Kai the AI

Technical Documentation

Chi Square Labs

2025-11-17

Table of contents

1. Kai the AI - Technical Documentation

Intelligent Teaching Assistant

Welcome

Welcome to the comprehensive technical documentation for **Kai the AI**, an intelligent teaching assistant developed by Chi Square Labs.

1.1. About This Documentation

This book provides complete technical documentation for Kai the AI, including:

- **Getting Started**: Introduction to Kai and quick start guides
- **API Reference**: Detailed API documentation and integration guides
- **Advanced Topics**: Migration guides and advanced configuration

1.2. About Kai the AI

Kai is an intelligent teaching assistant designed to enhance the learning experience by providing personalized support, answering questions, and facilitating interactive learning sessions.

1.3. Getting Help

If you need assistance or have questions:

- Visit our main website: chi2labs.com
- View the online documentation: chi2labs.com/docs
- Contact support: support@chi2labs.com

1.4. Version Information

- **Documentation Version**: 1.0.0
- **Last Updated**: 2025-11-17
- **Format**: PDF and HTML

1.5. Quick Navigation

Use the sidebar to navigate through different sections, or jump to:

- What is Kai? - Learn about Kai's features and capabilities
- Getting Started - Get up and running quickly
- API Reference - Detailed API documentation

Part I.

Introduction

2. What is Kai

An Introduction to Kai the AI

2.1. Your Autonomous Adaptive Teaching Assistant

Kai is an **Autonomous Adaptive Agent** that works as your AI teaching assistant—listening to lectures, learning from your teaching style, and continuously adapting to meet the evolving needs of both instructors and students.

Unlike generic AI tools, Kai integrates seamlessly with your existing lecture capture and learning management systems, transforming your lectures into an intelligent knowledge base while automating time-consuming administrative tasks. Kai operates autonomously in the background, freeing you to focus on what matters most: inspiring and educating your students.

2.2. How to Connect with Kai

Kai meets you where you work, offering multiple convenient ways to interact:

- **Instructor App**: Download the dedicated mobile app for on-the-go access to Kai's features and real-time notifications
- **Email Integration**: Simply email Kai using your institutional email address, and receive responses directly in your inbox
- **Chat Interfaces**: Connect through your preferred messaging platforms for quick queries and updates

2.3. How Kai Works

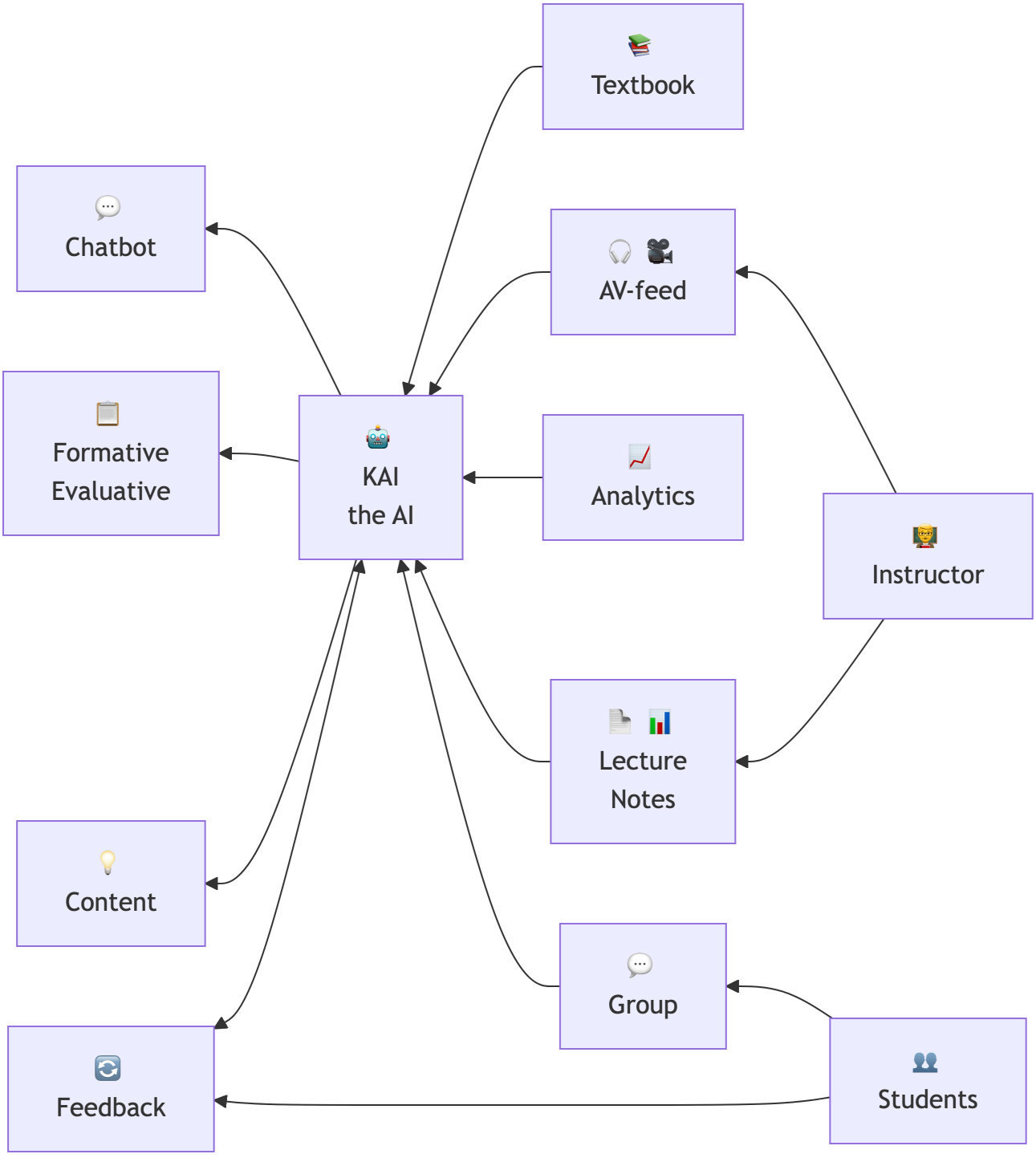
As an **autonomous agent**, Kai continuously gathers and analyzes information from multiple sources across your educational ecosystem, **adapting** its responses and recommendations based on what it learns:

Data Sources:

- **Lecture capture recordings** (Kai's primary learning source—it "attends" your classes)
- Student feedback and evaluations
- Learning analytics and engagement metrics
- Learning Management System (LMS) data
- Pop quiz results and assessments

- Course materials and curricula

This comprehensive, **adaptive** approach enables Kai to build an intelligent, course-specific knowledge base that evolves with your teaching—serving as a personalized resource hub for your students while providing you with actionable insights to improve learning outcomes.



Kai Overview

2.4. How Kai Helps Instructors Save Time

As an **autonomous assistant**, Kai transforms raw educational data into practical support—**taking over time-consuming tasks** so you can focus on teaching:

Lecture-Powered Student Support: Using insights from your lectures, Kai provides 24/7 answers to student questions based on YOUR course content—reducing repetitive email inquiries

Automated Reporting: Receive regular summaries of class performance, attendance patterns, and engagement metrics without manual data compilation

Adaptive Recommendations: Get data-driven suggestions for improving course delivery and addressing learning gaps, informed by lecture content and student interactions

Grading Assistance: Streamline repetitive grading tasks while maintaining consistency and your standards

Early Intervention: Kai autonomously identifies struggling students through learning analytics and can initiate timely interventions, helping you provide support exactly when it's needed

2.5. How Kai Helps Students Learn

Kai serves students as an **always-available learning companion** that adapts to their individual needs:

Course-Specific Answers: Get instant help based on lecture content and course materials—not generic AI responses

Personalized Learning: Receive tailored explanations and resources that adapt to your learning style and current understanding

Seamless Access: Connect with Kai through the platforms you already use—no new apps or logins required

3. Understanding Kai's Capabilities

An Intelligent Assistant That Knows Its Strengths

Kai is designed with built-in awareness of its own capabilities and limitations. This self-awareness means you can discover what Kai can do for you in the most natural way possible—simply by asking.

3.1. How to Learn What Kai Can Do

3.1.1. Browse the Documentation

You can explore Kai's features systematically through this documentation, where each capability is explained in detail with examples and best practices.

3.1.2. Ask Kai Directly

Alternatively, you can have a conversation with Kai about its abilities. Simply ask questions like:

- "How can I enroll students?"
- "Can you help me create quiz questions?"
- "What types of reports can you generate?"
- "Are you able to schedule office hours?"

Kai will provide clear, helpful responses about what it can and cannot do, guiding you naturally toward the features that best support your teaching needs.

3.2. Consistent Intelligence Across All Platforms

Whether you prefer the mobile app, email, or messaging apps like WhatsApp, Kai's self-awareness remains consistent. The same intelligent understanding of capabilities is available regardless of how you choose to interact:

3.2.1. Chat Interface (Mobile App)

The Kai mobile app provides an intuitive chat interface for immediate, conversational interactions with full feature discovery.

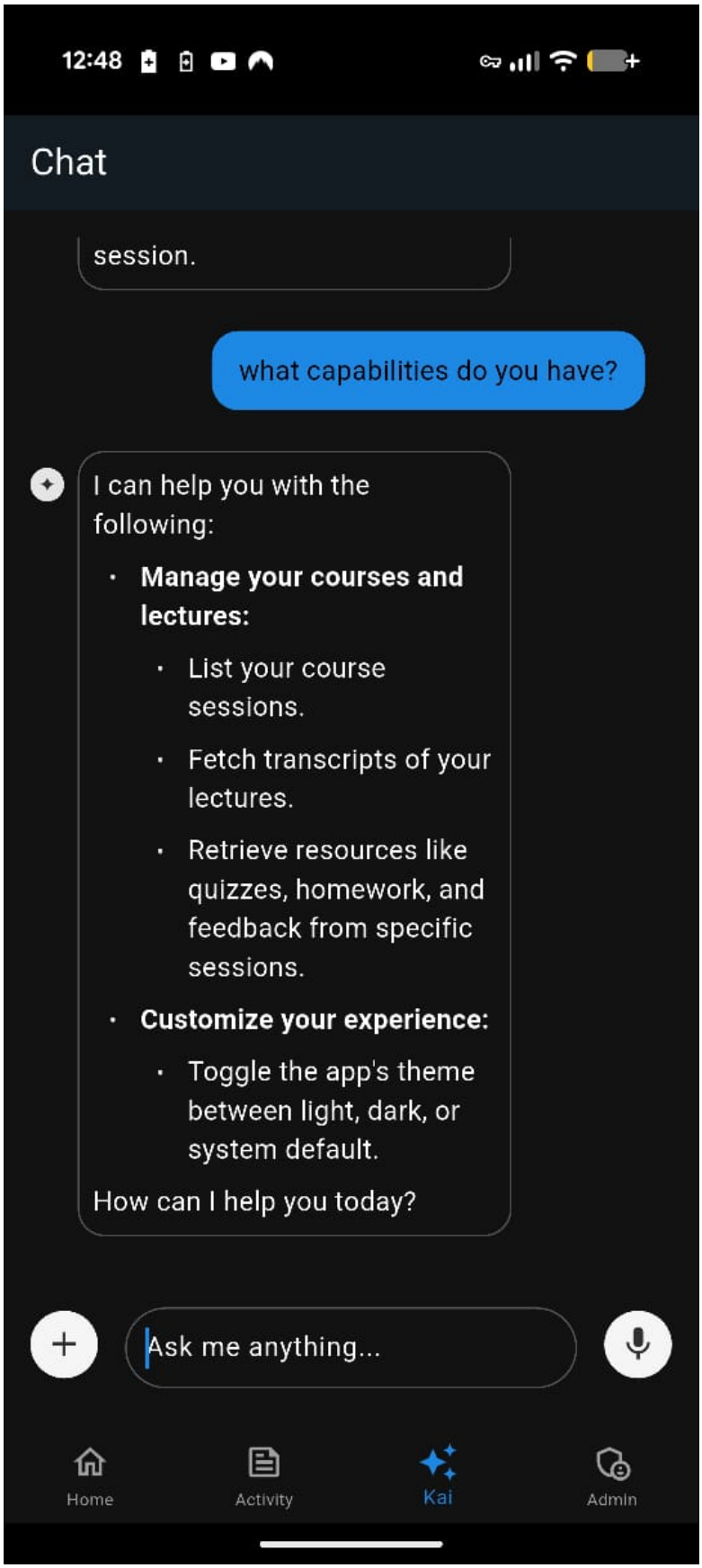


Figure 3.1.: Chat interface in the Kai mobile app showing capability discovery

[Screenshot: Chat interface showing a user asking "How can I enroll students?" with Kai's response]

3.2.2. Email Communication

Send your questions to Kai via email and receive detailed responses about available features and how to use them.

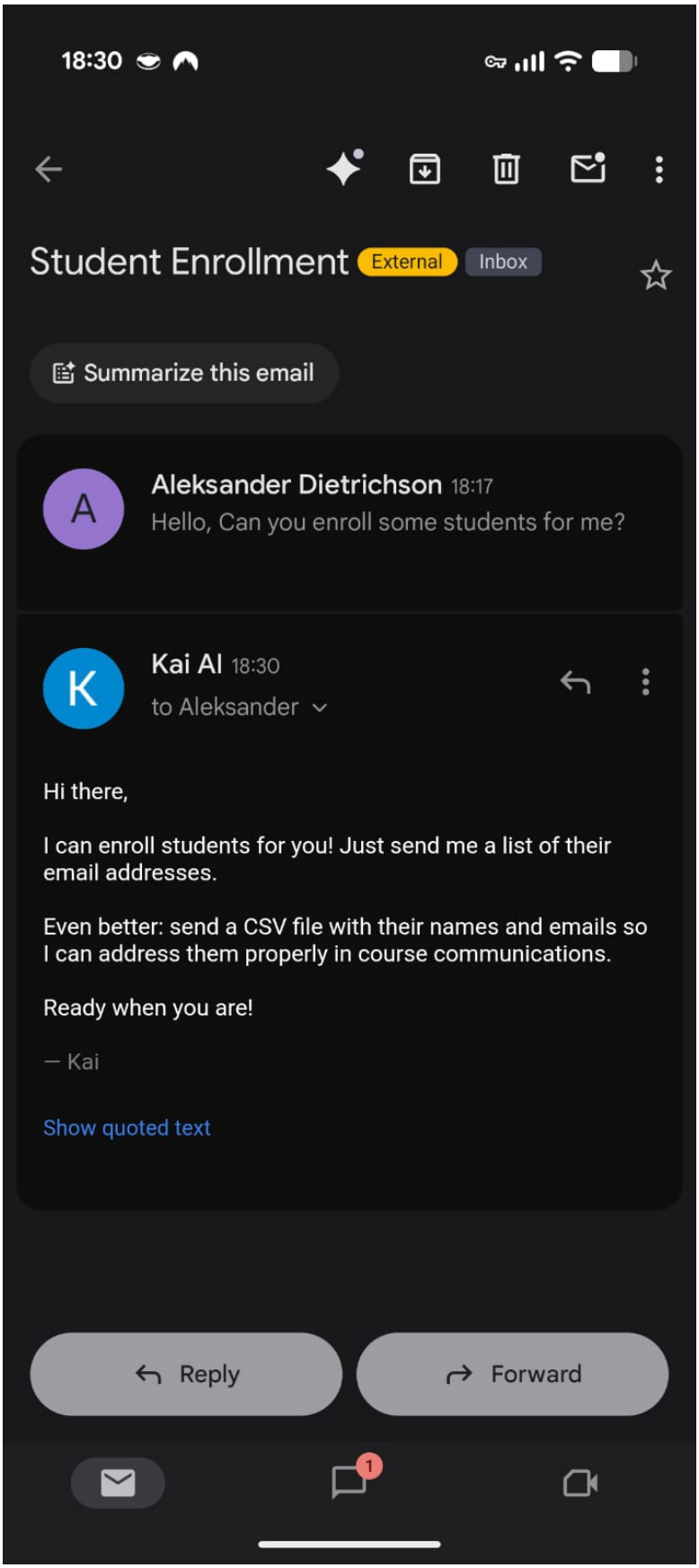


Figure 3.2.: Email showing Kai responding to a question about student enrollment

Email thread showing Kai's response to an enrollment inquiry

3.2.3. Messaging Apps

Connect through WhatsApp or other supported messaging platforms for quick capability checks on the go.

[Screenshot: WhatsApp conversation about Kai's features]

3.3. Natural Language Understanding

Kai interprets your questions intelligently, so you don't need to use specific keywords or phrases. Whether you ask "Can you grade essays?" or "I need help evaluating student writing," Kai understands your intent and responds appropriately.

This natural interaction design means you can focus on your teaching goals rather than learning complex commands or navigating through menus. Kai adapts to your communication style, making the discovery of its capabilities as effortless as having a conversation with a knowledgeable colleague.

3.4. Contextual Help Within the App

Throughout Kai's interface, you'll find **info icons ( )** that provide contextual help exactly when you need it. These unobtrusive icons appear next to features, settings, and options, offering quick explanations without requiring you to leave your current task or search through documentation.

Simply hover over or tap any info icon to see a helpful tooltip explaining the feature's purpose, how to use it effectively, and any relevant tips. This contextual approach ensures that help is always available at your fingertips, reducing the learning curve and helping you make the most of Kai's capabilities as you work.

Note: Kai will always be transparent about its limitations. If you ask for something outside its current capabilities, it will clearly explain what it cannot do and suggest alternative approaches when possible.

4. Getting Help

Multiple Ways to Get the Assistance You Need

When you need help with Kai, you have several convenient options available:

4.1. Ask Kai Directly

The fastest way to get help is often to ask Kai itself. Kai has built-in awareness of its capabilities and can guide you through any feature or task.

4.1.1. Chat Interface

Open the web-based chat and simply ask your question in natural language. Kai will provide immediate, helpful responses about what it can do and how to do it.

4.1.2. Email

Send your questions to Kai via email for detailed responses that you can reference later.

4.1.3. Messaging Apps

Connect through WhatsApp or other supported messaging platforms for quick help on the go.

4.2. Search Documentation

This documentation site includes comprehensive guides and references for all of Kai's features. Use the search bar to quickly find specific topics or browse through the organized sections.

4.3. Contact Support

If you're experiencing technical issues or need assistance beyond what Kai can provide directly:

Email: support@chi2labs.com

Our support team is ready to help with: - Technical troubleshooting - Account and billing questions - Feature requests and feedback - Training and onboarding assistance

---

Remember: Kai is designed to be self-explanatory. Before reaching out to support, try asking Kai directly—you might be surprised by how much it can help you on its own!

Part II.

Getting Started

5. Getting Started with Kai

Set up Kai the AI in your institution

5.1. Overview

Kai the AI is designed to integrate seamlessly into your existing educational infrastructure. This guide will walk you through the initial setup process.

5.2. Prerequisites

Before you begin, ensure you have:

- Administrative access to your institution's LMS
- A verified educator account with Chi Square Labs
- Basic familiarity with your LMS platform (Canvas, Blackboard, Moodle, etc.)

5.3. Installation Steps

5.3.1. Step 1: Create Your Institution Account

1. Visit chi2labs.com/signup
2. Select "Institution Setup"
3. Enter your institution details
4. Verify your email

> **Multi-user Setup**
>
> If you're setting up Kai for multiple educators, choose the "Department License" option for easier management.

5.3.2. Step 2: LMS Integration

5.3.2.1. For Canvas Users

```
// Add this to your Canvas custom JavaScript
const kaiConfig = {
  institution: "your-institution-id",
  apiKey: "your-api-key",
  features: ["grading", "content", "analytics"]
};
```

5.3.2.2. For Blackboard Users

1. Navigate to System Admin → Building Blocks
2. Upload the Kai integration package
3. Configure with your API credentials

5.3.3. Step 3: Configure Kai Features

Once connected, you can enable specific features:

Feature	Description	Setup Time
Smart Grading	AI-assisted grading with feedback	5 minutes
Content Generation	Create quizzes and assignments	2 minutes
Analytics Dashboard	Student performance insights	10 minutes
Office Hours Bot	24/7 student support	15 minutes

5.3.4. Step 4: Test the Integration

! Testing Checklist

- ☐ Create a test assignment
- ☐ Submit a sample student response
- ☐ Verify grading suggestions appear
- ☐ Check analytics dashboard
- ☐ Test student-facing features

5.4. Next Steps

- Configure grading rubrics
- Set up content templates
- Train Kai on your teaching style

5.5. Troubleshooting

5.5.1. Common Issues

5.5.2. Connection Errors

If you see "Unable to connect to Kai":

1. Verify your API key is correct
2. Check your institution's firewall settings
3. Ensure you're using HTTPS

5.5.3. Missing Features

Some features may require:

- Specific LMS permissions
- Account tier upgrade
- Additional configuration

5.6. Support

Need help? We're here for you:

- support@chi2labs.com
- Live chat: Available 9 AM - 6 PM EST
- Video tutorials

6. Logging In to Kai

Quick and Easy Access with Your Institutional Email

6.1. Quick Login

Getting started with Kai is simple and takes just seconds. We recommend using your **institutional email** (whether Google, Microsoft, or another provider) for seamless access.

Why Use Your Institutional Email?

If your institution has Kai installed with Single Sign-On (SSO), using your institutional email provides:

- **Instant access** - No separate password to remember
- **Automatic enrollment** - Direct connection to your courses
- **Secure authentication** - Leverages your institution's security

6.2. Login Process

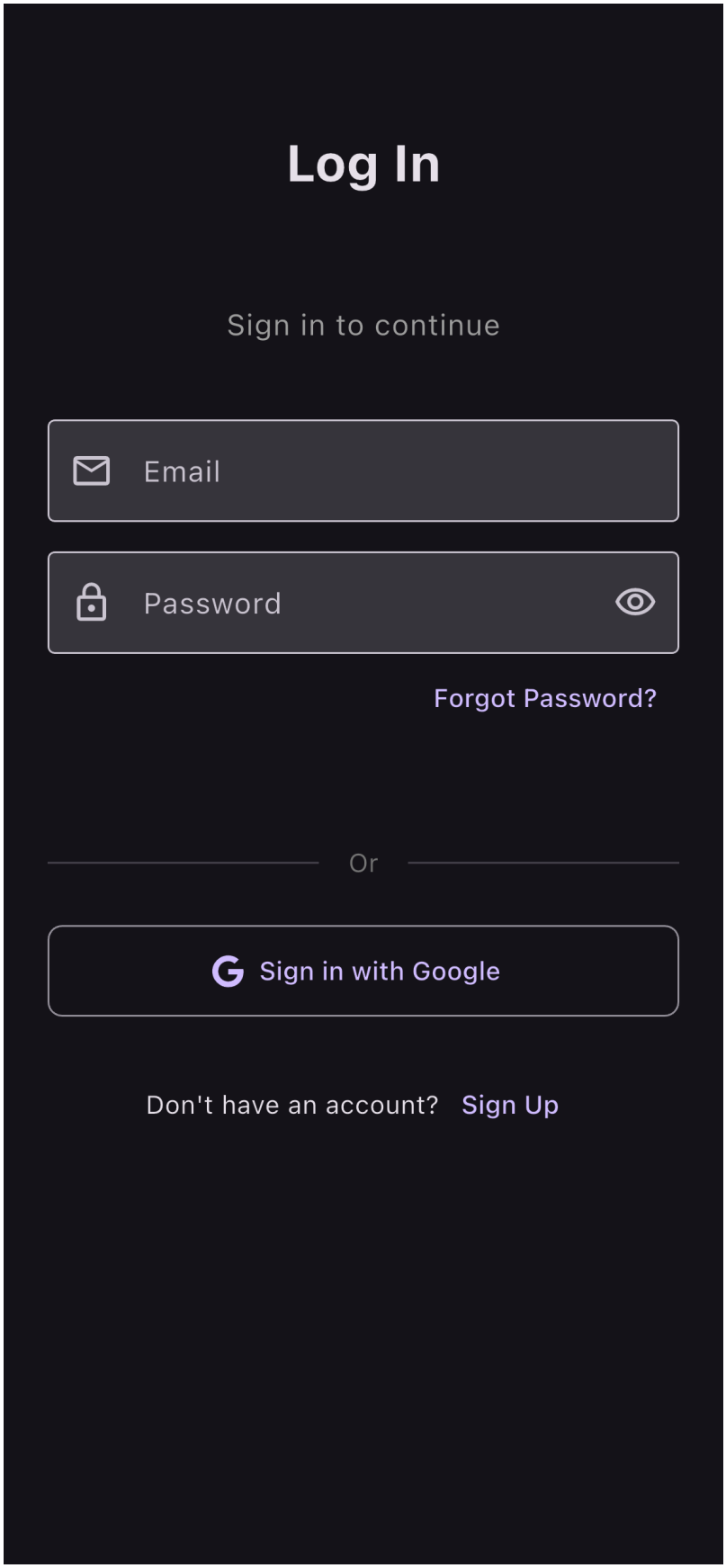


Figure 6.1.: Step 1: Select Login Method

Step 1: Choose your email provider

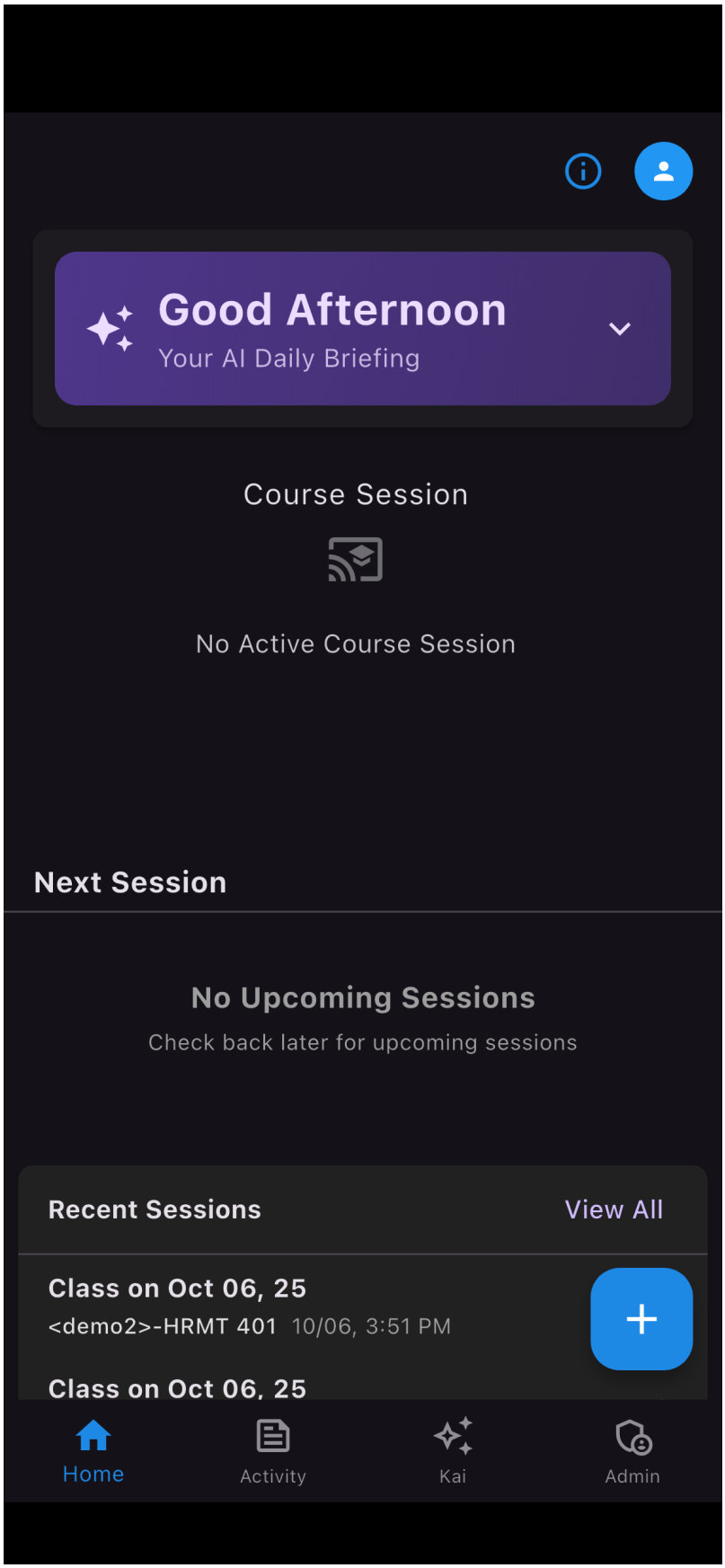


Figure 6.2.: Step 2: Authenticate

Step 2: Sign in with institutional credentials

That's it! Once authenticated, you'll be taken directly to your Kai dashboard with access to all your courses and features.

6.3. Supported Login Methods

- **Google** (recommended for G Suite institutions)
- **Microsoft** (recommended for Office 365 institutions)
- **Other institutional SSO providers**

Logging in takes less than 30 seconds. No complicated setup, no extra passwords to remember—just use your institutional email and get started.

7. Installation Guide

Step-by-step installation instructions for Kai the AI

7.1. Overview

This guide walks you through the complete installation process for Kai the AI, from initial account setup to full LMS integration.

7.2. Prerequisites

Before you begin, ensure you have:

- Administrative access to your Learning Management System (LMS)
- Institution email address for verification
- Basic familiarity with your LMS administration panel
- Permission to install third-party integrations

> **Estimated Setup Time**
>
> - **Basic Setup**: 10-15 minutes
> - **Full LMS Integration**: 30-45 minutes
> - **Team Onboarding**: 1-2 hours

7.3. Installation Steps

7.3.1. Step 1: Create Your Account

1. **Visit the signup page**: chi2labs.com/signup
2. **Choose your account type**:
    - Individual Educator (single user)
    - Department License (multiple educators)
    - Institution License (campus-wide)
3. **Enter institution details**:

- Institution name
- Your role (instructor, TA, administrator)
- Department/subject area

4. **Verify your email**: Check your inbox for the verification link

> Choosing the Right License
>
> - **Individual**: Best for trying out Kai or single-course use
> - **Department**: Ideal for 5-20 educators sharing best practices
> - **Institution**: Enterprise features, SSO, dedicated support

7.3.2. Step 2: Configure Your Profile

1. Navigate to **Settings** → **Profile**
2. Complete your teaching profile:
   - Subject areas taught
   - Typical class sizes
   - Teaching format (in-person, hybrid, online)
3. Set your preferences:
   - Notification frequency
   - Default grading rubric style
   - Privacy settings

7.3.3. Step 3: LMS Integration

Kai integrates with all major Learning Management Systems:

7.3.3.1. Canvas Integration

Option A: Using LTI (Recommended)

1. In Canvas, navigate to **Admin** → **Developer Keys**
2. Click **+ Developer Key** → **+ LTI Key**
3. Enter Kai's LTI configuration:

```
Name: Kai the AI
Redirect URIs: https://kaitheai.com/lti/callback
JWK Method: Public JWK URL
JWK URL: https://kaitheai.com/.well-known/jwks.json
```

4. Copy the generated Client ID
5. In Kai Settings, paste the Client ID under **Canvas Integration**

Option B: Using API Key

1. In Canvas, go to **Account** → **Settings** → **Approved Integrations**
2. Click **+ New Access Token**
3. Name: "Kai the AI", Expiration: Never
4. Copy the token
5. In Kai Settings, paste under **Canvas API Integration**

> ⚠ API Token Security
>
> Treat your Canvas API token like a password. Never share it or commit it to version control.

7.3.3.2. Blackboard Integration

1. Navigate to **System Admin** → **Integrations** → **Building Blocks**
2. Click **Upload Building Blocks**
3. Download Kai's Building Block from your dashboard
4. Upload the `.war` file
5. Configure settings:
   - API Endpoint: `https://kaitheai.com/api/v1`
   - Institution Code: (provided in your Kai dashboard)
   - API Key: (generated automatically)

7.3.3.3. Moodle Integration

1. Go to **Site Administration** → **Plugins** → **Install plugins**
2. Upload Kai plugin ZIP file (download from Kai dashboard)
3. Click **Install plugin from ZIP file**
4. Follow the configuration wizard:
   - Enter your Kai API key
   - Select features to enable
   - Configure webhook URL (automatic)

7.3.3.4. Google Classroom Integration

1. In Kai dashboard, click **Integrations** → **Google Classroom**
2. Click **Connect to Google**
3. Authenticate with your Google Workspace account
4. Grant permissions:
   - View courses and rosters
   - Create and grade assignments
   - Post announcements
5. Select which classes to sync with Kai

> **Multiple LMS Support**
>
> You can connect Kai to multiple LMS platforms simultaneously. Each integration is independent and can be configured separately.

7.3.4. Step 4: Mobile App Setup

The Kai mobile app enables real-time student engagement features.

7.3.4.1. For Instructors

iOS: 1. Download from App Store 2. Sign in with your Kai account 3. Enable notifications for student feedback 4. Sync with your LMS courses

Android: 1. Download from Google Play 2. Sign in with your Kai account 3. Enable notifications 4. Sync courses

7.3.4.2. For Students

Distribute these instructions to your students:

Installation: 1. Download "Kai Student" from App Store or Google Play 2. Use enrollment code: `[Found in your Kai dashboard]` 3. Or join via course link: `[Auto-generated per course]`

First-Time Setup: 1. Create student account with .edu email 2. Enter course enrollment code 3. Enable push notifications (required for feedback features) 4. Complete profile (anonymous ID or name, based on your settings)

7.3.5. Step 5: Feature Configuration

Enable and configure specific features:

7.3.5.1. Smart Grading

1. **Settings → Grading → Enable Smart Grading**
2. Configure grading preferences:
    - Feedback style: Detailed | Balanced | Concise
    - Auto-grade threshold: 0-100 (confidence level)
    - Review required: Always | Only low confidence | Never
3. Import or create grading rubrics
4. Test with sample assignments

7.3.5.2. Feedback Workflow

1. **Settings → Workflows → Enable Feedback**
2. Configure timing:
   - Suggested interval: 30 minutes (adjustable)
   - Auto-request on: Time | Content transition | Engagement drop
3. Set notification preferences for students
4. Customize feedback questions (optional)

7.3.5.3. Pop Quiz

1. **Settings → Workflows → Enable Pop Quiz**
2. Configure quiz settings:
   - Default question count: 5
   - Difficulty adaptation: Enabled
   - Time limit per question: 60 seconds
3. Set question bank sources:
   - Course materials
   - Textbook chapters
   - Custom topics

7.3.5.4. SafeStream

1. **Settings → Workflows → Enable SafeStream**
2. Configure moderation rules:
   - Sensitivity level: Low | Medium | High
   - Auto-flag keywords: (institution-specific)
   - Notification recipients: Instructors | Admin | Both
3. Review privacy policy compliance
4. Test with sample content

7.3.6. Step 6: Test Your Installation

> **!** Pre-Launch Checklist
>
> Complete these tests before using Kai in a live class:

LMS Integration Test: - [ ] Create a test assignment in your LMS - [ ] Verify it appears in Kai dashboard - [ ] Submit a sample response - [ ] Check that grading suggestions appear - [ ] Confirm grade syncs back to LMS

Mobile App Test: - [ ] Send test feedback request - [ ] Verify students receive push notification - [ ] Submit test feedback as student - [ ] Check feedback appears in instructor dashboard

Analytics Test: - [ ] Generate sample data (use test accounts) - [ ] View analytics dashboard - [ ] Export sample report - [ ] Verify data accuracy

7.4. Post-Installation

7.4.1. Onboarding Your Students

1. **First Day Introduction** (5 minutes):
   - Explain what Kai does and why you're using it
   - Demonstrate the feedback process
   - Address privacy concerns
   - Share enrollment instructions
2. **First Week Support**:
   - Monitor app installation rates
   - Send reminders to students who haven't enrolled
   - Answer questions in first class
   - Test all features with low stakes

7.4.2. Training Resources

- Video Walkthrough
- Student Onboarding Template
- First Week Guide

7.5. Troubleshooting Installation

7.5.1. Common Issues

7.5.1.1. "API Key Invalid" Error

Solution: 1. Verify you copied the complete API key 2. Check for extra spaces at beginning/end 3. Regenerate key if needed: Settings → API → Regenerate

7.5.1.2. LMS Integration Not Syncing

Solution: 1. Check firewall settings (Kai requires HTTPS) 2. Verify API permissions are correct 3. Test connection: Settings → Integrations → Test Connection 4. Review integration logs for specific errors

7.5.1.3. Students Can't Enroll

Solution: 1. Verify course is published in Kai 2. Check enrollment code hasn't expired 3. Ensure students are using correct app (Student vs. Instructor) 4. Confirm students have accepted app permissions

7.5.1.4. Mobile Notifications Not Working

Solution: 1. Check notification settings in phone settings 2. Verify app has notification permission 3. Test with a manual notification 4. Check Do Not Disturb settings

7.6. Support

7.6.1. Need Help?

- **Email**: support@chi2labs.com
- **Live Chat**: Available in dashboard (9 AM - 6 PM EST)
- **Video Support**: Schedule a call
- **Knowledge Base**: help.chi2labs.com

7.6.2. Enterprise Support

Institution license holders have access to: - Dedicated support representative - Priority response (< 2 hour SLA) - Custom integration assistance - On-site training sessions

7.7. Next Steps

Now that Kai is installed:

1. **Complete the Quick Start Guide**
2. **Explore Best Practices**
3. **Join the Educator Community**
4. **Schedule your first feedback session**

Installation taking longer than expected? Our support team is here to help get you up and running quickly.

8. Quick Start Guide

Get started with Kai in 15 minutes

8.1. Welcome!

This quick start guide will have you running your first Kai-enhanced class session in about 15 minutes. We'll focus on the most impactful feature: real-time student feedback.

> What You'll Accomplish
>
> By the end of this guide, you'll: - Have Kai connected to your course - Know how to request student feedback - Understand how to act on insights - Be ready for your first enhanced class session

8.2. Prerequisites

Before you begin: - Kai account created (Sign up here) - At least one course in your LMS - 5-10 students willing to install the mobile app (for testing)

8.3. 15-Minute Setup

8.3.1. Step 1: Connect Your Course (3 minutes)

1. **Log in to Kai**: dashboard.kaitheai.com
2. **Click "Add Course"** in the top right
3. **Select your LMS** (Canvas, Blackboard, Moodle, etc.)
4. **Authenticate** with your LMS credentials
5. **Choose the course** you want to enhance
6. **Click "Connect Course"**

> Already have LMS integration?
>
> If you completed the full Installation Guide, your courses should already be synced. Skip to Step 2.

8.3.2. Step 2: Invite Students (2 minutes)

Kai provides multiple ways to invite students:

Option A: Share Enrollment Link (Easiest) 1. In Kai dashboard, click your course name 2. Copy the enrollment link 3. Post in your LMS announcement or email students 4. Students click link → install app → automatic enrollment

Option B: Share Enrollment Code 1. Find your 6-digit course code in dashboard 2. Students download "Kai Student" app 3. Students enter code to join

Sample Announcement:

```
Welcome to [Course Name]!

This semester, we're using Kai the AI to make our class more
interactive and responsive to your needs.

To get started:
1. Install "Kai Student" from the App Store or Google Play
2. Click this link to join: [paste enrollment link]
3. Enable notifications

This helps me understand what's clear and what needs review,
so we can use our time together more effectively.

Questions? Ask in class or email me!
```

8.3.3. Step 3: Configure Feedback (5 minutes)

Let's set up your first feedback workflow:

1. **Go to Settings → Workflows → Feedback**
2. **Enable Feedback Workflow**
3. **Configure basic settings**:
    - **Suggested interval**: 30 minutes (Kai will remind you)
    - **Auto-request**: Leave OFF for your first time (manual control)
    - **Response time**: 2 minutes (how long students have to respond)
4. **Customize feedback questions** (optional):
    - Default: "What concepts are clear? What's confusing?"
    - Or create your own questions
5. **Save settings**

> Start Simple
>
> For your first session, use the default settings. You can customize later once you're comfortable with the workflow.

8.3.4. Step 4: Install Mobile App (2 minutes)

As an instructor, you'll use the mobile app to request feedback during class:

1. **Download "Kai Instructor"** from App Store or Google Play
2. **Sign in** with your Kai account
3. **Enable notifications**
4. **Select your course** from the list
5. **You're ready!**

8.3.5. Step 5: Test the System (3 minutes)

Before your first real class, do a test run:

1. **Ask 2-3 students to install the app** during office hours or before class
2. **Request test feedback**:
    - Open Kai Instructor app
    - Tap "Request Feedback"
    - Confirm the request
3. **Have students respond** with test data
4. **Review the results** in your dashboard:
    - See which topics they marked as clear/confusing
    - View the analysis (individual vs. class-wide issues)
    - Check that notifications worked

8.4. System Check

If you received student responses and saw them in your dashboard, you're all set!

8.5. Your First Enhanced Class

8.5.1. Before Class

1. **Remind students to install the app** (if they haven't already)
2. **Open Kai Instructor app** on your phone
3. **Have your laptop/tablet open** to the Kai dashboard (optional, for detailed analytics)

8.5.2. During Class

8.5.2.1. ~20 Minutes In

1. **Request feedback**:
    - Pull out your phone
    - Open Kai app
    - Tap "Request Feedback"
    - Students get push notification
    - Continue teaching while they respond (2 minutes)
2. **Quick glance at results**:
    - App shows real-time response count
    - Red flags = many students confused
    - Green checks = concept is clear

8.5.2.2. Act on Insights

If many students flagged a concept (e.g., 60%+ confused):

```
"I see a lot of you found [concept] confusing.
Let me revisit that from a different angle..."
```

If only a few students struggling (e.g., <20%):

```
"Most of you have [concept] down. For those who want
extra practice, I've just sent a resource link to
those specific students."
```

8.5.2.3. ~40 Minutes In

Repeat the feedback process. This time you might discover: - Your re-explanation worked (fewer confused students) - A new concept needs attention - Overall comprehension is high (move forward confidently)

8.5.3. After Class

1. **Review detailed analytics**:
    - Which concepts caused most confusion
    - Individual student patterns
    - Trends over time
2. **Send targeted resources**:
    - Kai can suggest relevant materials
    - Send to specific students or entire class

- Schedule for optimal timing

3. **Adjust next class**:
   - Plan to review high-confusion topics
   - Skip concepts that were universally clear
   - Prepare alternative explanations

8.6. Real-World Example

Professor Chen's Statistics Class:

Before Kai: - Covered entire chapter regardless of understanding - Found out students were confused only during exam - Spent office hours re-teaching to everyone

With Kai: - 20 min: Requested feedback after introducing "standard error" - Result: 70% of students confused - Action: Spent 10 minutes re-explaining with new example - 40 min: Requested feedback again - Result: Only 15% still confused - Action: Sent targeted video to those 15%, moved forward for rest

Outcome: - Class time used efficiently - Struggling students got help immediately - Rest of class didn't sit through redundant review - Exam scores improved 12% over previous semester

8.7. Quick Reference Card

Print this for your desk:

```
  KAI QUICK REFERENCE

Request Feedback:
  1. Open Kai app
  2. Tap "Request Feedback"
  3. Wait 2 minutes

Read Results:
    Red = many confused
    Yellow = some confused
    Green = mostly clear

Act on Results:
  High %: Review in class
  Low %: Send individual resources

Best Practice:
  Request every 20-30 minutes
```

8.8. Common First-Time Questions

Q: What if students don't have the app installed? A: Kai works with partial participation. Even 30-40% response rate gives valuable insights. Encourage installation over first week.

Q: Do I have to request feedback during class? A: No! You can also request feedback after class about pre-recorded content, readings, or homework.

Q: What if I forget to request feedback? A: If you enabled auto-suggest, Kai will send you a gentle reminder based on your configured interval.

Q: Can students see other students' responses? A: No. All feedback is private. You see aggregated patterns, not individual responses (unless configured otherwise).

Q: Does this work with large classes (100+ students)? A: Yes! In fact, it works especially well with large classes where individual check-ins are impossible.

8.9. Next Steps

Now that you've completed your first Kai session:

8.9.1. Immediate (This Week)

- Use feedback in your next 2-3 class sessions
- Monitor student app installation rate
- Adjust feedback timing based on your teaching rhythm

8.9.2. Short-term (Next 2 Weeks)

- Explore the Pop Quiz Workflow
- Review Best Practices
- Dive into analytics dashboard

8.9.3. Long-term (This Semester)

- Try Advanced Customization
- Explore API Integration
- Share your experience with colleagues

8.10. Getting Help

8.10.1. Resources

- Video: Your First Class with Kai
- Educator Community Forum
- Email Support

8.10.2. Live Support

- **Chat**: Available in dashboard (9 AM - 6 PM EST)
- **Phone**: 1-800-KAI-HELP (9 AM - 6 PM EST)
- **Video Call**: Schedule 15-min session

Congratulations! You're ready to teach your first Kai-enhanced class. Remember: start simple, build confidence, then explore advanced features.

Your students will appreciate the responsiveness, and you'll love the insights.

Part III.

User Guides

9. User Guide

Comprehensive guides for educators using Kai the AI

9.1. Overview

Welcome to the Kai the AI User Guide. This section provides comprehensive guides for educators to effectively integrate Kai into their teaching workflows.

9.2. Core Workflows

Kai provides several specialized workflows designed to enhance different aspects of teaching and learning:

9.2.1. Real-Time Feedback

Purpose: Get instant insights into student comprehension during class

The Feedback Workflow enables you to: - Request student feedback with one click - Identify topics that need class-wide review - Provide targeted resources for individual students - Adapt your teaching in real-time

Learn more about Feedback Workflow →

9.2.2. Formative Assessment

Purpose: Quick comprehension checks throughout lessons

The Pop Quiz Workflow helps you: - Generate instant quizzes based on recent content - Assess understanding before moving forward - Identify struggling students early - Track progress over time

Learn more about Pop Quiz Workflow →

9.2.3. Content Safety & Moderation

Purpose: Ensure appropriate and safe classroom interactions

The SafeStream Workflow provides: - Real-time content moderation for student submissions - Automated flagging of inappropriate content - Privacy-preserving monitoring - Compliance with institutional policies

Learn more about SafeStream Workflow →

9.3. Practical Guides

9.3.1. Getting Started

- Installation Guide
- Quick Start Tutorial
- First-Time Setup Checklist

9.3.2. Best Practices

- Effective Use Patterns
- Common Pitfalls to Avoid
- Optimization Tips

9.3.3. Integration

- LMS Integration Guide
- Custom Workflow Setup
- API Integration Examples

9.3.4. Troubleshooting

- Common Issues & Solutions
- Performance Optimization
- Support Resources

9.4. Teaching Scenarios

9.4.1. Scenario 1: Large Lecture Classes

Challenge: Maintaining engagement with 100+ students

Solution: Combine Feedback Workflow and Pop Quiz Workflow 1. Use Pop Quiz to check comprehension every 20 minutes 2. Request feedback at key concept transitions 3. Let Kai identify which topics need review 4. Provide targeted resources for struggling students

9.4.2. Scenario 2: Hybrid Learning

Challenge: Supporting both in-person and remote students

Solution: Leverage asynchronous features 1. Enable SafeStream for online discussions 2. Use Feedback Workflow across both groups 3. Track engagement metrics separately 4. Provide equitable support regardless of format

9.4.3. Scenario 3: Flipped Classroom

Challenge: Ensuring students complete pre-work

Solution: Pre-class assessment and adaptation 1. Assign pre-class content with embedded quizzes 2. Review completion and comprehension data 3. Adjust in-class activities based on results 4. Use class time for targeted support

9.5. Feature Matrix

Feature	Feedback	Pop Quiz	SafeStream
Real-time insights			
Student engagement			
Content moderation			
Analytics dashboard			
LMS integration			
Mobile app required			

Legend: Full support | Partial support

9.6. Tips for Success

Start Small

Don't try to implement all features at once. Start with one workflow that addresses your biggest challenge, master it, then expand to others.

Student Onboarding

Take time in the first week to introduce students to Kai. Show them how their participation helps improve the class experience for everyone.

! Data Privacy

Always review your institution's data privacy policies and ensure Kai is configured to comply with local regulations (FERPA, GDPR, etc.).

9.7. Next Steps

1. **New Users**: Start with the Quick Start Guide
2. **Experienced Users**: Explore Advanced Customization
3. **Developers**: Check out the API Reference
4. **Need Help**: Visit Troubleshooting

9.8. Additional Resources

- Video Tutorial Library
- Email Support
- Educator Community Forum
- Case Studies & Success Stories

10. Feedback Workflow

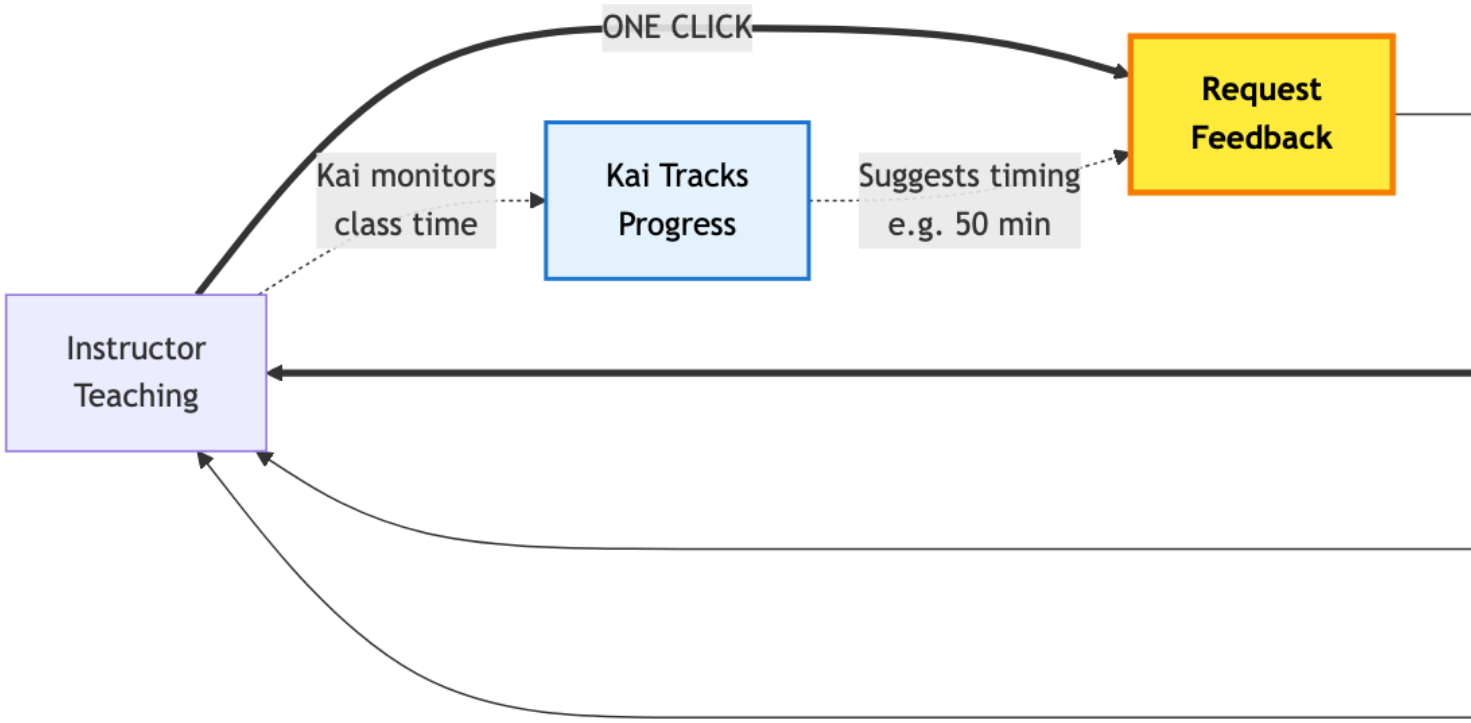
Real-Time Student Feedback and Adaptive Learning

10.1. Overview

The Feedback Workflow is incredibly simple: **One click gets you instant student insights**. While you teach, Kai monitors the class progress. With a single button press, you request feedback from students, and within moments you know exactly what they understood and what needs review. It's that simple - no forms to create, no data to analyze manually.

10.2. How It Works

Just teach, click, and get insights - the entire process is automated:



Student Feedback Workflow

The Entire Process in 3 Simple Steps

1. **You Teach** - Kai monitors class progress and timing
2. **You Click** - One button requests feedback from all students
3. **You Get Insights** - Instantly see what needs review vs. individual support

That's it! No surveys to design, no data to crunch, just actionable insights.

10.3. Workflow Steps

10.3.1. 1. Teaching and Recording

- Instructor conducts regular class session
- Kai monitors class duration and engagement metrics
- Class is recorded for later review and analysis

10.3.2. 2. Feedback Request Triggers

Two ways to initiate feedback:

- **Kai-Initiated**: Proactive recommendations based on:
  - Time elapsed (e.g., "50 minutes have passed")
  - Content complexity indicators
  - Student engagement metrics
- **Instructor-Initiated**: Manual request at any time through the app

10.3.3. 3. Student Feedback Collection

- Push notifications sent to all enrolled students with the app
- Students provide structured feedback:
  - Concepts they understood
  - Topics they found confusing
  - Specific areas requesting review

10.3.4. 4. Intelligent Analysis

Kai analyzes feedback patterns:

- **Frequency Analysis**: Identifies commonly misunderstood topics
- **Severity Assessment**: Evaluates depth of understanding gaps
- **Resource Allocation**: Determines optimal teaching strategy

10.3.5. 5. Adaptive Response

10.3.5.1. High-Frequency Issues (Many Students)

- **Action**: In-class review recommended
- **Example**: If 70% of students struggle with "standard error"
- **Result**: Instructor re-teaches concept to entire class
- **Benefit**: Efficient use of class time for maximum impact

10.3.5.2. Low-Frequency Issues (Few Students)

- **Action**: Individual online resources recommended
- **Example**: If 2 students struggle with "standard deviation"
- **Result**: Targeted students receive personalized content
- **Benefit**: Class time preserved for majority needs

10.4. See It In Action

The following screenshots demonstrate the instructor's feedback workflow from request creation through analysis:

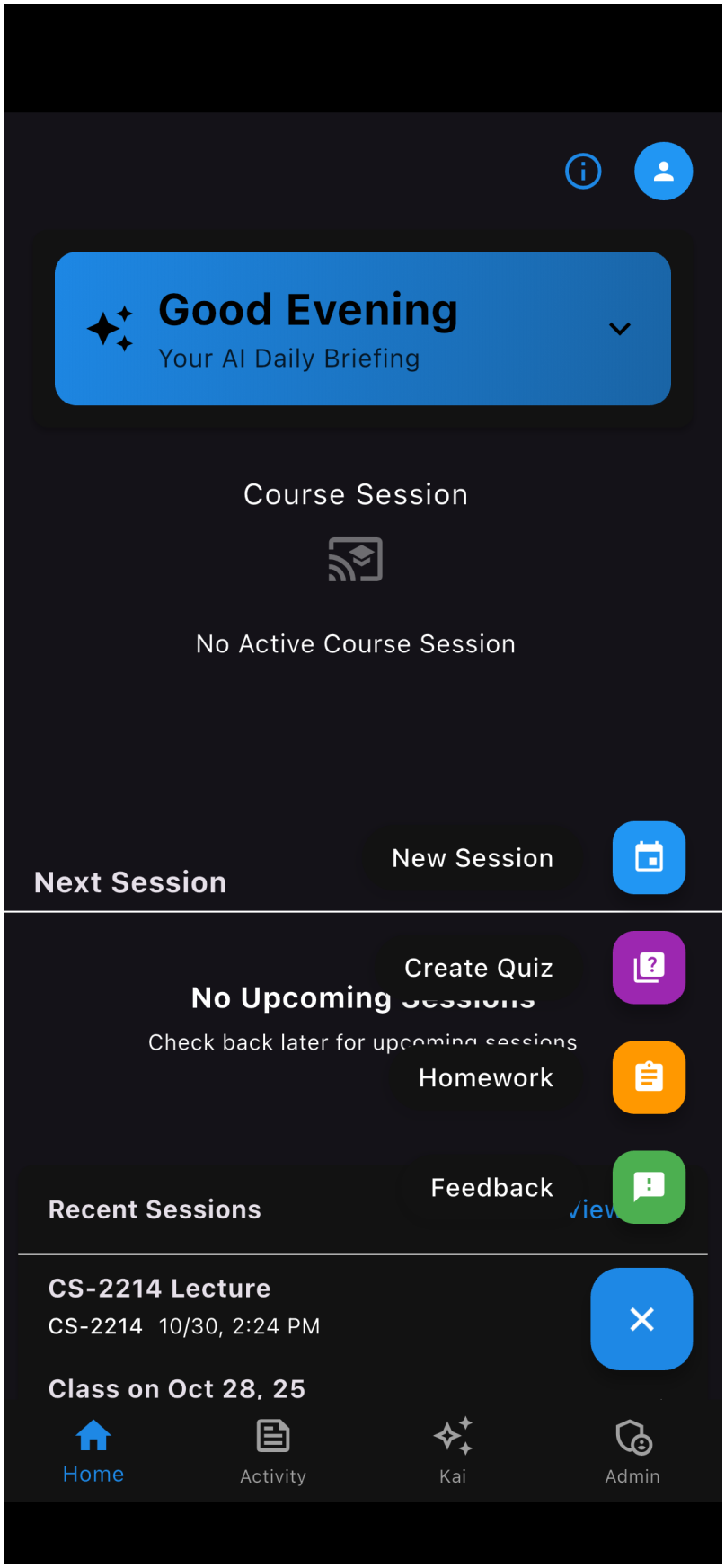


Figure 10.1.: Step 1: Create Feedback Request

Step 1: Create Feedback Request

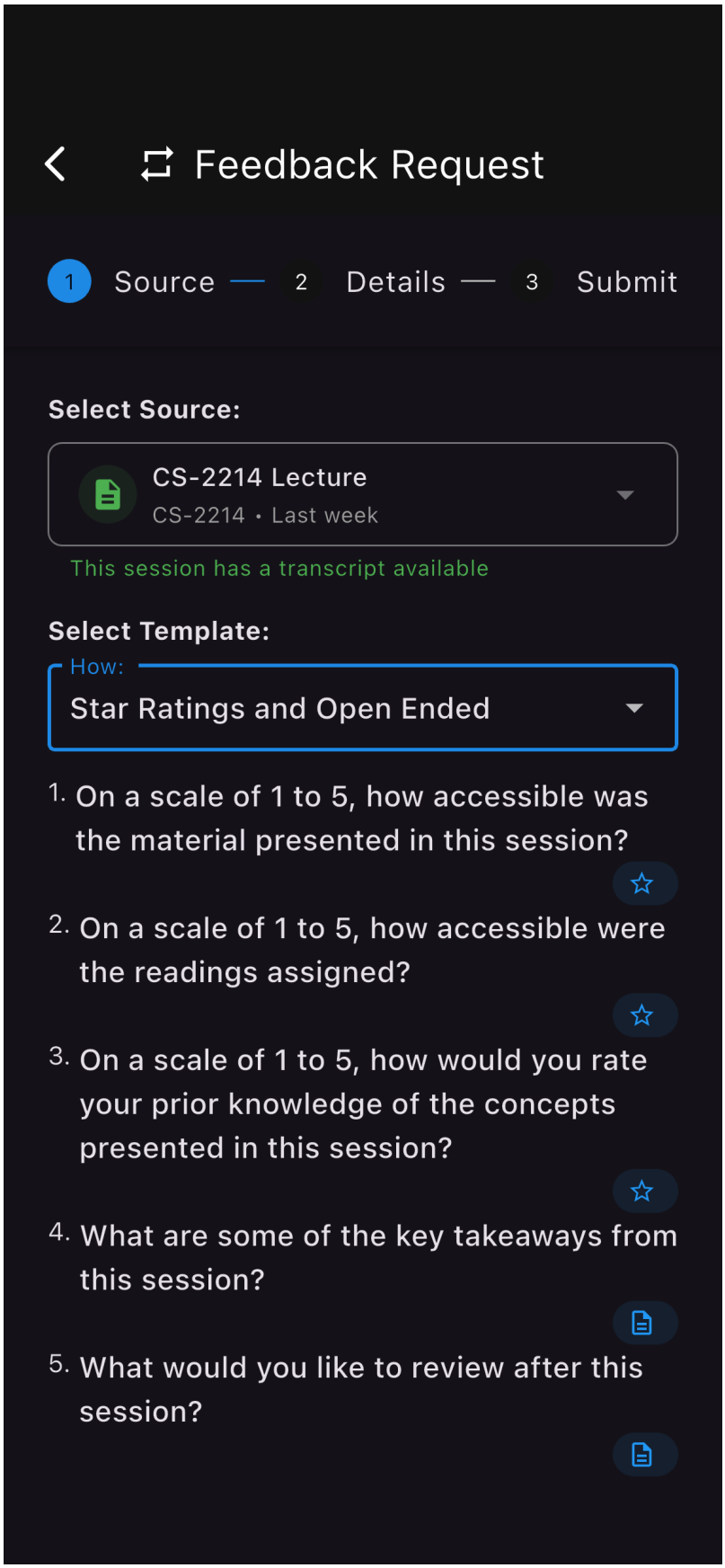


Figure 10.2.: Step 2: Configure Feedback Questions

Step 2: Configure Questions

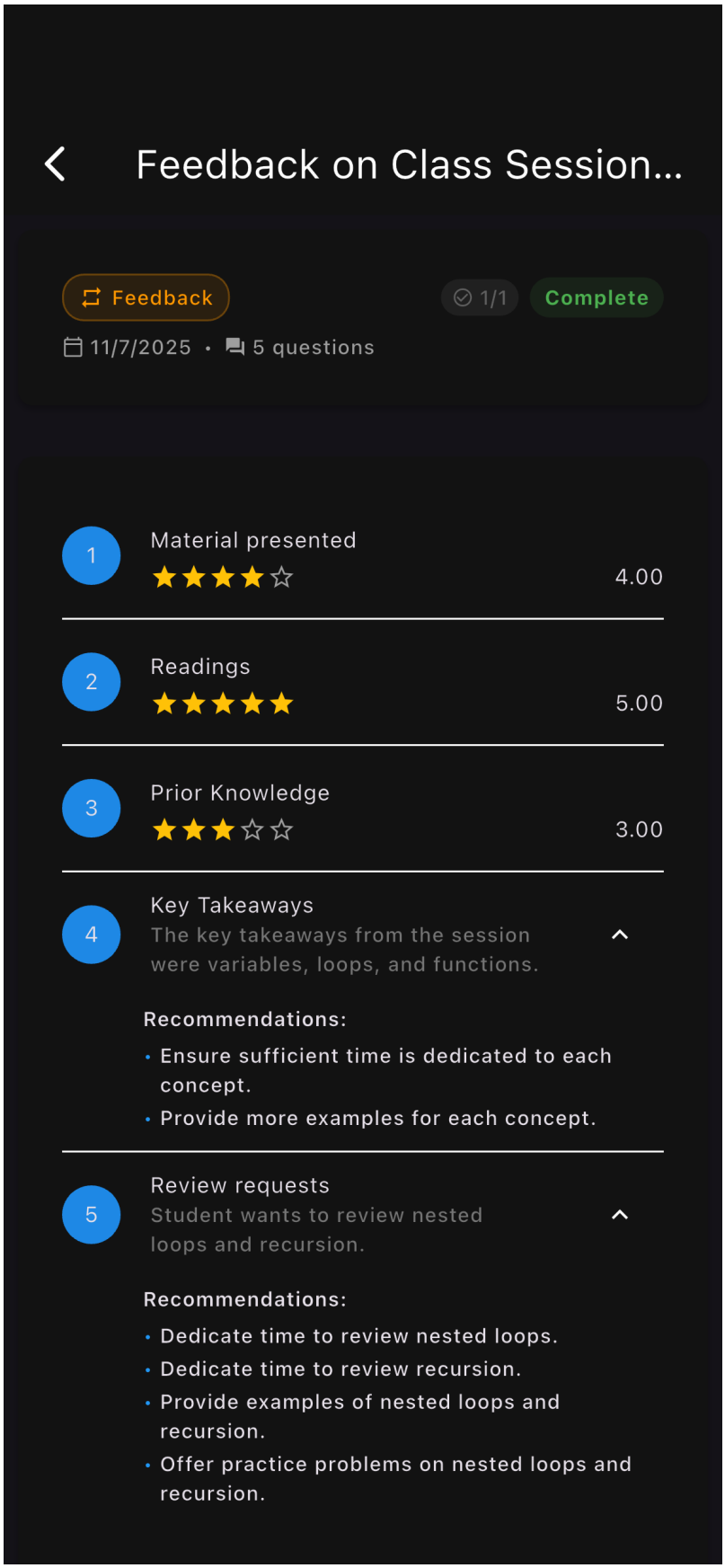


Figure 10.3.: Step 3: View Feedback Analysis

Step 3: View Analysis

10.5. Implementation Tips

10.5.1. Choosing the Right Feedback Template

Kai comes with pre-built feedback templates designed for different use cases. The **standard template** (shown in the screenshots above) is an excellent starting point for most courses:

- **3 Star-Rating Questions**: Quick quantitative assessment of comprehension
- **2 Open-Ended Questions**: Qualitative insights into student understanding

Why Open-Ended Questions Matter: While star ratings provide quick metrics, open-ended questions reveal the *specific* concepts causing confusion. This allows Kai to provide targeted recommendations rather than generic suggestions.

10.5.2. Template Consistency for Longitudinal Analysis

Pro Tip: Using the same or similar templates throughout your course enables powerful trend analysis over time. When Kai has consistent data points to compare, it can:

- Track improvement on specific topics across multiple sessions
- Identify persistent knowledge gaps that need additional attention
- Measure the effectiveness of your teaching interventions
- Show students measurable progress in their comprehension

Example: If you use the standard template after every lecture, Kai can show you that "student understanding of regression analysis improved from 2.3 to 4.1 after the in-class review" - actionable insights that inform your teaching strategy.

10.5.3. Best Practices for Instructors

1. **Start with Standard Template**: Use Kai's default 3-star + 2-open template for your first few sessions
2. **Regular Feedback Intervals**: Don't wait until the end of class - request feedback at natural transition points
3. **Act on Feedback Quickly**: Address high-frequency issues in the same class session when possible
4. **Communicate Actions**: Let students know their feedback was heard and how you're responding
5. **Monitor Trends**: Review Kai's longitudinal analysis to track improvement over multiple sessions
6. **Customize Gradually**: Once comfortable, customize templates for specific topics while maintaining core consistency

10.5.4. Student Engagement Strategies

1. **Incentivize Participation**: Recognize students who provide helpful feedback
2. **Keep It Simple**: Quick, easy-to-complete forms increase response rates
3. **Show Impact**: Demonstrate how feedback shapes the class in real-time
4. **Provide Options**: Mix star ratings (fast) with open-ended questions (detailed)
5. **Close the Loop**: Share aggregate results showing how the class is progressing

10.6. Benefits

10.6.1. For Instructors

- **Targeted Teaching**: Focus on concepts that need the most attention
- **Time Optimization**: Efficient allocation of limited class time
- **Data-Driven Decisions**: Real-time insights into student comprehension
- **Immediate Adaptation**: Adjust teaching on-the-fly based on feedback

10.6.2. For Students

- **Personalized Learning**: Individual support for specific needs
- **Collective Voice**: Class-wide issues addressed promptly
- **Accelerated Understanding**: Immediate clarification of confusion
- **Active Participation**: Direct input into learning process

10.6.3. For Educational Outcomes

- **Improved Comprehension**: Targeted interventions where needed
- **Higher Success Rates**: No student left behind approach
- **Resource Efficiency**: Optimal use of instructor and online resources
- **Measurable Impact**: Track improvement through feedback loops

10.7. Technical Requirements

- **Instructor App**: Installed and configured
- **Student App**: Distributed to all enrolled students

This adaptive feedback system ensures that every minute of class time is optimized for maximum learning impact, while no student is left behind through personalized online support.

11. Pop Quiz Workflow

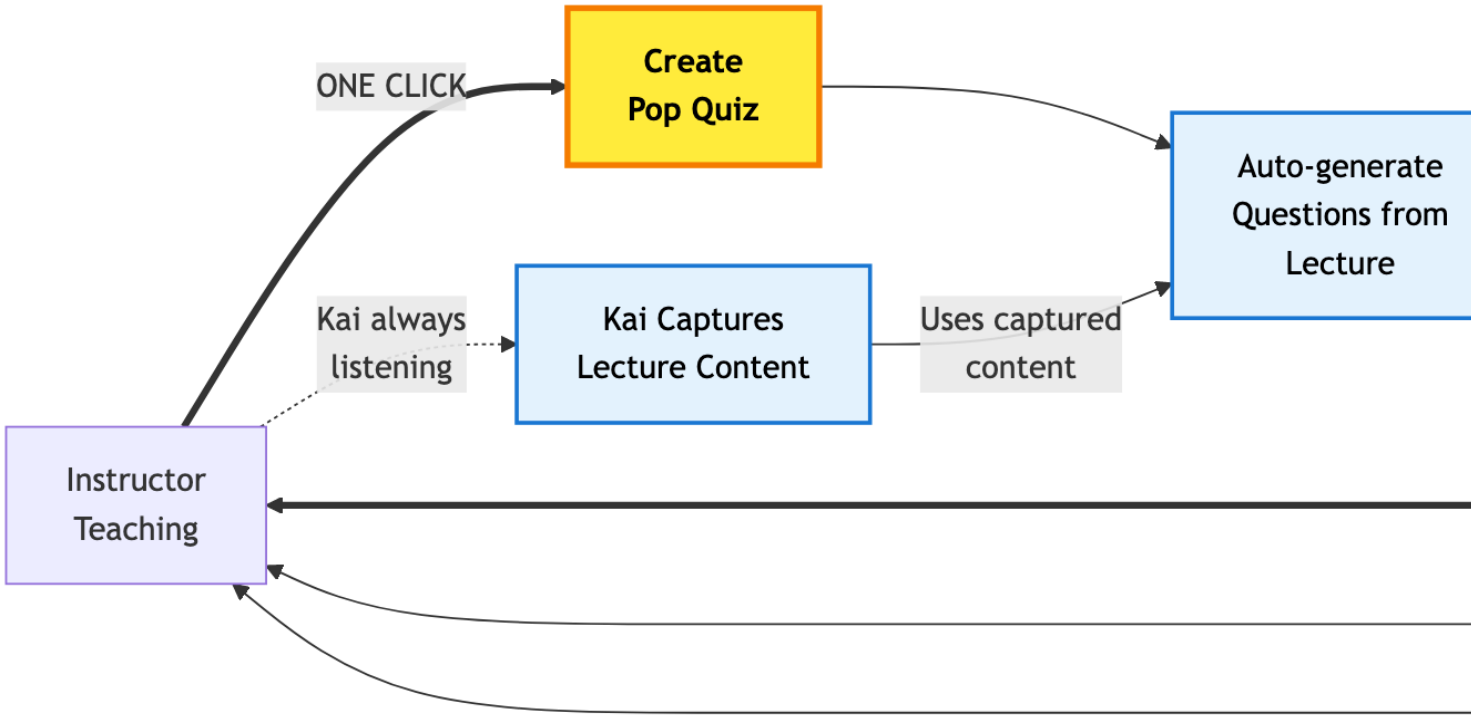
Real-Time Knowledge Assessment During Lectures

11.1. Overview

The Pop Quiz Workflow is remarkably simple: **One click creates a quiz from your live lecture**. That's it. Kai listens to your teaching, and when you click the button, it instantly generates relevant questions, sends them to students, and shows you real-time results. The entire process takes just seconds to initiate and gives you actionable insights within 15 minutes.

11.2. How It Works

The beauty is in the simplicity - while you teach, Kai listens. One click triggers everything else automatically:



Pop Quiz Workflow - From Lecture to Results in One Click

The Entire Process in 3 Simple Steps

1. **You Teach** - Kai listens and captures your lecture content
2. **You Click** - One button creates a quiz from what you just taught
3. **You Get Results** - Live dashboard shows student comprehension in real-time

That's it! No prep work, no manual question writing, no waiting for results.

11.3. Workflow Steps

11.3.1. 1. Lecture Capture & Monitoring

- Kai continuously captures lecture audio/video in real-time
- Transcribes and analyzes content for key concepts
- Identifies optimal quiz points based on topic completion

11.3.2. 2. Quiz Creation Triggers

Three strategic timing options: - **During Break**: Natural pause for assessment - **After Class**: Comprehensive review quiz - **Mid-Lecture**: Check understanding before proceeding

11.3.3. 3. One-Click Quiz Generation

Instructor initiates quiz with a single button press: - **Automatic Content Analysis**: Kai reviews last 10-20 minutes of lecture - **Intelligent Question Creation**: Questions directly tied to presented material - **Resource Integration**: Optional inclusion of supplementary materials - **Difficulty Calibration**: Questions matched to class level

11.3.4. 4. Quiz Content & Distribution

Automatically generated quiz includes: - **Multiple Choice**: Test conceptual understanding - **Short Answer**: Assess deeper comprehension - **True/False**: Quick knowledge checks - **Time Limit**: Default 15 minutes (customizable)

11.3.5. 5. Student Engagement

- **Instant Notification**: Push alert to all student devices
- **Timed Response**: Countdown timer creates urgency
- **Mobile Optimized**: Easy to complete on any device
- **Offline Support**: Answers sync when reconnected

11.3.6. 6. Real-Time Analytics

Within the 15-minute window: - **Live Score Tracking**: See results as students submit - **Pattern Recognition**: Identify common misconceptions - **Individual Performance**: Track each student's understanding - **Class Averages**: Overall comprehension metrics

11.3.7. 7. Immediate Action

Based on results, instructors can: - **Address Gaps**: Review poorly understood topics immediately - **Provide Resources**: Share targeted learning materials - **Adjust Pacing**: Speed up or slow down based on comprehension - **Identify Strugglers**: Offer individual support where needed

11.4. Benefits

11.4.1. For Instructors

- **Instant Assessment**: Real-time understanding check
- **Zero Prep Time**: Automated quiz generation
- **Data-Driven Teaching**: Immediate feedback on effectiveness
- **Adaptive Instruction**: Adjust teaching based on results

11.4.2. For Students

- **Active Learning**: Immediate application of concepts
- **Better Retention**: Testing effect enhances memory
- **Self-Assessment**: Understand own comprehension level
- **Timely Feedback**: Know performance while content is fresh

11.4.3. For Learning Outcomes

- **Higher Engagement**: Interactive learning experience
- **Measurable Progress**: Track understanding evolution
- **Targeted Interventions**: Address issues immediately
- **Improved Comprehension**: Reinforce key concepts

11.5. Key Features

11.5.1. Lecture-Based Question Generation

- **Contextual Relevance**: Questions directly from lecture content
- **Concept Extraction**: Kai identifies key teaching points
- **Natural Language**: Questions use instructor's terminology
- **Progressive Difficulty**: Mix of basic and advanced questions

11.5.2. Smart Timing Recommendations

Kai suggests optimal quiz times based on: - Topic completion markers - Natural lecture breaks - Student attention patterns - Content complexity levels

11.5.3. Result Visualization

Real-time dashboard displays: - **Heat Maps**: Question difficulty analysis - **Distribution Curves**: Class performance spread - **Time Analysis**: How long students spend per question - **Concept Mapping**: Links weak areas to lecture segments

11.6. Implementation Tips

11.6.1. Best Practices for Instructors

1. **Strategic Timing**: Use natural breaks in lecture flow
2. **Clear Expectations**: Inform students about pop quiz policy
3. **Quick Review**: Address misconceptions immediately
4. **Positive Reinforcement**: Celebrate good performance

11.6.2. Optimal Quiz Configuration

1. **Question Count**: 5-10 questions for 15-minute window
2. **Variety**: Mix question types for comprehensive assessment
3. **Difficulty Balance**: 60% medium, 20% easy, 20% challenging
4. **Clear Instructions**: Ensure students understand format

11.6.3. Student Preparation

1. **App Installation**: Ensure all students have the app
2. **Notification Settings**: Enable push notifications
3. **Practice Runs**: Conduct test quizzes early in term
4. **Expectations**: Communicate quiz frequency and weight

11.7. Technical Requirements

- **Instructor App**: Quiz creation and result viewing
- **Student App**: Quiz reception and submission
- **Audio Quality**: Clear lecture audio for accurate transcription

11.8. See It In Action

The following screenshots demonstrate the instructor's quiz workflow from creation through results analysis:

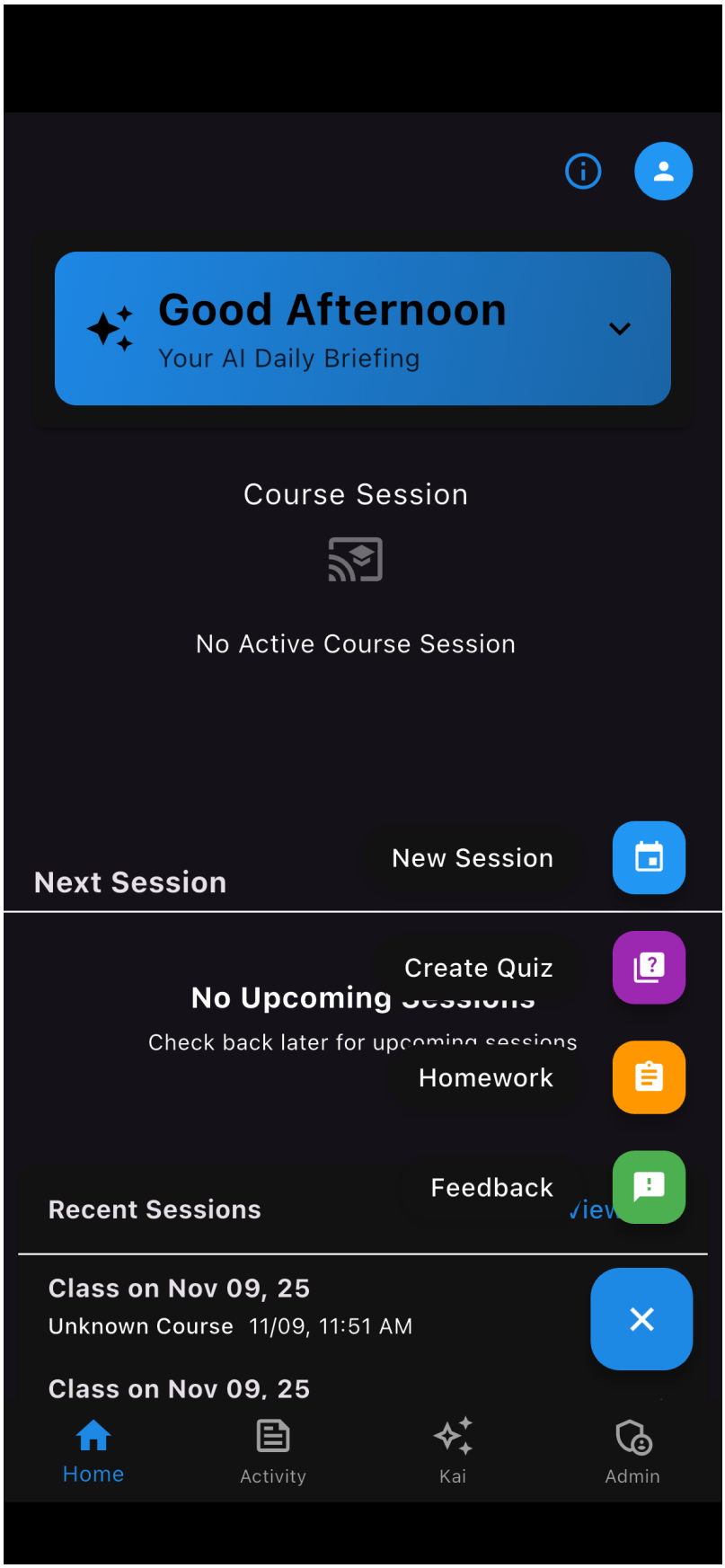


Figure 11.1.: Step 1: Create Quiz Request

Step 1: Create Quiz Request

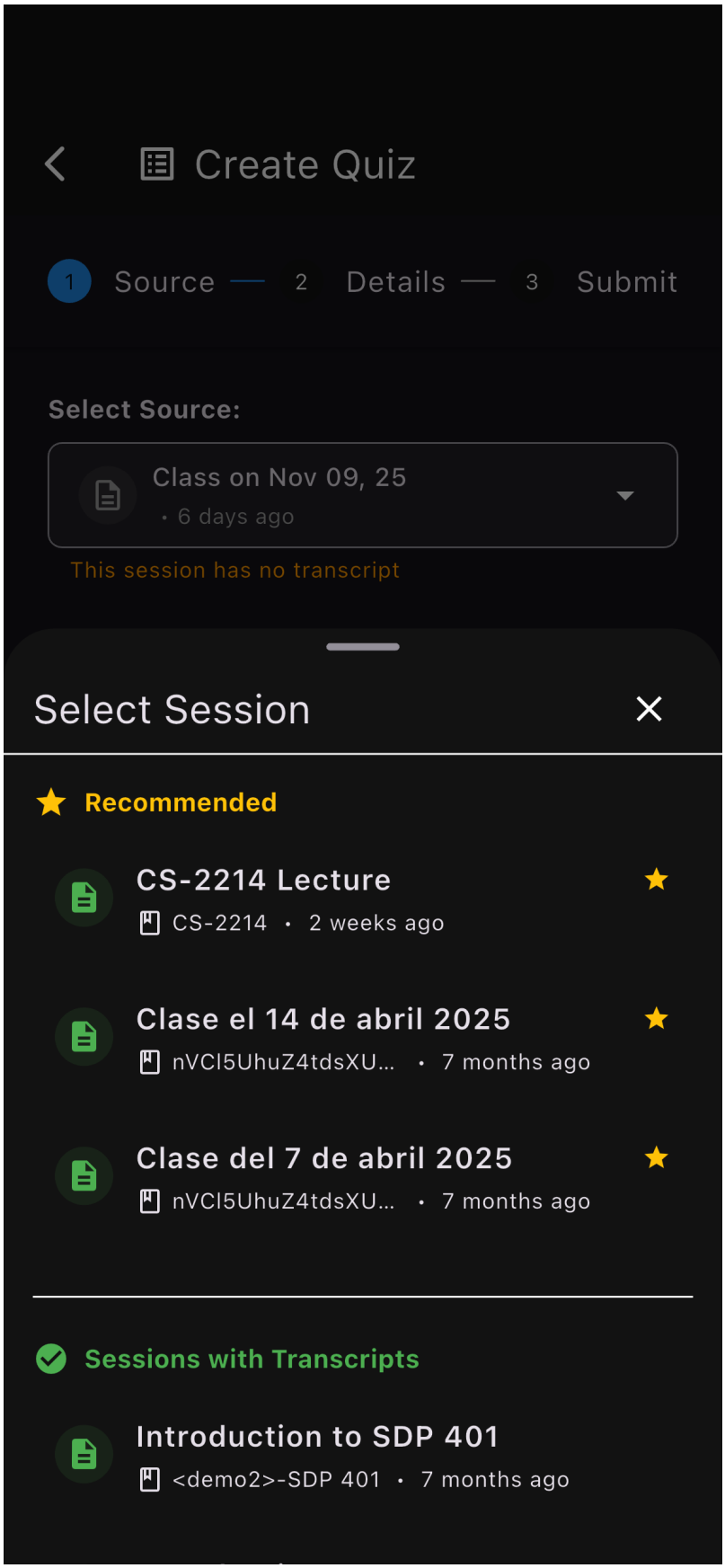


Figure 11.2.: Step 2: Configure Quiz Settings

Step 2: Configure Quiz Settings

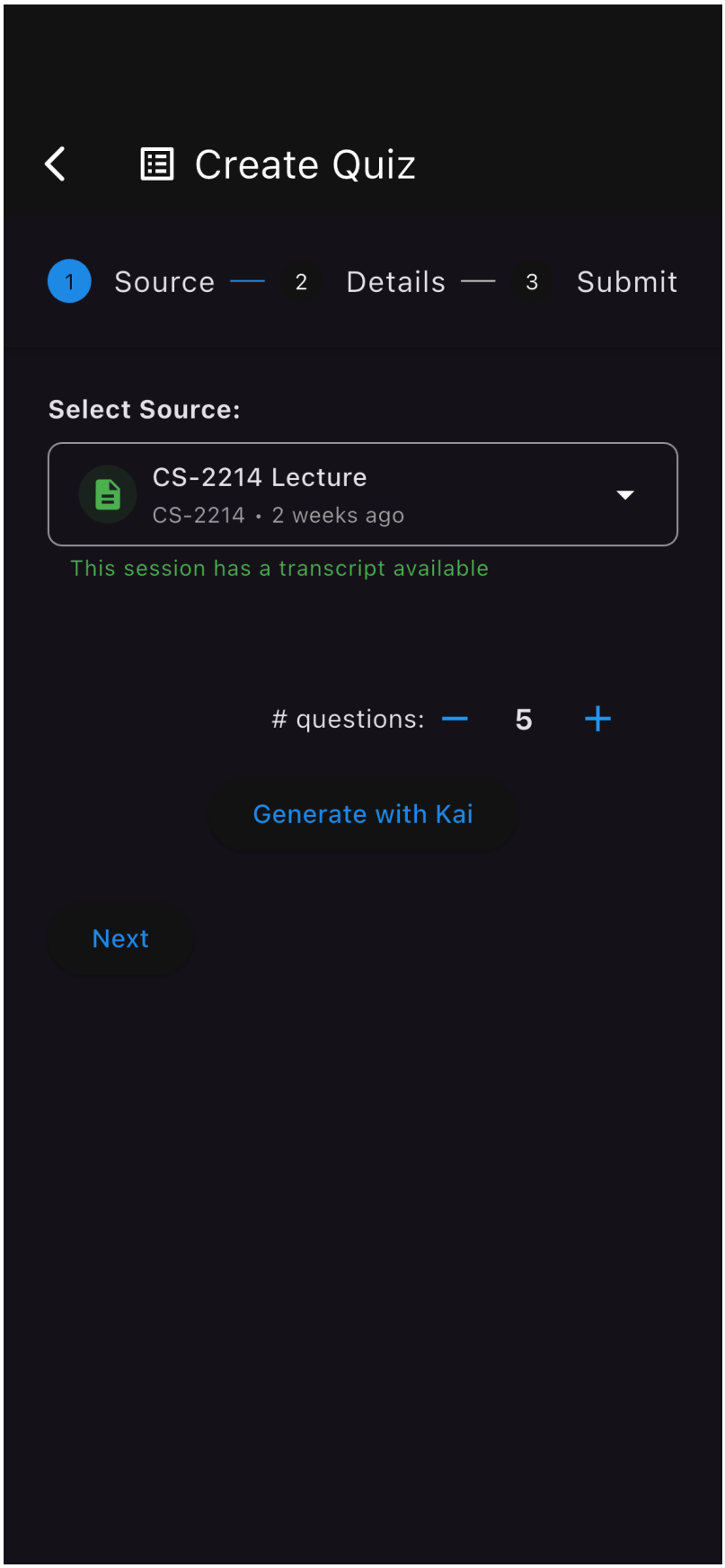


Figure 11.3.: Step 3: Review Quiz Content

Step 3: Review Quiz Content

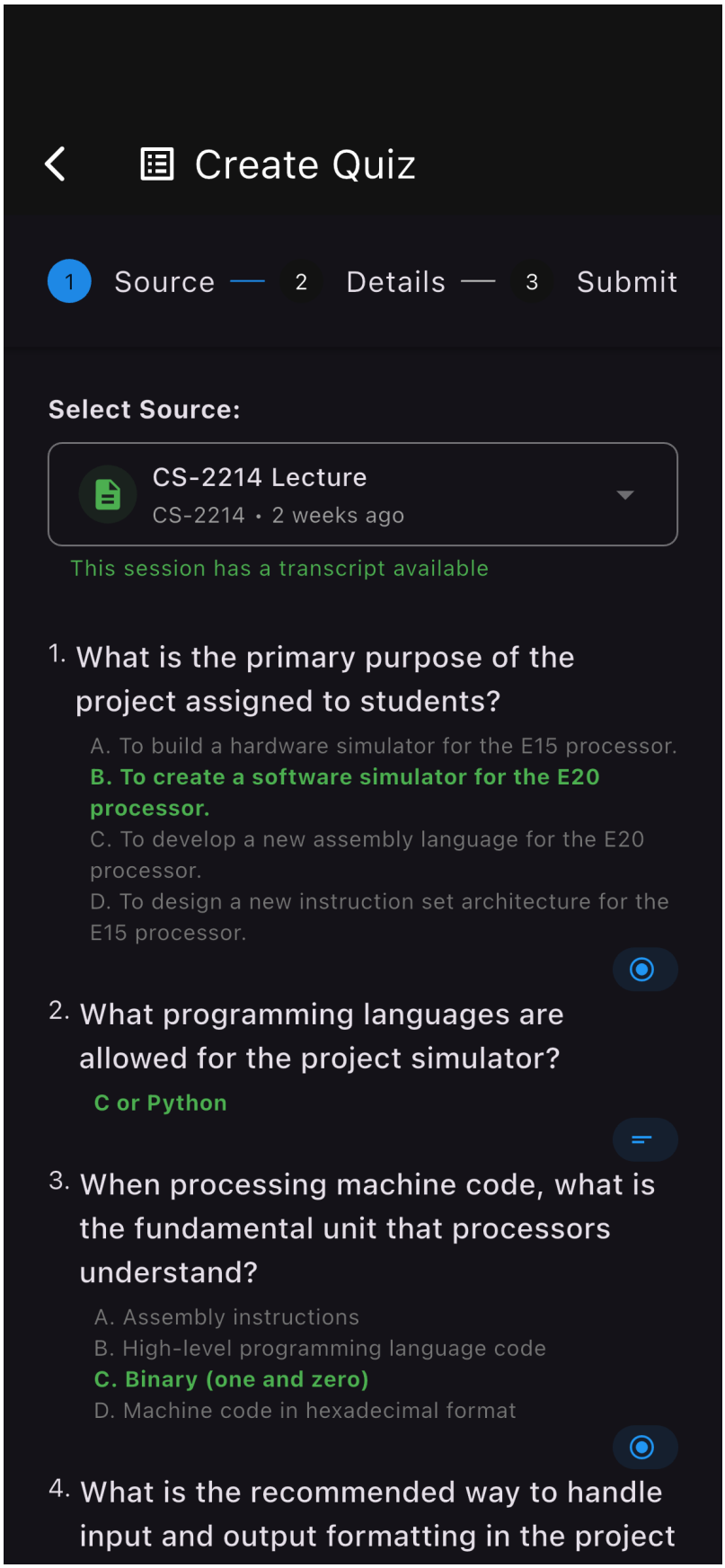


Figure 11.4.: Step 4: Send Quiz to Students

Step 4: Send Quiz to Students

11.8.1. Student Notification

Students receive instant push notifications when a quiz is available:

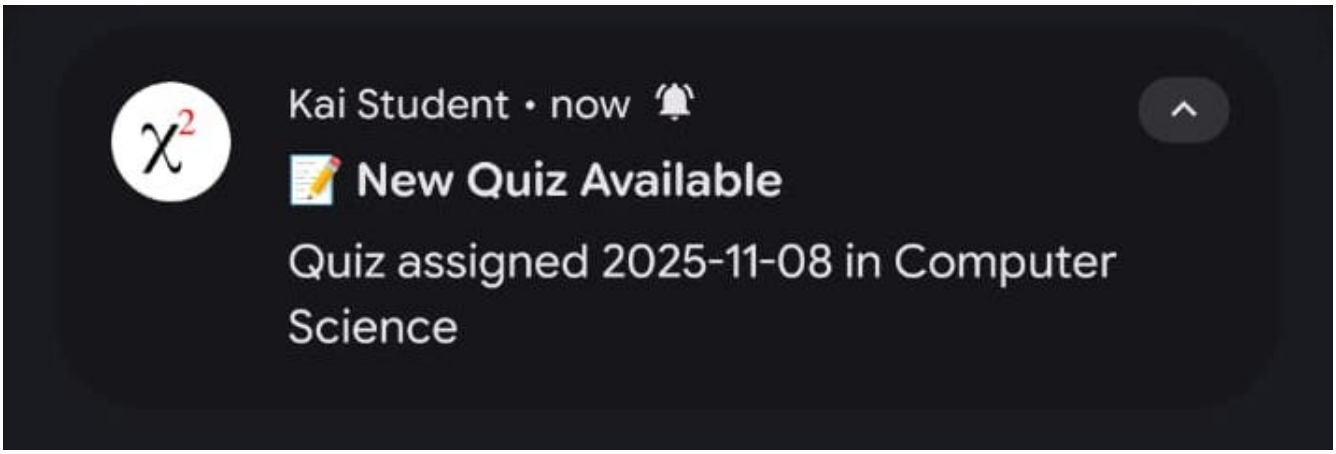


Figure 11.5.: Push notification sent to students when a new quiz is available

11.8.2. Quiz Results

After students complete the quiz, instructors can view detailed results and analytics:

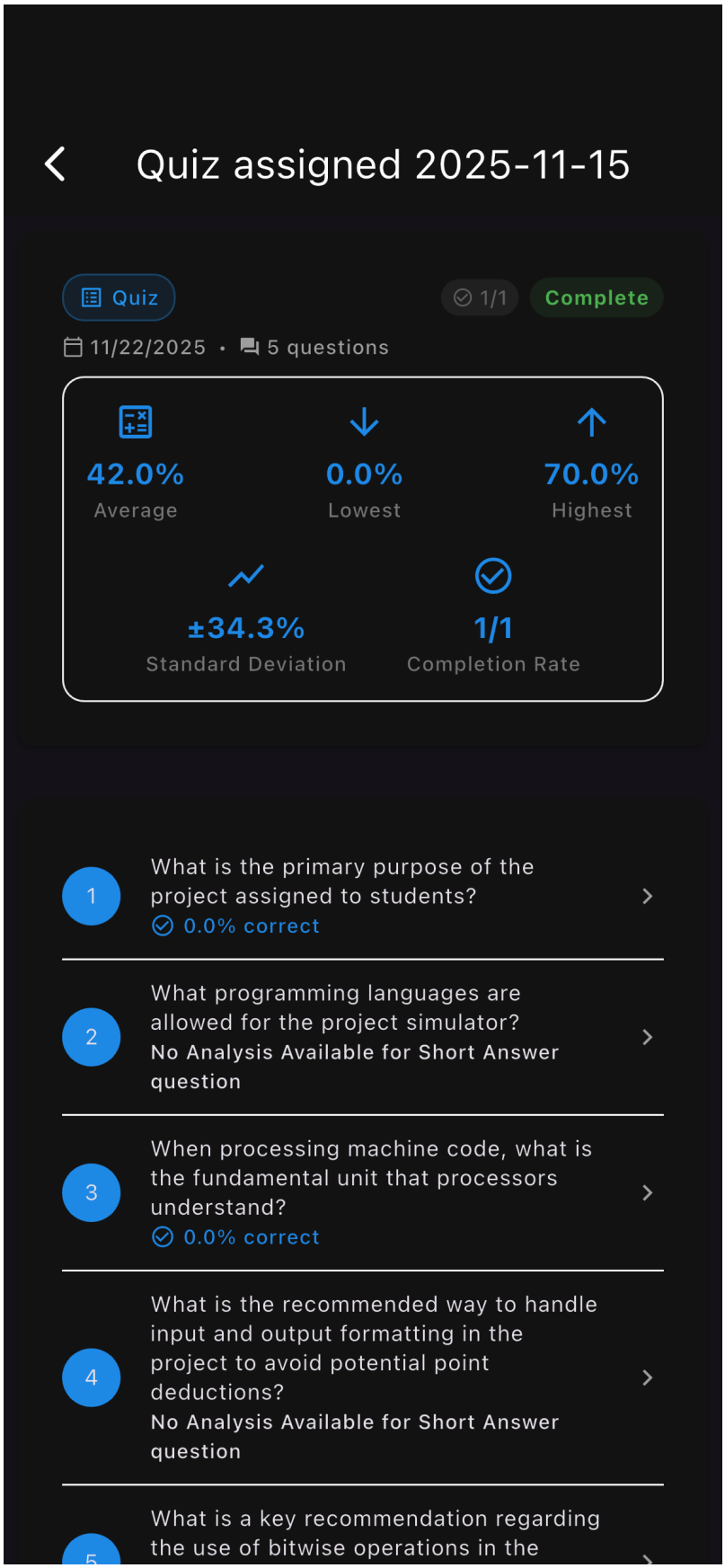


Figure 11.6.: Quiz Results Dashboard showing student performance and analytics

---

The Pop Quiz Workflow transforms passive lecture attendance into active learning engagement, providing instructors with immediate insights into student comprehension while the material is still fresh in everyone's minds.

12. SafeStream Workflow

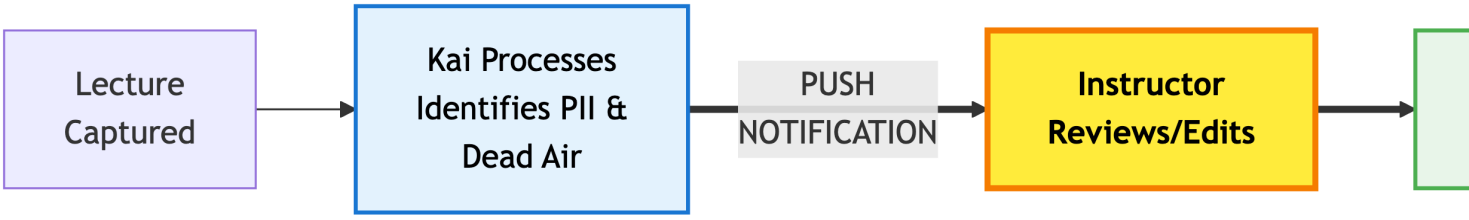
Automatic Privacy Protection and Content Enhancement for Lecture Recordings

12.1. Overview

The SafeStream Workflow automatically protects student privacy and enhances lecture recordings: **Kai reviews your lecture, suggests edits, and you approve with one click**. No more manual video editing, no more privacy concerns, no more boring dead air. SafeStream ensures your recordings are FERPA-compliant, engaging, and ready to publish—all with minimal effort from you.

12.2. How It Works

Record your lecture as usual, get notified when processing is complete, review suggestions, click submit—your video is automatically edited and published:



SafeStream Workflow - Automatic Privacy Protection and Content Enhancement

The Entire Process in 3 Simple Steps

1. **You Teach** - Lecture capture records your class as normal
2. **You Review** - Get push notification with Kai's edit suggestions
3. **You Click Submit** - Video is automatically edited and published

That's it! No manual editing, no privacy worries, just clean, engaging content.

12.3. Workflow Steps

12.3.1. 1. Standard Lecture Capture

- Regular lecture recording during class
- No change to your teaching routine
- Capture ends when class concludes

12.3.2. 2. Intelligent Processing

After lecture ends, Kai automatically: - Analyzes the entire recording - Identifies personally identifiable information (PII) - Detects periods of dead air or silence - Flags content requiring privacy protection

12.3.3. 3. Push Notification

- Instructor receives notification when processing complete
- Alert indicates video is ready for review
- Includes summary of recommended edits

12.3.4. 4. Review Interface

Simple editing interface shows: - **Privacy Concerns**: Student discussions about grades, personal information - **Dead Air Segments**: Setup time, breaks, long pauses - **Exam Detection**: Full lecture flagged if exam activity detected - **Suggested Actions**: - **Cut completely** (red strike-through): Removes video segment entirely - used for dead air or sensitive blackboard information - **Mute audio** (grey text): Video continues playing but audio is silenced - used for PII or grade discussions - **Cut entire lecture**: When AI detects an exam taking up the whole class period - **Visual Timeline**: Clear markers showing suggested edit points

What Happens When You Submit: When you approve edits, the system automatically: - Reprocesses the video on the server - Cuts out segments marked with red strike-through (dead air, sensitive content) - Mutes audio for segments marked in grey (PII protection) - Removes entire lecture if exam was detected - Re-uploads edited video to your LMS or CHI platform - Notifies students that new content is available

12.3.5. 5. One-Click Approval

- Review all suggestions at a glance
- Modify if needed (optional)
- Click “Submit” to approve edits
- Kai automatically applies all approved changes

12.3.6. 6. Automatic Reprocessing & Publication

After you approve edits, the system: - Reprocesses the entire video with approved changes - Cuts segments completely (for dead air or sensitive blackboard content) - Mutes audio sections (for PII or grade discussions) - Removes full lecture if exam detected - Uploads edited video to LMS or CHI platform - Makes content available to students - Maintains original recording in secure archive

12.4. Key Features

12.4.1. Privacy Protection

- **FERPA Compliance**: Automatically ensures recordings meet privacy regulations
- **PII Detection**: Identifies names, grades, personal discussions
- **Smart Muting**: Preserves lecture flow while protecting privacy
- **Institutional Alignment**: Meets your institution's privacy policies

12.4.2. Content Enhancement

- **Dead Air Removal**: Cuts setup time, breaks, and long pauses
- **Engagement Boost**: Creates more watchable, focused recordings
- **Time Savings**: Students get condensed, valuable content
- **Professional Polish**: Recordings feel intentionally edited

12.4.3. Intelligent Detection

Kai identifies and suggests edits for: - Student-instructor grade discussions - Personal information mentions - Beginning-of-class setup time - Mid-lecture breaks or pauses - End-of-class pack-up time - Technical difficulties or interruptions - **Exam detection**: Full class periods with exam activity → proposes cutting entire lecture

12.5. Benefits

12.5.1. For Instructors

- **Privacy Peace of Mind**: Never worry about accidental PII exposure
- **Zero Editing Time**: No manual video editing required
- **Mobile Management**: Review and approve from your phone
- **Compliance Assured**: Automatic FERPA compliance

12.5.2. For Students

- **Focused Content**: Get straight to the learning material
- **Time Efficient**: No sitting through dead air or setup
- **Privacy Protected**: Their personal information stays private
- **Better Experience**: More engaging, professional recordings

12.5.3. For Institutions

- **Policy Compliance**: Automatic adherence to privacy regulations
- **Risk Reduction**: Minimizes privacy breach potential
- **Quality Improvement**: Consistently edited, professional recordings
- **Cost Savings**: No need for manual video editing staff

12.6. Implementation Tips

12.6.1. Best Practices for Instructors

1. **Continue Normal Teaching**: Don't change your style for SafeStream
2. **Quick Reviews**: Spend 2-3 minutes reviewing suggestions
3. **Trust Kai's Analysis**: Recommendations are based on privacy best practices
4. **Batch Processing**: Review multiple lectures at once if needed

12.6.2. Maximizing Effectiveness

1. **Clear Audio**: Good audio quality helps Kai identify PII accurately
2. **Regular Processing**: Don't let recordings pile up—review promptly
3. **Feedback Loop**: Report any missed edits to improve Kai's detection
4. **Custom Settings**: Configure sensitivity levels for your needs

12.7. Technical Requirements

- **Instructor App**: Receive notifications and access editing interface
- **Lecture Capture System**: Standard recording equipment
- **Audio Quality**: Clear audio for accurate PII detection

12.8. See It In Action

The following screenshots demonstrate the SafeStream workflow from notification through editing approval:

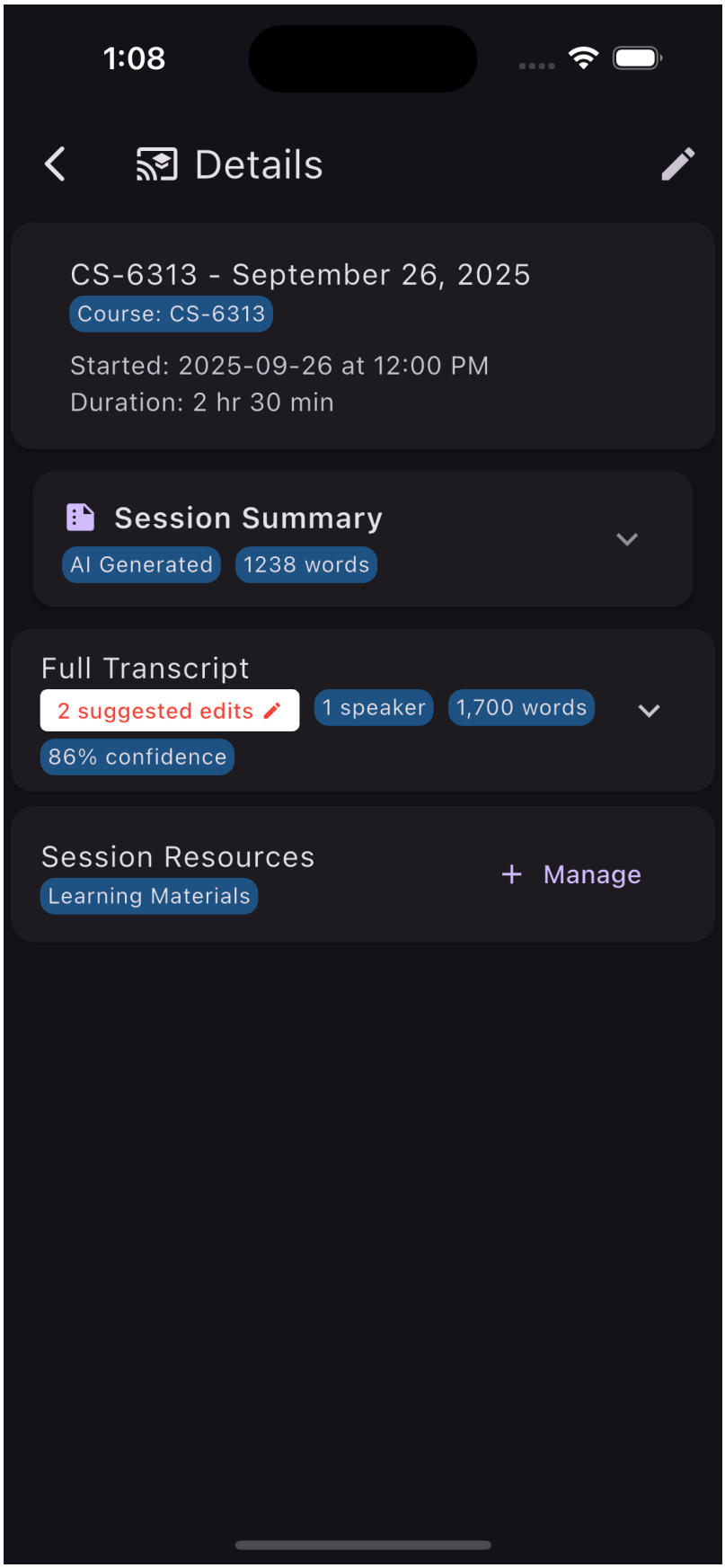


Figure 12.1.: Step 1: Review Notification

Step 1: Review Notification

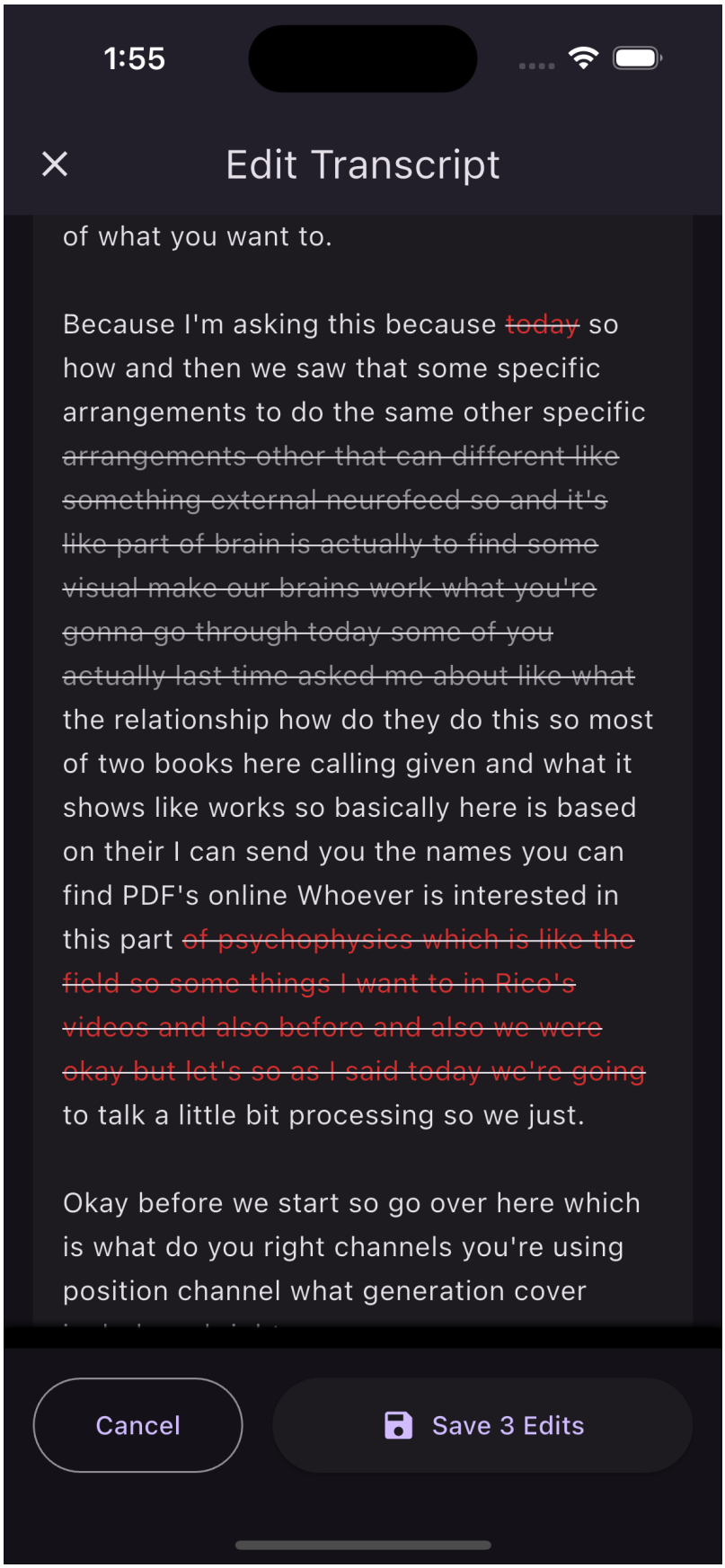


Figure 12.2.: Step 2: Editing Interface

Step 2: Editing Interface

12.8.1. Understanding the Editing Interface

The editing interface shows suggested edits with visual indicators for different edit types:

Edit Type Indicators:

- **Red strike-through**: Segments to be completely cut from video
  - Used for: Dead air, long pauses, setup time, sensitive blackboard content
  - Effect: Video segment is removed entirely from final recording
- **Grey text**: Segments where audio will be muted
  - Used for: PII discussions, student grade conversations, personal information
  - Effect: Video continues playing but audio is silenced
- **Full lecture flag**: Entire class period marked for removal
  - Used for: Exam sessions detected by AI
  - Effect: Complete lecture recording is not published

Review Process:

1. **Visual Markers**: Timeline shows color-coded edit suggestions
2. **Context Preview**: See surrounding transcript for each suggested edit
3. **One-Click Approval**: Review all suggestions and approve with single submit
4. **Optional Modifications**: Adjust edit boundaries if needed before submitting

After Submission:

- Video is automatically reprocessed on the server
- All approved edits are applied (cuts, mutes, removals)
- Edited video is uploaded to your LMS or CHI platform
- Students receive notification when new content is available
- Original recording is archived securely

12.9. Privacy & Compliance

12.9.1. FERPA Compliance

SafeStream ensures your recordings meet Family Educational Rights and Privacy Act requirements by: - Automatically detecting student information - Muting grade discussions - Removing personal identifiers - Maintaining audit logs of edits

12.9.2. Institutional Policies

Aligns with common institutional requirements: - Student privacy protection - Consent management - Data retention policies - Accessibility standards

12.9.3. Security Features

- Encrypted processing
- Secure storage of originals
- Edit history tracking
- Administrative oversight options

SafeStream transforms lecture recordings from potential privacy risks into polished, engaging, compliant educational content—all with just one click of approval from you.

13. Grading Setup Guide

Configure grading criteria, rubrics, and customize grading behavior

13.1. Overview

Kai's intelligent grading system combines AI-powered assessment with customizable rubrics and workflows to provide consistent, detailed feedback while saving you time. This guide covers everything from basic rubric setup to advanced grading configurations.

13.2. Quick Start

The fastest way to get started with grading:

1. **Create your first rubric** in the Dashboard
2. **Test with sample submissions** to calibrate AI
3. **Review and adjust** grading thresholds
4. **Enable auto-grading** for appropriate assignments

> First-Time Setup
>
> Start with Kai's pre-built rubrics and customize them to match your needs. This is faster than building from scratch and ensures you don't miss important criteria.

13.3. Creating Grading Rubrics

13.3.1. Basic Rubric Structure

A rubric in Kai consists of:

- **Criteria**: Individual aspects being evaluated (e.g., "Thesis Statement", "Evidence")
- **Point Values**: Weight of each criterion
- **Performance Levels**: Quality tiers (Excellent, Good, Fair, Poor)
- **Descriptors**: What defines each performance level

13.3.2. Creating via Dashboard

Step-by-step:

1. **Navigate**: Dashboard → Grading → Rubrics → "Create New Rubric"
2. **Name your rubric**: Use clear, descriptive names (e.g., "Research Paper - HIST 101")
3. **Set total points**: Typically 100 or aligned with your gradebook
4. **Add criteria**: Click "Add Criterion" for each aspect you want to evaluate

Example: Essay Rubric

```
Rubric Name: "Analytical Essay - ENGL 201"
Total Points: 100

Criterion 1: Thesis & Argument (25 points)
  - Excellent (25): Clear, specific, arguable thesis with sophisticated argument
  - Good (20): Clear thesis with solid argument, minor improvements possible
  - Fair (15): Thesis present but vague; argument needs development
  - Poor (10): Weak or missing thesis; argument unclear

Criterion 2: Evidence & Analysis (30 points)
  - Excellent (30): Strong textual evidence with deep, original analysis
  - Good (24): Good evidence with competent analysis
  - Fair (18): Some evidence but superficial analysis
  - Poor (12): Minimal evidence; analysis missing or incorrect

Criterion 3: Organization & Structure (20 points)
  - Excellent (20): Logical flow, clear transitions, coherent structure
  - Good (16): Generally organized with minor flow issues
  - Fair (12): Disorganized in places; unclear transitions
  - Poor (8): Poor organization; difficult to follow

Criterion 4: Writing Quality (15 points)
  - Excellent (15): Clear, precise writing; minimal errors
  - Good (12): Clear writing with minor errors
  - Fair (9): Unclear at times; multiple errors
  - Poor (6): Frequent errors impede understanding

Criterion 5: Citations & Format (10 points)
  - Excellent (10): Perfect citation format; all sources cited
  - Good (8): Minor citation errors
  - Fair (6): Multiple citation issues
  - Poor (4): Missing citations or major format errors
```

13.3.3. Creating via API

For programmatic rubric creation or bulk uploads:

```
import requests

rubric = {
    "name": "Analytical Essay - ENGL 201",
    "total_points": 100,
    "criteria": [
        {
            "name": "Thesis & Argument",
            "points": 25,
            "description": "Clear, arguable thesis with well-developed argument",
            "levels": [
                {
                    "name": "Excellent",
                    "points": 25,
                    "description": "Clear, specific, arguable thesis with sophisticated argument"
                },
                {
                    "name": "Good",
                    "points": 20,
                    "description": "Clear thesis with solid argument, minor improvements possible
                },
                {
                    "name": "Fair",
                    "points": 15,
                    "description": "Thesis present but vague; argument needs development"
                },
                {
                    "name": "Poor",
                    "points": 10,
                    "description": "Weak or missing thesis; argument unclear"
                }
            ]
        },
        {
            "name": "Evidence & Analysis",
            "points": 30,
            "description": "Use of textual evidence and quality of analysis",
            "levels": [
                {
                    "name": "Excellent",
                    "points": 30,
                    "description": "Strong textual evidence with deep, original analysis"
                },
                {
                    "name": "Good",
                    "points": 24,
                    "description": "Good evidence with competent analysis"
```

```
                },
                {
                    "name": "Fair",
                    "points": 18,
                    "description": "Some evidence but superficial analysis"
                },
                {
                    "name": "Poor",
                    "points": 12,
                    "description": "Minimal evidence; analysis missing or incorrect"
                }
            ]
        }
        # ... additional criteria
    ]
}

response = requests.post(
    "https://chi2api.com/v1/rubrics",
    headers={
        "Authorization": f"Bearer {API_KEY}",
        "Content-Type": "application/json"
    },
    json=rubric
)

rubric_id = response.json()['rubric_id']
print(f"Created rubric: {rubric_id}")
```

13.3.4. Subject-Specific Rubric Templates

Kai provides pre-built templates for common subjects:

Available Templates:

Subject	Template Name	Best For	Criteria Count
Writing	Essay Analysis	Analytical essays, literary analysis	5
Writing	Research Paper	Research-based writing	6
Writing	Creative Writing	Fiction, poetry, creative work	4
Science	Lab Report	Laboratory experiments	7
Science	Problem Sets	Math/physics problems	4

Subject	Template Name	Best For	Criteria Count
Social Science	Case Analysis	Business/policy cases	5
Programming	Code Quality	Programming assignments	6
Presentation	Oral Presentation	Speeches, presentations	5

Using a Template:

```
# Browse templates
templates = kai.rubrics.list_templates(subject="writing")

# Apply template
kai.rubrics.create_from_template(
    template_id="essay_analysis",
    course_id="course_abc",
    customizations={
        "name": "Literary Analysis - American Lit",
        "total_points": 100,
        "adjust_weights": {
            "Evidence & Analysis": 35,  # Increase from default 30
            "Writing Quality": 10        # Decrease from default 15
        }
    }
)
```

13.4. Grading Configuration

13.4.1. Auto-Grading Thresholds

Configure when Kai can automatically assign grades vs. when human review is required.

Confidence-Based Grading:

```
grading_policy = {
    "auto_grade_threshold": 0.85,  # Auto-grade if AI confidence >= 85%

    "review_required": {
        "low_confidence": True,        # Review if confidence < threshold
        "high_stakes": True,            # Review if assignment weight > 20%
        "boundary_scores": True,        # Review if near grade boundary (±2%)
        "flagged_content": True,        # Review if content flagged
        "student_request": True         # Review if student requests
    },
```

```
    "grade_boundaries": [90, 80, 70, 60],  # A, B, C, D cutoffs
    "boundary_margin": 2,                   # ±2 points triggers review

    "exceptions": {
        "final_exams": {"auto_grade": False},    # Always review finals
        "first_assignment": {"auto_grade": False} # Review first to calibrate
    }
}

kai.courses.update(
    course_id="course_abc",
    grading_settings=grading_policy
)
```

Example Scenarios:

Submission	AI Score	Confidence	Auto-Grade?	Reason
Excellent work, clear criteria match	95	0.96	Yes	High confidence, not near boundary
Good work, some ambiguity	82	0.78	No	Below 0.85 threshold
Borderline B/C (78%)	78	0.88	No	Within 2 points of 80% boundary
Final exam	92	0.94	No	Final exam exception
Plagiarism detected	88	0.91	No	Flagged content
Student requested review	85	0.89	No	Student request

13.4.2. Feedback Customization

Configure the style and detail of AI-generated feedback.

Feedback Styles:

Style	Description	Detail Level	Best For
Detailed	Comprehensive with specific suggestions	High	Major papers, final projects
Concise	Brief, actionable feedback	Medium	Regular assignments, quizzes
Encouraging	Positive, growth-focused	Medium	Struggling students, formative work

Style	Description	Detail Level	Best For
Professional	Formal, academic tone	High	Graduate work, professional programs
Points Only	Score with minimal commentary	Low	Self-graded work, completion-based

Configuration:

```
# Set course-wide default
kai.courses.update(
    course_id="course_abc",
    grading_settings={
        "feedback_style": "detailed",
        "feedback_options": {
            "show_rubric_scores": True,
            "include_suggestions": True,
            "highlight_strengths": True,
            "provide_examples": True,
            "suggest_resources": True
        }
    }
)

# Override for specific assignment
kai.assignments.update(
    assignment_id="assign_final",
    grading_settings={
        "feedback_style": "professional",
        "feedback_options": {
            "include_suggestions": False,  # Just evaluation
            "provide_examples": False
        }
    }
)

# Override for specific student (accommodation)
kai.students.update_grading_preferences(
    student_id="student_123",
    course_id="course_abc",
    preferences={
        "feedback_style": "encouraging",
        "additional_support": True
    }
)
```

Example Feedback Comparison:

```
Assignment: Essay on "The Great Gatsby"
Score: 82/100

--- DETAILED STYLE ---
Thesis & Argument (20/25):
Your thesis in paragraph 2 clearly identifies Fitzgerald's use of symbolism,
which is a strong foundation. However, consider making it more specific -
instead of "symbolism is important," argue WHY the green light specifically
represents Gatsby's impossible dream. See Smith (2018, p. 45) for examples
of strong thesis statements in literary analysis.

Evidence & Analysis (26/30):
Excellent use of textual evidence in paragraphs 3-5. Your analysis of the
green light scene is sophisticated and well-developed. To strengthen further,
connect your analysis back to your thesis more explicitly. For example, after
analyzing the "orgastic future" passage, add a sentence like "This demonstrates
how Fitzgerald uses temporal imagery to underscore the thesis that..."

--- CONCISE STYLE ---
Thesis & Argument (20/25): Strong foundation. Make thesis more specific about
green light symbolism.

Evidence & Analysis (26/30): Good textual evidence. Connect analysis back to
thesis more explicitly.

--- ENCOURAGING STYLE ---
Thesis & Argument (20/25): You're developing strong analytical skills! Your
thesis shows good understanding of symbolism. Next step: make it even more
specific and arguable.

Evidence & Analysis (26/30): Excellent work with textual evidence! Your
analysis shows real insight into Fitzgerald's techniques.
```

13.4.3. Grading Workflows

Define the process from submission to final grade.

Standard Workflow:

```
workflow = {
    "name": "Standard Essay Grading",
    "steps": [
        {
            "step": 1,
            "action": "ai_grade",
            "config": {
                "use_rubric": True,
```

```
                "rubric_id": "rubric_essay_001"
            }
        },
        {
            "step": 2,
            "action": "check_confidence",
            "config": {
                "threshold": 0.85,
                "if_low": "route_to_review"
            }
        },
        {
            "step": 3,
            "action": "check_policies",
            "config": {
                "check_plagiarism": True,
                "check_boundaries": True,
                "check_stakes": True
            }
        },
        {
            "step": 4,
            "action": "assign_grade",
            "config": {
                "auto_post": False,  # Instructor must approve
                "notify_student": True
            }
        }
    ]
}

kai.workflows.create(workflow)
```

Advanced Workflow with Peer Review:

```
peer_review_workflow = {
    "name": "Peer Review + AI Grading",
    "steps": [
        {
            "step": 1,
            "action": "peer_review",
            "config": {
                "reviewers_per_submission": 2,
                "review_rubric_id": "peer_review_rubric",
                "anonymize": True,
                "deadline_offset_hours": 48
            }
```

```
        },
        {
            "step": 2,
            "action": "ai_grade",
            "config": {
                "consider_peer_feedback": True,
                "peer_weight": 0.2,  # 20% peer, 80% AI/instructor
                "flag_discrepancies": True
            }
        },
        {
            "step": 3,
            "action": "instructor_review",
            "config": {
                "required_if": [
                    "large_peer_ai_discrepancy",
                    "flagged_content",
                    "low_confidence"
                ]
            }
        },
        {
            "step": 4,
            "action": "finalize_grade"
        }
    ]
}
```

13.5. Calibration and Training

13.5.1. Initial Calibration

Train Kai to match your grading style:

Calibration Process:

1. **Grade Sample Set**: Manually grade 5-10 representative submissions
2. **AI Comparison**: Kai grades the same submissions
3. **Review Differences**: Examine where AI differs from your grades
4. **Adjust Settings**: Modify thresholds and rubric weights
5. **Re-test**: Test on new sample set

Running Calibration:

```
# Create calibration set
calibration = kai.calibration.create(
    course_id="course_abc",
```

```
    assignment_id="assign_essay_001",
    rubric_id="rubric_essay_001",
    sample_size=10
)

# Grade manually
for submission in calibration.submissions:
    instructor_grade = manual_grading_interface(submission)
    kai.calibration.add_instructor_grade(
        calibration_id=calibration.id,
        submission_id=submission.id,
        grade=instructor_grade
    )

# Run AI grading on same submissions
kai.calibration.run_ai_grading(calibration.id)

# Compare and analyze
analysis = kai.calibration.analyze(calibration.id)

print(f"Average difference: {analysis.avg_difference} points")
print(f"Criteria with largest variance: {analysis.variance_by_criterion}")

# View detailed comparison
for comparison in analysis.comparisons:
    print(f"\nSubmission {comparison.submission_id}:")
    print(f"  Your grade: {comparison.instructor_grade}")
    print(f"  AI grade: {comparison.ai_grade}")
    print(f"  Difference: {comparison.difference}")
    print(f"  Largest criterion gap: {comparison.largest_gap}")
```

Interpreting Results:

Average Difference	Interpretation	Action
< 3 points	Excellent alignment	Proceed with confidence
3-5 points	Good alignment	Review criteria with largest gaps
5-8 points	Moderate alignment	Adjust rubric weights, re-calibrate
> 8 points	Poor alignment	Revise rubric descriptors significantly

13.5.2. Ongoing Adjustment

Continuously improve grading accuracy:

Feedback Loop:

```
# After each grading session, review AI performance
performance = kai.grading.get_performance_metrics(
    course_id="course_abc",
    timeframe="last_week"
)

print(f"Auto-graded: {performance.auto_graded_count}")
print(f"Required review: {performance.review_count}")
print(f"Instructor agreed with AI: {performance.agreement_rate}%")
print(f"Average adjustment: {performance.avg_adjustment} points")

# Identify patterns in adjustments
patterns = kai.grading.analyze_adjustments(course_id="course_abc")

for pattern in patterns:
    print(f"\nPattern: {pattern.description}")
    print(f"  Frequency: {pattern.frequency}")
    print(f"  Suggestion: {pattern.suggested_adjustment}")
```

13.6. Special Grading Scenarios

13.6.1. STEM Problem Sets

Configure for mathematical and scientific work:

```
stem_grading = {
    "partial_credit": {
        "enabled": True,
        "method": "step_by_step",  # Track work, not just final answer
        "steps": [
            {"name": "Problem Setup", "weight": 0.2},
            {"name": "Method Selection", "weight": 0.2},
            {"name": "Calculation", "weight": 0.3},
            {"name": "Final Answer", "weight": 0.2},
            {"name": "Units & Formatting", "weight": 0.1}
        ]
    },
    "answer_checking": {
        "numerical_tolerance": 0.01,  # 1% tolerance for rounding
        "accept_equivalent_forms": True,  # √2 = 1.414...
```

```
        "check_units": True
    },
    "common_errors": {
        "track": True,
        "provide_hints": True,
        "error_categories": [
            "sign_error",
            "unit_conversion",
            "algebraic_manipulation",
            "calculation_error"
        ]
    }
}

kai.assignments.update(
    assignment_id="physics_problemset_1",
    grading_settings=stem_grading
)
```

13.6.2. Code Assignments

Special configuration for programming:

```
code_grading = {
    "execution_testing": {
        "enabled": True,
        "test_cases": [
            {"input": "test1.txt", "expected_output": "output1.txt"},
            {"input": "test2.txt", "expected_output": "output2.txt"}
        ],
        "timeout_seconds": 5
    },
    "code_quality": {
        "check_style": True,
        "style_guide": "pep8",  # Or "google", "airbnb", etc.
        "check_documentation": True,
        "check_efficiency": True,
        "max_complexity": 10
    },
    "plagiarism_detection": {
        "enabled": True,
        "compare_to": ["current_class", "previous_semesters"],
        "code_similarity_threshold": 0.85
    },
    "rubric_weights": {
        "correctness": 0.50,
```

```
        "code_quality": 0.25,
        "documentation": 0.15,
        "efficiency": 0.10
    }
}
```

13.6.3. Creative Work

Configure for subjective assessments:

```
creative_grading = {
    "subjective_criteria": {
        "enabled": True,
        "criteria": [
            {
                "name": "Originality",
                "description": "Uniqueness and creativity of approach",
                "weight": 0.3,
                "allow_high_variance": True  # Subjectivity expected
            },
            {
                "name": "Emotional Impact",
                "description": "Effectiveness of emotional resonance",
                "weight": 0.3,
                "allow_high_variance": True
            },
            {
                "name": "Technical Execution",
                "description": "Craft and technical skill",
                "weight": 0.4,
                "allow_high_variance": False  # More objective
            }
        ]
    },
    "require_human_review": {
        "always": True,  # Creative work always needs human judgment
        "ai_provides": "suggestions_only"
    },
    "feedback_style": "encouraging"  # Foster creativity
}
```

13.7. Grade Management

13.7.1. Posting Grades

Control when and how grades are released:

```
posting_settings = {
    "auto_post": {
        "enabled": True,
        "conditions": {
            "only_if_auto_graded": True,
            "only_if_high_confidence": True,
            "delay_hours": 24  # Give instructor time to review
        }
    },
    "manual_post": {
        "batch_post": True,  # Post all at once
        "notify_on_post": True
    },
    "anonymity": {
        "hide_from_peers": True,
        "show_distribution": True,  # Show class distribution
        "show_percentile": True     # Show student's percentile
    }
}
```

13.7.2. Grade Analytics

Track grading patterns and student performance:

```
# View grade distribution
distribution = kai.analytics.grade_distribution(
    course_id="course_abc",
    assignment_id="essay_001"
)

print(f"Mean: {distribution.mean}")
print(f"Median: {distribution.median}")
print(f"Std Dev: {distribution.std_dev}")
print(f"Distribution: {distribution.histogram}")

# Identify outliers
outliers = kai.analytics.identify_outliers(
    course_id="course_abc",
    assignment_id="essay_001",
    threshold=2.0  # 2 standard deviations
```

```
)

# Track improvement over time
progress = kai.analytics.student_progress(
    student_id="student_123",
    course_id="course_abc"
)

print(f"Trend: {progress.trend}")  # "improving", "declining", "stable"
print(f"Average improvement: {progress.avg_improvement} points per assignment")
```

13.8. Best Practices

13.8.1. Rubric Design

> **Clear, Measurable Criteria**
>
> Use specific, observable criteria rather than vague descriptors. Instead of "good writing," use "clear topic sentences in each paragraph with supporting evidence."

Good vs. Poor Descriptors:

Poor	Good
"Good analysis"	"Analysis connects evidence to thesis with clear reasoning"
"Well organized"	"Logical paragraph order with transitions between ideas"
"Nice work"	"Correctly applies formulas with work shown for each step"

13.8.2. Consistency

- **Use same rubric** for similar assignments
- **Calibrate regularly** (every 3-4 assignments)
- **Document exceptions** when you override AI grades
- **Share rubrics** with students before assignment

13.8.3. Transparency

```
# Make rubrics visible to students
kai.rubrics.update(
    rubric_id="rubric_001",
```

```
    settings={
        "visible_to_students": True,
        "show_before_submission": True,
        "show_with_feedback": True
    }
)

# Provide sample graded work
kai.assignments.add_examples(
    assignment_id="essay_001",
    examples=[
        {
            "submission": "sample_excellent.pdf",
            "grade": 95,
            "feedback": "Example of excellent work",
            "rubric_scores": {...}
        },
        {
            "submission": "sample_good.pdf",
            "grade": 82,
            "feedback": "Example of good work with room for improvement",
            "rubric_scores": {...}
        }
    ]
)
```

13.8.4. Efficiency Tips

1. **Batch Grade**: Grade similar submissions together for consistency
2. **Use Templates**: Start with templates, customize as needed
3. **Review Strategically**: Focus human review on borderline/flagged cases
4. **Export Reports**: Use analytics to identify trends early

13.9. Troubleshooting

13.9.1. Common Issues

13.9.2. AI Grades Too Harsh/Lenient

Problem: AI consistently grades higher or lower than you would

Solutions:

```
# Adjust baseline
kai.courses.update(
    course_id="course_abc",
    grading_settings={
        "grade_adjustment": +3  # Add 3 points to all AI grades
        # or use multiplier: "grade_multiplier": 1.05
    }
)

# Or recalibrate with more examples
kai.calibration.run_full_recalibration(
    course_id="course_abc",
    sample_size=20  # Larger sample
)
```

13.9.3. Inconsistent Grading

Problem: Similar submissions receive different grades

Solutions: - Ensure rubric descriptors are specific and measurable - Increase auto-grade threshold to 0.90 (stricter) - Review and revise rubric after calibration - Check for ambiguous criteria

13.9.4. Low Confidence Scores

Problem: Too many submissions flagged for review

Solutions:

```
# Improve rubric specificity
# Add more performance level descriptors
# Provide more calibration examples

# Or adjust threshold temporarily
kai.courses.update(
    course_id="course_abc",
    grading_settings={
        "auto_grade_threshold": 0.75  # Lower threshold
    }
)
```

13.10. Support Resources

- **Email**: grading-support@chi2labs.com
- **Calibration Workshops**: Schedule a session
- **Video Tutorials**: Rubric creation walkthrough

- **Community**: Share rubrics

Next Steps: - Create content templates for assignments and quizzes - Personalize the learning experience for your students - Review best practices for effective grading

14. Personalization Guide

Customize learning experiences and enable adaptive learning features

14.1. Overview

Kai's personalization features allow you to tailor the learning experience to individual student needs, learning styles, and progress. From adaptive quizzing to customized feedback, these tools help you support every student effectively while scaling your teaching impact.

14.2. Why Personalization Matters

Benefits for Students: - Receive content at appropriate difficulty levels - Get targeted support in weak areas - Experience learning that matches their pace - Build confidence through appropriate challenges

Benefits for Instructors: - Identify struggling students early - Allocate support time efficiently - Track individual and group progress - Maintain engagement across diverse learners

14.3. Quick Start

14.3.1. Basic Personalization Setup

1. **Enable student profiles**: Track individual learning patterns
2. **Set adaptive parameters**: Configure how content adjusts
3. **Define learning paths**: Create flexible progression routes
4. **Monitor and adjust**: Review effectiveness and refine

> Start Simple
>
> Begin with one personalized element (e.g., adaptive quizzes) and expand as you become comfortable with the system.

14.4. Student Learning Profiles

14.4.1. Automatic Profile Building

Kai automatically builds student profiles based on:

- **Performance Data**: Quiz scores, assignment grades, improvement trends
- **Engagement Metrics**: Participation rates, time on task, resource usage
- **Learning Behaviors**: Response patterns, common errors, help-seeking
- **Preferences**: Feedback style preferences, study time patterns

Profile Components:

```
student_profile = {
    "student_id": "student_123",

    "performance": {
        "current_grade": 82,
        "trend": "improving",  # or "stable", "declining"
        "strong_areas": ["statistical_tests", "hypothesis_formulation"],
        "weak_areas": ["p_value_interpretation", "effect_size"],
        "mastery_levels": {
            "descriptive_statistics": 0.92,
            "inferential_statistics": 0.68,
            "experimental_design": 0.75
        }
    },

    "engagement": {
        "participation_rate": 0.88,
        "avg_time_per_assignment": 45,  # minutes
        "resource_usage": "high",
        "office_hours_attendance": 3,
        "peer_interaction": "moderate"
    },

    "learning_patterns": {
        "preferred_feedback": "detailed",
        "error_patterns": ["rushing_through_problems", "skipping_assumption_checks"],
        "strength_trajectory": "steady_improvement",
        "help_seeking_behavior": "proactive",
        "optimal_quiz_time": "evening"
    },

    "adaptations": {
        "recommended_difficulty": "medium_to_hard",
        "extra_support_topics": ["p_value_interpretation"],
        "enrichment_ready": ["statistical_tests"],
```

```
        "pacing": "standard"
    }
}

# Access student profile
profile = kai.students.get_profile(
    student_id="student_123",
    course_id="course_abc"
)
```

14.4.2. Manual Profile Adjustments

Override or supplement automatic profiles:

```
# Add known information
kai.students.update_profile(
    student_id="student_123",
    course_id="course_abc",
    updates={
        "accommodations": {
            "extended_time": 1.5,  # 50% extra time
            "alternative_format": "large_print",
            "distraction_free_environment": True
        },
        "background": {
            "prior_statistics_course": False,
            "math_confidence": "low",
            "learning_style_preference": "visual"
        },
        "goals": {
            "target_grade": "B+",
            "career_path": "clinical_psychology",
            "specific_interests": ["psychological_testing", "research_methods"]
        }
    }
)
```

14.5. Adaptive Learning Features

14.5.1. Adaptive Quizzing

Quizzes that adjust difficulty based on student performance.

Configuration:

```
adaptive_quiz = {
    "name": "Adaptive Practice: Hypothesis Testing",
    "type": "adaptive",

    "algorithm": {
        "start_difficulty": "auto",  # Based on student profile
        "adaptation_speed": "moderate",  # slow, moderate, fast

        "rules": {
            "correct_answer": {
                "action": "increase_difficulty",
                "increment": 0.5  # Difficulty scale 1-5
            },
            "incorrect_answer": {
                "action": "decrease_difficulty",
                "increment": 0.5
            },
            "two_consecutive_correct": {
                "action": "skip_easier_levels"
            },
            "two_consecutive_incorrect": {
                "action": "provide_hint_and_retry"
            }
        }
    },

    "termination_criteria": {
        "mastery_achieved": {
            "threshold": 0.85,
            "consecutive_correct": 3
        },
        "max_questions": 20,
        "time_limit_minutes": 30,
        "struggling_threshold": {
            "incorrect_streak": 5,
            "action": "end_and_suggest_review"
        }
    },

    "question_pool": {
        "total_questions": 100,
        "difficulty_distribution": {
            "easy": 30,
            "medium": 40,
            "hard": 30
        },
        "ensure_coverage": ["null_hypothesis", "alternative_hypothesis", "p_values", "alpha_level
```

```
    },

    "feedback": {
        "immediate": True,
        "adaptive_detail": {
            "struggling_students": "detailed_with_hints",
            "average_students": "moderate",
            "advanced_students": "concise"
        },
        "suggest_resources": True,
        "track_for_later_review": True
    }
}

# Create adaptive quiz
quiz = kai.quizzes.create(
    course_id="course_abc",
    config=adaptive_quiz,
    personalization="enabled"
)
```

Student Experience Example:

```
Student A (struggling with p-values):
1. Question (Medium): "What does p-value represent?" → Incorrect
2. Question (Easy): "Is p-value a probability?" → Correct
3. Question (Easy-Medium): "Which p-value is more significant: 0.03 or 0.08?" → Correct
4. Question (Medium): Definition question → Correct
5. [Hint provided] → Medium question → Correct
6. Quiz ends after achieving mastery at appropriate level

Student B (strong understanding):
1. Question (Medium): Definition → Correct
2. Question (Medium-Hard): Interpretation → Correct
3. Question (Hard): Application scenario → Correct
4. Question (Hard): Complex analysis → Correct
5. Question (Very Hard): Multiple concepts → Correct
6. Quiz ends with mastery at advanced level
```

14.5.2. Personalized Learning Paths

Create flexible progression routes through content.

Learning Path Structure:

```
learning_path = {
    "name": "Statistics Fundamentals",
    "type": "personalized",

    "modules": [
        {
            "module_id": "mod_1",
            "name": "Descriptive Statistics",
            "required": True,
            "estimated_hours": 4,

            "prerequisites": [],

            "content": {
                "lecture": "video_01.mp4",
                "reading": ["chapter_1.pdf"],
                "practice": "quiz_descriptive"
            },

            "mastery_criteria": {
                "quiz_score_min": 0.80,
                "practice_problems_min": 10
            },

            "differentiation": {
                "struggling": {
                    "additional_resources": ["tutorial_video_basic.mp4"],
                    "simplified_examples": True,
                    "extra_practice": "quiz_descriptive_review"
                },
                "advanced": {
                    "enrichment": ["advanced_applications.pdf"],
                    "challenge_problems": True,
                    "skip_basic_practice": True
                }
            }
        },
        {
            "module_id": "mod_2",
            "name": "Probability Basics",
            "required": True,
            "estimated_hours": 5,

            "prerequisites": ["mod_1"],

            "adaptive_sequencing": {
                "if_strong_in_mod_1": {
```

```
                    "start_difficulty": "medium",
                    "skip_content": ["probability_intro_basic"]
                },
                "if_weak_in_mod_1": {
                    "start_difficulty": "easy",
                    "add_content": ["math_review", "probability_intuition"]
                }
            }
        },
        {
            "module_id": "mod_3",
            "name": "Sampling Distributions",
            "required": True,
            "branch_point": True,

            "branches": {
                "conceptual_focus": {
                    "description": "Emphasis on intuition and interpretation",
                    "suitable_for": ["low_math_confidence", "applied_focus"],
                    "content_variant": "conceptual"
                },
                "mathematical_focus": {
                    "description": "Emphasis on derivations and proofs",
                    "suitable_for": ["high_math_confidence", "theoretical_focus"],
                    "content_variant": "mathematical"
                },
                "balanced": {
                    "description": "Mix of concepts and mathematics",
                    "suitable_for": ["default"],
                    "content_variant": "standard"
                }
            }
        }
    ],

    "path_selection": {
        "method": "automatic",  # or "student_choice", "instructor_assigned"
        "based_on": [
            "profile_background",
            "performance_to_date",
            "stated_goals",
            "confidence_levels"
        ],
        "allow_path_switching": True,
        "review_checkpoints": [3, 6, 9]  # Module numbers
    }
}
```

```
# Assign learning path
kai.learning_paths.assign(
    course_id="course_abc",
    path_config=learning_path,
    students="all",  # or specific student IDs
    start_date="2024-09-01"
)
```

14.5.3. Personalized Content Recommendations

Suggest resources based on individual needs.

Recommendation Engine:

```
recommendations = {
    "triggers": {
        "low_quiz_score": {
            "threshold": 0.70,
            "action": "recommend_review_materials"
        },
        "missed_concept": {
            "threshold": 0.60,
            "action": "recommend_focused_practice"
        },
        "high_performance": {
            "threshold": 0.95,
            "action": "recommend_enrichment"
        },
        "upcoming_assignment": {
            "days_before": 3,
            "action": "recommend_preparation_materials"
        }
    },

    "resource_types": {
        "videos": {
            "short_tutorials": "5-10 minutes, concept-focused",
            "detailed_lectures": "20-30 minutes, comprehensive",
            "worked_examples": "10-15 minutes, step-by-step"
        },
        "readings": {
            "textbook_sections": "main reference",
            "supplementary_articles": "alternative explanations",
            "simplified_guides": "accessible introductions"
        },
        "practice": {
```

```
            "targeted_drills": "specific weak areas",
            "mixed_practice": "multiple concepts",
            "challenge_problems": "advanced applications"
        },
        "interactive": {
            "simulations": "visual/interactive learning",
            "tutorials": "guided practice",
            "self_assessments": "check understanding"
        }
    },

    "personalization_factors": [
        "current_performance",
        "learning_style_preference",
        "time_available",
        "recent_struggles",
        "upcoming_assessments"
    ]
}

# Get recommendations for student
recs = kai.recommendations.get(
    student_id="student_123",
    course_id="course_abc",
    context="preparation_for_exam"
)

print("Recommended for you:")
for rec in recs:
    print(f"- {rec.title} ({rec.type})")
    print(f"  Why: {rec.reason}")
    print(f"  Time: {rec.estimated_minutes} minutes")
    print(f"  Priority: {rec.priority}")
```

14.6. Personalized Feedback

14.6.1. Adaptive Feedback Styles

Customize feedback based on student needs and context.

Configuration:

```
feedback_personalization = {
    "by_student_profile": {
        "struggling": {
            "style": "encouraging",
```

```
            "detail_level": "high",
            "include_hints": True,
            "suggest_resources": True,
            "tone": "supportive"
        },
        "average": {
            "style": "balanced",
            "detail_level": "medium",
            "include_hints": "for_errors_only",
            "suggest_resources": "if_relevant",
            "tone": "professional"
        },
        "advanced": {
            "style": "concise",
            "detail_level": "low",
            "include_hints": False,
            "suggest_challenges": True,
            "tone": "professional"
        }
    },

    "by_assignment_type": {
        "formative": {
            "emphasis": "learning_and_improvement",
            "show_solutions": "after_submission",
            "allow_retry": True
        },
        "summative": {
            "emphasis": "evaluation",
            "show_solutions": "after_deadline",
            "allow_retry": False
        }
    },

    "by_performance": {
        "excellent_work": {
            "acknowledge_strengths": True,
            "suggest_next_challenges": True,
            "keep_brief": True
        },
        "good_with_room": {
            "acknowledge_strengths": True,
            "specific_improvements": True,
            "encouragement": True
        },
        "needs_improvement": {
            "focus_on_key_issues": True,
```

```
            "provide_concrete_steps": True,
            "offer_support_resources": True,
            "emphasize_growth": True
        }
    },

    "special_contexts": {
        "first_assignment": {
            "extra_guidance": True,
            "orient_to_expectations": True
        },
        "after_struggle": {
            "acknowledge_improvement": True,
            "rebuild_confidence": True
        },
        "accommodation_needs": {
            "format_adjustments": True,
            "clarity_emphasis": True
        }
    }
}

# Apply personalized feedback
kai.courses.update(
    course_id="course_abc",
    feedback_settings=feedback_personalization
)
```

Example Feedback Comparison:

```
Assignment: Problem Set on Hypothesis Testing
Question: "Calculate and interpret the p-value"

--- STRUGGLING STUDENT (Score: 55%) ---
Great effort on this problem set! I can see you're working hard to understand
hypothesis testing. Let's focus on building your understanding of p-values.

For question 3, you calculated p = 0.08 correctly - nice work on the
calculation! However, the interpretation needs some adjustment. Remember:
a p-value tells us the probability of getting our result (or more extreme)
if the null hypothesis were true.

  Key Point: Compare your p-value (0.08) to alpha (0.05). Since 0.08 > 0.05,
we FAIL TO REJECT the null hypothesis.

Try thinking of it this way: If p is low (< alpha), the null must go!
If p is high (> alpha), the null can stay!
```

```
 Helpful Resources:
- Video: "Understanding P-values Intuitively" (8 min)
- Practice: "P-value Interpretation Quiz" (10 problems)

You're making progress! Focus on these concepts before the next assignment.

--- AVERAGE STUDENT (Score: 82%) ---
Good work on this problem set. Your calculations are accurate and you're
demonstrating solid understanding of the core concepts.

For question 3: Your p-value calculation (p = 0.08) is correct. Your
interpretation is mostly right, but be more specific about what we conclude.
Rather than saying "the result is not significant," state clearly: "We fail
to reject the null hypothesis at   = 0.05 because p = 0.08 > 0.05."

For question 5: Good interpretation! Consider also mentioning the practical
significance when discussing your conclusion.

Keep up the consistent work!

--- ADVANCED STUDENT (Score: 96%) ---
Excellent work. Calculations accurate, interpretations clear and precise.

Question 3: Consider discussing Type II error risk when p = 0.08 (close
to boundary).

Challenge: For your next assignment, try incorporating effect size measures
(Cohen's d) alongside p-values for more complete inference.

Strong performance overall.
```

14.6.2. Growth-Focused Feedback

Emphasize improvement and learning trajectories.

```
growth_feedback = {
    "track_progress": True,
    "compare_to_self": True,  # Not to peers
    "highlight_improvements": True,

    "feedback_elements": [
        {
            "element": "progress_acknowledgment",
            "when": "improvement_detected",
            "template": "Your understanding of {topic} has improved significantly! "
                        "Your score increased from {previous_score} to {current_score}."
```

```
        },
        {
            "element": "specific_growth",
            "when": "skill_mastered",
            "template": "You've mastered {skill}! This shows in your {example}. "
                        "This is a big step forward."
        },
        {
            "element": "growth_opportunity",
            "when": "struggle_detected",
            "template": "You're still developing your understanding of {topic}. "
                        "This is normal - it's a challenging concept. Focus on {specific_step}."
        },
        {
            "element": "trajectory_statement",
            "when": "always",
            "template": "You're on a {trajectory} path. Keep {recommendation}."
        }
    ]
}

# Example generated feedback
"""
Your understanding of confidence intervals has improved significantly!
Your score increased from 68% to 85%.

You've mastered the calculation process! This shows in how you correctly
applied the formula in all 5 problems. This is a big step forward.

You're still developing your understanding of interpretation. This is
normal - it's a challenging concept. Focus on connecting the numerical
interval to what it means in context.

You're on an improving path. Keep practicing interpretation with the
real-world examples in Module 4.
"""
```

14.7. Intelligent Study Support

14.7.1. Personalized Study Plans

Generate customized study recommendations.

```
study_plan = kai.study_plans.generate(
    student_id="student_123",
    course_id="course_abc",
```

```
    goal="prepare_for_midterm",

    parameters={
        "exam_date": "2024-10-20",
        "current_date": "2024-10-06",
        "available_study_hours": 12,  # Total hours available

        "focus_areas": "auto",  # Automatically identify from profile

        "preferences": {
            "session_length": 60,  # 60-minute study sessions
            "break_length": 15,
            "study_times": ["evening", "weekend_morning"]
        }
    }
)

"""
Generated Plan:

  YOUR PERSONALIZED MIDTERM STUDY PLAN

Time available: 14 days, 12 hours total

  Focus Areas (based on your performance):
1. P-value interpretation (PRIORITY - current: 58%)
2. Confidence intervals (current: 72%)
3. Hypothesis testing review (current: 81%)

  Week 1: Building Foundation
Monday, Oct 7 (7-8 PM):
  - Review: P-value concepts
  - Watch: "P-values Explained" (15 min)
  - Practice: 10 interpretation problems
  - Goal: Understand basic interpretation

Wednesday, Oct 9 (7-8 PM):
  - Practice: P-value calculations
  - Complete: Practice quiz
  - Goal: 80% accuracy on quiz

Friday, Oct 11 (7-8 PM):
  - Review: Confidence intervals
  - Redo: Problem set #4 (focus on weak areas)
  - Goal: Improve to 85%

Saturday, Oct 12 (10 AM-12 PM):
```

```
  - Deep work: Hypothesis testing
  - Complete: Practice exam sections 1-2
  - Review: Errors and concepts

 Week 2: Integration and Practice
Monday, Oct 14 (7-8 PM):
  - Mixed practice: All concepts
  - Complete: Practice exam section 3
  - Goal: Identify remaining weak areas

Wednesday, Oct 16 (7-8 PM):
  - Targeted review: Weak areas from practice exam
  - Office hours: Ask specific questions
  - Goal: Clarify confusion points

Friday, Oct 18 (7-8 PM):
  - Full practice exam
  - Time yourself: 90 minutes
  - Goal: 85% overall

Saturday, Oct 19 (10 AM-12 PM):
  - Review: Practice exam errors
  - Final clarifications
  - Light review: Key formulas and concepts
  - Goal: Confident and prepared

Sunday, Oct 20: EXAM DAY
  - Morning: Quick review of note sheet
  - Stay calm and confident!

 Track your progress:
 Complete each session
 Note what's still unclear
 Adjust plan if needed
"""
```

14.7.2. Just-in-Time Help

Provide support exactly when students need it.

```
jit_support = {
    "triggers": {
        "stuck_on_problem": {
            "detect": "time_spent > 5_minutes_no_progress",
            "action": "offer_hint"
        },
```

```
        "repeated_error": {
            "detect": "same_mistake_3_times",
            "action": "provide_explanation_and_example"
        },
        "low_confidence": {
            "detect": "student_indicates_uncertainty",
            "action": "offer_similar_worked_example"
        },
        "pre_assignment": {
            "detect": "assignment_due_within_24_hours",
            "action": "suggest_preparation_checklist"
        }
    },

    "help_types": {
        "hints": {
            "progressive": True,  # Start subtle, get more specific
            "levels": ["general_direction", "specific_step", "partial_solution"]
        },
        "examples": {
            "similar_problems": True,
            "worked_solutions": True,
            "video_tutorials": True
        },
        "explanations": {
            "concept_review": True,
            "common_errors": True,
            "why_it_matters": True
        }
    },

    "escalation": {
        "if_still_stuck": "suggest_office_hours",
        "if_urgent": "offer_tutoring_connection",
        "if_widespread": "notify_instructor"
    }
}

# Example interaction
"""
[Student spending 7 minutes on problem 3]

 Need a hint?

[Student clicks yes]

Hint 1: Remember to check your assumptions before choosing a test.
```

```
What type of data do you have?

[Student still stuck after 3 more minutes]

Hint 2: You have two independent groups and continuous data.
Which test is designed for this scenario?

[Student enters wrong answer]

Let me show you a similar problem:

Example: Comparing test scores between two different classrooms
- Independent groups: Classroom A vs Classroom B
- Continuous data: Test scores
- Appropriate test: Independent samples t-test

Now try your problem with this pattern in mind.

[If still struggling]
 Office hours are available tomorrow 2-4 PM. Would you like to
save this problem to discuss?
"""
```

14.8. Differentiation Strategies

14.8.1. Multi-Level Assignments

Offer options at different difficulty levels.

```
differentiated_assignment = {
    "name": "Hypothesis Testing Application",
    "type": "tiered",

    "core_assignment": {
        "required_for_all": True,
        "description": "Basic hypothesis test with provided data",
        "points": 70
    },

    "tiers": {
        "tier_1_foundations": {
            "description": "Guided steps with templates",
            "suitable_for": "students_scoring_below_75",
            "scaffolding": "high",
            "points_possible": 70,
```

```
            "supports": [
                "step_by_step_template",
                "formula_sheet_provided",
                "example_worked_problem",
                "calculation_hints"
            ]
        },

        "tier_2_standard": {
            "description": "Standard problems with minimal scaffolding",
            "suitable_for": "students_scoring_75_to_90",
            "scaffolding": "low",
            "points_possible": 85,

            "additional_challenge": [
                "choose_appropriate_test",
                "explain_assumptions",
                "interpret_in_context"
            ]
        },

        "tier_3_advanced": {
            "description": "Complex scenarios requiring synthesis",
            "suitable_for": "students_scoring_above_90",
            "scaffolding": "none",
            "points_possible": 100,

            "extensions": [
                "multiple_valid_approaches",
                "discuss_limitations",
                "suggest_alternative_analyses",
                "real_world_application"
            ]
        }
    },

    "tier_selection": {
        "method": "student_choice",  # or "auto_assigned", "instructor_assigned"
        "show_recommendation": True,
        "allow_tier_switching": True,
        "deadline_for_switch": "24_hours_after_release"
    }
}

# Student experience
"""
New Assignment Available: Hypothesis Testing Application
```

```
Based on your performance, we recommend: Tier 2 (Standard)

Choose your level:
  Tier 1: Foundations (more guidance, 70 points possible)
  Tier 2: Standard (some guidance, 85 points possible) [RECOMMENDED]
  Tier 3: Advanced (independent work, 100 points possible)

You can switch tiers within 24 hours of starting.
All tiers cover the same core concepts - choose the level where
you'll learn best!
"""
```

14.8.2. Flexible Pacing

Allow students to progress at appropriate speeds.

```
flexible_pacing = {
    "course_id": "course_abc",

    "pacing_options": {
        "accelerated": {
            "description": "Fast pace for strong background",
            "weeks": 10,
            "recommended_for": "prior_knowledge_or_high_confidence"
        },
        "standard": {
            "description": "Regular semester pace",
            "weeks": 14,
            "recommended_for": "most_students"
        },
        "supported": {
            "description": "Extended time with extra support",
            "weeks": 16,
            "recommended_for": "need_more_time_or_support"
        }
    },

    "flexibility": {
        "allow_pace_changes": True,
        "change_deadline": "week_4",
        "core_deadlines_fixed": ["midterm", "final"],
        "flexible_deadlines": ["homework", "quizzes", "labs"]
    },

    "support_by_pace": {
        "accelerated": {
```

```
            "enrichment_content": True,
            "advanced_applications": True,
            "minimal_review": True
        },
        "supported": {
            "extra_practice": True,
            "review_sessions": True,
            "checkpoint_meetings": True
        }
    }
}
```

14.9. Monitoring and Analytics

14.9.1. Personalization Effectiveness

Track how personalization impacts learning.

```
# View personalization analytics
analytics = kai.personalization.get_analytics(
    course_id="course_abc",
    timeframe="semester"
)

print("Personalization Impact:")
print(f"Students receiving personalized content: {analytics.personalized_pct}%")
print(f"Avg improvement with personalization: +{analytics.improvement_points} points")
print(f"Engagement increase: +{analytics.engagement_increase}%")
print(f"Student satisfaction: {analytics.satisfaction_rating}/5")

# By student segment
for segment in analytics.segments:
    print(f"\n{segment.name}:")
    print(f"  Benefit from personalization: {segment.impact}")
    print(f"  Most effective features: {segment.top_features}")

# A/B test results (if running)
if analytics.ab_test_results:
    print("\nA/B Test: Personalized vs. Standard")
    print(f"Personalized group avg: {analytics.personalized_avg}")
    print(f"Control group avg: {analytics.control_avg}")
    print(f"Difference: {analytics.difference} (p = {analytics.p_value})")
```

14.9.2. Student Progress Dashboards

Provide students with personalized progress views.

```
student_dashboard = {
    "components": [
        {
            "widget": "progress_overview",
            "shows": {
                "current_grade": True,
                "trend": True,
                "comparison_to_goals": True,
                "not_comparison_to_peers": True  # Avoid peer comparison
            }
        },
        {
            "widget": "strengths_and_opportunities",
            "shows": {
                "mastered_concepts": True,
                "in_progress_concepts": True,
                "needs_work_concepts": True,
                "recommended_focus": True
            }
        },
        {
            "widget": "personalized_recommendations",
            "shows": {
                "suggested_activities": True,
                "preparation_for_upcoming": True,
                "study_plan": True
            }
        },
        {
            "widget": "engagement_insights",
            "shows": {
                "time_on_task": True,
                "participation_rate": True,
                "resource_usage": True,
                "optimal_study_times": True
            }
        }
    ],

    "privacy": {
        "student_controls_visibility": True,
        "no_peer_comparison": True,
        "emphasize_growth": True
    }
```

```
}
```

14.10. Best Practices

14.10.1. Ethical Personalization

> **!** Privacy and Equity
>
> Always prioritize student privacy and ensure personalization enhances rather than limits opportunities.

Guidelines:

- **Transparency**: Inform students how personalization works
- **Control**: Let students opt out or adjust settings
- **Equity**: Ensure all students have path to success
- **Growth Mindset**: Frame as support, not limitation
- **Avoid**: Labeling students or limiting opportunities
- **Avoid**: Sharing personal learning data without consent

Implementation:

```
personalization_settings = {
    "transparency": {
        "explain_to_students": True,
        "show_why_content_selected": True,
        "privacy_policy_visible": True
    },

    "student_control": {
        "allow_opt_out": True,
        "allow_preference_changes": True,
        "allow_path_override": True
    },

    "equity_safeguards": {
        "all_students_access_all_content": True,
        "recommendations_not_restrictions": True,
        "prevent_tracking_harm": True
    }
}
```

14.10.2. Start Small, Scale Gradually

Recommended Progression:

1. **Semester 1**: Adaptive quizzes only

2. **Semester 2**: Add personalized feedback
3. **Semester 3**: Introduce learning paths
4. **Semester 4+**: Full personalization suite

14.10.3. Regular Review

```
review_schedule = {
    "weekly": [
        "Check personalization accuracy",
        "Review student feedback",
        "Adjust immediate issues"
    ],
    "monthly": [
        "Analyze effectiveness data",
        "Identify patterns",
        "Refine algorithms"
    ],
    "semester": [
        "Comprehensive evaluation",
        "Student surveys",
        "Plan improvements"
    ]
}
```

14.11. Troubleshooting

14.11.1. Personalization Not Helping

Problem: Students not benefiting from personalized features

Diagnosis:

```
diagnosis = kai.personalization.diagnose(course_id="course_abc")
print(diagnosis.issues)
```

Common Causes: - Insufficient data for accurate profiling (first few weeks) - Personalization parameters too conservative - Students not engaging with recommended content

Solutions: - Allow 3-4 weeks for profile building - Adjust adaptation speed - Survey students about barriers - Provide incentives for engaging with recommendations

14.11.2. Too Much Differentiation

Problem: Managing multiple versions is overwhelming

Solutions: - Reduce tier options (3 is often maximum) - Use automatic tier assignment - Create templates for differentiated content - Start with one differentiated assignment type

14.11.3. Privacy Concerns

Problem: Students worried about data collection

Solutions:

```
# Implement transparency dashboard
kai.privacy.create_student_dashboard(
    show_what_data_collected=True,
    show_how_data_used=True,
    provide_export_option=True,
    provide_delete_option=True
)

# Send clear communication
kai.communications.send_privacy_notice(
    to="all_students",
    content="personalization_transparency_template"
)
```

14.12. Support Resources

- **Personalization Workshops**: Schedule training
- **Research Base**: Evidence for adaptive learning
- **Email**: personalization@chi2labs.com
- **Community**: Share strategies

Next Steps: - Review best practices for effective personalization - Set up grading to work with adaptive assessments - Create templates for differentiated content - Explore analytics to measure personalization impact

15. Best Practices

Proven strategies for maximizing Kai's impact

15.1. Overview

This guide compiles best practices from hundreds of educators who have successfully integrated Kai into their teaching workflows. Learn from their experiences to maximize impact while avoiding common pitfalls.

15.2. Core Principles

15.2.1. 1. Start Small, Scale Gradually

> The 80/20 Rule
>
> 80% of Kai's value comes from 20% of its features. Master the Feedback Workflow first, then add other features as you become comfortable.

Recommended Progression: - **Week 1-2**: Feedback Workflow only - **Week 3-4**: Add Pop Quiz for formative assessment - **Week 5+**: Explore advanced features (SafeStream, custom integrations)

Why This Works: - Reduces cognitive load on you and students - Builds confidence through quick wins - Allows time to establish routines - Creates data baseline for comparison

15.2.2. 2. Communicate Transparently

Students are more receptive when they understand the "why" behind new tools.

First Day Introduction Template:

```
"This semester, I'm using a tool called Kai to help me teach
more responsively. Here's what that means for you:

  What it does:
   - Lets you give me quick feedback during class
```

```
  - Helps me know what to review vs. move forward
  - Gets you help faster when you're stuck

  What it doesn't do:
  - Track your attendance or behavior
  - Grade your participation
  - Share your responses with other students

  Why I'm using it:
  - Our class time is limited
  - Everyone learns differently and at different paces
  - I want to focus on what YOU need, not what the schedule says

  What I need from you:
  - Install the app in the first week
  - Respond honestly when I request feedback
  - Give it a fair try for the first month

Questions?"
```

15.2.3. 3. Establish Consistent Routines

Consistency helps students know what to expect and increases participation rates.

Example Routine: - Request feedback at ~25 and ~50 minutes in a 75-minute class - Use the same notification sound/pattern - Always acknowledge feedback: “Thanks for the quick responses…” - Act on feedback within the same class session when possible

15.3. Feature-Specific Best Practices

15.3.1. Feedback Workflow

15.3.1.1. Timing

Optimal Request Frequency: | Class Length | Recommended Requests | |————–|————————| | 50 minutes | 1-2 times | | 75 minutes | 2-3 times | | 110 minutes | 3-4 times | | 3+ hours | Every 30-40 minutes |

Best Times to Request: - After introducing a complex new concept - Before transitioning to next major topic - When you sense confusion (body language, questions) - After worked examples - During active discussions or group work - During exams or individual work time

15.3.1.2. Response Rate Optimization

Achieving 70%+ Response Rates:

1. **Make it effortless**:
   - Keep app in easy-to-reach place
   - Use simple, clear questions
   - Limit to 2-minute response windows
   - Enable one-tap responses
2. **Demonstrate value**:
   - Act on feedback immediately when possible
   - Explicitly mention: "Based on your feedback, we're going to…"
   - Show trends over time: "You've all improved on…"
3. **Gentle accountability**:
   - Display response count (not individuals): "23 out of 28 responded - thanks!"
   - Celebrate high participation weeks
   - Occasionally explain how specific feedback helped
4. **Remove barriers**:
   - Ensure app notifications work
   - Address privacy concerns early
   - Make installation dead simple
   - Provide tech support in first week

15.3.1.3. Interpreting Results

Decision Matrix:

% Confused	Student Count	Recommended Action
60%+	Any	Full class review required
30-60%	Any	Quick re-explanation + resources
10-30%	<5 students	Individual resources only
<10%	Any	Move forward, note for future

Reading Between the Lines: - **High confusion + low responses**: Concept may be more confusing than you think (confused students didn't respond) - **Low confusion + high responses**: Good understanding, confident students - **Varied responses**: Consider multiple explanation approaches needed

15.3.2. Pop Quiz Workflow

15.3.2.1. Quiz Design

Effective Quiz Characteristics: - **Length**: 3-5 questions (completable in 5 minutes) - **Timing**: End of topic segment, not middle - **Difficulty**: Mix of easy (confidence) and medium (diagnostic) - **Stakes**: Low/no stakes (formative assessment only) - **Frequency**: 1-2 per class session

Question Types by Purpose:

Purpose	Question Type	Example
Recall	Multiple choice	"Which formula represents standard error?"
Understanding	True/false	"Increasing sample size decreases standard error"
Application	Short answer	"Calculate SE for this dataset"
Analysis	Scenario-based	"Why does this result violate assumptions?"

15.3.2.2. Using Quiz Data

Immediate Actions: - Review questions with <50% correct rate - Identify students who consistently struggle - Adjust pace based on overall performance

Long-term Tracking: - Monitor improvement over time - Identify challenging concepts for future semesters - Correlate with exam performance

15.3.3. SafeStream Workflow

15.3.3.1. Configuration

Sensitivity Settings by Context:

Context	Recommended Level	Rationale
Undergraduate discussion	Medium	Balance safety with open dialogue
Graduate seminar	Low	More mature, self-regulating
Online forums	High	Less supervision, more risk
Peer review	Medium	Encourage constructive criticism

Custom Keyword Lists: - Include institution-specific terms - Add course-specific sensitive topics - Update based on emerging issues - Review flagged content monthly

15.3.3.2. Responding to Flags

Action Protocol:

1. **Immediate (within 1 hour)**:
   - Review flagged content
   - Assess severity (low/medium/high)
   - Document incident
2. **Follow-up (within 24 hours)**:
   - **High severity**: Contact student, administrator, or counseling
   - **Medium severity**: Private conversation with student
   - **Low severity**: Monitor for patterns
3. **Prevention (ongoing)**:
   - Reinforce community guidelines
   - Address patterns in class (anonymously)
   - Adjust sensitivity if over-flagging

15.4. Optimization Tips

15.4.1. Maximizing Student Engagement

Proven Strategies:

1. **Gamification Elements**:
   - Weekly "participation hero" recognition
   - Class-wide response rate goals
   - Semester progress tracking
2. **Intrinsic Motivation**:
   - Show how feedback directly improved class
   - Share anonymous success stories
   - Connect to learning outcomes
3. **Remove Friction**:
   - First-week app installation support
   - Troubleshooting during office hours
   - Simple, consistent requests

15.4.2. Time Management

Making Kai Time-Neutral:

Many educators worry Kai will add work. Here's how to make it time-neutral or even time-saving:

Time Saved: - Less time re-teaching to entire class (focus on actual needs) - Fewer office hours answering the same questions - Less time grading (with smart grading) - Faster curriculum adjustments (data-driven)

Time Invested: - Initial setup (one-time): 30-60 minutes - Weekly review: 15-20 minutes - In-class requests: 2-3 minutes per request

Net Result: Most educators report 1-3 hours saved per week after the first month.

Efficiency Tips: - Review analytics once weekly, not daily - Set up automated responses for common issues - Use templates for resource distribution - Batch similar tasks (all quiz creation on Fridays)

15.4.3. Data Privacy and Ethics

FERPA Compliance Checklist: - [ ] Student data encrypted in transit and at rest - [ ] No sharing of individual responses without consent - [ ] Configurable anonymization options - [ ] Data retention policies documented - [ ] Students can opt out without penalty

Ethical Guidelines:

1. **Transparency**: Students should know what data is collected and how it's used
2. **Consent**: Provide clear opt-in (or opt-out) mechanisms
3. **Equity**: Ensure non-participants aren't disadvantaged
4. **Privacy**: Never publicly identify struggling students
5. **Purpose**: Use data only for educational improvement

Sample Privacy Statement:

```
Data Collection and Use:
- Kai collects your feedback responses and quiz answers
- Data is used only to improve this class
- Individual responses are private to the instructor
- Aggregated, anonymous data may be used to improve Kai
- You can request data deletion at semester end
- Non-participation will not affect your grade
```

15.5. Common Pitfalls

15.5.1. What to Avoid

⚠ Pitfall #1: Over-reliance on Automation

Problem: Letting Kai make all teaching decisions **Solution**: Use Kai as a tool for insights, not a replacement for judgment

⚠ Pitfall #2: Feedback Fatigue

Problem: Requesting feedback too frequently **Solution**: Stick to 2-3 times per class maximum; quality over quantity

⚠ Pitfall #3: Ignoring Low Response Rates

Problem: Making decisions based on 20% participation **Solution**: Address barriers; need >50% for reliable insights

⚠ Pitfall #4: Public Shaming

Problem: Identifying struggling students publicly **Solution**: Always keep individual data private; celebrate effort

⚠ Pitfall #5: Analysis Paralysis

Problem: Spending hours analyzing every data point **Solution**: Focus on actionable insights; weekly review is enough

15.5.2. Red Flags

Stop and reassess if you notice: - Response rates dropping below 40% - Students complaining about frequency - You're spending >1 hour/week on analytics - Data contradicts your observations - Technology overshadowing teaching

15.6. Success Metrics

15.6.1. How to Measure Impact

Short-term Indicators (Week 1-4): - App installation rate: Target 70%+ - Average response rate: Target 60%+ - Time to act on feedback: Target <5 minutes - Student feedback sentiment: Target "helpful" or "neutral"

Medium-term Indicators (Month 2-3): - Quiz score trends: Target upward - Office hours volume: Target slight decrease - Concept confusion trends: Target improving - Student engagement: Target stable or improving

Long-term Indicators (Semester): - Exam performance: Compare to previous semesters - Course evaluations: Look for "responsive" comments - Drop/withdrawal rates: Target decrease - Student-reported learning: Target increase

Sample Survey Questions:

```
1. Kai helped me learn better in this class:
   [ ] Strongly Agree [ ] Agree [ ] Neutral [ ] Disagree [ ] Strongly Disagree

2. The instructor used feedback to adjust teaching:
   [ ] Always [ ] Often [ ] Sometimes [ ] Rarely [ ] Never

3. Responding to feedback requests was:
   [ ] Very easy [ ] Easy [ ] Neutral [ ] Difficult [ ] Very difficult

4. What worked well about Kai?
   [Open response]

5. What could be improved?
   [Open response]
```

15.7. Case Studies

15.7.1. Large Lecture: Dr. Martinez, Physics 101 (250 students)

Challenge: Maintaining engagement in large lecture hall

Kai Implementation: - Feedback requests every 25 minutes - Pop quizzes before problem-solving sessions - Automated resource distribution to struggling students

Results: - Response rate: 73% average - Exam scores: +8% vs. previous semester - Student evaluation: +0.6 points (5-point scale) - Time saved: 2 hours/week (fewer office hour repeats)

Key Insight: "Kai let me feel like I was teaching a class of 30, not 250."

15.7.2. Hybrid Course: Prof. Johnson, Literature Seminar (18 students)

Challenge: Balancing in-person and remote student needs

Kai Implementation: - Asynchronous feedback on readings - In-class discussion prep quizzes - Real-time engagement tracking for both groups

Results: - Participation parity: Remote students 92% vs. in-person 95% - Discussion quality: Increased per instructor assessment - Student satisfaction: 94% "felt equally engaged"

Key Insight: "Kai eliminated the 'second-class' feeling for remote students."

15.7.3. Flipped Classroom: Prof. Kim, Calculus (45 students)

Challenge: Ensuring pre-class work completion and comprehension

Kai Implementation: - Pre-class quizzes on video content - Class time optimized based on quiz results - Targeted interventions for struggling students

Results: - Pre-class completion: 89% (vs. 62% previous semester) - Class time efficiency: +30% (less re-teaching) - Student mastery: +12% on cumulative final

Key Insight: "I finally know what students learned before class starts."

15.8. Advanced Strategies

15.8.1. Cross-Course Coordination

For departments with multiple instructors using Kai:

1. **Shared Best Practices**:
   - Weekly lunch meetings to share insights
   - Common feedback question bank
   - Coordinated quiz standards
2. **Longitudinal Tracking**:
   - Identify prerequisite gaps early
   - Track student improvement across courses
   - Inform curriculum sequencing
3. **Resource Pooling**:
   - Shared content library
   - Collaborative rubric development
   - Joint analytics review

15.8.2. Research Integration

Use Kai data for scholarship of teaching and learning (SoTL):

1. **IRB Approval**: Get approval for using aggregated, de-identified data
2. **Hypothesis Testing**: Test specific pedagogical interventions
3. **Publication**: Share findings in discipline-specific journals
4. **Student Co-authors**: Involve students in research design

15.9. Next Steps

15.9.1. Immediate Actions

1. Review your current Kai usage against these best practices
2. Identify 1-2 areas for improvement
3. Set specific, measurable goals for next month

15.9.2. Resources

- Educator Community Forum
- Download: Best Practices Checklist
- Self-Assessment Tool
- Monthly Best Practices Newsletter

15.9.3. Get Help

- 1-on-1 Coaching
- Advanced Training Sessions
- Peer Mentorship Program

Remember: The best practice is the one that works for you and your students. Use these guidelines as a starting point, not rigid rules.

16. Troubleshooting Guide

Solutions to common issues and problems

16.1. Overview

This guide provides solutions to the most common issues encountered when using Kai the AI. Issues are organized by category for quick reference.

> Quick Diagnosis
>
> Most issues fall into one of these categories: - **Connection**: API keys, network, LMS integration - **Mobile App**: Notifications, enrollment, sync issues - **Features**: Feedback, quizzes, analytics not working - **Performance**: Slow response, timeouts, data lag

16.2. Connection Issues

16.2.1. API Key Errors

16.2.1.1. "Invalid API Key" or "Authentication Failed"

Symptoms: - Cannot connect to Kai services - 401 Unauthorized errors - Integration tests fail

Solutions:

1. **Verify API Key**:

```
# Check for common issues:
- Extra spaces at beginning/end
- Incomplete copy (key should start with "kai_")
- Wrong key type (using test key in production)
```

2. **Regenerate Key**:

   - Dashboard → Settings → API Keys
   - Click "Regenerate Key"
   - Update in all integration points
   - Wait 5 minutes for propagation

3. **Check Key Permissions**:
    - Ensure key has required scopes
    - Verify not expired
    - Confirm account status is active

16.2.1.2. "API Key Limit Exceeded"

Symptoms: - 429 Too Many Requests errors - Features stop working sporadically

Solutions:

1. **Check Usage**:
    - Dashboard → Analytics → API Usage
    - Review current plan limits
    - Identify usage spikes
2. **Optimize Requests**:
    - Reduce feedback request frequency
    - Implement client-side caching
    - Batch API calls where possible
3. **Upgrade Plan**:
    - Contact sales@chi2labs.com
    - Consider institution license for higher limits

16.2.2. LMS Integration Issues

16.2.2.1. Canvas Integration Not Syncing

Symptoms: - Courses not appearing in Kai - Assignments not syncing - Grade updates not pushing back

Diagnostic Steps:

```
# Test connection
Dashboard → Integrations → Canvas → Test Connection

# Check logs
Dashboard → Integrations → Canvas → View Logs
```

Solutions:

16.2.3. LTI Integration

1. **Verify LTI Configuration**:
   - Client ID matches in both systems
   - Redirect URIs exactly match
   - JWK URL accessible from Canvas servers
2. **Check Canvas Permissions**:
   - Admin → Settings → Apps → Kai
   - Ensure all required permissions granted
   - Re-authorize if needed
3. **Test Individual Components**:
   - Launch LTI tool from Canvas
   - Check if courses appear
   - Verify user mapping

16.2.4. API Integration

1. **Verify Access Token**:
   - Token not expired
   - Has required scopes:
     - `url:GET|/api/v1/courses`
     - `url:POST|/api/v1/courses/:course_id/assignments`
     - `url:PUT|/api/v1/courses/:course_id/assignments/:id/submissions/:user_id`
2. **Check Network Access**:
   - Kai can reach Canvas API endpoint
   - No firewall blocking
   - HTTPS properly configured
3. **Review Rate Limits**:
   - Canvas API rate limits not exceeded
   - Implement exponential backoff
   - Contact Canvas admin if needed

16.2.4.1. Blackboard Integration Issues

Common Issues:

1. **Building Block Not Installing**:

   ```
   Error: "Incompatible with current Blackboard version"
   Solution: Download version-specific .war file from Kai dashboard
   ```

2. **Features Not Appearing**:

```
Error: Building block installed but no Kai features visible
Solution:
- System Admin → Building Blocks → Kai → Set to Available
- Clear Blackboard cache
- Wait 5-10 minutes for propagation
```

3. **Grade Sync Failing**:

```
Error: Grades not appearing in Blackboard gradebook
Solution:
- Verify grade column exists in Blackboard
- Check Kai has write permissions
- Review Blackboard logs: System Admin → Logs → Filter "Kai"
```

16.2.4.2. Moodle Integration Issues

Plugin Not Recognized:

1. **Installation Path**: Ensure plugin installed to correct directory

```
Correct: /path/to/moodle/local/kai/
Incorrect: /path/to/moodle/kai/
```

2. **Permissions**: Set proper file permissions

```
chmod -R 755 /path/to/moodle/local/kai/
chown -R www-data:www-data /path/to/moodle/local/kai/
```

3. **Database Update**: Manually trigger update

```
Site Administration → Notifications → Update Database Now
```

16.3. Mobile App Issues

16.3.1. Notifications Not Working

16.3.1.1. Students Not Receiving Push Notifications

Systematic Diagnosis:

1. **Device Settings**:

```
iOS: Settings → Notifications → Kai Student → Allow Notifications
Android: Settings → Apps → Kai Student → Notifications → Enabled
```

2. **App Settings**:

```
Open Kai app → Settings → Notifications → Feedback Requests: ON
```

3. **Network Issues**:

- Ensure device has internet connection
- Check if other apps receive notifications
- Try WiFi vs. cellular data

4. **Server-Side Check**:

   ```
   Dashboard → Course → Students → [Student Name] → Notification Status
   Should show: "Last notification delivered: [timestamp]"
   ```

Solutions by Platform:

16.3.2. iOS

Do Not Disturb / Focus Mode:

```
Settings → Focus → [Current Mode] → Apps → Kai → Allow
```

Notification Summary:

```
Settings → Notifications → Scheduled Summary
Ensure Kai is NOT in scheduled summary (needs immediate delivery)
```

Reset Notifications:

```
1. Delete Kai app
2. Restart device
3. Reinstall Kai
4. Grant notification permission
5. Re-enroll in course
```

16.3.3. Android

Battery Optimization:

```
Settings → Apps → Kai Student → Battery → Don't optimize
```

Data Saver:

```
Settings → Network & Internet → Data Saver → OFF
Or: Allow Kai to use data when Data Saver is on
```

Notification Channels:

```
Settings → Apps → Kai Student → Notifications
Enable ALL channels, especially "Feedback Requests"
```

16.3.4. Enrollment Issues

16.3.4.1. Students Can't Enroll in Course

Error: "Invalid Enrollment Code"

Solutions:

1. **Verify Code**:
   - Check for typos (0 vs O, 1 vs l)
   - Confirm code hasn't expired
   - Ensure using current semester code
2. **Regenerate Code**:

   ```
   Dashboard → Course → Settings → Enrollment → Regenerate Code
   ```

3. **Use Direct Link Instead**:

   ```
   Dashboard → Course → Settings → Enrollment → Copy Link
   Share link directly with students
   ```

Error: "Course Not Found"

Solutions:

1. **Publish Course**:

   ```
   Dashboard → Course → Settings → Status → Published
   ```

2. **Check Enrollment Period**:

   ```
   Dashboard → Course → Settings → Enrollment Window
   Adjust start/end dates if needed
   ```

16.3.4.2. App Won't Sync Course Data

Symptoms: - Course appears empty - Assignments missing - Outdated information

Solutions:

1. **Manual Sync**:

   ```
   Pull down on course list to refresh
   Or: Settings → Sync → Sync Now
   ```

2. **Clear Cache**:

   ```
   Settings → Storage → Clear Cache
   (Does NOT delete enrollment or data)
   ```

3. **Re-enrollment**:

```
Settings → Courses → [Course] → Unenroll
Re-enroll using enrollment code/link
```

16.4. Feature Issues

16.4.1. Feedback Workflow Not Working

16.4.1.1. No Feedback Requests Appearing

Instructor Side:

1. **Verify Feature Enabled**:

   ```
   Dashboard → Course → Settings → Workflows → Feedback: ON
   ```

2. **Check Student Enrollment**:

   ```
   Dashboard → Course → Students → View Enrolled
   Should show student count > 0
   ```

3. **Test with Dummy Account**:

   ```
   Create test student account
   Enroll in course
   Request feedback
   Check test account receives it
   ```

Student Side:

1. **Verify App Installation**:

   ```
   Correct app: "Kai Student" (not "Kai Instructor")
   Latest version installed
   ```

2. **Check Course Enrollment**:

   ```
   Open app → Courses tab
   Course should appear in list
   ```

3. **Notification Settings**:

   ```
   App → Settings → Notifications → Enabled
   ```

16.4.1.2. Low Response Rates (<40%)

Diagnosis:

1. **Check Notification Delivery Rate**:

   ```
   Dashboard → Analytics → Engagement → Notification Delivery
   If <80%: notification issue
   If >80%: engagement issue
   ```

2. **Survey Students**:

   ```
   Quick poll: "Why don't you always respond to feedback requests?"
   Common answers:
   - Didn't see notification
   - Too busy during class
   - Forgot to enable notifications
   - Don't understand the purpose
   ```

Solutions:

Cause	Solution
Technical issues	Address notification problems (see above)
Time pressure	Increase response window (3-5 minutes)
Lack of motivation	Demonstrate value, show impact
Confusion	Re-explain purpose and privacy

16.4.2. Pop Quiz Issues

16.4.2.1. Quiz Questions Not Generating

Error Messages:

```
"Insufficient content for quiz generation"
Solution: Provide more course materials or select broader topic

"Topic not recognized"
Solution: Use more specific or standard terminology

"Generation timeout"
Solution: Reduce question count or simplify topic
```

Diagnostic Steps:

1. **Check Content Library**:

```
Dashboard → Course → Content → Verify uploads
Need: sylllabus, readings, or lecture notes
```

2. **Verify Topic Alignment**:

```
Topic must match content in library
Example: Don't ask about "Regression" if only "Statistics" uploaded
```

3. **Test with Sample Topic**:

```
Use pre-loaded topic: "Introduction to [Your Subject]"
If works: content issue
If fails: system issue (contact support)
```

16.4.2.2. Students See Different Quiz Versions

This is normal: Kai adapts difficulty based on student performance.

If unintended:

```
Dashboard → Quiz → Settings → Adaptive Difficulty → OFF
Enable "Same quiz for all students"
```

16.4.3. Analytics Dashboard Issues

16.4.3.1. Data Not Updating

Symptoms: - Dashboard shows old data - Recent feedback not appearing - Student counts incorrect

Solutions:

1. **Check Data Freshness**:

```
Dashboard → Analytics → Last Updated: [timestamp]
If >15 minutes old, force refresh
```

2. **Force Refresh**:

```
Click refresh icon ( ) in top right
Or: Clear browser cache and reload
```

3. **Verify Data Pipeline**:

```
Dashboard → System Status → Data Pipeline: Should be "Healthy"
If degraded: wait 30 minutes and retry
```

16.4.3.2. Export Failing

Error: "Export timeout" or "File too large"

Solutions:

1. **Reduce Date Range**:

   ```
   Instead of: Full semester
   Try: One month at a time
   ```

2. **Filter Data**:

   ```
   Export specific metrics instead of "All data"
   ```

3. **Alternative Export Methods**:

   ```
   Use API for programmatic access
   Or: Request bulk export from support
   ```

16.5. Performance Issues

16.5.1. Slow Response Times

16.5.1.1. Dashboard Loading Slowly

Quick Fixes:

1. **Browser Cache**:

   ```
   Chrome: Ctrl+Shift+Delete → Clear cache
   Firefox: Ctrl+Shift+Delete → Clear cache
   Safari: Cmd+Option+E
   ```

2. **Disable Browser Extensions**:

   ```
   Try incognito/private mode
   If faster: browser extension conflict
   ```

3. **Check Internet Speed**:

   ```
   Visit fast.com or speedtest.net
   Need: >5 Mbps download for smooth experience
   ```

Long-term Solutions:

1. **Optimize Data Views**:

   ```
   Dashboard → Settings → Performance
   Enable: "Lazy load analytics"
   Enable: "Reduce animation"
   ```

2. **Archive Old Courses**:

```
Dashboard → Courses → [Old Course] → Archive
Reduces dashboard load
```

16.5.1.2. API Requests Timing Out

Symptoms: - 504 Gateway Timeout errors - Requests taking >30 seconds - Features fail sporadically

Solutions:

1. **Check System Status**:

```
Visit: status.chi2labs.com
If incident: wait for resolution
If operational: proceed to next step
```

2. **Reduce Payload Size**:

```
Instead of: Batch uploading 100 assignments
Try: Upload 10 at a time
```

3. **Implement Retry Logic**:

```javascript
// Example for developers
async function callKaiAPI(endpoint, data, retries = 3) {
  try {
    return await fetch(endpoint, { body: data });
  } catch (error) {
    if (retries > 0) {
      await sleep(1000 * (4 - retries)); // exponential backoff
      return callKaiAPI(endpoint, data, retries - 1);
    }
    throw error;
  }
}
```

16.6. Error Messages

16.6.1. Common Error Codes

Error Code	Meaning	Solution
400	Bad Request	Check request parameters
401	Unauthorized	Verify API key
403	Forbidden	Check account permissions
404	Not Found	Verify resource ID
429	Rate Limited	Wait and retry, or upgrade plan

Error Code	Meaning	Solution
500	Server Error	Contact support
503	Service Unavailable	Check status.chi2labs.com
504	Gateway Timeout	Reduce request size or retry

16.6.2. Error Message Decoder

"FEATURE_NOT_ENABLED"

```
Cause: Trying to use feature not in your plan
Solution: Upgrade plan or disable feature
```

"STUDENT_LIMIT_EXCEEDED"

```
Cause: More students enrolled than plan allows
Solution: Upgrade plan or remove inactive students
```

"INVALID_COURSE_ID"

```
Cause: Course ID doesn't exist or wrong format
Solution: Verify course ID from dashboard
```

"WEBHOOK_DELIVERY_FAILED"

```
Cause: Your webhook endpoint not responding
Solution: Check endpoint URL, ensure publicly accessible
```

16.7. Platform-Specific Issues

16.7.1. Browser Compatibility

Recommended Browsers: - Chrome 90+ (best performance) - Firefox 88+ - Safari 14+ - Edge 90+ - Internet Explorer (not supported)

If using unsupported browser:

```
Warning message will appear
Solution: Upgrade to supported browser
```

16.7.2. Mobile OS Requirements

Minimum Versions: - iOS: 13.0+ - Android: 8.0 (API level 26)+

If on older version:

```
App won't install from App Store/Play Store
Solution: Update OS or use web interface
```

16.8. Getting Additional Help

16.8.1. Self-Service Resources

1. **Knowledge Base**: help.chi2labs.com
2. **Video Tutorials**: youtube.com/@KaiTheAI
3. **Community Forum**: community.chi2labs.com
4. **System Status**: status.chi2labs.com

16.8.2. Contact Support

16.8.2.1. Support Tiers

Free & Educator Plans: - Email: support@chi2labs.com - Response time: 24-48 hours - Community forum support

Institution Plans: - Email: priority@chi2labs.com - Response time: 2-4 hours - Dedicated Slack channel - Phone: 1-800-KAI-HELP - Video call support

16.8.2.2. What to Include in Support Requests

Essential Information:

```
1. Issue Description:
   - What were you trying to do?
   - What happened instead?
   - Error message (exact text or screenshot)

2. Environment:
   - Browser/App version
   - Operating system
   - Account type (educator/institution)

3. Steps to Reproduce:
   - Step-by-step instructions
   - Expected vs. actual result
```

```
4. Impact:
   - How many users affected?
   - Is it blocking critical work?

5. What You've Tried:
   - Troubleshooting steps already attempted
   - Any temporary workarounds found
```

Example Good Support Request:

```
Subject: Students not receiving feedback notifications on iOS

Description:
I requested feedback at 2:30 PM EST today. 25 out of 30 students
have iOS devices. None of them received the push notification,
but the 5 Android students did receive it.

Environment:
- Course: PSYCH 101, Section 03
- iOS versions: mostly 15.x and 16.x
- Kai Student app version: 2.4.1

Steps to Reproduce:
1. Open Kai Instructor app
2. Tap "Request Feedback"
3. Wait 2 minutes
4. iOS students show no notification

Impact:
- High: Cannot use feedback workflow
- Affects 80% of my students

Already Tried:
- Verified students have notifications enabled
- Tested with my own iOS device (same issue)
- Checked system status (all green)
- Android students working fine

Request: Please investigate iOS notification delivery issue.
```

16.8.3. Reporting Bugs

Bug Report Template:

```
1. Bug summary (one sentence)
```

```
2. Expected behavior
3. Actual behavior
4. Steps to reproduce
5. Screenshots/videos (if applicable)
6. Environment details
7. Frequency (always/sometimes/once)
```

Submit bugs: - GitHub: github.com/chi2labs/kai-issues - Email: bugs@chi2labs.com

16.8.4. Feature Requests

How to Request Features:

```
1. Check existing requests: community.chi2labs.com/features
2. If new: Create detailed description
3. Include use case and expected benefit
4. Vote on existing requests that matter to you
```

Popular recent requests: - Integration with Zoom for attendance tracking - Advanced analytics for longitudinal studies - Custom grading rubric templates - Multi-language support

16.9. Preventive Maintenance

16.9.1. Regular Health Checks

Weekly: - Check notification delivery rate (should be >90%) - Review API usage vs. limits - Test mobile app connectivity

Monthly: - Update mobile apps to latest version - Review and archive old courses - Audit student enrollment lists - Check LMS integration status

Semester Start/End: - Verify all integrations still active - Update student lists - Export analytics data for records - Clean up test data

16.9.2. Backup and Recovery

What's Automatically Backed Up: - All course data and analytics - Student responses and feedback - Quiz results and grading data - Configuration settings

What You Should Back Up: - Custom rubrics (export from dashboard) - Integration credentials (store securely) - Custom content libraries

How to Export Data:

```
Dashboard → Course → Settings → Export
Select: "All data" or specific datasets
Format: CSV, JSON, or PDF
```

Still stuck? Our support team is here to help. Don't hesitate to reach out!

Part IV.

API Reference

17. API Reference

Complete API documentation for Kai the AI

17.1. Base URL

All API requests should be made to:

`https://chi2api.com/v1`

17.2. Authentication

Kai uses API keys for authentication. Include your API key in the Authorization header:

```
curl -H "Authorization: Bearer YOUR_API_KEY" \
  https://chi2api.com/v1/status
```

> **Security Note**
>
> Never expose your API key in client-side code or public repositories.

17.3. Endpoints

17.3.1. Grading

17.3.1.1. Submit Assignment for Grading

```
POST /assignments/grade
```

Request Body:

```
{
  "assignment_id": "string",
  "student_submission": "string",
  "rubric_id": "string (optional)",
  "grading_style": "detailed|concise|points_only"
}
```

Response:

```
{
  "grade_id": "string",
  "score": 85,
  "feedback": "string",
  "suggestions": ["string"],
  "timestamp": "2024-01-01T00:00:00Z"
}
```

Example:

17.3.2. Python

```
import requests

response = requests.post(
    "https://chi2api.com/v1/assignments/grade",
    headers={"Authorization": f"Bearer {API_KEY}"},
    json={
        "assignment_id": "CS101-HW1",
        "student_submission": "Student's essay text here...",
        "grading_style": "detailed"
    }
)

result = response.json()
print(f"Score: {result['score']}")
print(f"Feedback: {result['feedback']}")
```

17.3.3. JavaScript

```
const response = await fetch('https://chi2api.com/v1/assignments/grade', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${API_KEY}`,
    'Content-Type': 'application/json'
```

```
  },
  body: JSON.stringify({
    assignment_id: 'CS101-HW1',
    student_submission: 'Student essay text here...',
    grading_style: 'detailed'
  })
});

const result = await response.json();
console.log(`Score: ${result.score}`);
console.log(`Feedback: ${result.feedback}`);
```

17.3.4. cURL

```
curl -X POST https://chi2api.com/v1/assignments/grade \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "assignment_id": "CS101-HW1",
    "student_submission": "Student essay text here...",
    "grading_style": "detailed"
  }'
```

17.3.5. Content Generation

17.3.5.1. Generate Quiz Questions

```
POST /content/quiz
```

Request Body:

```
{
  "topic": "string",
  "difficulty": "easy|medium|hard",
  "question_count": 10,
  "question_types": ["multiple_choice", "true_false", "short_answer"]
}
```

17.3.6. Analytics

17.3.6.1. Get Student Performance

```
GET /analytics/students/{student_id}/performance
```

Query Parameters:

Parameter	Type	Description
start_date	string	ISO 8601 date
end_date	string	ISO 8601 date
course_id	string	Filter by specific course

17.4. Rate Limits

Plan	Requests per minute	Requests per day
Free	10	1,000
Educator	60	10,000
Institution	300	100,000

17.5. Error Handling

Kai uses standard HTTP response codes:

Code	Meaning
200	Success
400	Bad Request - Invalid parameters
401	Unauthorized - Invalid API key
429	Too Many Requests - Rate limit exceeded
500	Internal Server Error

Error Response Format:

```
{
  "error": {
    "code": "invalid_parameter",
    "message": "The 'assignment_id' field is required",
    "details": {}
  }
}
```

17.6. Webhooks

Configure webhooks to receive real-time updates:

```
POST /webhooks
```

```
{
  "url": "https://your-domain.com/webhook",
  "events": ["grading.completed", "content.generated"],
  "active": true
}
```

17.7. SDKs

Official SDKs are available for:

- Python - Python SDK documentation with installation, examples, and API reference
- JavaScript/TypeScript - JavaScript and TypeScript SDK with Node.js and browser support
- Java - Java SDK with Maven/Gradle integration and async support
- Ruby - Ruby SDK with Rails integration and idiomatic patterns

17.8. Support

- API Status: status.chi2api.com
- Developer Forum: forum.chi2labs.com
- Email: developers@chi2labs.com

18. API Endpoints Reference

Complete reference for all Kai API endpoints

18.1. Overview

This page provides detailed documentation for all Kai API endpoints. Each endpoint includes request/response formats, parameters, examples, and common error scenarios.

> **Base URL**
>
> All API requests should use the base URL:
>
> `https://chi2api.com/v1`

18.2. Authentication

All endpoints require authentication via API key in the Authorization header:

```
Authorization: Bearer YOUR_API_KEY
```

See Authentication Guide for details on obtaining and managing API keys.

18.3. Grading Endpoints

18.3.1. Submit Assignment for Grading

Grade a student submission using AI-powered assessment.

```
POST /assignments/grade
```

Request Body:

Parameter	Type	Required	Description
`assignment_id`	string	Yes	Unique identifier for the assignment
`student_submission`	string	Yes	Student's work to be graded (text or URL)
`rubric_id`	string	No	ID of grading rubric to use
`grading_style`	string	No	Style of feedback: `detailed`, `concise`, `points_only` (default: `detailed`)
`max_score`	number	No	Maximum possible score (default: 100)
`return_suggestions`	boolean	No	Include improvement suggestions (default: true)

Response:

```
{
  "grade_id": "grd_abc123",
  "score": 85,
  "max_score": 100,
  "percentage": 85.0,
  "feedback": "Excellent analysis of the main themes...",
  "suggestions": [
    "Consider adding more specific examples",
    "Conclusion could be strengthened"
  ],
  "rubric_scores": {
    "content": 28,
    "organization": 22,
    "grammar": 20,
    "citations": 15
  },
  "confidence": 0.92,
  "timestamp": "2024-01-15T14:30:00Z",
  "processing_time_ms": 1847
}
```

Example:

18.3.2. cURL

```
curl -X POST https://chi2api.com/v1/assignments/grade \
  -H "Authorization: Bearer kai_abc123..." \
  -H "Content-Type: application/json" \
  -d '{
```

```
    "assignment_id": "PSYCH101-ESSAY1",
    "student_submission": "The fundamental attribution error...",
    "grading_style": "detailed",
    "max_score": 100
  }'
```

18.3.3. Python

```python
import requests

response = requests.post(
    "https://chi2api.com/v1/assignments/grade",
    headers={
        "Authorization": f"Bearer {API_KEY}",
        "Content-Type": "application/json"
    },
    json={
        "assignment_id": "PSYCH101-ESSAY1",
        "student_submission": "The fundamental attribution error...",
        "grading_style": "detailed",
        "max_score": 100
    }
)

result = response.json()
print(f"Score: {result['score']}/{result['max_score']}")
print(f"Feedback: {result['feedback']}")
```

18.3.4. JavaScript

```javascript
const response = await fetch('https://chi2api.com/v1/assignments/grade', {
  method: 'POST',
  headers: {
    'Authorization': `Bearer ${API_KEY}`,
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    assignment_id: 'PSYCH101-ESSAY1',
    student_submission: 'The fundamental attribution error...',
    grading_style: 'detailed',
    max_score: 100
  })
});
```

```
const result = await response.json();
console.log(`Score: ${result.score}/${result.max_score}`);
console.log(`Feedback: ${result.feedback}`);
```

18.3.5. Batch Grade Submissions

Grade multiple submissions in a single request.

```
POST /assignments/grade/batch
```

Request Body:

```
{
  "assignment_id": "PSYCH101-ESSAY1",
  "grading_style": "concise",
  "submissions": [
    {
      "student_id": "student123",
      "submission": "Student 1's essay text..."
    },
    {
      "student_id": "student124",
      "submission": "Student 2's essay text..."
    }
  ]
}
```

Response:

```
{
  "batch_id": "batch_xyz789",
  "status": "processing",
  "total_submissions": 2,
  "estimated_completion": "2024-01-15T14:35:00Z",
  "results_url": "/assignments/grade/batch/batch_xyz789"
}
```

18.3.6. Get Grading Result

Retrieve results from a batch grading job.

```
GET /assignments/grade/batch/{batch_id}
```

Response:

```
{
  "batch_id": "batch_xyz789",
  "status": "completed",
  "total_submissions": 2,
  "completed": 2,
  "failed": 0,
  "results": [
    {
      "student_id": "student123",
      "grade_id": "grd_abc123",
      "score": 85,
      "feedback": "..."
    },
    {
      "student_id": "student124",
      "grade_id": "grd_abc124",
      "score": 92,
      "feedback": "..."
    }
  ]
}
```

18.4. Content Generation Endpoints

18.4.1. Generate Quiz Questions

Create quiz questions based on course content.

```
POST /content/quiz
```

Request Body:

Parameter	Type	Required	Description
`topic`	string	Yes	Subject matter for quiz
`difficulty`	string	No	Question difficulty: `easy`, `medium`, `hard` (default: `medium`)
`question_count`	number	No	Number of questions (1-50, default: 10)
`question_types`	array	No	Types to include: `multiple_choice`, `true_false`, `short_answer`
`course_id`	string	No	Course context for question generation

Response:

```
{
  "quiz_id": "quiz_123",
  "topic": "Introduction to Statistics",
  "difficulty": "medium",
  "questions": [
    {
      "question_id": "q1",
      "type": "multiple_choice",
      "text": "What is the primary purpose of standard error?",
      "options": [
        "Measure variability in population",
        "Estimate sampling distribution spread",
        "Calculate correlation",
        "Determine causation"
      ],
      "correct_answer": 1,
      "explanation": "Standard error estimates the standard deviation of the sampling distributio
      "points": 1
    }
  ],
  "total_points": 10
}
```


18.4.2. Generate Assignment Prompts

Create assignment prompts based on learning objectives.

```
POST /content/assignment
```

Request Body:


```
{
  "learning_objectives": [
    "Understand hypothesis testing",
    "Apply t-tests to real data"
  ],
  "assignment_type": "problem_set",
  "difficulty": "intermediate",
  "estimated_time": 60
}
```


Response:

```
{
  "assignment_id": "assign_456",
  "title": "Hypothesis Testing with Real-World Data",
  "description": "Apply your knowledge of hypothesis testing...",
  "prompts": [
    {
      "prompt_number": 1,
      "text": "Given the following dataset...",
      "rubric_criteria": ["Correct null hypothesis", "Appropriate test selection"]
    }
  ],
  "estimated_time_minutes": 60
}
```

18.5. Feedback Endpoints

18.5.1. Request Student Feedback

Trigger a feedback request to enrolled students.

```
POST /courses/{course_id}/feedback/request
```

Request Body:

Parameter	Type	Required	Description
`questions`	array	No	Custom questions (uses default if omitted)
`response_window_minutes`	number	No	How long students have to respond (default: 2)
`anonymous`	boolean	No	Collect anonymously (default: true)

Response:

```
{
  "request_id": "freq_789",
  "course_id": "course_abc",
  "sent_to_students": 45,
  "delivery_status": {
    "sent": 45,
    "delivered": 43,
    "failed": 2
  },
  "response_window_closes": "2024-01-15T14:35:00Z"
}
```

18.5.2. Get Feedback Results

Retrieve responses from a feedback request.

```
GET /courses/{course_id}/feedback/{request_id}
```

Response:

```
{
  "request_id": "freq_789",
  "course_id": "course_abc",
  "response_rate": 0.73,
  "total_responses": 33,
  "total_students": 45,
  "analysis": {
    "high_frequency_confusion": [
      {
        "topic": "Standard Error",
        "mentioned_by_count": 23,
        "percentage": 0.70,
        "action": "class_review_recommended"
      }
    ],
    "low_frequency_confusion": [
      {
        "topic": "Z-scores",
        "mentioned_by_count": 4,
        "percentage": 0.12,
        "action": "individual_resources_recommended"
      }
    ]
  },
  "sentiment": {
    "positive": 0.65,
    "neutral": 0.25,
    "negative": 0.10
  }
}
```

18.6. Analytics Endpoints

18.6.1. Get Course Analytics

Retrieve comprehensive analytics for a course.

```
GET /analytics/courses/{course_id}
```

Query Parameters:

Parameter	Type	Required	Description
`start_date`	string	No	ISO 8601 date (default: semester start)
`end_date`	string	No	ISO 8601 date (default: today)
`metrics`	string	No	Comma-separated list of metrics

Response:

```
{
  "course_id": "course_abc",
  "period": {
    "start": "2024-01-01",
    "end": "2024-05-01"
  },
  "enrollment": {
    "total": 45,
    "active": 43,
    "dropped": 2
  },
  "engagement": {
    "avg_feedback_response_rate": 0.73,
    "avg_quiz_completion_rate": 0.88,
    "app_active_users": 41
  },
  "performance": {
    "avg_quiz_score": 82.5,
    "avg_assignment_score": 85.2,
    "improvement_trend": 0.08
  },
  "common_confusion_topics": [
    "Standard Error",
    "Hypothesis Testing",
    "P-values"
  ]
}
```

18.6.2. Get Student Performance

Retrieve individual student analytics.

```
GET /analytics/students/{student_id}
```

Query Parameters:

Parameter	Type	Description
`course_id`	string	Filter by specific course
`start_date`	string	ISO 8601 date
`end_date`	string	ISO 8601 date

Response:

```
{
  "student_id": "student123",
  "courses": [
    {
      "course_id": "course_abc",
      "course_name": "Introduction to Statistics",
      "performance": {
        "current_grade": 87.5,
        "quiz_average": 85.0,
        "assignment_average": 90.0,
        "participation_score": 88.0
      },
      "engagement": {
        "feedback_response_rate": 0.95,
        "app_login_frequency": "daily",
        "last_active": "2024-01-15T14:30:00Z"
      },
      "struggling_topics": [
        "Hypothesis Testing"
      ],
      "strong_topics": [
        "Descriptive Statistics",
        "Probability"
      ]
    }
  ]
}
```

18.7. Course Management Endpoints

18.7.1. List Courses

Get all courses for the authenticated account.

```
GET /courses
```

Query Parameters:

Parameter	Type	Description
`status`	string	Filter by status: `active`, `archived`, `draft`
`semester`	string	Filter by semester
`limit`	number	Results per page (max 100)
`offset`	number	Pagination offset

Response:

```
{
  "courses": [
    {
      "course_id": "course_abc",
      "name": "Introduction to Statistics",
      "code": "STAT101",
      "semester": "Spring 2024",
      "status": "active",
      "enrollment_count": 45,
      "lms_integration": "canvas"
    }
  ],
  "total": 1,
  "limit": 100,
  "offset": 0
}
```

18.7.2. Create Course

Create a new course.

```
POST /courses
```

Request Body:

```
{
  "name": "Introduction to Statistics",
  "code": "STAT101",
  "semester": "Spring 2024",
  "description": "Introduction to statistical concepts...",
  "lms_integration": {
    "platform": "canvas",
```

```
    "course_id": "12345"
  }
}
```

Response:

```
{
  "course_id": "course_abc",
  "name": "Introduction to Statistics",
  "code": "STAT101",
  "enrollment_code": "ABC123",
  "enrollment_link": "https://kaitheai.com/enroll/ABC123",
  "created_at": "2024-01-15T14:30:00Z"
}
```

18.7.3. Update Course

Update course settings.

```
PATCH /courses/{course_id}
```

Request Body:

```
{
  "name": "Introduction to Statistics (Updated)",
  "settings": {
    "feedback_enabled": true,
    "quiz_enabled": true,
    "anonymous_feedback": true
  }
}
```

18.7.4. Delete Course

Archive or delete a course.

```
DELETE /courses/{course_id}
```

Query Parameters:

Parameter	Type	Description
`permanent`	boolean	Permanently delete (default: false, archives instead)

18.8. Webhook Endpoints

18.8.1. Register Webhook

Configure a webhook to receive real-time updates.

```
POST /webhooks
```

Request Body:

```
{
  "url": "https://your-domain.com/webhook",
  "events": [
    "grading.completed",
    "feedback.received",
    "quiz.submitted"
  ],
  "active": true,
  "secret": "your_webhook_secret"
}
```

Response:

```
{
  "webhook_id": "wh_123",
  "url": "https://your-domain.com/webhook",
  "events": ["grading.completed", "feedback.received", "quiz.submitted"],
  "active": true,
  "created_at": "2024-01-15T14:30:00Z"
}
```

18.8.2. List Webhooks

Get all configured webhooks.

```
GET /webhooks
```

18.8.3. Test Webhook

Send a test event to verify webhook configuration.

```
POST /webhooks/{webhook_id}/test
```

18.9. Rate Limits

All endpoints are subject to rate limiting based on your plan:

Plan	Requests/min	Requests/day	Burst
Free	10	1,000	20
Educator	60	10,000	100
Department	180	50,000	300
Institution	600	500,000	1000

Rate Limit Headers:

```
X-RateLimit-Limit: 60
X-RateLimit-Remaining: 45
X-RateLimit-Reset: 1642262400
```

18.10. Error Responses

All endpoints return errors in a consistent format:

```
{
  "error": {
    "code": "invalid_parameter",
    "message": "The 'assignment_id' field is required",
    "details": {
      "field": "assignment_id",
      "constraint": "required"
    },
    "request_id": "req_abc123",
    "timestamp": "2024-01-15T14:30:00Z"
  }
}
```

Common Error Codes:

Code	HTTP Status	Description
`invalid_parameter`	400	Request parameter invalid or missing
`unauthorized`	401	Invalid or missing API key
`forbidden`	403	Insufficient permissions
`not_found`	404	Resource not found
`rate_limit_exceeded`	429	Too many requests
`internal_error`	500	Server error
`service_unavailable`	503	Temporary service outage

18.11. SDKs and Client Libraries

Official SDKs available for easy integration:

- Python SDK - Full-featured Python client
- JavaScript/TypeScript SDK - Node.js and browser support
- Java SDK - Maven/Gradle integration
- Ruby SDK - Idiomatic Ruby patterns

18.12. Pagination

List endpoints support cursor-based pagination:

Request:

```
GET /courses?limit=20&offset=40
```

Response:

```
{
  "data": [...],
  "pagination": {
    "limit": 20,
    "offset": 40,
    "total": 156,
    "has_more": true
  }
}
```

18.13. Versioning

The API uses URL-based versioning. Current version: `v1`

Base URL includes version:

```
https://chi2api.com/v1/endpoint
```

Version sunset policy: - New versions announced 6 months in advance - Old versions supported for minimum 12 months - Deprecation notices sent via email and headers

18.14. Support

- **API Status**: status.chi2api.com
- **Developer Forum**: forum.chi2labs.com
- **Email**: developers@chi2labs.com
- **Authentication Issues**: See Authentication Guide
- **Integration Help**: See Integration Guide

This reference is updated regularly. Last updated: 2025-11-17

19. Authentication Guide

Secure authentication and API key management

19.1. Overview

Kai uses API key authentication for all API requests. This guide covers how to obtain, manage, and securely use API keys to authenticate your applications and integrations.

> **!** Security First
>
> API keys provide full access to your Kai account. Treat them like passwords and never expose them in client-side code or public repositories.

19.2. Authentication Methods

Kai supports multiple authentication methods depending on your use case:

Method	Use Case	Security Level
API Keys	Server-to-server, scripts, CI/CD	High
OAuth 2.0	Third-party integrations, user delegation	Very High
LTI 1.3	LMS integrations (Canvas, Blackboard, etc.)	Very High
Session Tokens	Web dashboard, temporary access	Medium

19.3. API Keys

19.3.1. Obtaining an API Key

1. **Log in to your Kai dashboard**: dashboard.kaitheai.com
2. **Navigate to Settings → API Keys**
3. **Click "Generate New API Key"**
4. **Configure key settings**:
   - **Name**: Descriptive name (e.g., "Production Server", "Development")
   - **Permissions**: Select required scopes

- **Expiration**: Set expiration date (or never)

5. **Copy the key**: It will only be shown once

> ⚠ Save Your Key
>
> API keys are only displayed once during creation. Store it securely immediately. If lost, you must generate a new key.

19.3.2. API Key Format

Kai API keys follow this format:

```
kai_<environment>_<32-character-random-string>
```

Examples:

```
kai_live_a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6  # Production
kai_test_x9y8z7w6v5u4t3s2r1q0p9o8n7m6l5k4  # Test/sandbox
```

Key Prefixes: - `kai_live_`: Production environment - `kai_test_`: Test/sandbox environment

19.3.3. Using API Keys

Include your API key in the `Authorization` header of every API request:

HTTP Header:

```
Authorization: Bearer kai_live_a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6
```

Example Request:

19.3.4. cURL

```
curl https://chi2api.com/v1/courses \
  -H "Authorization: Bearer kai_live_a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6"
```

19.3.5. Python

```python
import requests

API_KEY = "kai_live_a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6"

headers = {
    "Authorization": f"Bearer {API_KEY}"
}

response = requests.get(
    "https://chi2api.com/v1/courses",
    headers=headers
)
```

19.3.6. JavaScript

```javascript
const API_KEY = "kai_live_a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6";

const response = await fetch("https://chi2api.com/v1/courses", {
  headers: {
    "Authorization": `Bearer ${API_KEY}`
  }
});
```

19.3.7. Java

```java
import java.net.http.*;

String API_KEY = "kai_live_a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6";

HttpClient client = HttpClient.newHttpClient();
HttpRequest request = HttpRequest.newBuilder()
    .uri(URI.create("https://chi2api.com/v1/courses"))
    .header("Authorization", "Bearer " + API_KEY)
    .build();

HttpResponse<String> response = client.send(request,
    HttpResponse.BodyHandlers.ofString());
```

19.3.8. Ruby

```ruby
require 'net/http'
require 'uri'
```

```ruby
api_key = "kai_live_a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6"

uri = URI.parse("https://chi2api.com/v1/courses")
request = Net::HTTP::Get.new(uri)
request["Authorization"] = "Bearer #{api_key}"

response = Net::HTTP.start(uri.hostname, uri.port, use_ssl: true) do |http|
  http.request(request)
end
```

19.3.9. API Key Permissions (Scopes)

Control what each API key can access using scopes:

Scope	Description	Risk Level
`courses:read`	View course information	Low
`courses:write`	Create/update courses	Medium
`students:read`	View student data	Medium
`students:write`	Modify student records	High
`grading:read`	View grades	Medium
`grading:write`	Submit grades	High
`analytics:read`	Access analytics	Low
`feedback:read`	View feedback responses	Medium
`feedback:write`	Request feedback	Low
`admin:*`	Full administrative access	Very High

Best Practice: Principle of Least Privilege

Only grant the minimum scopes needed for each key:

```
Good: Integration key with only required scopes
  Scopes: courses:read, grading:write

Bad: Everything gets admin:*
  Risk: Compromised key = full account access
```

19.3.10. Managing API Keys

19.3.10.1. List All Keys

View all active API keys:

```
GET /api-keys
```

Response:

```
{
  "keys": [
    {
      "key_id": "key_abc123",
      "name": "Production Server",
      "prefix": "kai_live_a1b2...",
      "scopes": ["courses:read", "grading:write"],
      "created": "2024-01-01T00:00:00Z",
      "last_used": "2024-01-15T14:30:00Z",
      "expires": null
    }
  ]
}
```

19.3.10.2. Rotate API Keys

When to rotate: - Periodically (every 90 days recommended) - After suspected compromise - When team member leaves - After major system changes

Rotation process:

1. **Generate new key** with same permissions
2. **Update all integrations** to use new key
3. **Test thoroughly** in staging/development
4. **Deploy to production**
5. **Revoke old key** after confirming new key works

> **Zero-Downtime Rotation**
>
> Create the new key before revoking the old one. Both keys work simultaneously during transition period.

19.3.10.3. Revoke API Keys

Immediately disable a compromised key:

```
DELETE /api-keys/{key_id}
```

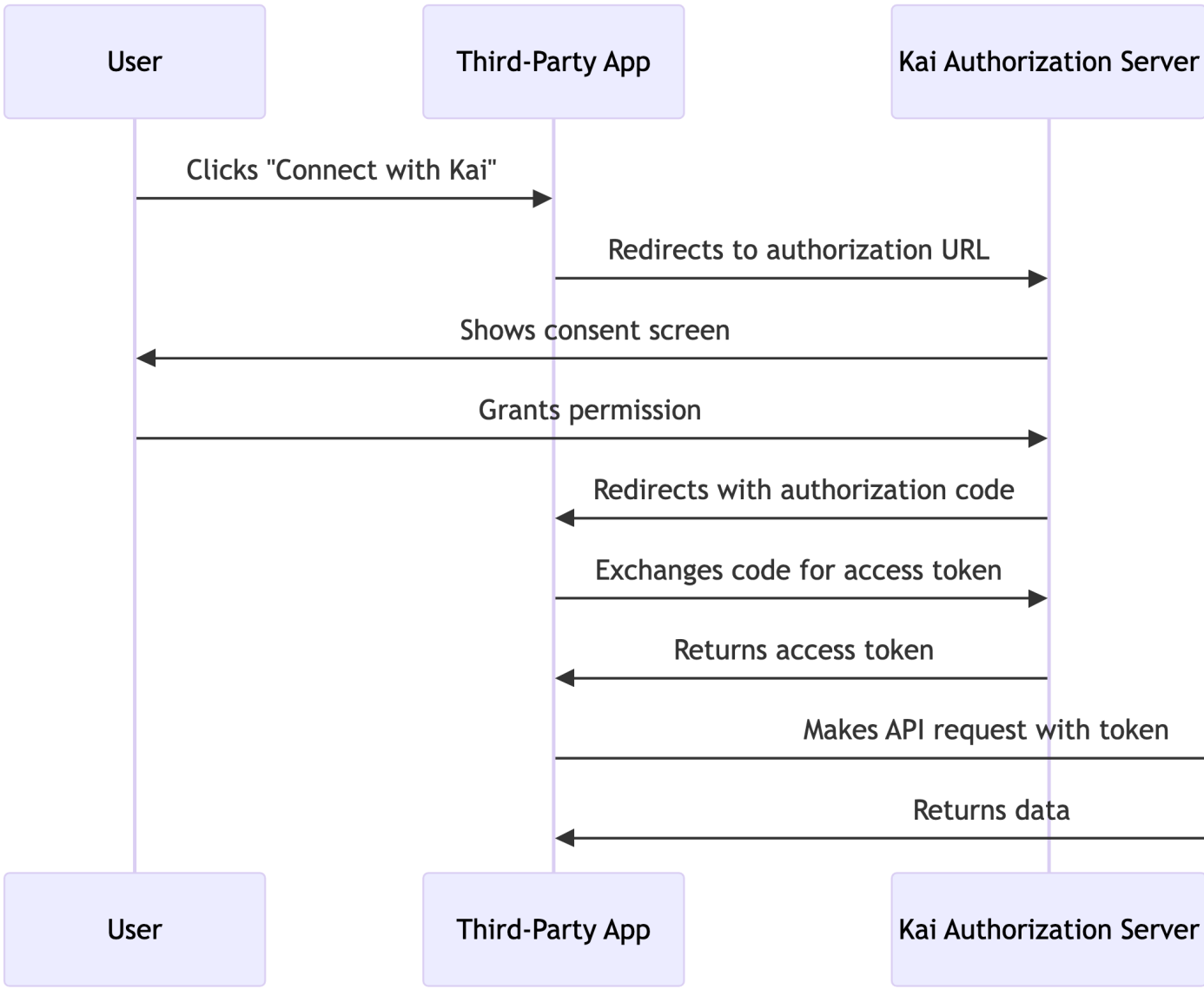
Dashboard: 1. Settings → API Keys 2. Find the key to revoke 3. Click "Revoke" 4. Confirm action

> **Immediate Effect**
>
> Revoking a key takes effect immediately. Any requests using that key will fail.

19.4. OAuth 2.0

For third-party integrations or user-delegated access, use OAuth 2.0.

19.4.1. OAuth Flow



19.4.2. Registering an OAuth Application

1. **Dashboard → Settings → OAuth Applications**
2. **Click "New Application"**
3. **Configure settings**:
   - **Application Name**: Your app's name

- **Redirect URIs**: Where to send users after authorization
- **Scopes**: Permissions your app needs

4. **Save** to receive Client ID and Client Secret

19.4.3. Authorization Request

Redirect users to:

```
GET https://chi2api.com/oauth/authorize?
  response_type=code&
  client_id=YOUR_CLIENT_ID&
  redirect_uri=https://your-app.com/callback&
  scope=courses:read%20grading:write&
  state=random_string
```

Parameters:

Parameter	Required	Description
`response_type`	Yes	Always `code` for authorization code flow
`client_id`	Yes	Your application's client ID
`redirect_uri`	Yes	Must match registered redirect URI
`scope`	Yes	Space-separated list of scopes
`state`	Recommended	Random string to prevent CSRF

19.4.4. Token Exchange

Exchange authorization code for access token:

```
POST https://chi2api.com/oauth/token
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&
code=AUTHORIZATION_CODE&
client_id=YOUR_CLIENT_ID&
client_secret=YOUR_CLIENT_SECRET&
redirect_uri=https://your-app.com/callback
```

Response:

```
{
  "access_token": "kai_oauth_abc123...",
  "token_type": "Bearer",
  "expires_in": 3600,
  "refresh_token": "kai_refresh_xyz789...",
  "scope": "courses:read grading:write"
}
```

19.4.5. Using Access Tokens

Use access tokens the same way as API keys:

```
Authorization: Bearer kai_oauth_abc123...
```

19.4.6. Refreshing Tokens

Access tokens expire after 1 hour. Use refresh token to get new access token:

```
POST https://chi2api.com/oauth/token
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token&
refresh_token=kai_refresh_xyz789...&
client_id=YOUR_CLIENT_ID&
client_secret=YOUR_CLIENT_SECRET
```

19.5. LTI 1.3 Authentication

For LMS integrations, Kai supports LTI 1.3 (Learning Tools Interoperability).

19.5.1. LTI Configuration

Required Information:

Field	Value
OpenID Connect Login URL	`https://chi2api.com/lti/login`
Target Link URI	`https://chi2api.com/lti/launch`
Public JWK URL	`https://chi2api.com/.well-known/jwks.json`
Redirect URIs	`https://chi2api.com/lti/callback`

Required Claims:

- `sub` - User identifier
- `https://purl.imsglobal.org/spec/lti/claim/context` - Course context
- `https://purl.imsglobal.org/spec/lti/claim/roles` - User roles

19.5.2. Canvas LTI Setup

See Installation Guide - Canvas Integration for detailed instructions.

19.5.3. Blackboard LTI Setup

See Installation Guide - Blackboard Integration for detailed instructions.

19.6. Security Best Practices

19.6.1. Storing API Keys

19.7. Never Do This

```
#  Hardcoded in source code
API_KEY = "kai_live_a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6"

#  Committed to Git
# config.py
API_KEY = "kai_live_..."

#  In client-side JavaScript
const API_KEY = "kai_live_...";
```

Best Practices

```
#  Environment variables
import os
API_KEY = os.environ.get("KAI_API_KEY")

#  Secret management service
from secret_manager import get_secret
API_KEY = get_secret("kai-api-key")

#  Configuration files (gitignored)
import configparser
config = configparser.ConfigParser()
config.read("config.ini")  # .gitignore includes config.ini
API_KEY = config["kai"]["api_key"]
```

19.7.1. Environment Variables

Development:

```
# .env file (add to .gitignore)
KAI_API_KEY=kai_test_x9y8z7w6v5u4t3s2r1q0p9o8n7m6l5k4
```

Production:

```
# Set in deployment environment
export KAI_API_KEY=kai_live_a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6
```

Docker:

```
# docker-compose.yml
services:
  app:
    environment:
      - KAI_API_KEY=${KAI_API_KEY}
```

19.7.2. Secret Management Services

AWS Secrets Manager:

```
import boto3
import json

client = boto3.client('secretsmanager')
response = client.get_secret_value(SecretId='kai-api-key')
API_KEY = json.loads(response['SecretString'])['api_key']
```

HashiCorp Vault:

```
import hvac

client = hvac.Client(url='https://vault.example.com')
client.auth.approle.login(role_id='...', secret_id='...')
API_KEY = client.secrets.kv.v2.read_secret_version(
    path='kai-api-key'
)['data']['data']['key']
```

Google Secret Manager:

```
from google.cloud import secretmanager

client = secretmanager.SecretManagerServiceClient()
name = "projects/PROJECT_ID/secrets/kai-api-key/versions/latest"
response = client.access_secret_version(request={"name": name})
API_KEY = response.payload.data.decode("UTF-8")
```

19.7.3. Network Security

Use HTTPS Only:

```
#  Always HTTPS
response = requests.get("https://chi2api.com/v1/courses")

#  Never HTTP
response = requests.get("http://chi2api.com/v1/courses")
```

Verify SSL Certificates:

```
#  Verify certificates (default)
response = requests.get(url, verify=True)

#  Don't disable verification
response = requests.get(url, verify=False)  # Dangerous!
```

19.7.4. IP Allowlisting

Restrict API key usage to specific IP addresses:

Dashboard Configuration: 1. Settings → API Keys → [Your Key] → IP Restrictions 2. Add allowed IP addresses or CIDR ranges 3. Save changes

Example:

```
Allowed IPs:
- 203.0.113.0/24 (Office network)
- 198.51.100.42 (Production server)
```

19.7.5. Monitoring and Alerts

Enable alerts for: - Unusual API usage patterns - Failed authentication attempts - API key used from new IP address - Rate limit approaching

Configure in: Dashboard → Settings → Security → Alerts

19.8. Testing Authentication

19.8.1. Verify API Key

Test your API key is working:

```
curl https://chi2api.com/v1/auth/verify \
  -H "Authorization: Bearer YOUR_API_KEY"
```

Success Response:

```
{
  "valid": true,
  "key_id": "key_abc123",
  "scopes": ["courses:read", "grading:write"],
  "expires": null,
  "rate_limit": {
    "limit": 60,
    "remaining": 58,
    "reset": 1642262400
  }
}
```

Failure Response:

```
{
  "error": {
    "code": "invalid_api_key",
    "message": "The provided API key is invalid or has been revoked"
  }
}
```

19.8.2. Test OAuth Flow

Use Kai's OAuth testing tool:

1. Dashboard → Settings → OAuth Applications → [Your App] → Test
2. Click "Start OAuth Flow"
3. Complete authorization
4. Verify token received

19.9. Troubleshooting

19.9.1. Common Authentication Errors

19.9.1.1. "Invalid API Key"

Possible causes: - Key was revoked - Key contains typos or extra spaces - Using test key in production (or vice versa)

Solution: - Verify key in dashboard - Regenerate if needed - Check environment variables

19.9.1.2. "Insufficient Permissions"

Possible causes: - API key lacks required scope - Trying to access resource outside permissions

Solution: - Check required scopes in Endpoints Reference - Generate new key with appropriate scopes

19.9.1.3. "Rate Limit Exceeded"

Possible causes: - Too many requests in short period - Multiple services sharing one key

Solution: - Implement exponential backoff - Create separate keys for different services - Upgrade plan for higher limits

19.9.2. Getting Help

- **Security Issues**: security@chi2labs.com
- **Authentication Support**: support@chi2labs.com
- **Documentation**: API Reference, Endpoints

For integration examples and workflows, see the Integration Guide.

20. Integration Guide

Integrate Kai into your educational technology stack

20.1. Overview

This guide provides comprehensive instructions for integrating Kai the AI with various platforms, services, and custom applications. Whether you're connecting to an LMS, building custom workflows, or creating third-party applications, this guide has you covered.

20.2. Learning Management Systems

20.2.1. Canvas Integration

Canvas is one of the most popular LMS platforms. Kai integrates seamlessly via LTI 1.3 or API.

20.2.1.1. LTI 1.3 Integration (Recommended)

Advantages: - Deep integration with Canvas UI - Single sign-on (SSO) - Automatic roster sync - Grade passback

Setup Steps:

1. **In Canvas (as Admin)**:

   ```
   Admin → Developer Keys → + Developer Key → + LTI Key
   ```

2. **Configure LTI Key**:

   ```
   Key Name: Kai the AI
   Redirect URIs: https://kaitheai.com/lti/callback
   JWK Method: Public JWK URL
   JWK URL: https://kaitheai.com/.well-known/jwks.json
   Target Link URI: https://kaitheai.com/lti/launch
   OpenID Connect Initiation URL: https://kaitheai.com/lti/login
   ```

3. **Enable Scopes**:

- Can create and view assignment data
- Can view assignment data
- Can view course content
- Can view user data
- Can submit grades

4. **Copy Client ID**: Generated after saving
5. **In Kai Dashboard**:

   ```
   Settings → Integrations → Canvas → Add LTI Integration
   Paste Client ID
   Save
   ```

6. **Add to Course**:

   ```
   Canvas Course → Settings → Apps → View App Configurations
   → + App → Kai the AI → Configure
   ```

20.2.1.2. Canvas API Integration

Use when: You need programmatic access or LTI isn't available

Setup:

1. **Generate Canvas API Token**:

   ```
   Canvas → Account → Settings → + New Access Token
   Purpose: Kai Integration
   Expires: [Choose appropriate expiration]
   ```

2. **Configure in Kai**:

   ```
   Kai Dashboard → Settings → Integrations → Canvas
   Canvas URL: https://your-institution.instructure.com
   API Token: [paste token]
   Test Connection → Save
   ```

Example API Usage:

20.2.2. Sync Courses

```python
import requests

# Fetch Canvas courses
canvas_response = requests.get(
    "https://your-institution.instructure.com/api/v1/courses",
    headers={"Authorization": f"Bearer {CANVAS_TOKEN}"}
)
```

```
# Create courses in Kai
for course in canvas_response.json():
    kai_response = requests.post(
        "https://chi2api.com/v1/courses",
        headers={"Authorization": f"Bearer {KAI_API_KEY}"},
        json={
            "name": course["name"],
            "code": course["course_code"],
            "lms_integration": {
                "platform": "canvas",
                "course_id": str(course["id"])
            }
        }
    )
```

20.2.3. Sync Grades

```
# Get grades from Kai
kai_grades = requests.get(
    f"https://chi2api.com/v1/courses/{course_id}/grades",
    headers={"Authorization": f"Bearer {KAI_API_KEY}"}
).json()

# Submit to Canvas
for grade in kai_grades["grades"]:
    canvas_response = requests.put(
        f"https://your-institution.instructure.com/api/v1/courses/{canvas_course_id}"
        f"/assignments/{assignment_id}/submissions/{student_id}",
        headers={"Authorization": f"Bearer {CANVAS_TOKEN}"},
        json={
            "submission": {
                "posted_grade": grade["score"]
            }
        }
    )
```

20.2.4. Blackboard Integration

Blackboard integration uses the Building Block architecture.

Installation:

1. **Download Building Block**:

   ```
   Kai Dashboard → Settings → Integrations → Blackboard
   → Download Building Block
   ```

2. **Install in Blackboard**:

   ```
   System Admin → Building Blocks → Installed Tools
   → Upload Building Blocks → Choose File
   → Select Kai .war file → Submit
   ```

3. **Configure**:

   ```
   Find "Kai the AI" in list → Settings
   API Endpoint: https://chi2api.com/v1
   Institution Code: [from Kai dashboard]
   API Key: [from Kai dashboard]
   → Submit
   ```

4. **Make Available**:

   ```
   Set Availability to "Available" → Submit
   ```

Adding to Courses:

```
Course → Content → Build Content → Kai the AI
Select feature: Feedback, Quiz, or Grading
Configure settings → Submit
```

20.2.5. Moodle Integration

Moodle integration uses a local plugin.

Installation:

1. **Download Plugin**:

   ```
   Kai Dashboard → Settings → Integrations → Moodle
   → Download Plugin (.zip)
   ```

2. **Install via UI**:

   ```
   Site Administration → Plugins → Install plugins
   → Upload plugin → Choose file → Upload
   → Install plugin from ZIP file
   ```

3. **Or Manual Installation**:

   ```
   cd /path/to/moodle
   unzip kai-moodle-plugin.zip -d local/
   ```

4. **Update Database**:

   ```
   Site Administration → Notifications
   → Upgrade Moodle database now
   ```

5. **Configure**:

```
Site Administration → Plugins → Local plugins → Kai
API Key: [from Kai dashboard]
Enable features: [select desired features]
→ Save changes
```

20.2.6. Google Classroom Integration

OAuth-based integration for seamless Google Workspace integration.

Setup:

1. **In Kai Dashboard**:

   ```
   Settings → Integrations → Google Classroom → Connect
   ```

2. **Authenticate**:

   ```
   Sign in with Google Workspace admin account
   ```

3. **Grant Permissions**:

   - View courses and rosters
   - Create and grade assignments
   - Post announcements

4. **Select Classes**:

   ```
   Choose which Google Classroom classes to sync
   → Save
   ```

Features:

- Automatic roster sync
- Assignment creation from Kai
- Grade synchronization
- Feedback via Google Classroom stream

20.3. Custom Integrations

20.3.1. Webhooks

Receive real-time notifications when events occur in Kai.

20.3.1.1. Setting Up Webhooks

1. Create Webhook Endpoint

Your endpoint must: - Accept POST requests - Return 200 OK within 5 seconds - Verify webhook signature

Example Endpoint (Python/Flask):

```
from flask import Flask, request, jsonify
import hmac
import hashlib

app = Flask(__name__)
WEBHOOK_SECRET = "your_webhook_secret"

@app.route('/kai-webhook', methods=['POST'])
def kai_webhook():
    # Verify signature
    signature = request.headers.get('X-Kai-Signature')
    payload = request.data

    expected_signature = hmac.new(
        WEBHOOK_SECRET.encode(),
        payload,
        hashlib.sha256
    ).hexdigest()

    if not hmac.compare_digest(signature, expected_signature):
        return jsonify({"error": "Invalid signature"}), 401

    # Process event
    event = request.json
    event_type = event['type']

    if event_type == 'grading.completed':
        handle_grading_completed(event['data'])
    elif event_type == 'feedback.received':
        handle_feedback_received(event['data'])

    return jsonify({"status": "received"}), 200

def handle_grading_completed(data):
    grade_id = data['grade_id']
    score = data['score']
    # Your custom logic here
    print(f"Grade {grade_id} completed with score {score}")

def handle_feedback_received(data):
```

```
    request_id = data['request_id']
    response_count = data['response_count']
    # Your custom logic here
    print(f"Feedback request {request_id} received {response_count} responses")
```

2. Register Webhook

```
curl -X POST https://chi2api.com/v1/webhooks \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "url": "https://your-domain.com/kai-webhook",
    "events": [
      "grading.completed",
      "feedback.received",
      "quiz.submitted"
    ],
    "secret": "your_webhook_secret"
  }'
```

20.3.1.2. Available Events

Event	Triggered When	Payload Includes
`grading.completed`	Assignment graded	`grade_id`, `score`, `feedback`
`feedback.received`	Student submits feedback	`request_id`, `student_id`, `response`
`feedback.request_closed`	Feedback window closes	`request_id`, `response_count`, `analysis`
`quiz.submitted`	Student submits quiz	`quiz_id`, `student_id`, `score`
`course.enrollment_added`	Student enrolls	`course_id`, `student_id`
`course.enrollment_removed`	Student unenrolls	`course_id`, `student_id`

20.3.1.3. Webhook Payload Structure

```
{
  "id": "evt_abc123",
  "type": "grading.completed",
  "created": "2024-01-15T14:30:00Z",
  "data": {
    "grade_id": "grd_xyz789",
    "assignment_id": "assign_123",
    "student_id": "student_456",
    "score": 87,
    "feedback": "Excellent work...",
    "confidence": 0.92
```

```
  },
  "account_id": "acc_abc",
  "api_version": "v1"
}
```

20.3.2. REST API Integration

Build custom applications using Kai's REST API.

20.3.2.1. Complete Integration Example

Scenario: Custom dashboard showing real-time class engagement

20.3.3. Python

```
import requests
from datetime import datetime, timedelta

class KaiIntegration:
    def __init__(self, api_key):
        self.api_key = api_key
        self.base_url = "https://chi2api.com/v1"
        self.headers = {
            "Authorization": f"Bearer {api_key}",
            "Content-Type": "application/json"
        }

    def get_courses(self):
        """Fetch all active courses"""
        response = requests.get(
            f"{self.base_url}/courses",
            headers=self.headers,
            params={"status": "active"}
        )
        response.raise_for_status()
        return response.json()["courses"]

    def request_feedback(self, course_id):
        """Request feedback from students"""
        response = requests.post(
            f"{self.base_url}/courses/{course_id}/feedback/request",
            headers=self.headers,
            json={
                "response_window_minutes": 2,
                "anonymous": True
```

```
            }
        )
        response.raise_for_status()
        return response.json()

    def get_feedback_results(self, course_id, request_id):
        """Get feedback analysis"""
        response = requests.get(
            f"{self.base_url}/courses/{course_id}/feedback/{request_id}",
            headers=self.headers
        )
        response.raise_for_status()
        return response.json()

    def get_course_analytics(self, course_id, days=7):
        """Get course analytics for past N days"""
        end_date = datetime.now()
        start_date = end_date - timedelta(days=days)

        response = requests.get(
            f"{self.base_url}/analytics/courses/{course_id}",
            headers=self.headers,
            params={
                "start_date": start_date.isoformat(),
                "end_date": end_date.isoformat()
            }
        )
        response.raise_for_status()
        return response.json()

# Usage
kai = KaiIntegration("kai_live_...")

# Get all courses
courses = kai.get_courses()
for course in courses:
    print(f"Course: {course['name']}")

    # Get analytics
    analytics = kai.get_course_analytics(course['course_id'])
    print(f"  Engagement: {analytics['engagement']['avg_feedback_response_rate']:.1%}")
    print(f"  Avg Quiz Score: {analytics['performance']['avg_quiz_score']:.1f}")
```

20.3.4. Node.js

```
const axios = require('axios');

class KaiIntegration {
  constructor(apiKey) {
    this.apiKey = apiKey;
    this.baseURL = 'https://chi2api.com/v1';
    this.client = axios.create({
      baseURL: this.baseURL,
      headers: {
        'Authorization': `Bearer ${apiKey}`,
        'Content-Type': 'application/json'
      }
    });
  }

  async getCourses() {
    const response = await this.client.get('/courses', {
      params: { status: 'active' }
    });
    return response.data.courses;
  }

  async requestFeedback(courseId) {
    const response = await this.client.post(
      `/courses/${courseId}/feedback/request`,
      {
        response_window_minutes: 2,
        anonymous: true
      }
    );
    return response.data;
  }

  async getFeedbackResults(courseId, requestId) {
    const response = await this.client.get(
      `/courses/${courseId}/feedback/${requestId}`
    );
    return response.data;
  }

  async getCourseAnalytics(courseId, days = 7) {
    const endDate = new Date();
    const startDate = new Date();
    startDate.setDate(startDate.getDate() - days);
```

```
    const response = await this.client.get(
      `/analytics/courses/${courseId}`,
      {
        params: {
          start_date: startDate.toISOString(),
          end_date: endDate.toISOString()
        }
      }
    );
    return response.data;
  }
}

// Usage
const kai = new KaiIntegration('kai_live_...');

(async () => {
  const courses = await kai.getCourses();

  for (const course of courses) {
    console.log(`Course: ${course.name}`);

    const analytics = await kai.getCourseAnalytics(course.course_id);
    console.log(`  Engagement: ${(analytics.engagement.avg_feedback_response_rate * 100).toFixed(
    console.log(`  Avg Quiz Score: ${analytics.performance.avg_quiz_score.toFixed(1)}`);
  }
})();
```

20.3.5. SDK Usage

Official SDKs simplify integration:

20.3.6. Python SDK

```
from kai import KaiClient

# Initialize client
kai = KaiClient(api_key="kai_live_...")

# List courses
courses = kai.courses.list(status="active")

# Request feedback
feedback_request = kai.feedback.request(
    course_id="course_abc",
```

```
    response_window_minutes=2
)

# Get results
results = kai.feedback.get_results(
    course_id="course_abc",
    request_id=feedback_request.request_id
)

# Print analysis
for topic in results.analysis.high_frequency_confusion:
    print(f"Review needed: {topic.topic} ({topic.percentage:.0%} confused)")
```

20.3.7. JavaScript SDK

```
const { KaiClient } = require('@kai/sdk');

// Initialize client
const kai = new KaiClient({ apiKey: 'kai_live_...' });

// List courses
const courses = await kai.courses.list({ status: 'active' });

// Request feedback
const feedbackRequest = await kai.feedback.request({
  courseId: 'course_abc',
  responseWindowMinutes: 2
});

// Get results
const results = await kai.feedback.getResults({
  courseId: 'course_abc',
  requestId: feedbackRequest.requestId
});

// Print analysis
results.analysis.highFrequencyConfusion.forEach(topic => {
  console.log(`Review needed: ${topic.topic} (${(topic.percentage * 100).toFixed(0)}% confused)`)
});
```

See SDK documentation for complete reference: - Python SDK - JavaScript/TypeScript SDK - Java SDK - Ruby SDK

20.4. Third-Party Integrations

20.4.1. Slack Integration

Get Kai notifications in Slack channels.

Setup:

1. **In Kai Dashboard**:

   ```
   Settings → Integrations → Slack → Add to Slack
   ```

2. **Authorize**:

   ```
   Select Slack workspace → Authorize
   ```

3. **Configure Channels**:

   ```
   Course alerts → #course-updates
   Grading complete → #grading-notifications
   System alerts → #kai-system
   ```

Example Slack Message:

```
Kai the AI
Feedback Request Complete - PSYCH 101

  Response Rate: 32/45 students (71%)

  High-frequency confusion:
  • Standard Error (70% confused)
  → Recommend: In-class review

  Low-frequency confusion:
  • Z-scores (12% confused)
  → Recommend: Individual resources

View Details → [Link to dashboard]
```

20.4.2. Microsoft Teams Integration

Similar to Slack, receive notifications in Teams channels.

Setup: Settings → Integrations → Microsoft Teams → Connect

20.4.3. Zapier Integration

Create custom workflows connecting Kai to 5,000+ apps.

Example Zaps:

1. **New Feedback → Google Sheets**:

   ```
   Trigger: Feedback received (Kai)
   Action: Add row to spreadsheet (Google Sheets)
   ```

2. **Low Quiz Score → Email**:

   ```
   Trigger: Quiz submitted (Kai)
   Filter: Score < 70
   Action: Send email (Gmail)
   ```

3. **Grading Complete → Notion**:

   ```
   Trigger: Grading completed (Kai)
   Action: Create database item (Notion)
   ```

20.5. Data Export and Reporting

20.5.1. Scheduled Exports

Automate data exports for institutional reporting.

Configuration:

```
Dashboard → Settings → Data Export
Schedule: Weekly (Monday 6 AM)
Format: CSV
Include: Analytics, grades, engagement
Destination: SFTP, S3, or email
```

20.5.2. API-Based Export

```python
# Export all course data
response = requests.get(
    f"https://chi2api.com/v1/courses/{course_id}/export",
    headers={"Authorization": f"Bearer {API_KEY}"},
    params={
        "format": "json",
        "include": "grades,feedback,analytics"
    }
)
```

```
# Save to file
with open("course_data.json", "w") as f:
    json.dump(response.json(), f, indent=2)
```

20.5.3. PowerBI/Tableau Integration

Connect BI tools directly to Kai's API.

PowerBI Setup:

1. **Get Data** → **Web**
2. **Enter URL**: `https://chi2api.com/v1/analytics/courses/COURSE_ID`
3. **Add Header**: `Authorization: Bearer YOUR_API_KEY`
4. **Transform data** as needed
5. **Create visualizations**

20.6. Security Considerations

20.6.1. API Key Management

- Use separate keys for each integration
- Rotate keys every 90 days
- Revoke immediately if compromised
- Store in secret management system

20.6.2. Network Security

- Always use HTTPS
- Verify SSL certificates
- Use IP allowlisting when possible
- Implement rate limiting on your side

20.6.3. Data Privacy

- Comply with FERPA, GDPR, and local regulations
- Encrypt data in transit and at rest
- Implement proper access controls
- Regular security audits

20.7. Troubleshooting

20.7.1. Common Integration Issues

LMS Sync Not Working: - Verify API credentials - Check firewall settings - Review integration logs - Test with minimal permissions first

Webhook Not Receiving Events: - Ensure endpoint is publicly accessible - Verify signature validation - Check webhook URL is HTTPS - Review webhook logs in dashboard

Rate Limits Exceeded: - Implement exponential backoff - Cache data when possible - Use batch endpoints - Consider upgrading plan

20.8. Support

- **Integration Support**: integrations@chi2labs.com
- **API Documentation**: API Reference
- **Authentication**: Authentication Guide
- **Endpoints**: Endpoints Reference

Have a custom integration need? Contact our integrations team for personalized support.

21. Customization Guide

Customize Kai to match your teaching style and institutional needs

21.1. Overview

Kai the AI is designed to be highly customizable, allowing you to tailor features, workflows, and integrations to your specific teaching style and institutional requirements. This guide covers all available customization options.

21.2. Grading Customization

21.2.1. Custom Rubrics

Create custom grading rubrics that match your assessment criteria.

21.2.1.1. Creating a Rubric

Via Dashboard:

1. **Navigate**: Dashboard → Settings → Grading → Rubrics
2. **Click**: "Create New Rubric"
3. **Configure**:
    - **Name**: "Essay Rubric - PSYCH 101"
    - **Total Points**: 100
    - **Add Criteria**

Example Rubric Structure:

Criterion	Points	Description	Levels
Thesis Statement	20	Clear, arguable thesis	Excellent (20), Good (15), Fair (10), Poor (5)
Evidence	30	Supporting evidence and analysis	Excellent (30), Good (22), Fair (15), Poor (8)

Criterion	Points	Description	Levels
Organization	20	Logical flow and structure	Excellent (20), Good (15), Fair (10), Poor (5)
Writing Quality	20	Grammar, style, clarity	Excellent (20), Good (15), Fair (10), Poor (5)
Citations	10	Proper citation format	Excellent (10), Good (7), Fair (5), Poor (2)

21.2.1.2. Via API

```
rubric = {
    "name": "Essay Rubric - PSYCH 101",
    "total_points": 100,
    "criteria": [
        {
            "name": "Thesis Statement",
            "points": 20,
            "description": "Clear, arguable thesis",
            "levels": [
                {"name": "Excellent", "points": 20, "description": "Clear, specific, arguable the
                {"name": "Good", "points": 15, "description": "Clear thesis with minor issues"},
                {"name": "Fair", "points": 10, "description": "Vague or unclear thesis"},
                {"name": "Poor", "points": 5, "description": "Missing or fundamentally flawed the
            ]
        },
        # ... more criteria
    ]
}

response = requests.post(
    "https://chi2api.com/v1/rubrics",
    headers={"Authorization": f"Bearer {API_KEY}"},
    json=rubric
)
```

21.2.2. Feedback Styles

Customize the tone and detail level of AI-generated feedback.

Available Styles:

Style	Description	Best For	Example
Detailed	Comprehensive feedback with specific suggestions	Major assignments, essays	"Your thesis in paragraph 2 effectively argues… However, consider strengthening…"
Concise	Brief, actionable feedback	Quizzes, frequent assignments	"Strong analysis. Add more examples in body paragraphs."
Points Only	Score with minimal commentary	Quick assessments, self-graded work	"Score: 85/100"
Encouraging	Positive tone, growth-focused	Struggling students, formative assessment	"Great progress! You're developing strong analysis skills…"
Professional	Formal, academic tone	Graduate courses, professional programs	"The analysis demonstrates competent understanding of…"

Configuration:

```
# Set default for course
kai.courses.update(
    course_id="course_abc",
    grading_settings={
        "feedback_style": "encouraging",
        "show_rubric_scores": True,
        "include_suggestions": True
    }
)

# Override for specific assignment
kai.assignments.grade(
    assignment_id="assign_123",
    student_submission="...",
    grading_style="detailed"  # Override course default
)
```

21.2.3. Auto-Grading Thresholds

Configure when AI can auto-grade vs. requiring human review.

Confidence-Based Grading:

```
grading_policy = {
    "auto_grade_threshold": 0.85,  # Auto-grade if confidence >= 85%
    "review_required": {
        "low_confidence": True,       # Always review if confidence < threshold
        "high_stakes": True,           # Review if assignment weight > 20%
        "boundary_scores": True,       # Review if score near grade boundary
        "flagged_content": True        # Review if content flagged
    },
    "grade_boundaries": [90, 80, 70, 60]  # A, B, C, D thresholds
}

kai.courses.update(
    course_id="course_abc",
    grading_settings=grading_policy
)
```

Example Scenarios:

Scenario	Confidence	Auto-Grade?	Reason
Clear excellent work	0.95	Yes	Above threshold
Borderline B/C (79%)	0.88	No	Near boundary
Low-stakes quiz	0.80	Depends	Check assignment weight
Plagiarism detected	0.92	No	Flagged content

21.3. Feedback Workflow Customization

21.3.1. Custom Feedback Questions

Replace default questions with your own.

Default Questions: 1. "What concepts from today's class were clear to you?" 2. "What topics would you like me to review or explain differently?"

Custom Questions Example:

```
feedback_config = {
    "questions": [
        {
            "id": "clarity",
            "text": "On a scale of 1-5, how clear was today's lecture?",
            "type": "scale",
            "min": 1,
            "max": 5
        },
        {
            "id": "confused_topics",
```

```
            "text": "Which specific topics need more explanation?",
            "type": "multiple_choice",
            "options": [
                "Standard Error",
                "Hypothesis Testing",
                "P-values",
                "Effect Size",
                "None - all clear"
            ],
            "allow_multiple": True
        },
        {
            "id": "pace",
            "text": "Was the pace of today's class...",
            "type": "single_choice",
            "options": ["Too fast", "Just right", "Too slow"]
        },
        {
            "id": "additional",
            "text": "Any other feedback?",
            "type": "free_text",
            "optional": True
        }
    ]
}

kai.courses.update(
    course_id="course_abc",
    feedback_settings=feedback_config
)
```

21.3.2. Feedback Triggers

Customize when feedback requests are suggested or automatic.

Trigger Configuration:

```
triggers = {
    "time_based": {
        "enabled": True,
        "interval_minutes": 30,
        "max_per_class": 3
    },
    "content_based": {
        "enabled": True,
        "trigger_on": [
            "topic_transition",      # New major topic
```

```
            "complex_concept",        # AI detects difficult concept
            "question_spike"          # Many student questions
        ]
    },
    "engagement_based": {
        "enabled": True,
        "trigger_on": [
            "attention_drop",         # Engagement metrics decline
            "confusion_indicators"    # Body language, facial expressions
        ],
        "threshold": 0.3              # 30% engagement drop
    },
    "manual_override": {
        "allow_disable_suggestions": True,
        "require_confirmation": False
    }
}

kai.courses.update(
    course_id="course_abc",
    feedback_triggers=triggers
)
```

21.3.3. Response Window Customization

Adjust how long students have to respond.

```
response_settings = {
    "default_window_minutes": 2,
    "allow_late_responses": True,
    "late_window_minutes": 5,
    "send_reminder": {
        "enabled": True,
        "after_seconds": 60
    }
}
```

21.4. Quiz Customization

21.4.1. Question Generation Rules

Control how Kai generates quiz questions.

```
quiz_settings = {
    "difficulty_distribution": {
        "easy": 0.3,      # 30% easy questions
        "medium": 0.5,    # 50% medium questions
        "hard": 0.2       # 20% hard questions
    },
    "question_types": {
        "multiple_choice": 0.5,
        "true_false": 0.2,
        "short_answer": 0.3
    },
    "content_sources": [
        "lecture_notes",
        "textbook_chapters",
        "assigned_readings",
        "previous_exams"
    ],
    "avoid_repetition": {
        "enabled": True,
        "lookback_quizzes": 5  # Don't repeat from last 5 quizzes
    },
    "adaptive_difficulty": {
        "enabled": True,
        "adjust_based_on": "student_performance"
    }
}

kai.courses.update(
    course_id="course_abc",
    quiz_settings=quiz_settings
)
```

21.4.2. Question Templates

Create custom question templates for consistency.

Template Example:

```
template = {
    "name": "Concept Application Template",
    "structure": {
        "stem": "Given the following scenario: {scenario}",
        "question": "Which statistical test would be most appropriate?",
        "options": [
            "{correct_test}",
            "{plausible_alternative_1}",
            "{plausible_alternative_2}",
```

```
            "{clearly_wrong_option}"
        ],
        "explanation": "The correct answer is {correct_test} because {reasoning}. "
                       "{plausible_alternative_1} would not work because {why_not}."
    },
    "variables": {
        "scenario": {
            "type": "generated",
            "context": "real-world statistical scenario"
        },
        "correct_test": {
            "type": "from_content",
            "category": "statistical_tests"
        }
    }
}

kai.quiz_templates.create(template)
```

21.4.3. Time Limits and Attempts

```
quiz_policies = {
    "time_limit": {
        "enabled": True,
        "minutes": 10,
        "auto_submit": True
    },
    "attempts": {
        "allowed": 2,
        "use_highest": True,  # Or "average", "latest"
        "time_between_attempts_hours": 24
    },
    "question_randomization": {
        "randomize_order": True,
        "randomize_options": True,
        "pool_size": 20,  # Select 10 from pool of 20
        "questions_shown": 10
    }
}
```

21.5. SafeStream Customization

21.5.1. Content Moderation Rules

Fine-tune what content gets flagged.

Sensitivity Levels:

```
moderation_config = {
    "sensitivity": "medium",  # low, medium, high, custom
    "categories": {
        "profanity": {
            "enabled": True,
            "action": "flag",  # flag, block, or allow
            "severity_threshold": "medium"
        },
        "harassment": {
            "enabled": True,
            "action": "block",
            "notify_admin": True
        },
        "academic_dishonesty": {
            "enabled": True,
            "action": "flag",
            "patterns": [
                "plagiarism_indicators",
                "contract_cheating_language",
                "suspicious_collaboration"
            ]
        },
        "personal_information": {
            "enabled": True,
            "action": "flag",
            "types": ["phone", "address", "ssn", "credit_card"]
        }
    }
}

kai.courses.update(
    course_id="course_abc",
    safestream_settings=moderation_config
)
```

21.5.2. Custom Keyword Lists

Add institution-specific or course-specific keywords.

```
custom_keywords = {
    "flag_words": [
        # Academic integrity
        "write my paper for me",
        "homework help service",

        # Institution-specific
        "campus-specific-concern",

        # Course-specific sensitive topics
        "sensitive-topic-1",
        "sensitive-topic-2"
    ],
    "allow_words": [
        # Context where flagged words are acceptable
        "discussing the ethics of cheating",  # Academic discussion
        "analyzing plagiarism in literature"  # Course content
    ],
    "escalate_words": [
        # Immediate admin notification
        "self-harm",
        "violence",
        "threat"
    ]
}

kai.safestream.update_keywords(
    course_id="course_abc",
    keywords=custom_keywords
)
```

21.5.3. Action Workflows

Define what happens when content is flagged.

```
action_workflow = {
    "low_severity": {
        "action": "log",
        "notify_instructor": False,
        "allow_post": True
    },
    "medium_severity": {
        "action": "flag",
        "notify_instructor": True,
        "allow_post": True,
        "require_review_within_hours": 24
```

```
    },
    "high_severity": {
        "action": "block",
        "notify_instructor": True,
        "notify_admin": True,
        "notify_counseling": ["self_harm", "violence"],
        "immediate_review": True
    }
}
```

21.6. UI/UX Customization

21.6.1. Branding

Customize the appearance of Kai to match your institution.

Dashboard Customization:

```
branding = {
    "logo_url": "https://your-institution.edu/logo.png",
    "primary_color": "#003366",      # Institution blue
    "secondary_color": "#FFD700",    # Institution gold
    "font_family": "Proxima Nova, sans-serif",
    "custom_css_url": "https://your-institution.edu/kai-custom.css"
}

kai.settings.update_branding(branding)
```

Mobile App Customization (Institution License):

```
app_branding = {
    "app_name": "University XYZ Learning Assistant",
    "app_icon_url": "https://your-institution.edu/app-icon.png",
    "splash_screen_url": "https://your-institution.edu/splash.png",
    "theme": {
        "primary_color": "#003366",
        "accent_color": "#FFD700",
        "dark_mode_available": True
    }
}
```

21.6.2. Notification Preferences

Customize notification timing and frequency.

Instructor Notifications:

```
instructor_notifications = {
    "feedback_complete": {
        "enabled": True,
        "method": ["push", "email"],
        "immediate": True
    },
    "grading_complete": {
        "enabled": True,
        "method": ["email"],
        "digest": "daily"  # Or "immediate", "weekly"
    },
    "low_engagement_alert": {
        "enabled": True,
        "threshold": 0.4,  # Alert if <40% participation
        "method": ["push"]
    },
    "quiet_hours": {
        "enabled": True,
        "start": "22:00",
        "end": "07:00",
        "timezone": "America/New_York"
    }
}
```

Student Notifications:

```
student_notifications = {
    "feedback_request": {
        "enabled": True,
        "method": ["push"],
        "sound": "gentle_chime"
    },
    "quiz_available": {
        "enabled": True,
        "method": ["push", "email"],
        "advance_notice_hours": 24
    },
    "grade_posted": {
        "enabled": True,
        "method": ["push"],
        "include_score": False  # Privacy option
    },
    "reminder_frequency": "once"  # Or "twice", "persistent"
}
```

21.7. Analytics Customization

21.7.1. Custom Metrics

Define custom metrics to track.

```
custom_metrics = {
    "engagement_score": {
        "formula": "(feedback_response_rate * 0.4) + (quiz_completion * 0.3) + (app_usage * 0.3)"
        "display_name": "Engagement Index",
        "threshold_warning": 0.6,
        "threshold_critical": 0.4
    },
    "comprehension_velocity": {
        "formula": "improvement_rate / time_period",
        "display_name": "Learning Velocity",
        "unit": "points/week"
    },
    "intervention_effectiveness": {
        "formula": "post_intervention_score - pre_intervention_score",
        "display_name": "Intervention Impact",
        "track_by": "topic"
    }
}

kai.analytics.add_custom_metrics(custom_metrics)
```

21.7.2. Dashboard Layout

Customize what appears on your dashboard.

```
dashboard_config = {
    "widgets": [
        {
            "type": "engagement_overview",
            "position": {"row": 1, "col": 1},
            "size": "large"
        },
        {
            "type": "recent_feedback",
            "position": {"row": 1, "col": 2},
            "size": "medium",
            "settings": {
                "show_analysis": True,
                "limit": 5
            }
```

```
        },
        {
            "type": "struggling_students",
            "position": {"row": 2, "col": 1},
            "size": "medium",
            "settings": {
                "threshold": 0.7,
                "metric": "quiz_average"
            }
        },
        {
            "type": "topic_confusion_trends",
            "position": {"row": 2, "col": 2},
            "size": "large",
            "settings": {
                "timeframe": "2_weeks"
            }
        }
    ],
    "refresh_interval_seconds": 60
}

kai.settings.update_dashboard(dashboard_config)
```

21.7.3. Export Templates

Create custom export formats for reporting.

```
export_template = {
    "name": "Weekly Institutional Report",
    "format": "pdf",
    "sections": [
        {
            "title": "Executive Summary",
            "include": ["total_students", "avg_engagement", "top_challenges"]
        },
        {
            "title": "Course Performance",
            "include": ["course_list", "completion_rates", "avg_scores"]
        },
        {
            "title": "Student Support Needs",
            "include": ["struggling_students", "intervention_recommendations"]
        },
        {
            "title": "Appendix",
```

```
            "include": ["detailed_analytics", "raw_data"]
        }
    ],
    "schedule": {
        "frequency": "weekly",
        "day": "Monday",
        "time": "06:00",
        "recipients": ["dean@institution.edu", "analytics@institution.edu"]
    }
}

kai.exports.create_template(export_template)
```

21.8. Integration Customization

21.8.1. Webhook Customization

Filter and transform webhook payloads.

```
webhook_config = {
    "url": "https://your-server.com/webhook",
    "events": ["feedback.received", "quiz.submitted"],
    "filters": {
        "course_ids": ["course_abc", "course_xyz"],  # Only these courses
        "min_confidence": 0.8,  # Only high-confidence events
        "exclude_test_data": True
    },
    "transform": {
        "include_fields": ["student_id", "score", "timestamp"],
        "exclude_fields": ["personal_info"],
        "rename_fields": {
            "student_id": "learner_id"
        }
    },
    "retry_policy": {
        "max_attempts": 3,
        "backoff": "exponential"
    }
}
```

21.8.2. LMS Sync Customization

Control what syncs between Kai and your LMS.

```
lms_sync_config = {
    "sync_direction": {
        "courses": "bidirectional",      # LMS ↔ Kai
        "students": "from_lms",          # LMS → Kai only
        "assignments": "bidirectional",
        "grades": "to_lms"               # Kai → LMS only
    },
    "sync_frequency": {
        "courses": "daily",
        "students": "hourly",
        "assignments": "hourly",
        "grades": "immediate"
    },
    "conflict_resolution": {
        "grade_conflicts": "keep_highest",  # Or "keep_latest", "manual_review"
        "enrollment_conflicts": "lms_wins"
    },
    "selective_sync": {
        "only_published_courses": True,
        "only_active_students": True,
        "exclude_past_courses": True
    }
}

kai.integrations.update_lms_sync(
    platform="canvas",
    config=lms_sync_config
)
```

21.9. Accessibility Customization

21.9.1. Language and Localization

Configure language preferences and translations.

```
localization = {
    "default_language": "en",
    "available_languages": ["en", "es", "fr", "zh"],
    "student_choice": True,  # Students can choose their language
    "auto_translate": {
        "enabled": True,
        "feedback": True,
        "questions": True,
        "ui_elements": True
    },
    "regional_settings": {
```

```
        "date_format": "MM/DD/YYYY",  # Or "DD/MM/YYYY", "YYYY-MM-DD"
        "time_format": "12h",         # Or "24h"
        "timezone": "America/New_York"
    }
}
```

21.9.2. Accessibility Features

```
accessibility = {
    "screen_reader": {
        "enabled": True,
        "verbose_mode": True
    },
    "visual": {
        "high_contrast_mode": True,
        "font_size_options": ["small", "medium", "large", "x-large"],
        "dyslexia_friendly_font": True
    },
    "mobile_accessibility": {
        "voice_input": True,
        "haptic_feedback": True,
        "simplified_ui": True
    }
}
```

21.10. Template Library

21.10.1. Pre-built Configurations

Kai provides templates for common scenarios:

Available Templates:

Template	Description	Best For
Large Lecture	High-frequency feedback, auto-grading	100+ students
Seminar Discussion	Rich feedback, peer review	10-25 students
Flipped Classroom	Pre-class quizzes, targeted support	Any size
Lab Course	Practical assessments, safety monitoring	Hands-on courses
Online Asynchronous	Flexible deadlines, discussion monitoring	Remote learning
Hybrid	Balanced online/in-person features	Mixed format

Applying a Template:

```
kai.courses.apply_template(
    course_id="course_abc",
    template="large_lecture",
    customize={
        "feedback_frequency": "high",  # Override template default
        "auto_grade_threshold": 0.90   # Stricter than template
    }
)
```

21.11. Advanced Customization

21.11.1. Custom Algorithms

Institution license holders can upload custom algorithms.

Example: Custom Engagement Score

```
def custom_engagement_score(student_data):
    """
    Custom engagement calculation emphasizing attendance
    """
    attendance_weight = 0.5
    participation_weight = 0.3
    completion_weight = 0.2

    score = (
        student_data['attendance_rate'] * attendance_weight +
        student_data['participation_rate'] * participation_weight +
        student_data['completion_rate'] * completion_weight
    )

    return {
        'score': score,
        'level': 'high' if score > 0.8 else 'medium' if score > 0.5 else 'low',
        'components': {
            'attendance': student_data['attendance_rate'],
            'participation': student_data['participation_rate'],
            'completion': student_data['completion_rate']
        }
    }

# Upload custom algorithm
kai.algorithms.upload(
    name="custom_engagement",
    function=custom_engagement_score,
```

```
    test_data=sample_student_data
)
```

21.12. Best Practices

21.12.1. Start Simple

> **Incremental Customization**
>
> Don't customize everything at once. Start with defaults, identify pain points, then customize specific features.

Recommended Progression: 1. **Week 1-2**: Use defaults, identify needs 2. **Week 3-4**: Customize feedback questions and timing 3. **Week 5-6**: Add custom rubrics 4. **Week 7+**: Advanced features (custom metrics, algorithms)

21.12.2. Version Control

Track your customizations:

```
# Export current configuration
config = kai.settings.export(course_id="course_abc")

# Save as version
with open("kai_config_v2.json", "w") as f:
    json.dump(config, f, indent=2)

# Import configuration
with open("kai_config_v2.json", "r") as f:
    config = json.load(f)

kai.settings.import_config(
    course_id="course_new",
    config=config
)
```

21.12.3. Test Before Deploying

Always test customizations in a sandbox course:

```
# Create test course
test_course = kai.courses.create(
    name="TEST - Configuration Testing",
    is_sandbox=True
```

```
)

# Apply customizations
kai.settings.import_config(
    course_id=test_course.course_id,
    config=custom_config
)

# Test with sample data
# ... testing ...

# If successful, apply to production
kai.settings.import_config(
    course_id="course_abc",
    config=custom_config
)
```

21.13. Support

- **Customization Help**: customization@chi2labs.com
- **API Reference**: Complete API documentation
- **Training**: Advanced customization workshops
- **Community**: Share configurations

Customization capabilities vary by plan. Contact sales for institution-specific features.

Part V.

SDKs

22. Python SDK

Official Python SDK for Kai the AI

22.1. Overview

The official Python SDK for Kai the AI provides a modern, type-safe interface for integrating intelligent teaching assistance into your Python applications. Built with Python 3.8+ features, the SDK supports both synchronous and asynchronous operations, comprehensive error handling, and follows PEP 8 style guidelines.

22.2. Installation

Install the SDK using pip:

```
pip install kai-sdk
```

For development installations with optional dependencies:

```
pip install kai-sdk[dev]  # Includes testing and development tools
pip install kai-sdk[async]  # Includes async HTTP client dependencies
```

22.2.1. Requirements

- Python 3.8 or higher
- Active Kai the AI API key
- Internet connection for API requests

22.3. Quick Start

Here's a minimal example to get you started:

```
from kai_sdk import KaiClient

# Initialize the client
client = KaiClient(api_key="your_api_key_here")

# Grade a student submission
result = client.assignments.grade(
    assignment_id="CS101-HW1",
    student_submission="The sorting algorithm works by...",
    grading_style="detailed"
)

print(f"Score: {result.score}/100")
print(f"Feedback: {result.feedback}")
```

22.4. Authentication

22.4.1. API Key Setup

The SDK requires an API key for authentication. You can provide the key in several ways:

22.4.2. Environment Variable

```
export KAI_API_KEY="your_api_key_here"
```

```
from kai_sdk import KaiClient

# Automatically reads from KAI_API_KEY environment variable
client = KaiClient()
```

22.4.3. Direct Initialization

```
from kai_sdk import KaiClient

client = KaiClient(api_key="your_api_key_here")
```

22.4.4. Configuration File

```python
from kai_sdk import KaiClient
import os

# Load from configuration file
with open("config/kai_config.txt") as f:
    api_key = f.read().strip()

client = KaiClient(api_key=api_key)
```

> **Security Best Practice**
>
> Never hardcode API keys in your source code or commit them to version control. Use environment variables or secure configuration management.

22.5. Core Classes

22.5.1. KaiClient

The main client class for interacting with Kai's API.

```python
class KaiClient:
    """
    Main client for Kai the AI API.

    Args:
        api_key: Your API key (defaults to KAI_API_KEY env variable)
        base_url: API base URL (default: https://chi2api.com/v1)
        timeout: Request timeout in seconds (default: 30)
        max_retries: Maximum number of retry attempts (default: 3)
    """

    def __init__(
        self,
        api_key: str | None = None,
        base_url: str = "https://chi2api.com/v1",
        timeout: int = 30,
        max_retries: int = 3
    ) -> None:
        ...
```

Attributes:

- `assignments`: Assignment grading operations
- `content`: Content generation operations
- `analytics`: Student analytics and performance metrics
- `quizzes`: Quiz generation and management

22.5.2. Assignment

Handle assignment grading and feedback operations.

```
class Assignment:
    """Assignment grading operations."""

    def grade(
        self,
        assignment_id: str,
        student_submission: str,
        rubric_id: str | None = None,
        grading_style: str = "detailed"
    ) -> GradeResult:
        """
        Grade a student submission.

        Args:
            assignment_id: Unique identifier for the assignment
            student_submission: Student's submitted work
            rubric_id: Optional rubric to apply
            grading_style: "detailed", "concise", or "points_only"

        Returns:
            GradeResult object containing score and feedback

        Raises:
            ValidationError: Invalid parameters
            RateLimitError: Rate limit exceeded
            APIError: API request failed
        """
        ...

    def get_grade(self, grade_id: str) -> GradeResult:
        """Retrieve a previously generated grade."""
        ...

    def batch_grade(
        self,
        submissions: list[dict]
    ) -> list[GradeResult]:
        """Grade multiple submissions in a single request."""
        ...
```

22.5.3. Quiz

Generate and manage quiz content.

```python
class Quiz:
    """Quiz generation and management operations."""

    def generate(
        self,
        topic: str,
        difficulty: str = "medium",
        question_count: int = 10,
        question_types: list[str] | None = None
    ) -> QuizResult:
        """
        Generate quiz questions on a topic.

        Args:
            topic: Subject matter for quiz questions
            difficulty: "easy", "medium", or "hard"
            question_count: Number of questions (1-50)
            question_types: List of question types to include

        Returns:
            QuizResult with generated questions
        """
        ...
```

22.5.4. Analytics

Access student performance data and analytics.

```python
class Analytics:
    """Student analytics and performance metrics."""

    def get_student_performance(
        self,
        student_id: str,
        start_date: str | None = None,
        end_date: str | None = None,
        course_id: str | None = None
    ) -> PerformanceData:
        """
        Retrieve student performance metrics.

        Args:
            student_id: Student identifier
            start_date: ISO 8601 date string (optional)
            end_date: ISO 8601 date string (optional)
            course_id: Filter by specific course (optional)
```

```
        Returns:
            PerformanceData with metrics and trends
        """
        ...

    def get_class_analytics(
        self,
        course_id: str,
        metric_types: list[str] | None = None
    ) -> ClassAnalytics:
        """Retrieve aggregate class-level analytics."""
        ...
```

22.6. Response Models

22.6.1. GradeResult

```
from dataclasses import dataclass
from datetime import datetime

@dataclass
class GradeResult:
    """Result from grading operation."""

    grade_id: str
    score: int  # 0-100
    feedback: str
    suggestions: list[str]
    timestamp: datetime
    rubric_breakdown: dict[str, int] | None = None

    @property
    def passed(self) -> bool:
        """Check if grade meets passing threshold (>= 70)."""
        return self.score >= 70
```

22.6.2. QuizResult

```
@dataclass
class QuizQuestion:
    """Individual quiz question."""

    question_id: str
```

```
    question_text: str
    question_type: str
    options: list[str] | None = None
    correct_answer: str | None = None
    explanation: str | None = None

@dataclass
class QuizResult:
    """Result from quiz generation."""

    quiz_id: str
    topic: str
    difficulty: str
    questions: list[QuizQuestion]
    estimated_time_minutes: int
```

22.6.3. PerformanceData

```
@dataclass
class PerformanceData:
    """Student performance metrics."""

    student_id: str
    average_score: float
    assignments_completed: int
    assignments_total: int
    strength_areas: list[str]
    improvement_areas: list[str]
    trend: str  # "improving", "stable", "declining"
    last_updated: datetime
```

22.7. Complete API Methods

22.7.1. Assignment Grading

22.7.2. Synchronous

```
from kai_sdk import KaiClient
from kai_sdk.exceptions import ValidationError, RateLimitError

client = KaiClient(api_key="your_api_key")

try:
```

```
    # Grade a single submission
    result = client.assignments.grade(
        assignment_id="CS101-HW1",
        student_submission="My algorithm implementation...",
        rubric_id="standard_coding_rubric",
        grading_style="detailed"
    )

    print(f"Grade ID: {result.grade_id}")
    print(f"Score: {result.score}/100")
    print(f"Feedback: {result.feedback}")

    if result.rubric_breakdown:
        print("\nRubric Breakdown:")
        for criterion, points in result.rubric_breakdown.items():
            print(f"  {criterion}: {points}")

    print("\nSuggestions:")
    for suggestion in result.suggestions:
        print(f"  - {suggestion}")

except ValidationError as e:
    print(f"Invalid parameters: {e}")
except RateLimitError as e:
    print(f"Rate limit exceeded: {e}")
```

22.7.3. Asynchronous

```
import asyncio
from kai_sdk import AsyncKaiClient

async def grade_submission():
    async with AsyncKaiClient(api_key="your_api_key") as client:
        result = await client.assignments.grade(
            assignment_id="CS101-HW1",
            student_submission="My algorithm implementation...",
            grading_style="detailed"
        )

        print(f"Score: {result.score}/100")
        return result

# Run async function
result = asyncio.run(grade_submission())
```

22.7.4. Batch Grading

Grade multiple submissions efficiently:

```
from kai_sdk import KaiClient

client = KaiClient()

submissions = [
    {
        "assignment_id": "CS101-HW1",
        "student_id": "student_001",
        "student_submission": "First student's work..."
    },
    {
        "assignment_id": "CS101-HW1",
        "student_id": "student_002",
        "student_submission": "Second student's work..."
    },
    # ... more submissions
]

results = client.assignments.batch_grade(submissions)

for result in results:
    print(f"Student {result.student_id}: {result.score}/100")
```

22.7.5. Quiz Generation

22.7.6. Basic Quiz

```
from kai_sdk import KaiClient

client = KaiClient()

quiz = client.quizzes.generate(
    topic="Python Data Structures",
    difficulty="medium",
    question_count=10,
    question_types=["multiple_choice", "true_false"]
)

print(f"Quiz ID: {quiz.quiz_id}")
print(f"Estimated time: {quiz.estimated_time_minutes} minutes")
print(f"Questions: {len(quiz.questions)}")
```

```
for i, question in enumerate(quiz.questions, 1):
    print(f"\nQuestion {i}: {question.question_text}")
    if question.options:
        for opt in question.options:
            print(f"  {opt}")
```

22.7.7. Advanced Quiz with Async

```
import asyncio
from kai_sdk import AsyncKaiClient

async def generate_multiple_quizzes():
    async with AsyncKaiClient() as client:
        # Generate multiple quizzes concurrently
        topics = [
            "Python Basics",
            "Object-Oriented Programming",
            "Data Structures"
        ]

        tasks = [
            client.quizzes.generate(
                topic=topic,
                difficulty="medium",
                question_count=5
            )
            for topic in topics
        ]

        quizzes = await asyncio.gather(*tasks)
        return quizzes

quizzes = asyncio.run(generate_multiple_quizzes())
print(f"Generated {len(quizzes)} quizzes")
```

22.7.8. Student Analytics

```
from kai_sdk import KaiClient
from datetime import datetime, timedelta

client = KaiClient()

# Get performance for the last 30 days
end_date = datetime.now()
```

```
start_date = end_date - timedelta(days=30)

performance = client.analytics.get_student_performance(
    student_id="student_12345",
    start_date=start_date.isoformat(),
    end_date=end_date.isoformat(),
    course_id="CS101"
)

print(f"Average Score: {performance.average_score:.1f}")
print(f"Completion Rate: {performance.assignments_completed}/{performance.assignments_total}")
print(f"Trend: {performance.trend}")

print("\nStrengths:")
for area in performance.strength_areas:
    print(f"    {area}")

print("\nAreas for Improvement:")
for area in performance.improvement_areas:
    print(f"  ! {area}")
```

22.8. Error Handling

The SDK provides specific exception types for different error scenarios:

```
from kai_sdk import KaiClient
from kai_sdk.exceptions import (
    KaiSDKError,           # Base exception
    ValidationError,       # Invalid parameters
    AuthenticationError,   # Invalid API key
    RateLimitError,       # Rate limit exceeded
    APIError,             # General API error
    TimeoutError,         # Request timeout
)

client = KaiClient()

try:
    result = client.assignments.grade(
        assignment_id="CS101-HW1",
        student_submission="Student work...",
        grading_style="detailed"
    )

except ValidationError as e:
    # Handle invalid parameters
```

```
    print(f"Invalid input: {e}")
    print(f"Details: {e.details}")

except AuthenticationError as e:
    # Handle authentication failures
    print(f"Authentication failed: {e}")
    print("Please check your API key")

except RateLimitError as e:
    # Handle rate limiting
    print(f"Rate limit exceeded: {e}")
    print(f"Retry after: {e.retry_after} seconds")

except TimeoutError as e:
    # Handle timeouts
    print(f"Request timed out: {e}")
    print("Try increasing the timeout parameter")

except APIError as e:
    # Handle general API errors
    print(f"API error: {e}")
    print(f"Status code: {e.status_code}")
    print(f"Error code: {e.error_code}")

except KaiSDKError as e:
    # Catch-all for SDK errors
    print(f"SDK error: {e}")
```

Error Recovery Pattern

Implement exponential backoff for rate limit errors:

```python
import time
from kai_sdk.exceptions import RateLimitError

max_retries = 3
retry_delay = 1

for attempt in range(max_retries):
    try:
        result = client.assignments.grade(...)
        break
    except RateLimitError as e:
        if attempt < max_retries - 1:
            wait_time = retry_delay * (2 ** attempt)
            print(f"Rate limited, waiting {wait_time}s...")
            time.sleep(wait_time)
        else:
            raise
```

22.9. Async/Await Support

The SDK provides full async support through the `AsyncKaiClient`:

22.9.1. Basic Async Usage

```python
import asyncio
from kai_sdk import AsyncKaiClient

async def main():
    async with AsyncKaiClient(api_key="your_api_key") as client:
        # All methods are awaitable
        result = await client.assignments.grade(
            assignment_id="CS101-HW1",
            student_submission="Student work..."
        )
        print(f"Score: {result.score}")

asyncio.run(main())
```

22.9.2. Concurrent Operations

```python
import asyncio
from kai_sdk import AsyncKaiClient
```

```
async def process_multiple_submissions():
    async with AsyncKaiClient() as client:
        # Grade multiple submissions concurrently
        submissions = [
            ("assignment_1", "submission text 1"),
            ("assignment_2", "submission text 2"),
            ("assignment_3", "submission text 3"),
        ]

        tasks = [
            client.assignments.grade(
                assignment_id=aid,
                student_submission=text
            )
            for aid, text in submissions
        ]

        # Execute all grading requests concurrently
        results = await asyncio.gather(*tasks)

        for i, result in enumerate(results, 1):
            print(f"Submission {i}: {result.score}/100")

asyncio.run(process_multiple_submissions())
```

22.9.3. Async Error Handling

```
import asyncio
from kai_sdk import AsyncKaiClient
from kai_sdk.exceptions import RateLimitError, APIError

async def safe_grade(client, assignment_id, submission):
    try:
        result = await client.assignments.grade(
            assignment_id=assignment_id,
            student_submission=submission
        )
        return result
    except RateLimitError:
        # Wait and retry
        await asyncio.sleep(5)
        return await safe_grade(client, assignment_id, submission)
    except APIError as e:
        print(f"Failed to grade: {e}")
        return None
```

```
async def main():
    async with AsyncKaiClient() as client:
        result = await safe_grade(
            client,
            "CS101-HW1",
            "Student work..."
        )
        if result:
            print(f"Score: {result.score}")

asyncio.run(main())
```

22.10. Rate Limiting and Best Practices

22.10.1. Understanding Rate Limits

Kai enforces the following rate limits based on your plan:

Plan	Requests/Minute	Requests/Day
Free	10	1,000
Educator	60	10,000
Institution	300	100,000

22.10.2. Handling Rate Limits

The SDK automatically handles rate limiting with exponential backoff:

```
from kai_sdk import KaiClient

# Configure retry behavior
client = KaiClient(
    api_key="your_api_key",
    max_retries=5,  # Retry up to 5 times
    timeout=60      # 60 second timeout
)

# The client will automatically retry on rate limit errors
result = client.assignments.grade(...)
```

22.10.3. Best Practices

Performance Optimization Tips

1. **Use Batch Operations**: Grade multiple submissions in one request

```
results = client.assignments.batch_grade(submissions)
```

2. **Enable Async for Concurrent Operations**: Use `AsyncKaiClient` for parallel requests

```
async with AsyncKaiClient() as client:
    results = await asyncio.gather(*tasks)
```

3. **Implement Caching**: Cache frequently accessed data like rubrics

```
from functools import lru_cache

@lru_cache(maxsize=100)
def get_rubric(rubric_id):
    return client.rubrics.get(rubric_id)
```

4. **Monitor Rate Limits**: Check response headers for rate limit status

```
result = client.assignments.grade(...)
remaining = client.last_response.headers.get('X-RateLimit-Remaining')
print(f"Requests remaining: {remaining}")
```

5. **Use Webhooks for Long Operations**: For batch processing, use webhooks instead of polling

22.11. Configuration Options

22.11.1. Client Configuration

```
from kai_sdk import KaiClient
from kai_sdk.config import Config

# Method 1: Direct configuration
client = KaiClient(
    api_key="your_api_key",
    base_url="https://chi2api.com/v1",
    timeout=30,
    max_retries=3,
    verify_ssl=True,
```

```
    user_agent="MyApp/1.0"
)

# Method 2: Using Config object
config = Config(
    api_key="your_api_key",
    timeout=60,
    max_retries=5
)
client = KaiClient(config=config)

# Method 3: Environment-based configuration
# Reads from KAI_API_KEY, KAI_BASE_URL, etc.
client = KaiClient.from_env()
```

22.11.2. Logging Configuration

```
import logging
from kai_sdk import KaiClient

# Enable debug logging
logging.basicConfig(level=logging.DEBUG)
kai_logger = logging.getLogger('kai_sdk')
kai_logger.setLevel(logging.DEBUG)

client = KaiClient()

# Now all SDK operations will log debug information
result = client.assignments.grade(...)
```

22.11.3. Custom HTTP Session

```
from kai_sdk import KaiClient
import requests

# Create custom session with specific settings
session = requests.Session()
session.headers.update({'X-Custom-Header': 'value'})
session.proxies = {'https': 'http://proxy.example.com:8080'}

client = KaiClient(
    api_key="your_api_key",
    session=session
)
```

22.12. Complete Working Examples

22.12.1. Example 1: Automated Grading Pipeline

```
#!/usr/bin/env python3
"""
Automated grading pipeline for processing student submissions.
"""
from pathlib import Path
from typing import List
from kai_sdk import KaiClient
from kai_sdk.exceptions import KaiSDKError
import csv

def load_submissions(csv_path: Path) -> List[dict]:
    """Load student submissions from CSV file."""
    submissions = []
    with open(csv_path) as f:
        reader = csv.DictReader(f)
        for row in reader:
            submissions.append({
                'student_id': row['student_id'],
                'assignment_id': row['assignment_id'],
                'submission_text': row['submission_text']
            })
    return submissions

def grade_submissions(client: KaiClient, submissions: List[dict]) -> List[dict]:
    """Grade all submissions and return results."""
    results = []

    for submission in submissions:
        try:
            result = client.assignments.grade(
                assignment_id=submission['assignment_id'],
                student_submission=submission['submission_text'],
                grading_style='detailed'
            )

            results.append({
                'student_id': submission['student_id'],
                'score': result.score,
                'feedback': result.feedback,
                'suggestions': ', '.join(result.suggestions)
            })
```

```
            print(f"  Graded {submission['student_id']}: {result.score}/100")

        except KaiSDKError as e:
            print(f"  Failed to grade {submission['student_id']}: {e}")
            results.append({
                'student_id': submission['student_id'],
                'score': None,
                'feedback': f"Error: {e}",
                'suggestions': ''
            })

    return results

def save_results(results: List[dict], output_path: Path):
    """Save grading results to CSV."""
    with open(output_path, 'w', newline='') as f:
        writer = csv.DictWriter(f, fieldnames=['student_id', 'score', 'feedback', 'suggestions'])
        writer.writeheader()
        writer.writerows(results)

    print(f"\n  Results saved to {output_path}")

def main():
    # Initialize client
    client = KaiClient()

    # Load submissions
    submissions = load_submissions(Path('submissions.csv'))
    print(f"Loaded {len(submissions)} submissions")

    # Grade all submissions
    results = grade_submissions(client, submissions)

    # Save results
    save_results(results, Path('grading_results.csv'))

    # Print summary
    successful = sum(1 for r in results if r['score'] is not None)
    avg_score = sum(r['score'] for r in results if r['score']) / successful

    print(f"\n=== Summary ===")
    print(f"Total submissions: {len(submissions)}")
    print(f"Successfully graded: {successful}")
    print(f"Average score: {avg_score:.1f}/100")

if __name__ == '__main__':
    main()
```

22.12.2. Example 2: Quiz Generator with Export

```
#!/usr/bin/env python3
"""
Generate quizzes for multiple topics and export to various formats.
"""
from kai_sdk import KaiClient
from typing import List
import json

def generate_quiz(
    client: KaiClient,
    topic: str,
    difficulty: str = "medium",
    count: int = 10
):
    """Generate a quiz for a specific topic."""
    quiz = client.quizzes.generate(
        topic=topic,
        difficulty=difficulty,
        question_count=count,
        question_types=["multiple_choice", "true_false", "short_answer"]
    )

    print(f"\n Generated quiz: {topic}")
    print(f"  ID: {quiz.quiz_id}")
    print(f"  Questions: {len(quiz.questions)}")
    print(f"  Time: ~{quiz.estimated_time_minutes} minutes")

    return quiz

def export_quiz_to_json(quiz, filename: str):
    """Export quiz to JSON format."""
    quiz_data = {
        'quiz_id': quiz.quiz_id,
        'topic': quiz.topic,
        'difficulty': quiz.difficulty,
        'estimated_time_minutes': quiz.estimated_time_minutes,
        'questions': [
            {
                'id': q.question_id,
                'text': q.question_text,
                'type': q.question_type,
                'options': q.options,
                'correct_answer': q.correct_answer,
                'explanation': q.explanation
            }
```

```
            for q in quiz.questions
        ]
    }

    with open(filename, 'w') as f:
        json.dump(quiz_data, f, indent=2)

    print(f" Exported to {filename}")

def export_quiz_to_markdown(quiz, filename: str):
    """Export quiz to Markdown format."""
    with open(filename, 'w') as f:
        f.write(f"# {quiz.topic} Quiz\n\n")
        f.write(f"**Difficulty:** {quiz.difficulty}\n")
        f.write(f"**Estimated Time:** {quiz.estimated_time_minutes} minutes\n\n")
        f.write("---\n\n")

        for i, q in enumerate(quiz.questions, 1):
            f.write(f"## Question {i}\n\n")
            f.write(f"{q.question_text}\n\n")

            if q.options:
                for opt in q.options:
                    f.write(f"- {opt}\n")
                f.write("\n")

            if q.explanation:
                f.write(f"**Explanation:** {q.explanation}\n\n")

            f.write("---\n\n")

    print(f" Exported to {filename}")

def main():
    client = KaiClient()

    # Topics to generate quizzes for
    topics = [
        ("Python Fundamentals", "easy", 15),
        ("Data Structures", "medium", 20),
        ("Algorithm Design", "hard", 10)
    ]

    for topic, difficulty, count in topics:
        # Generate quiz
        quiz = generate_quiz(client, topic, difficulty, count)
```

```
        # Export to different formats
        base_name = topic.lower().replace(' ', '_')
        export_quiz_to_json(quiz, f"{base_name}_quiz.json")
        export_quiz_to_markdown(quiz, f"{base_name}_quiz.md")

    print("\n  All quizzes generated and exported successfully!")

if __name__ == '__main__':
    main()
```

22.12.3. Example 3: Student Analytics Dashboard

```
#!/usr/bin/env python3
"""
Generate comprehensive student analytics reports.
"""
from kai_sdk import KaiClient
from datetime import datetime, timedelta
from typing import List
import matplotlib.pyplot as plt
from pathlib import Path

def get_class_roster(csv_path: Path) -> List[str]:
    """Load student IDs from roster."""
    import csv
    with open(csv_path) as f:
        reader = csv.DictReader(f)
        return [row['student_id'] for row in reader]

def analyze_student(client: KaiClient, student_id: str, course_id: str):
    """Get comprehensive analytics for a student."""
    end_date = datetime.now()
    start_date = end_date - timedelta(days=90)  # Last 90 days

    performance = client.analytics.get_student_performance(
        student_id=student_id,
        start_date=start_date.isoformat(),
        end_date=end_date.isoformat(),
        course_id=course_id
    )

    return {
        'student_id': student_id,
        'average_score': performance.average_score,
        'completion_rate': performance.assignments_completed / performance.assignments_total,
```

```
        'trend': performance.trend,
        'strengths': performance.strength_areas,
        'improvements': performance.improvement_areas
    }

def generate_class_report(analytics_data: List[dict], output_dir: Path):
    """Generate visual report for the class."""
    output_dir.mkdir(exist_ok=True)

    # Extract data for visualization
    scores = [data['average_score'] for data in analytics_data]
    completion_rates = [data['completion_rate'] * 100 for data in analytics_data]

    # Create visualizations
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

    # Score distribution
    ax1.hist(scores, bins=10, edgecolor='black')
    ax1.set_xlabel('Average Score')
    ax1.set_ylabel('Number of Students')
    ax1.set_title('Score Distribution')
    ax1.axvline(sum(scores)/len(scores), color='red', linestyle='--', label='Class Average')
    ax1.legend()

    # Completion rate distribution
    ax2.hist(completion_rates, bins=10, edgecolor='black')
    ax2.set_xlabel('Completion Rate (%)')
    ax2.set_ylabel('Number of Students')
    ax2.set_title('Assignment Completion Distribution')

    plt.tight_layout()
    plt.savefig(output_dir / 'class_analytics.png', dpi=300, bbox_inches='tight')
    print(f"  Saved visualization to {output_dir / 'class_analytics.png'}")

    # Generate text report
    with open(output_dir / 'class_report.txt', 'w') as f:
        f.write("CLASS ANALYTICS REPORT\n")
        f.write("=" * 50 + "\n\n")

        f.write(f"Total Students: {len(analytics_data)}\n")
        f.write(f"Class Average: {sum(scores)/len(scores):.1f}\n")
        f.write(f"Average Completion Rate: {sum(completion_rates)/len(completion_rates):.1f}%\n\n

        # Students by trend
        improving = sum(1 for d in analytics_data if d['trend'] == 'improving')
        stable = sum(1 for d in analytics_data if d['trend'] == 'stable')
        declining = sum(1 for d in analytics_data if d['trend'] == 'declining')
```

```
        f.write("Student Trends:\n")
        f.write(f"  Improving: {improving} ({improving/len(analytics_data)*100:.1f}%)\n")
        f.write(f"  Stable: {stable} ({stable/len(analytics_data)*100:.1f}%)\n")
        f.write(f"  Declining: {declining} ({declining/len(analytics_data)*100:.1f}%)\n\n")

        # Students needing attention
        f.write("Students Needing Attention:\n")
        for data in analytics_data:
            if data['average_score'] < 70 or data['trend'] == 'declining':
                f.write(f"  - {data['student_id']}: Score {data['average_score']:.1f}, {data['tre

    print(f" Saved report to {output_dir / 'class_report.txt'}")

def main():
    client = KaiClient()
    course_id = "CS101"

    # Load student roster
    student_ids = get_class_roster(Path('roster.csv'))
    print(f"Analyzing {len(student_ids)} students...")

    # Gather analytics for all students
    analytics_data = []
    for student_id in student_ids:
        try:
            data = analyze_student(client, student_id, course_id)
            analytics_data.append(data)
            print(f" Analyzed {student_id}")
        except Exception as e:
            print(f" Failed to analyze {student_id}: {e}")

    # Generate comprehensive report
    output_dir = Path('analytics_report')
    generate_class_report(analytics_data, output_dir)

    print(f"\n Analytics complete! Check {output_dir} for results")

if __name__ == '__main__':
    main()
```

22.13. Type Hints and Modern Python

The SDK leverages modern Python features for better type safety:

```python
from typing import Optional, Union, List, Dict, Any
from kai_sdk import KaiClient
from kai_sdk.models import GradeResult

def process_grade(
    client: KaiClient,
    assignment_id: str,
    submission: str,
    rubric_id: Optional[str] = None
) -> Union[GradeResult, None]:
    """
    Type-safe grading function with proper annotations.

    Args:
        client: Initialized KaiClient instance
        assignment_id: Assignment identifier
        submission: Student submission text
        rubric_id: Optional rubric identifier

    Returns:
        GradeResult if successful, None if failed
    """
    try:
        result = client.assignments.grade(
            assignment_id=assignment_id,
            student_submission=submission,
            rubric_id=rubric_id
        )
        return result
    except Exception as e:
        print(f"Grading failed: {e}")
        return None

# Type checkers (mypy, pyright) will validate all types
result: Optional[GradeResult] = process_grade(
    client=client,
    assignment_id="CS101-HW1",
    submission="Student work..."
)

if result is not None:
    score: int = result.score
    print(f"Score: {score}")
```

22.14. Links and Resources

22.14.1. Related Documentation

- API Reference - Complete REST API documentation
- Getting Started - Initial setup and configuration
- Guides - Integration guides and tutorials

22.14.2. SDK Resources

- GitHub Repository - Source code and issues
- PyPI Package - Package distribution
- Changelog - Version history
- Examples - More code examples

22.14.3. Support

- API Status: status.kaitheai.com
- Developer Forum: forum.kaitheai.com
- Email: developers@chi2labs.com

> **Stay Updated**
>
> Star the GitHub repository to receive notifications about new releases and features.

23. JavaScript/TypeScript SDK

Official JavaScript and TypeScript SDK for Kai the AI

23.1. Overview

The Kai the AI JavaScript/TypeScript SDK provides a powerful, type-safe interface for integrating intelligent teaching assistant capabilities into your Node.js and browser-based applications. Built with modern JavaScript standards and full TypeScript support, the SDK offers comprehensive access to all Kai API features.

23.2. Installation

Install the SDK using your preferred package manager:

23.2.1. npm

```
npm install @kai-ai/sdk
```

23.2.2. yarn

```
yarn add @kai-ai/sdk
```

23.2.3. pnpm

```
pnpm add @kai-ai/sdk
```

> **Note**
>
> The SDK works in both Node.js (v16+) and modern browsers with ES2020+ support. TypeScript 5.0+ is recommended for the best development experience.

23.3. Quick Start

Get started with the SDK in just a few lines of code:

23.3.1. TypeScript

```typescript
import { KaiClient } from '@kai-ai/sdk';

// Initialize the client
const kai = new KaiClient({
  apiKey: 'your-api-key',
  baseUrl: 'https://chi2api.com/v1' // optional
});

// Grade an assignment
const result = await kai.assignments.grade({
  studentId: 'student-123',
  assignmentId: 'assign-456',
  submission: 'The mitochondria is the powerhouse of the cell...',
  rubric: {
    criteria: [
      { name: 'accuracy', weight: 0.6 },
      { name: 'clarity', weight: 0.4 }
    ]
  }
});

console.log(`Grade: ${result.score}/100`);
console.log(`Feedback: ${result.feedback}`);
```

23.3.2. JavaScript

```javascript
const { KaiClient } = require('@kai-ai/sdk');

// Initialize the client
const kai = new KaiClient({
  apiKey: 'your-api-key',
  baseUrl: 'https://chi2api.com/v1' // optional
});

// Grade an assignment
const result = await kai.assignments.grade({
  studentId: 'student-123',
  assignmentId: 'assign-456',
  submission: 'The mitochondria is the powerhouse of the cell...',
```

```
  rubric: {
    criteria: [
      { name: 'accuracy', weight: 0.6 },
      { name: 'clarity', weight: 0.4 }
    ]
  }
});

console.log(`Grade: ${result.score}/100`);
console.log(`Feedback: ${result.feedback}`);
```

23.4. Authentication

The SDK requires an API key for authentication. You can obtain your API key from the Kai the AI Dashboard.

23.4.1. Environment Variables

```
// Recommended: Use environment variables
import { KaiClient } from '@kai-ai/sdk';

const kai = new KaiClient({
  apiKey: process.env.KAI_API_KEY
});
```

23.4.2. Direct Configuration

```
// Direct API key (not recommended for production)
import { KaiClient } from '@kai-ai/sdk';

const kai = new KaiClient({
  apiKey: 'kai_sk_1234567890abcdef'
});
```

23.4.3. Custom Headers

```
// Add custom headers for advanced use cases
import { KaiClient } from '@kai-ai/sdk';

const kai = new KaiClient({
  apiKey: process.env.KAI_API_KEY,
```

```
  headers: {
    'X-Custom-Header': 'value',
    'X-Request-ID': generateRequestId()
  }
});
```

> **Warning**
>
> **Security Best Practice**: Never hardcode API keys in your source code. Always use environment variables or secure configuration management systems.

23.5. Core Classes and Types

23.5.1. KaiClient

The main client class for interacting with the Kai API.

23.5.2. TypeScript

```
interface KaiClientConfig {
  apiKey: string;
  baseUrl?: string;
  timeout?: number;
  retries?: number;
  headers?: Record<string, string>;
}

class KaiClient {
  constructor(config: KaiClientConfig);

  // Namespaced API methods
  assignments: AssignmentsAPI;
  quizzes: QuizzesAPI;
  analytics: AnalyticsAPI;
  students: StudentsAPI;
}
```

23.5.3. JavaScript

```
// Configuration object
const config = {
  apiKey: 'your-api-key',
```

```
  baseUrl: 'https://chi2api.com/v1', // optional
  timeout: 30000, // optional, in milliseconds
  retries: 3, // optional, number of retry attempts
  headers: {} // optional, custom headers
};

const kai = new KaiClient(config);
```

23.5.4. Assignment Types

23.5.5. TypeScript

```
// Branded types for type safety
type StudentId = string & { readonly __brand: 'StudentId' };
type AssignmentId = string & { readonly __brand: 'AssignmentId' };

interface RubricCriterion {
  name: string;
  description?: string;
  weight: number; // 0.0 to 1.0
  maxScore?: number;
}

interface Rubric {
  criteria: RubricCriterion[];
  totalPoints?: number;
}

interface GradeAssignmentRequest {
  studentId: StudentId;
  assignmentId: AssignmentId;
  submission: string;
  rubric: Rubric;
  provideFeedback?: boolean;
  detailedAnalysis?: boolean;
}

interface GradeResult {
  assignmentId: AssignmentId;
  studentId: StudentId;
  score: number;
  maxScore: number;
  percentage: number;
  feedback: string;
  criteriaScores: CriterionScore[];
  suggestions: string[];
```

```
  gradedAt: Date;
}

interface CriterionScore {
  criterion: string;
  score: number;
  maxScore: number;
  feedback: string;
}
```

23.5.6. JavaScript

```
// Example assignment grading request
const gradeRequest = {
  studentId: 'student-123',
  assignmentId: 'assign-456',
  submission: 'Student answer text...',
  rubric: {
    criteria: [
      {
        name: 'accuracy',
        description: 'Correctness of information',
        weight: 0.5,
        maxScore: 50
      },
      {
        name: 'clarity',
        description: 'Clear communication',
        weight: 0.3,
        maxScore: 30
      },
      {
        name: 'completeness',
        description: 'Thoroughness of answer',
        weight: 0.2,
        maxScore: 20
      }
    ],
    totalPoints: 100
  },
  provideFeedback: true,
  detailedAnalysis: true
};
```

23.5.7. Quiz Types

23.5.8. TypeScript

```typescript
type QuizId = string & { readonly __brand: 'QuizId' };
type QuestionId = string & { readonly __brand: 'QuestionId' };

type QuestionType = 'multiple_choice' | 'true_false' | 'short_answer' | 'essay';
type DifficultyLevel = 'easy' | 'medium' | 'hard';

interface GenerateQuizRequest {
  topic: string;
  numQuestions: number;
  questionTypes?: QuestionType[];
  difficulty?: DifficultyLevel;
  learningObjectives?: string[];
  excludeTopics?: string[];
}

interface QuizQuestion {
  id: QuestionId;
  type: QuestionType;
  question: string;
  options?: string[]; // For multiple choice
  correctAnswer: string | number;
  explanation: string;
  difficulty: DifficultyLevel;
  tags: string[];
  points: number;
}

interface Quiz {
  id: QuizId;
  title: string;
  description: string;
  questions: QuizQuestion[];
  totalPoints: number;
  estimatedTime: number; // in minutes
  createdAt: Date;
}
```

23.5.9. JavaScript

```javascript
// Example quiz generation request
const quizRequest = {
  topic: 'Cell Biology',
```

```
  numQuestions: 10,
  questionTypes: ['multiple_choice', 'short_answer'],
  difficulty: 'medium',
  learningObjectives: [
    'Understand cell structure',
    'Explain cellular processes'
  ],
  excludeTopics: ['advanced genetics']
};
```

23.5.10. Analytics Types

23.5.11. TypeScript

```
interface StudentAnalytics {
  studentId: StudentId;
  overallPerformance: PerformanceMetrics;
  subjectBreakdown: SubjectMetrics[];
  recentActivity: ActivityRecord[];
  strengths: string[];
  weaknesses: string[];
  recommendations: string[];
  lastUpdated: Date;
}

interface PerformanceMetrics {
  averageGrade: number;
  assignmentsCompleted: number;
  assignmentsTotal: number;
  completionRate: number;
  improvementTrend: 'improving' | 'stable' | 'declining';
  percentile: number;
}

interface SubjectMetrics {
  subject: string;
  averageGrade: number;
  assignmentsCount: number;
  masteryLevel: 'beginner' | 'intermediate' | 'advanced' | 'expert';
  timeSpent: number; // in minutes
}

interface ActivityRecord {
  timestamp: Date;
  type: 'assignment' | 'quiz' | 'study_session';
  itemId: string;
```

```
  score?: number;
  duration: number; // in minutes
}
```

23.5.12. JavaScript

```
// Example analytics response
const analytics = {
  studentId: 'student-123',
  overallPerformance: {
    averageGrade: 87.5,
    assignmentsCompleted: 24,
    assignmentsTotal: 30,
    completionRate: 0.8,
    improvementTrend: 'improving',
    percentile: 78
  },
  subjectBreakdown: [
    {
      subject: 'Biology',
      averageGrade: 92,
      assignmentsCount: 12,
      masteryLevel: 'advanced',
      timeSpent: 480
    }
  ],
  strengths: ['Scientific reasoning', 'Lab analysis'],
  weaknesses: ['Mathematical applications'],
  recommendations: ['Review algebra concepts', 'Practice graph interpretation']
};
```

23.6. API Methods

23.6.1. Assignments API

23.6.2. TypeScript

```
class AssignmentsAPI {
  /**
   * Grade a student assignment
   * @param request - Grading request parameters
   * @returns Promise resolving to grade result
   */
  async grade(request: GradeAssignmentRequest): Promise<GradeResult> {
```

```
    // Implementation
  }

  /**
   * Get assignment details
   * @param assignmentId - Assignment identifier
   * @returns Promise resolving to assignment details
   */
  async get(assignmentId: AssignmentId): Promise<Assignment> {
    // Implementation
  }

  /**
   * List assignments for a student
   * @param studentId - Student identifier
   * @param options - Optional filtering parameters
   * @returns Promise resolving to assignment list
   */
  async list(
    studentId: StudentId,
    options?: ListAssignmentsOptions
  ): Promise<Assignment[]> {
    // Implementation
  }

  /**
   * Provide additional feedback on an assignment
   * @param assignmentId - Assignment identifier
   * @param feedback - Additional feedback text
   * @returns Promise resolving to updated grade result
   */
  async addFeedback(
    assignmentId: AssignmentId,
    feedback: string
  ): Promise<GradeResult> {
    // Implementation
  }
}

// Example usage
const gradeResult = await kai.assignments.grade({
  studentId: 'student-123' as StudentId,
  assignmentId: 'assign-456' as AssignmentId,
  submission: 'The process of photosynthesis...',
  rubric: {
    criteria: [
      { name: 'accuracy', weight: 0.6, maxScore: 60 },
```

```
      { name: 'clarity', weight: 0.4, maxScore: 40 }
    ],
    totalPoints: 100
  },
  provideFeedback: true,
  detailedAnalysis: true
});

console.log(`Score: ${gradeResult.score}/${gradeResult.maxScore}`);
console.log(`Feedback: ${gradeResult.feedback}`);
```

23.6.3. JavaScript

```
// Grade an assignment
const gradeResult = await kai.assignments.grade({
  studentId: 'student-123',
  assignmentId: 'assign-456',
  submission: 'The process of photosynthesis...',
  rubric: {
    criteria: [
      { name: 'accuracy', weight: 0.6, maxScore: 60 },
      { name: 'clarity', weight: 0.4, maxScore: 40 }
    ],
    totalPoints: 100
  },
  provideFeedback: true,
  detailedAnalysis: true
});

// Get assignment details
const assignment = await kai.assignments.get('assign-456');

// List student assignments
const assignments = await kai.assignments.list('student-123', {
  status: 'completed',
  limit: 20,
  offset: 0
});

// Add additional feedback
const updated = await kai.assignments.addFeedback(
  'assign-456',
  'Great improvement in understanding the light-dependent reactions!'
);
```

23.6.4. Quizzes API

23.6.5. TypeScript

```typescript
class QuizzesAPI {
  /**
   * Generate a new quiz
   * @param request - Quiz generation parameters
   * @returns Promise resolving to generated quiz
   */
  async generate(request: GenerateQuizRequest): Promise<Quiz> {
    // Implementation
  }

  /**
   * Get quiz details
   * @param quizId - Quiz identifier
   * @returns Promise resolving to quiz details
   */
  async get(quizId: QuizId): Promise<Quiz> {
    // Implementation
  }

  /**
   * Submit quiz answers for grading
   * @param quizId - Quiz identifier
   * @param studentId - Student identifier
   * @param answers - Student's answers
   * @returns Promise resolving to quiz results
   */
  async submit(
    quizId: QuizId,
    studentId: StudentId,
    answers: QuizAnswers
  ): Promise<QuizResult> {
    // Implementation
  }

  /**
   * Regenerate specific questions in a quiz
   * @param quizId - Quiz identifier
   * @param questionIds - IDs of questions to regenerate
   * @returns Promise resolving to updated quiz
   */
  async regenerateQuestions(
    quizId: QuizId,
    questionIds: QuestionId[]
```

```
  ): Promise<Quiz> {
    // Implementation
  }
}

// Example usage
const quiz = await kai.quizzes.generate({
  topic: 'Cellular Respiration',
  numQuestions: 15,
  questionTypes: ['multiple_choice', 'short_answer'],
  difficulty: 'medium',
  learningObjectives: [
    'Explain the steps of cellular respiration',
    'Compare aerobic and anaerobic respiration'
  ]
});

console.log(`Generated quiz: ${quiz.title}`);
console.log(`Questions: ${quiz.questions.length}`);
console.log(`Total points: ${quiz.totalPoints}`);
```

23.6.6. JavaScript

```
// Generate a quiz
const quiz = await kai.quizzes.generate({
  topic: 'Cellular Respiration',
  numQuestions: 15,
  questionTypes: ['multiple_choice', 'short_answer'],
  difficulty: 'medium',
  learningObjectives: [
    'Explain the steps of cellular respiration',
    'Compare aerobic and anaerobic respiration'
  ]
});

// Get quiz details
const quizDetails = await kai.quizzes.get('quiz-789');

// Submit quiz answers
const result = await kai.quizzes.submit(
  'quiz-789',
  'student-123',
  {
    'question-1': 'A',
    'question-2': 'The mitochondria produces ATP...',
    'question-3': 'True'
```

```
  }
);

console.log(`Quiz score: ${result.score}/${result.totalPoints}`);

// Regenerate specific questions
const updatedQuiz = await kai.quizzes.regenerateQuestions(
  'quiz-789',
  ['question-5', 'question-8']
);
```

23.6.7. Analytics API

23.6.8. TypeScript

```
class AnalyticsAPI {
  /**
   * Get comprehensive analytics for a student
   * @param studentId - Student identifier
   * @param options - Optional time range and filters
   * @returns Promise resolving to student analytics
   */
  async getStudentAnalytics(
    studentId: StudentId,
    options?: AnalyticsOptions
  ): Promise<StudentAnalytics> {
    // Implementation
  }

  /**
   * Get class-wide analytics
   * @param classId - Class identifier
   * @param options - Optional filters
   * @returns Promise resolving to class analytics
   */
  async getClassAnalytics(
    classId: string,
    options?: AnalyticsOptions
  ): Promise<ClassAnalytics> {
    // Implementation
  }

  /**
   * Get performance trends over time
   * @param studentId - Student identifier
   * @param timeRange - Time range for trends
```

```
   * @returns Promise resolving to trend data
   */
  async getTrends(
    studentId: StudentId,
    timeRange: TimeRange
  ): Promise<TrendData> {
    // Implementation
  }

  /**
   * Get personalized recommendations
   * @param studentId - Student identifier
   * @returns Promise resolving to recommendations
   */
  async getRecommendations(
    studentId: StudentId
  ): Promise<Recommendation[]> {
    // Implementation
  }
}

interface AnalyticsOptions {
  startDate?: Date;
  endDate?: Date;
  subjects?: string[];
  includeDetails?: boolean;
}

interface TimeRange {
  start: Date;
  end: Date;
  granularity: 'day' | 'week' | 'month';
}

// Example usage
const analytics = await kai.analytics.getStudentAnalytics(
  'student-123' as StudentId,
  {
    startDate: new Date('2024-01-01'),
    endDate: new Date('2024-12-31'),
    subjects: ['Biology', 'Chemistry'],
    includeDetails: true
  }
);

console.log(`Average grade: ${analytics.overallPerformance.averageGrade}`);
console.log(`Completion rate: ${analytics.overallPerformance.completionRate * 100}%`);
```

```
console.log(`Strengths: ${analytics.strengths.join(', ')}`);
```

23.6.9. JavaScript

```
// Get student analytics
const analytics = await kai.analytics.getStudentAnalytics(
  'student-123',
  {
    startDate: new Date('2024-01-01'),
    endDate: new Date('2024-12-31'),
    subjects: ['Biology', 'Chemistry'],
    includeDetails: true
  }
);

// Get class analytics
const classAnalytics = await kai.analytics.getClassAnalytics('class-456', {
  includeDetails: true
});

// Get performance trends
const trends = await kai.analytics.getTrends('student-123', {
  start: new Date('2024-01-01'),
  end: new Date('2024-12-31'),
  granularity: 'week'
});

// Get personalized recommendations
const recommendations = await kai.analytics.getRecommendations('student-123');

console.log('Recommendations:');
recommendations.forEach(rec => {
  console.log(`- ${rec.title}: ${rec.description}`);
});
```

23.7. Error Handling

The SDK provides comprehensive error handling with typed error classes:

23.7.1. TypeScript

```typescript
import {
  KaiClient,
  KaiError,
  AuthenticationError,
  RateLimitError,
  ValidationError,
  NetworkError
} from '@kai-ai/sdk';

const kai = new KaiClient({ apiKey: process.env.KAI_API_KEY });

try {
  const result = await kai.assignments.grade({
    studentId: 'student-123' as StudentId,
    assignmentId: 'assign-456' as AssignmentId,
    submission: 'Student answer...',
    rubric: { criteria: [{ name: 'accuracy', weight: 1.0 }] }
  });

  console.log(`Grade: ${result.score}`);
} catch (error) {
  if (error instanceof AuthenticationError) {
    console.error('Invalid API key:', error.message);
    // Handle authentication failure
  } else if (error instanceof RateLimitError) {
    console.error('Rate limit exceeded:', error.message);
    console.log(`Retry after: ${error.retryAfter} seconds`);
    // Implement exponential backoff
  } else if (error instanceof ValidationError) {
    console.error('Invalid request:', error.message);
    console.log('Validation errors:', error.errors);
    // Show validation errors to user
  } else if (error instanceof NetworkError) {
    console.error('Network error:', error.message);
    // Implement retry logic
  } else if (error instanceof KaiError) {
    console.error('Kai API error:', error.message);
    console.log('Status code:', error.statusCode);
  } else {
    console.error('Unexpected error:', error);
  }
}
```

23.7.2. JavaScript

```
const { KaiClient } = require('@kai-ai/sdk');

const kai = new KaiClient({ apiKey: process.env.KAI_API_KEY });

try {
  const result = await kai.assignments.grade({
    studentId: 'student-123',
    assignmentId: 'assign-456',
    submission: 'Student answer...',
    rubric: { criteria: [{ name: 'accuracy', weight: 1.0 }] }
  });

  console.log(`Grade: ${result.score}`);
} catch (error) {
  // Check error type by name
  switch (error.name) {
    case 'AuthenticationError':
      console.error('Invalid API key:', error.message);
      break;
    case 'RateLimitError':
      console.error('Rate limit exceeded:', error.message);
      console.log(`Retry after: ${error.retryAfter} seconds`);
      break;
    case 'ValidationError':
      console.error('Invalid request:', error.message);
      console.log('Validation errors:', error.errors);
      break;
    case 'NetworkError':
      console.error('Network error:', error.message);
      break;
    default:
      console.error('Error:', error.message);
  }
}
```

Tip

Best Practice: Always implement proper error handling and retry logic for production applications. The SDK provides detailed error information to help diagnose issues quickly.

23.8. Rate Limiting and Best Practices

The Kai API implements rate limiting to ensure fair usage. The SDK handles rate limits automatically with built-in retry logic.

23.8.1. TypeScript

```typescript
import { KaiClient } from '@kai-ai/sdk';

// Configure rate limiting behavior
const kai = new KaiClient({
  apiKey: process.env.KAI_API_KEY,
  retries: 3, // Number of retry attempts
  retryDelay: 1000, // Initial retry delay in ms
  retryBackoff: 2, // Exponential backoff multiplier
  timeout: 30000 // Request timeout in ms
});

// The SDK automatically handles rate limits
// with exponential backoff retry logic
const results = await Promise.all(
  assignments.map(async (assignment) => {
    try {
      return await kai.assignments.grade(assignment);
    } catch (error) {
      if (error instanceof RateLimitError) {
        // Rate limit hit, SDK will auto-retry
        console.log(`Rate limited, retrying after ${error.retryAfter}s`);
      }
      throw error;
    }
  })
);
```

23.8.2. JavaScript

```javascript
const { KaiClient } = require('@kai-ai/sdk');

// Configure rate limiting behavior
const kai = new KaiClient({
  apiKey: process.env.KAI_API_KEY,
  retries: 3, // Number of retry attempts
  retryDelay: 1000, // Initial retry delay in ms
  retryBackoff: 2, // Exponential backoff multiplier
  timeout: 30000 // Request timeout in ms
});

// Batch processing with rate limit handling
async function gradeAssignments(assignments) {
  const results = [];
```

```
  for (const assignment of assignments) {
    try {
      const result = await kai.assignments.grade(assignment);
      results.push(result);
    } catch (error) {
      if (error.name === 'RateLimitError') {
        console.log(`Rate limited, waiting ${error.retryAfter}s`);
        await new Promise(resolve => setTimeout(resolve, error.retryAfter * 1000));
        // Retry the request
        const result = await kai.assignments.grade(assignment);
        results.push(result);
      } else {
        throw error;
      }
    }
  }

  return results;
}
```

23.8.3. Best Practices

> **Note**
>
> **Rate Limit Guidelines**:
>
> - Default limit: 60 requests per minute
> - Burst limit: 10 requests per second
> - Use batch operations when available
> - Implement exponential backoff for retries
> - Cache responses when appropriate
> - Monitor rate limit headers in responses

23.9. Configuration Options

Complete configuration reference for the SDK:

23.9.1. TypeScript

```
interface KaiClientConfig {
  /** API key for authentication (required) */
  apiKey: string;
```

```typescript
  /** Base URL for API requests (default: https://chi2api.com/v1) */
  baseUrl?: string;

  /** Request timeout in milliseconds (default: 30000) */
  timeout?: number;

  /** Number of retry attempts for failed requests (default: 3) */
  retries?: number;

  /** Initial delay for retry attempts in ms (default: 1000) */
  retryDelay?: number;

  /** Exponential backoff multiplier (default: 2) */
  retryBackoff?: number;

  /** Custom HTTP headers for all requests */
  headers?: Record<string, string>;

  /** Enable debug logging (default: false) */
  debug?: boolean;

  /** Custom HTTP agent for Node.js (default: default agent) */
  agent?: any;

  /** Maximum number of concurrent requests (default: 10) */
  maxConcurrent?: number;
}

// Example: Production configuration
const kai = new KaiClient({
  apiKey: process.env.KAI_API_KEY!,
  baseUrl: 'https://chi2api.com/v1',
  timeout: 60000, // 60 seconds for long-running operations
  retries: 5,
  retryDelay: 2000,
  retryBackoff: 1.5,
  headers: {
    'X-Application-ID': 'my-app',
    'X-Application-Version': '1.0.0'
  },
  debug: false,
  maxConcurrent: 5
});
```

23.9.2. JavaScript

```
// Example: Development configuration
const devKai = new KaiClient({
  apiKey: process.env.KAI_API_KEY,
  baseUrl: 'https://chi2api.com/v1',
  timeout: 30000,
  retries: 3,
  debug: true, // Enable debug logging
  headers: {
    'X-Environment': 'development'
  }
});

// Example: Production configuration with custom agent
const https = require('https');
const prodKai = new KaiClient({
  apiKey: process.env.KAI_API_KEY,
  baseUrl: 'https://chi2api.com/v1',
  timeout: 60000,
  retries: 5,
  agent: new https.Agent({
    keepAlive: true,
    maxSockets: 10
  }),
  maxConcurrent: 5
});
```

23.10. Complete Examples

23.10.1. Example 1: Grading Assignments with Detailed Feedback

23.10.2. TypeScript

```
import { KaiClient, type GradeAssignmentRequest, type StudentId, type AssignmentId } from '@kai-a

const kai = new KaiClient({
  apiKey: process.env.KAI_API_KEY!
});

async function gradeEssayAssignment(
  studentId: StudentId,
  assignmentId: AssignmentId,
  essayText: string
) {
```

```
const request: GradeAssignmentRequest = {
  studentId,
  assignmentId,
  submission: essayText,
  rubric: {
    criteria: [
      {
        name: 'thesis_statement',
        description: 'Clear and focused thesis statement',
        weight: 0.2,
        maxScore: 20
      },
      {
        name: 'evidence_support',
        description: 'Strong supporting evidence and examples',
        weight: 0.3,
        maxScore: 30
      },
      {
        name: 'organization',
        description: 'Logical organization and flow',
        weight: 0.2,
        maxScore: 20
      },
      {
        name: 'grammar_mechanics',
        description: 'Grammar, spelling, and mechanics',
        weight: 0.15,
        maxScore: 15
      },
      {
        name: 'conclusion',
        description: 'Effective conclusion',
        weight: 0.15,
        maxScore: 15
      }
    ],
    totalPoints: 100
  },
  provideFeedback: true,
  detailedAnalysis: true
};

try {
  const result = await kai.assignments.grade(request);

  console.log('=== GRADING RESULTS ===');
```

```
    console.log(`Overall Score: ${result.score}/${result.maxScore} (${result.percentage.toFixed(1
    console.log('\nGeneral Feedback:');
    console.log(result.feedback);

    console.log('\n=== CRITERIA BREAKDOWN ===');
    result.criteriaScores.forEach(criterion => {
      console.log(`\n${criterion.criterion}:`);
      console.log(`  Score: ${criterion.score}/${criterion.maxScore}`);
      console.log(`  Feedback: ${criterion.feedback}`);
    });

    if (result.suggestions.length > 0) {
      console.log('\n=== SUGGESTIONS FOR IMPROVEMENT ===');
      result.suggestions.forEach((suggestion, index) => {
        console.log(`${index + 1}. ${suggestion}`);
      });
    }

    return result;
  } catch (error) {
    console.error('Failed to grade assignment:', error);
    throw error;
  }
}

// Usage
const studentId = 'student-123' as StudentId;
const assignmentId = 'essay-001' as AssignmentId;
const essay = `
Photosynthesis is the process by which plants convert light energy into chemical energy.
This essay will explore the mechanisms and importance of photosynthesis in the ecosystem...
[Full essay text]
`;

gradeEssayAssignment(studentId, assignmentId, essay);
```

23.10.3. JavaScript

```
const { KaiClient } = require('@kai-ai/sdk');

const kai = new KaiClient({
  apiKey: process.env.KAI_API_KEY
});

async function gradeEssayAssignment(studentId, assignmentId, essayText) {
  const request = {
```

```
  studentId,
  assignmentId,
  submission: essayText,
  rubric: {
    criteria: [
      {
        name: 'thesis_statement',
        description: 'Clear and focused thesis statement',
        weight: 0.2,
        maxScore: 20
      },
      {
        name: 'evidence_support',
        description: 'Strong supporting evidence and examples',
        weight: 0.3,
        maxScore: 30
      },
      {
        name: 'organization',
        description: 'Logical organization and flow',
        weight: 0.2,
        maxScore: 20
      },
      {
        name: 'grammar_mechanics',
        description: 'Grammar, spelling, and mechanics',
        weight: 0.15,
        maxScore: 15
      },
      {
        name: 'conclusion',
        description: 'Effective conclusion',
        weight: 0.15,
        maxScore: 15
      }
    ],
    totalPoints: 100
  },
  provideFeedback: true,
  detailedAnalysis: true
};

try {
  const result = await kai.assignments.grade(request);

  console.log('=== GRADING RESULTS ===');
  console.log(`Overall Score: ${result.score}/${result.maxScore} (${result.percentage.toFixed(1
```

```
    console.log('\nGeneral Feedback:');
    console.log(result.feedback);

    console.log('\n=== CRITERIA BREAKDOWN ===');
    result.criteriaScores.forEach(criterion => {
      console.log(`\n${criterion.criterion}:`);
      console.log(`  Score: ${criterion.score}/${criterion.maxScore}`);
      console.log(`  Feedback: ${criterion.feedback}`);
    });

    if (result.suggestions.length > 0) {
      console.log('\n=== SUGGESTIONS FOR IMPROVEMENT ===');
      result.suggestions.forEach((suggestion, index) => {
        console.log(`${index + 1}. ${suggestion}`);
      });
    }

    return result;
  } catch (error) {
    console.error('Failed to grade assignment:', error);
    throw error;
  }
}

// Usage
const essay = `
Photosynthesis is the process by which plants convert light energy into chemical energy.
This essay will explore the mechanisms and importance of photosynthesis in the ecosystem...
[Full essay text]
`;

gradeEssayAssignment('student-123', 'essay-001', essay);
```

23.10.4. Example 2: Generating and Managing Quizzes

23.10.5. TypeScript

```
import { KaiClient, type Quiz, type QuizId, type StudentId } from '@kai-ai/sdk';

const kai = new KaiClient({
  apiKey: process.env.KAI_API_KEY!
});

async function createAndAdministerQuiz(topic: string, studentIds: StudentId[]) {
  try {
    // Generate quiz
```

```
    console.log(`Generating quiz on topic: ${topic}...`);
    const quiz = await kai.quizzes.generate({
      topic,
      numQuestions: 10,
      questionTypes: ['multiple_choice', 'short_answer'],
      difficulty: 'medium',
      learningObjectives: [
        `Understand key concepts in ${topic}`,
        `Apply knowledge to real-world scenarios`
      ]
    });

    console.log(`\nQuiz created: ${quiz.title}`);
    console.log(`Questions: ${quiz.questions.length}`);
    console.log(`Total points: ${quiz.totalPoints}`);
    console.log(`Estimated time: ${quiz.estimatedTime} minutes`);

    // Display quiz questions
    console.log('\n=== QUIZ QUESTIONS ===');
    quiz.questions.forEach((question, index) => {
      console.log(`\n${index + 1}. ${question.question}`);
      console.log(`   Type: ${question.type}`);
      console.log(`   Difficulty: ${question.difficulty}`);
      console.log(`   Points: ${question.points}`);

      if (question.options) {
        question.options.forEach((option, optIndex) => {
          console.log(`   ${String.fromCharCode(65 + optIndex)}) ${option}`);
        });
      }
    });

    // Simulate student taking quiz
    const studentResults = await Promise.all(
      studentIds.map(studentId => gradeStudentQuiz(quiz.id, studentId))
    );

    // Calculate class statistics
    const classAverage = studentResults.reduce((sum, r) => sum + r.percentage, 0) / studentResult
    console.log(`\n=== CLASS STATISTICS ===`);
    console.log(`Class average: ${classAverage.toFixed(1)}%`);
    console.log(`Students completed: ${studentResults.length}`);

    return { quiz, studentResults };
  } catch (error) {
    console.error('Failed to create and administer quiz:', error);
    throw error;
```

```
  }
}

async function gradeStudentQuiz(quizId: QuizId, studentId: StudentId) {
  // Simulate student answers (in real app, get from form submission)
  const answers = {
    'question-1': 'B',
    'question-2': 'Photosynthesis converts light energy to chemical energy through...',
    'question-3': 'A',
    // ... more answers
  };

  const result = await kai.quizzes.submit(quizId, studentId, answers);

  console.log(`\nStudent ${studentId}:`);
  console.log(`  Score: ${result.score}/${result.totalPoints} (${result.percentage.toFixed(1)}%)`
  console.log(`  Time taken: ${result.timeTaken} minutes`);

  return result;
}

// Usage
const studentIds: StudentId[] = [
  'student-001' as StudentId,
  'student-002' as StudentId,
  'student-003' as StudentId
];

createAndAdministerQuiz('Cellular Respiration', studentIds);
```

23.10.6. JavaScript

```
const { KaiClient } = require('@kai-ai/sdk');

const kai = new KaiClient({
  apiKey: process.env.KAI_API_KEY
});

async function createAndAdministerQuiz(topic, studentIds) {
  try {
    // Generate quiz
    console.log(`Generating quiz on topic: ${topic}...`);
    const quiz = await kai.quizzes.generate({
      topic,
      numQuestions: 10,
      questionTypes: ['multiple_choice', 'short_answer'],
```

```
      difficulty: 'medium',
      learningObjectives: [
        `Understand key concepts in ${topic}`,
        `Apply knowledge to real-world scenarios`
      ]
    });

    console.log(`\nQuiz created: ${quiz.title}`);
    console.log(`Questions: ${quiz.questions.length}`);
    console.log(`Total points: ${quiz.totalPoints}`);
    console.log(`Estimated time: ${quiz.estimatedTime} minutes`);

    // Display quiz questions
    console.log('\n=== QUIZ QUESTIONS ===');
    quiz.questions.forEach((question, index) => {
      console.log(`\n${index + 1}. ${question.question}`);
      console.log(`   Type: ${question.type}`);
      console.log(`   Difficulty: ${question.difficulty}`);
      console.log(`   Points: ${question.points}`);

      if (question.options) {
        question.options.forEach((option, optIndex) => {
          console.log(`   ${String.fromCharCode(65 + optIndex)}) ${option}`);
        });
      }
    });

    // Simulate student taking quiz
    const studentResults = await Promise.all(
      studentIds.map(studentId => gradeStudentQuiz(quiz.id, studentId))
    );

    // Calculate class statistics
    const classAverage = studentResults.reduce((sum, r) => sum + r.percentage, 0) / studentResult
    console.log(`\n=== CLASS STATISTICS ===`);
    console.log(`Class average: ${classAverage.toFixed(1)}%`);
    console.log(`Students completed: ${studentResults.length}`);

    return { quiz, studentResults };
  } catch (error) {
    console.error('Failed to create and administer quiz:', error);
    throw error;
  }
}

async function gradeStudentQuiz(quizId, studentId) {
  // Simulate student answers (in real app, get from form submission)
```

```
  const answers = {
    'question-1': 'B',
    'question-2': 'Photosynthesis converts light energy to chemical energy through...',
    'question-3': 'A',
    // ... more answers
  };

  const result = await kai.quizzes.submit(quizId, studentId, answers);

  console.log(`\nStudent ${studentId}:`);
  console.log(`  Score: ${result.score}/${result.totalPoints} (${result.percentage.toFixed(1)}%)`
  console.log(`  Time taken: ${result.timeTaken} minutes`);

  return result;
}

// Usage
const studentIds = ['student-001', 'student-002', 'student-003'];
createAndAdministerQuiz('Cellular Respiration', studentIds);
```

23.10.7. Example 3: Comprehensive Student Analytics Dashboard

23.10.8. TypeScript

```
import { KaiClient, type StudentId, type StudentAnalytics } from '@kai-ai/sdk';

const kai = new KaiClient({
  apiKey: process.env.KAI_API_KEY!
});

async function generateStudentDashboard(studentId: StudentId) {
  try {
    // Fetch comprehensive analytics
    const analytics = await kai.analytics.getStudentAnalytics(studentId, {
      startDate: new Date(new Date().setMonth(new Date().getMonth() - 3)),
      endDate: new Date(),
      includeDetails: true
    });

    // Fetch performance trends
    const trends = await kai.analytics.getTrends(studentId, {
      start: new Date(new Date().setMonth(new Date().getMonth() - 3)),
      end: new Date(),
      granularity: 'week'
    });
```

```
// Fetch personalized recommendations
const recommendations = await kai.analytics.getRecommendations(studentId);

// Display dashboard
console.log('                                    ');
console.log('          STUDENT PERFORMANCE DASHBOARD                    ');
console.log('                                    ');

console.log('\n OVERALL PERFORMANCE');
console.log(`   Average Grade: ${analytics.overallPerformance.averageGrade.toFixed(1)}`);
console.log(`   Completion Rate: ${(analytics.overallPerformance.completionRate * 100).toFixe
console.log(`   Assignments: ${analytics.overallPerformance.assignmentsCompleted}/${analytics
console.log(`   Trend: ${getTrendEmoji(analytics.overallPerformance.improvementTrend)} ${anal
console.log(`   Class Percentile: ${analytics.overallPerformance.percentile}th`);

console.log('\n SUBJECT BREAKDOWN');
analytics.subjectBreakdown.forEach(subject => {
  console.log(`\n   ${subject.subject}:`);
  console.log(`     Average: ${subject.averageGrade.toFixed(1)}`);
  console.log(`     Mastery: ${getMasteryEmoji(subject.masteryLevel)} ${subject.masteryLevel}
  console.log(`     Assignments: ${subject.assignmentsCount}`);
  console.log(`     Time Spent: ${subject.timeSpent} minutes`);
});

console.log('\n STRENGTHS');
analytics.strengths.forEach((strength, index) => {
  console.log(`   ${index + 1}. ${strength}`);
});

console.log('\n AREAS FOR IMPROVEMENT');
analytics.weaknesses.forEach((weakness, index) => {
  console.log(`   ${index + 1}. ${weakness}`);
});

console.log('\n PERSONALIZED RECOMMENDATIONS');
recommendations.forEach((rec, index) => {
  console.log(`\n   ${index + 1}. ${rec.title}`);
  console.log(`      ${rec.description}`);
  console.log(`      Priority: ${rec.priority}`);
  if (rec.estimatedTime) {
    console.log(`      Estimated Time: ${rec.estimatedTime} minutes`);
  }
});

console.log('\n RECENT TRENDS');
displayTrendChart(trends);
```

```
    return { analytics, trends, recommendations };
  } catch (error) {
    console.error('Failed to generate dashboard:', error);
    throw error;
  }
}

function getTrendEmoji(trend: string): string {
  const emojis: Record<string, string> = {
    'improving': ' ',
    'stable': ' ',
    'declining': ' '
  };
  return emojis[trend] || ' ';
}

function getMasteryEmoji(level: string): string {
  const emojis: Record<string, string> = {
    'beginner': ' ',
    'intermediate': ' ',
    'advanced': ' ',
    'expert': ' '
  };
  return emojis[level] || ' ';
}

function displayTrendChart(trends: any) {
  console.log('\n   Week    Grade    Progress');
  console.log('                    ');
  trends.dataPoints.forEach((point: any, index: number) => {
    const bar = ' '.repeat(Math.round(point.value / 10));
    console.log(`   Week ${index + 1}  ${point.value.toFixed(1)}    ${bar}`);
  });
}

// Usage
generateStudentDashboard('student-123' as StudentId);
```

23.10.9. JavaScript

```
const { KaiClient } = require('@kai-ai/sdk');

const kai = new KaiClient({
  apiKey: process.env.KAI_API_KEY
});
```

```
async function generateStudentDashboard(studentId) {
  try {
    // Fetch comprehensive analytics
    const analytics = await kai.analytics.getStudentAnalytics(studentId, {
      startDate: new Date(new Date().setMonth(new Date().getMonth() - 3)),
      endDate: new Date(),
      includeDetails: true
    });

    // Fetch performance trends
    const trends = await kai.analytics.getTrends(studentId, {
      start: new Date(new Date().setMonth(new Date().getMonth() - 3)),
      end: new Date(),
      granularity: 'week'
    });

    // Fetch personalized recommendations
    const recommendations = await kai.analytics.getRecommendations(studentId);

    // Display dashboard
    console.log('                                    ');
    console.log('          STUDENT PERFORMANCE DASHBOARD                  ');
    console.log('                                    ');

    console.log('\n OVERALL PERFORMANCE');
    console.log(`   Average Grade: ${analytics.overallPerformance.averageGrade.toFixed(1)}`);
    console.log(`   Completion Rate: ${(analytics.overallPerformance.completionRate * 100).toFixe
    console.log(`   Assignments: ${analytics.overallPerformance.assignmentsCompleted}/${analytics
    console.log(`   Trend: ${getTrendEmoji(analytics.overallPerformance.improvementTrend)} ${anal
    console.log(`   Class Percentile: ${analytics.overallPerformance.percentile}th`);

    console.log('\n SUBJECT BREAKDOWN');
    analytics.subjectBreakdown.forEach(subject => {
      console.log(`\n   ${subject.subject}:`);
      console.log(`     Average: ${subject.averageGrade.toFixed(1)}`);
      console.log(`     Mastery: ${getMasteryEmoji(subject.masteryLevel)} ${subject.masteryLevel}
      console.log(`     Assignments: ${subject.assignmentsCount}`);
      console.log(`     Time Spent: ${subject.timeSpent} minutes`);
    });

    console.log('\n STRENGTHS');
    analytics.strengths.forEach((strength, index) => {
      console.log(`   ${index + 1}. ${strength}`);
    });

    console.log('\n AREAS FOR IMPROVEMENT');
    analytics.weaknesses.forEach((weakness, index) => {
```

```
      console.log(`   ${index + 1}. ${weakness}`);
    });

    console.log('\n PERSONALIZED RECOMMENDATIONS');
    recommendations.forEach((rec, index) => {
      console.log(`\n   ${index + 1}. ${rec.title}`);
      console.log(`      ${rec.description}`);
      console.log(`      Priority: ${rec.priority}`);
      if (rec.estimatedTime) {
        console.log(`      Estimated Time: ${rec.estimatedTime} minutes`);
      }
    });

    console.log('\n RECENT TRENDS');
    displayTrendChart(trends);

    return { analytics, trends, recommendations };
  } catch (error) {
    console.error('Failed to generate dashboard:', error);
    throw error;
  }
}

function getTrendEmoji(trend) {
  const emojis = {
    'improving': ' ',
    'stable': ' ',
    'declining': ' '
  };
  return emojis[trend] || ' ';
}

function getMasteryEmoji(level) {
  const emojis = {
    'beginner': ' ',
    'intermediate': ' ',
    'advanced': ' ',
    'expert': ' '
  };
  return emojis[level] || ' ';
}

function displayTrendChart(trends) {
  console.log('\n   Week    Grade    Progress');
  console.log('                  ');
  trends.dataPoints.forEach((point, index) => {
    const bar = ' '.repeat(Math.round(point.value / 10));
```

```
    console.log(`   Week ${index + 1}  ${point.value.toFixed(1)}    ${bar}`);
  });
}

// Usage
generateStudentDashboard('student-123');
```

23.11. Node.js vs Browser Usage

The SDK works seamlessly in both Node.js and browser environments:

23.11.1. Node.js

```
// server.ts - Node.js backend
import { KaiClient } from '@kai-ai/sdk';
import express from 'express';

const app = express();
const kai = new KaiClient({
  apiKey: process.env.KAI_API_KEY!
});

app.post('/api/grade-assignment', async (req, res) => {
  try {
    const result = await kai.assignments.grade(req.body);
    res.json(result);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

app.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

23.11.2. Browser

```
// client.ts - Browser frontend
import { KaiClient } from '@kai-ai/sdk';

// In production, use a backend proxy to protect API key
// This example is for demonstration only
const kai = new KaiClient({
```

```
  apiKey: getApiKeyFromBackend() // Fetch from your backend
});

async function gradeAssignment(formData: FormData) {
  const submission = formData.get('submission') as string;

  try {
    const result = await kai.assignments.grade({
      studentId: getCurrentStudentId(),
      assignmentId: formData.get('assignmentId') as string,
      submission,
      rubric: JSON.parse(formData.get('rubric') as string)
    });

    // Update UI with results
    displayGradingResults(result);
  } catch (error) {
    showError(error.message);
  }
}
```

> **Warning**
>
> **Security Note**: Never expose your API key in client-side code. Always use a backend proxy to make API calls from the browser.

23.12. TypeScript Strict Typing Benefits

The SDK leverages TypeScript's advanced type system for maximum safety:

23.12.1. Branded Types

```
// Type-safe IDs prevent mixing different entity types
type StudentId = string & { readonly __brand: 'StudentId' };
type AssignmentId = string & { readonly __brand: 'AssignmentId' };
type QuizId = string & { readonly __brand: 'QuizId' };

// This will cause a compile-time error
const studentId: StudentId = 'student-123' as StudentId;
const assignmentId: AssignmentId = studentId; //  Error: Type mismatch

// Correct usage
const validStudentId = 'student-123' as StudentId;
const validAssignmentId = 'assign-456' as AssignmentId;
```

```
await kai.assignments.grade({
  studentId: validStudentId,
  assignmentId: validAssignmentId,
  // ... rest of parameters
});
```

23.12.2. Discriminated Unions

```
// Type-safe error handling with discriminated unions
type Result<T, E = Error> =
  | { ok: true; data: T }
  | { ok: false; error: E };

// Compiler ensures exhaustive checking
function handleResult<T>(result: Result<T>) {
  if (result.ok) {
    // TypeScript knows result.data exists here
    console.log(result.data);
  } else {
    // TypeScript knows result.error exists here
    console.error(result.error);
  }
}
```

23.12.3. Generic Constraints

```
// Type-safe API client with constrained generics
interface ApiEndpoints {
  '/assignments/grade': GradeResult;
  '/quizzes/generate': Quiz;
  '/analytics/student': StudentAnalytics;
}

async function apiCall<K extends keyof ApiEndpoints>(
  endpoint: K,
  body: unknown
): Promise<ApiEndpoints[K]> {
  // Implementation with full type safety
  // Return type is inferred based on endpoint
}

// Usage with type inference
const gradeResult = await apiCall('/assignments/grade', { /* ... */ });
// gradeResult is typed as GradeResult
```

23.13. Related Documentation

- API Reference - Complete REST API documentation
- Python SDK - Python SDK documentation
- Authentication Guide - Authentication and security
- Quick Start Guide - Getting started with Kai the AI
- Best Practices - Development best practices

23.14. Support and Resources

- **Documentation**: https://chi2labs.com/docs
- **API Status**: https://status.chi2labs.com
- **GitHub Issues**: https://github.com/chi2labs/kai-sdk-ts/issues
- **Community Forum**: https://community.chi2labs.com
- **Email Support**: support@chi2labs.com

Last updated: 2025-10-07

24. Java SDK

Official Java SDK for Kai the AI

24.1. Overview

The official Java SDK for Kai the AI provides a robust, enterprise-ready interface for integrating intelligent teaching assistance into your Java applications. Built with Java 11+ features, the SDK supports both synchronous and asynchronous operations, comprehensive error handling, thread safety, and follows modern Java best practices.

24.2. Installation

Add the SDK to your project using Maven or Gradle:

24.2.1. Maven

```
<dependency>
    <groupId>com.chi2labs</groupId>
    <artifactId>kai-sdk</artifactId>
    <version>1.0.0</version>
</dependency>
```

24.2.2. Gradle (Groovy)

```
dependencies {
    implementation 'com.chi2labs:kai-sdk:1.0.0'
}
```

24.2.3. Gradle (Kotlin DSL)

```
dependencies {
    implementation("com.chi2labs:kai-sdk:1.0.0")
}
```

24.2.4. Requirements

- Java 11 or higher
- Active Kai the AI API key
- Internet connection for API requests

24.3. Quick Start

Here's a minimal example to get you started:

```
import com.chi2labs.kai.KaiClient;
import com.chi2labs.kai.models.GradeResult;

public class QuickStart {
    public static void main(String[] args) {
        // Initialize the client
        KaiClient client = new KaiClient("your_api_key_here");

        // Grade a student submission
        GradeResult result = client.assignments().grade()
            .assignmentId("CS101-HW1")
            .studentSubmission("The sorting algorithm works by...")
            .gradingStyle("detailed")
            .execute();

        System.out.printf("Score: %d/100%n", result.getScore());
        System.out.printf("Feedback: %s%n", result.getFeedback());
    }
}
```

24.4. Authentication

24.4.1. API Key Setup

The SDK requires an API key for authentication. You can provide the key in several ways:

24.4.2. Environment Variable

```
export KAI_API_KEY="your_api_key_here"
```

```
import com.chi2labs.kai.KaiClient;

// Automatically reads from KAI_API_KEY environment variable
KaiClient client = new KaiClient();
```

24.4.3. Direct Initialization

```
import com.chi2labs.kai.KaiClient;

KaiClient client = new KaiClient("your_api_key_here");
```

24.4.4. Configuration Builder

```
import com.chi2labs.kai.KaiClient;
import com.chi2labs.kai.config.KaiConfig;

KaiConfig config = KaiConfig.builder()
    .apiKey("your_api_key_here")
    .baseUrl("https://chi2api.com/v1")
    .timeout(30000)
    .build();

KaiClient client = new KaiClient(config);
```

24.4.5. Properties File

```
# kai.properties
kai.api.key=your_api_key_here
kai.api.baseUrl=https://chi2api.com/v1
kai.api.timeout=30000
```

```
import com.chi2labs.kai.KaiClient;
import java.util.Properties;
import java.io.FileInputStream;

Properties props = new Properties();
props.load(new FileInputStream("kai.properties"));

KaiClient client = new KaiClient(props.getProperty("kai.api.key"));
```

> **Security Best Practice**
>
> Never hardcode API keys in your source code or commit them to version control. Use environment variables or secure configuration management systems like AWS Secrets Manager or HashiCorp Vault.

24.5. Core Classes

24.5.1. KaiClient

The main client class for interacting with Kai's API.

```
public class KaiClient implements AutoCloseable {
    /**
     * Create a new KaiClient with the specified API key.
     *
     * @param apiKey Your API key (or null to read from KAI_API_KEY env variable)
     */
    public KaiClient(String apiKey) { }

    /**
     * Create a new KaiClient with custom configuration.
     *
     * @param config Configuration object
     */
    public KaiClient(KaiConfig config) { }

    /**
     * Access assignment grading operations.
     *
     * @return AssignmentOperations instance
     */
    public AssignmentOperations assignments() { }

    /**
     * Access content generation operations.
     *
     * @return ContentOperations instance
     */
    public ContentOperations content() { }

    /**
     * Access student analytics operations.
     *
     * @return AnalyticsOperations instance
     */
    public AnalyticsOperations analytics() { }

    /**
     * Access quiz generation operations.
     *
     * @return QuizOperations instance
     */
```

```
    public QuizOperations quizzes() { }

    @Override
    public void close() { }
}
```

Usage with try-with-resources:

```
try (KaiClient client = new KaiClient()) {
    GradeResult result = client.assignments().grade()
        .assignmentId("CS101-HW1")
        .studentSubmission("Student work...")
        .execute();

    System.out.println("Score: " + result.getScore());
}
```

24.5.2. AssignmentOperations

Handle assignment grading and feedback operations.

```
public interface AssignmentOperations {
    /**
     * Create a new grade request builder.
     *
     * @return GradeRequestBuilder for fluent API
     */
    GradeRequestBuilder grade();

    /**
     * Retrieve a previously generated grade.
     *
     * @param gradeId The grade identifier
     * @return GradeResult object
     * @throws KaiException if grade not found or API error
     */
    GradeResult getGrade(String gradeId) throws KaiException;

    /**
     * Grade multiple submissions in a single request.
     *
     * @param submissions List of submission data
     * @return List of GradeResult objects
     * @throws KaiException if batch grading fails
     */
    List<GradeResult> batchGrade(List<SubmissionData> submissions)
```

```
        throws KaiException;
}
```

24.5.3. GradeRequestBuilder

Fluent API for building grade requests:

```
public interface GradeRequestBuilder {
    GradeRequestBuilder assignmentId(String assignmentId);
    GradeRequestBuilder studentSubmission(String submission);
    GradeRequestBuilder rubricId(String rubricId);
    GradeRequestBuilder gradingStyle(String style);
    GradeResult execute() throws KaiException;
    CompletableFuture<GradeResult> executeAsync();
}
```

24.5.4. QuizOperations

Generate and manage quiz content.

```
public interface QuizOperations {
    /**
     * Create a new quiz generation request builder.
     *
     * @return QuizRequestBuilder for fluent API
     */
    QuizRequestBuilder generate();

    /**
     * Retrieve a previously generated quiz.
     *
     * @param quizId The quiz identifier
     * @return QuizResult object
     * @throws KaiException if quiz not found or API error
     */
    QuizResult getQuiz(String quizId) throws KaiException;
}

public interface QuizRequestBuilder {
    QuizRequestBuilder topic(String topic);
    QuizRequestBuilder difficulty(String difficulty);
    QuizRequestBuilder questionCount(int count);
    QuizRequestBuilder questionTypes(String... types);
    QuizResult execute() throws KaiException;
    CompletableFuture<QuizResult> executeAsync();
}
```

24.5.5. AnalyticsOperations

Access student performance data and analytics.

```
public interface AnalyticsOperations {
    /**
     * Retrieve student performance metrics.
     *
     * @param studentId Student identifier
     * @return PerformanceData with metrics and trends
     * @throws KaiException if analytics retrieval fails
     */
    PerformanceData getStudentPerformance(String studentId)
        throws KaiException;

    /**
     * Retrieve student performance metrics with filters.
     *
     * @param request Performance request with filters
     * @return PerformanceData with metrics and trends
     * @throws KaiException if analytics retrieval fails
     */
    PerformanceData getStudentPerformance(PerformanceRequest request)
        throws KaiException;

    /**
     * Retrieve aggregate class-level analytics.
     *
     * @param courseId Course identifier
     * @return ClassAnalytics object
     * @throws KaiException if analytics retrieval fails
     */
    ClassAnalytics getClassAnalytics(String courseId)
        throws KaiException;
}
```

24.6. Response Models

24.6.1. GradeResult

```
public class GradeResult {
    private final String gradeId;
    private final int score;
    private final String feedback;
    private final List<String> suggestions;
```

```
    private final Instant timestamp;
    private final Map<String, Integer> rubricBreakdown;

    // Getters
    public String getGradeId() { return gradeId; }
    public int getScore() { return score; }
    public String getFeedback() { return feedback; }
    public List<String> getSuggestions() { return suggestions; }
    public Instant getTimestamp() { return timestamp; }
    public Map<String, Integer> getRubricBreakdown() { return rubricBreakdown; }

    /**
     * Check if grade meets passing threshold (>= 70).
     *
     * @return true if passing, false otherwise
     */
    public boolean isPassed() {
        return score >= 70;
    }

    @Override
    public String toString() {
        return String.format("GradeResult{gradeId='%s', score=%d, passed=%s}",
            gradeId, score, isPassed());
    }
}
```

24.6.2. QuizResult

```
public class QuizQuestion {
    private final String questionId;
    private final String questionText;
    private final String questionType;
    private final List<String> options;
    private final String correctAnswer;
    private final String explanation;

    // Getters omitted for brevity
}

public class QuizResult {
    private final String quizId;
    private final String topic;
    private final String difficulty;
    private final List<QuizQuestion> questions;
```

```
    private final int estimatedTimeMinutes;

    // Getters
    public String getQuizId() { return quizId; }
    public String getTopic() { return topic; }
    public String getDifficulty() { return difficulty; }
    public List<QuizQuestion> getQuestions() { return questions; }
    public int getEstimatedTimeMinutes() { return estimatedTimeMinutes; }

    /**
     * Get the total number of questions in the quiz.
     *
     * @return number of questions
     */
    public int getQuestionCount() {
        return questions.size();
    }
}
```

24.6.3. PerformanceData

```
public class PerformanceData {
    private final String studentId;
    private final double averageScore;
    private final int assignmentsCompleted;
    private final int assignmentsTotal;
    private final List<String> strengthAreas;
    private final List<String> improvementAreas;
    private final String trend;
    private final Instant lastUpdated;

    // Getters
    public String getStudentId() { return studentId; }
    public double getAverageScore() { return averageScore; }
    public int getAssignmentsCompleted() { return assignmentsCompleted; }
    public int getAssignmentsTotal() { return assignmentsTotal; }
    public List<String> getStrengthAreas() { return strengthAreas; }
    public List<String> getImprovementAreas() { return improvementAreas; }
    public String getTrend() { return trend; }
    public Instant getLastUpdated() { return lastUpdated; }

    /**
     * Calculate completion rate as a percentage.
     *
     * @return completion rate (0-100)
```

```
     */
    public double getCompletionRate() {
        return (double) assignmentsCompleted / assignmentsTotal * 100;
    }
}
```

24.7. Complete API Methods

24.7.1. Assignment Grading

24.7.2. Synchronous

```
import com.chi2labs.kai.KaiClient;
import com.chi2labs.kai.models.GradeResult;
import com.chi2labs.kai.exceptions.*;

public class GradingExample {
    public static void main(String[] args) {
        try (KaiClient client = new KaiClient()) {
            // Grade a single submission
            GradeResult result = client.assignments().grade()
                .assignmentId("CS101-HW1")
                .studentSubmission("My algorithm implementation...")
                .rubricId("standard_coding_rubric")
                .gradingStyle("detailed")
                .execute();

            System.out.printf("Grade ID: %s%n", result.getGradeId());
            System.out.printf("Score: %d/100%n", result.getScore());
            System.out.printf("Feedback: %s%n", result.getFeedback());

            if (result.getRubricBreakdown() != null) {
                System.out.println("\nRubric Breakdown:");
                result.getRubricBreakdown().forEach((criterion, points) ->
                    System.out.printf("  %s: %d%n", criterion, points)
                );
            }

            System.out.println("\nSuggestions:");
            result.getSuggestions().forEach(suggestion ->
                System.out.println("  - " + suggestion)
            );

        } catch (ValidationException e) {
            System.err.println("Invalid parameters: " + e.getMessage());
```

```
        } catch (RateLimitException e) {
            System.err.println("Rate limit exceeded: " + e.getMessage());
        } catch (KaiException e) {
            System.err.println("API error: " + e.getMessage());
        }
    }
}
```

24.7.3. Asynchronous

```
import com.chi2labs.kai.KaiClient;
import com.chi2labs.kai.models.GradeResult;
import java.util.concurrent.CompletableFuture;

public class AsyncGradingExample {
    public static void main(String[] args) {
        try (KaiClient client = new KaiClient()) {
            CompletableFuture<GradeResult> future = client.assignments().grade()
                .assignmentId("CS101-HW1")
                .studentSubmission("My algorithm implementation...")
                .gradingStyle("detailed")
                .executeAsync();

            // Handle result asynchronously
            future.thenAccept(result -> {
                System.out.printf("Score: %d/100%n", result.getScore());
                System.out.println("Feedback: " + result.getFeedback());
            }).exceptionally(ex -> {
                System.err.println("Grading failed: " + ex.getMessage());
                return null;
            });

            // Wait for completion (in real app, do other work)
            future.join();
        }
    }
}
```

24.7.4. CompletableFuture Composition

```
import com.chi2labs.kai.KaiClient;
import com.chi2labs.kai.models.GradeResult;
import java.util.concurrent.CompletableFuture;
import java.util.List;
```

```
public class ComposedAsyncExample {
    public static void main(String[] args) {
        try (KaiClient client = new KaiClient()) {
            List<String> submissions = List.of(
                "First submission...",
                "Second submission...",
                "Third submission..."
            );

            // Grade all submissions concurrently
            CompletableFuture<?>[] futures = submissions.stream()
                .map(submission -> client.assignments().grade()
                    .assignmentId("CS101-HW1")
                    .studentSubmission(submission)
                    .executeAsync()
                    .thenAccept(result ->
                        System.out.printf("Score: %d/100%n", result.getScore())
                    ))
                .toArray(CompletableFuture[]::new);

            // Wait for all to complete
            CompletableFuture.allOf(futures).join();
            System.out.println("All grading completed!");
        }
    }
}
```

24.7.5. Batch Grading

Grade multiple submissions efficiently:

```
import com.chi2labs.kai.KaiClient;
import com.chi2labs.kai.models.GradeResult;
import com.chi2labs.kai.models.SubmissionData;
import java.util.List;

public class BatchGradingExample {
    public static void main(String[] args) {
        try (KaiClient client = new KaiClient()) {
            List<SubmissionData> submissions = List.of(
                new SubmissionData("CS101-HW1", "student_001", "First student's work..."),
                new SubmissionData("CS101-HW1", "student_002", "Second student's work..."),
                new SubmissionData("CS101-HW1", "student_003", "Third student's work...")
            );

            List<GradeResult> results = client.assignments().batchGrade(submissions);
```

```
            results.forEach(result ->
                System.out.printf("Student: Score %d/100%n", result.getScore())
            );

            // Calculate class statistics
            double averageScore = results.stream()
                .mapToInt(GradeResult::getScore)
                .average()
                .orElse(0.0);

            System.out.printf("Class average: %.1f/100%n", averageScore);

        } catch (Exception e) {
            System.err.println("Batch grading failed: " + e.getMessage());
        }
    }
}
```

24.7.6. Quiz Generation

24.7.7. Basic Quiz

```
import com.chi2labs.kai.KaiClient;
import com.chi2labs.kai.models.QuizResult;
import com.chi2labs.kai.models.QuizQuestion;

public class QuizExample {
    public static void main(String[] args) {
        try (KaiClient client = new KaiClient()) {
            QuizResult quiz = client.quizzes().generate()
                .topic("Java Data Structures")
                .difficulty("medium")
                .questionCount(10)
                .questionTypes("multiple_choice", "true_false")
                .execute();

            System.out.printf("Quiz ID: %s%n", quiz.getQuizId());
            System.out.printf("Estimated time: %d minutes%n",
                quiz.getEstimatedTimeMinutes());
            System.out.printf("Questions: %d%n", quiz.getQuestionCount());

            int questionNum = 1;
            for (QuizQuestion question : quiz.getQuestions()) {
                System.out.printf("%nQuestion %d: %s%n",
                    questionNum++, question.getQuestionText());
```

```
                if (question.getOptions() != null) {
                    question.getOptions().forEach(opt ->
                        System.out.println("  " + opt)
                    );
                }
            }

        } catch (Exception e) {
            System.err.println("Quiz generation failed: " + e.getMessage());
        }
    }
}
```

24.7.8. Async Multiple Quizzes

```
import com.chi2labs.kai.KaiClient;
import com.chi2labs.kai.models.QuizResult;
import java.util.List;
import java.util.concurrent.CompletableFuture;

public class MultipleQuizzesExample {
    public static void main(String[] args) {
        try (KaiClient client = new KaiClient()) {
            List<String> topics = List.of(
                "Java Basics",
                "Object-Oriented Programming",
                "Data Structures"
            );

            // Generate multiple quizzes concurrently
            CompletableFuture<?>[] futures = topics.stream()
                .map(topic -> client.quizzes().generate()
                    .topic(topic)
                    .difficulty("medium")
                    .questionCount(5)
                    .executeAsync()
                    .thenAccept(quiz ->
                        System.out.printf("Generated quiz: %s (%d questions)%n",
                            quiz.getTopic(), quiz.getQuestionCount())
                    ))
                .toArray(CompletableFuture[]::new);

            CompletableFuture.allOf(futures).join();
            System.out.printf("Generated %d quizzes%n", topics.size());
```

```
        } catch (Exception e) {
            System.err.println("Quiz generation failed: " + e.getMessage());
        }
    }
}
```

24.7.9. Student Analytics

```
import com.chi2labs.kai.KaiClient;
import com.chi2labs.kai.models.PerformanceData;
import com.chi2labs.kai.models.PerformanceRequest;
import java.time.Instant;
import java.time.temporal.ChronoUnit;

public class AnalyticsExample {
    public static void main(String[] args) {
        try (KaiClient client = new KaiClient()) {
            // Get performance for the last 30 days
            Instant endDate = Instant.now();
            Instant startDate = endDate.minus(30, ChronoUnit.DAYS);

            PerformanceRequest request = new PerformanceRequest.Builder()
                .studentId("student_12345")
                .startDate(startDate)
                .endDate(endDate)
                .courseId("CS101")
                .build();

            PerformanceData performance = client.analytics()
                .getStudentPerformance(request);

            System.out.printf("Average Score: %.1f%n",
                performance.getAverageScore());
            System.out.printf("Completion Rate: %d/%d (%.1f%%)%n",
                performance.getAssignmentsCompleted(),
                performance.getAssignmentsTotal(),
                performance.getCompletionRate());
            System.out.printf("Trend: %s%n", performance.getTrend());

            System.out.println("\nStrengths:");
            performance.getStrengthAreas().forEach(area ->
                System.out.println("    " + area)
            );

            System.out.println("\nAreas for Improvement:");
```

```
            performance.getImprovementAreas().forEach(area ->
                System.out.println("  ! " + area)
            );

        } catch (Exception e) {
            System.err.println("Analytics retrieval failed: " + e.getMessage());
        }
    }
}
```

24.8. Error Handling

The SDK provides specific exception types for different error scenarios:

```
import com.chi2labs.kai.KaiClient;
import com.chi2labs.kai.models.GradeResult;
import com.chi2labs.kai.exceptions.*;

public class ErrorHandlingExample {
    public static void main(String[] args) {
        try (KaiClient client = new KaiClient()) {
            GradeResult result = client.assignments().grade()
                .assignmentId("CS101-HW1")
                .studentSubmission("Student work...")
                .gradingStyle("detailed")
                .execute();

            System.out.println("Score: " + result.getScore());

        } catch (ValidationException e) {
            // Handle invalid parameters
            System.err.println("Invalid input: " + e.getMessage());
            System.err.println("Details: " + e.getDetails());

        } catch (AuthenticationException e) {
            // Handle authentication failures
            System.err.println("Authentication failed: " + e.getMessage());
            System.err.println("Please check your API key");

        } catch (RateLimitException e) {
            // Handle rate limiting
            System.err.println("Rate limit exceeded: " + e.getMessage());
            System.err.printf("Retry after: %d seconds%n",
                e.getRetryAfterSeconds());

        } catch (TimeoutException e) {
```

```
            // Handle timeouts
            System.err.println("Request timed out: " + e.getMessage());
            System.err.println("Try increasing the timeout parameter");

        } catch (ApiException e) {
            // Handle general API errors
            System.err.println("API error: " + e.getMessage());
            System.err.println("Status code: " + e.getStatusCode());
            System.err.println("Error code: " + e.getErrorCode());

        } catch (KaiException e) {
            // Catch-all for SDK errors
            System.err.println("SDK error: " + e.getMessage());
        }
    }
}
```

24.8.1. Exception Hierarchy

```
KaiException (base)
   ValidationException        // Invalid parameters
   AuthenticationException    // Invalid API key
   RateLimitException        // Rate limit exceeded
   TimeoutException          // Request timeout
   ApiException              // General API error
```

> **Error Recovery Pattern**
>
> Implement exponential backoff for rate limit errors:

```java
import com.chi2labs.kai.exceptions.RateLimitException;

public GradeResult gradeWithRetry(KaiClient client, String assignmentId,
                                  String submission) throws KaiException {
    int maxRetries = 3;
    int retryDelay = 1000; // milliseconds

    for (int attempt = 0; attempt < maxRetries; attempt++) {
        try {
            return client.assignments().grade()
                .assignmentId(assignmentId)
                .studentSubmission(submission)
                .execute();

        } catch (RateLimitException e) {
            if (attempt < maxRetries - 1) {
                int waitTime = retryDelay * (int) Math.pow(2, attempt);
                System.out.printf("Rate limited, waiting %dms...%n", waitTime);
                Thread.sleep(waitTime);
            } else {
                throw e;
            }
        }
    }
    throw new KaiException("Max retries exceeded");
}
```

24.9. Thread Safety

The `KaiClient` is thread-safe and can be shared across multiple threads:

```java
import com.chi2labs.kai.KaiClient;
import java.util.concurrent.*;

public class ThreadSafetyExample {
    private static final KaiClient client = new KaiClient();

    public static void main(String[] args) throws InterruptedException {
        ExecutorService executor = Executors.newFixedThreadPool(10);

        // Submit multiple grading tasks concurrently
        for (int i = 0; i < 100; i++) {
            final int index = i;
            executor.submit(() -> {
                try {
```

```
                    var result = client.assignments().grade()
                        .assignmentId("CS101-HW" + index)
                        .studentSubmission("Submission " + index)
                        .execute();

                    System.out.printf("Thread %s: Score %d%n",
                        Thread.currentThread().getName(),
                        result.getScore());

                } catch (Exception e) {
                    System.err.println("Error in thread: " + e.getMessage());
                }
            });
        }

        executor.shutdown();
        executor.awaitTermination(5, TimeUnit.MINUTES);
        client.close();
    }
}
```

Connection Pooling

The SDK uses connection pooling by default for efficient resource usage. You can configure pool settings:

```
KaiConfig config = KaiConfig.builder()
    .apiKey("your_api_key")
    .maxConnections(50)
    .connectionTimeout(30000)
    .build();

KaiClient client = new KaiClient(config);
```

24.10. Configuration Options

24.10.1. Client Configuration

```
import com.chi2labs.kai.KaiClient;
import com.chi2labs.kai.config.KaiConfig;
import java.time.Duration;

// Method 1: Builder pattern
KaiConfig config = KaiConfig.builder()
```

```
    .apiKey("your_api_key")
    .baseUrl("https://chi2api.com/v1")
    .timeout(Duration.ofSeconds(30))
    .maxRetries(3)
    .maxConnections(20)
    .verifySSL(true)
    .userAgent("MyApp/1.0")
    .build();

KaiClient client = new KaiClient(config);

// Method 2: From properties file
KaiClient client = KaiClient.fromProperties("kai.properties");

// Method 3: From environment variables
KaiClient client = KaiClient.fromEnvironment();
```

24.10.2. Logging Configuration

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import com.chi2labs.kai.KaiClient;

public class LoggingExample {
    private static final Logger logger = LoggerFactory.getLogger(LoggingExample.class);

    public static void main(String[] args) {
        // The SDK uses SLF4J for logging
        // Configure your logging framework (Logback, Log4j2, etc.)

        try (KaiClient client = new KaiClient()) {
            logger.info("Initializing Kai client");

            var result = client.assignments().grade()
                .assignmentId("CS101-HW1")
                .studentSubmission("Student work...")
                .execute();

            logger.info("Grading completed: score={}", result.getScore());
        }
    }
}
```

Logback configuration (logback.xml):

```
<configuration>
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>

    <!-- Set SDK logging level -->
    <logger name="com.chi2labs.kai" level="DEBUG"/>

    <root level="INFO">
        <appender-ref ref="STDOUT"/>
    </root>
</configuration>
```

24.11. Spring Framework Integration

24.11.1. Spring Boot Configuration

```
import com.chi2labs.kai.KaiClient;
import com.chi2labs.kai.config.KaiConfig;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class KaiConfiguration {

    @Bean
    @ConfigurationProperties(prefix = "kai")
    public KaiConfig kaiConfig() {
        return KaiConfig.builder().build();
    }

    @Bean
    public KaiClient kaiClient(KaiConfig config) {
        return new KaiClient(config);
    }
}
```

application.yml:

```
kai:
  api-key: ${KAI_API_KEY}
```

```
  base-url: https://chi2api.com/v1
  timeout: 30000
  max-retries: 3
```

24.11.2. Spring Service Example

```
import com.chi2labs.kai.KaiClient;
import com.chi2labs.kai.models.GradeResult;
import com.chi2labs.kai.exceptions.KaiException;
import org.springframework.stereotype.Service;
import org.springframework.beans.factory.annotation.Autowired;

@Service
public class GradingService {

    private final KaiClient kaiClient;

    @Autowired
    public GradingService(KaiClient kaiClient) {
        this.kaiClient = kaiClient;
    }

    public GradeResult gradeAssignment(String assignmentId, String submission) {
        try {
            return kaiClient.assignments().grade()
                .assignmentId(assignmentId)
                .studentSubmission(submission)
                .gradingStyle("detailed")
                .execute();
        } catch (KaiException e) {
            throw new RuntimeException("Grading failed", e);
        }
    }

    public CompletableFuture<GradeResult> gradeAssignmentAsync(
            String assignmentId, String submission) {
        return kaiClient.assignments().grade()
            .assignmentId(assignmentId)
            .studentSubmission(submission)
            .executeAsync();
    }
}
```

24.11.3. Spring REST Controller

```
import com.chi2labs.kai.models.GradeResult;
import org.springframework.web.bind.annotation.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;

@RestController
@RequestMapping("/api/grading")
public class GradingController {

    private final GradingService gradingService;

    @Autowired
    public GradingController(GradingService gradingService) {
        this.gradingService = gradingService;
    }

    @PostMapping("/grade")
    public ResponseEntity<GradeResult> gradeSubmission(
            @RequestBody SubmissionRequest request) {
        GradeResult result = gradingService.gradeAssignment(
            request.getAssignmentId(),
            request.getSubmission()
        );
        return ResponseEntity.ok(result);
    }

    @PostMapping("/grade/async")
    public CompletableFuture<ResponseEntity<GradeResult>> gradeSubmissionAsync(
            @RequestBody SubmissionRequest request) {
        return gradingService.gradeAssignmentAsync(
            request.getAssignmentId(),
            request.getSubmission()
        ).thenApply(ResponseEntity::ok);
    }
}
```

24.12. Complete Working Examples

24.12.1. Example 1: Automated Grading Pipeline

```
import com.chi2labs.kai.KaiClient;
import com.chi2labs.kai.models.GradeResult;
```

```java
import com.chi2labs.kai.models.SubmissionData;
import com.chi2labs.kai.exceptions.KaiException;
import java.io.*;
import java.nio.file.*;
import java.util.*;
import java.util.stream.Collectors;

/**
 * Automated grading pipeline for processing student submissions from CSV.
 */
public class AutomatedGradingPipeline {

    private final KaiClient client;

    public AutomatedGradingPipeline(KaiClient client) {
        this.client = client;
    }

    public static class Submission {
        private final String studentId;
        private final String assignmentId;
        private final String submissionText;

        public Submission(String studentId, String assignmentId, String submissionText) {
            this.studentId = studentId;
            this.assignmentId = assignmentId;
            this.submissionText = submissionText;
        }

        // Getters
        public String getStudentId() { return studentId; }
        public String getAssignmentId() { return assignmentId; }
        public String getSubmissionText() { return submissionText; }
    }

    public static class GradingRecord {
        private final String studentId;
        private final Integer score;
        private final String feedback;
        private final String suggestions;

        public GradingRecord(String studentId, Integer score,
                             String feedback, String suggestions) {
            this.studentId = studentId;
            this.score = score;
            this.feedback = feedback;
            this.suggestions = suggestions;
```

```
        }

        // Getters omitted for brevity
    }

    public List<Submission> loadSubmissions(Path csvPath) throws IOException {
        List<Submission> submissions = new ArrayList<>();

        try (BufferedReader reader = Files.newBufferedReader(csvPath)) {
            String line = reader.readLine(); // Skip header
            while ((line = reader.readLine()) != null) {
                String[] parts = line.split(",", 3);
                if (parts.length == 3) {
                    submissions.add(new Submission(
                        parts[0].trim(),
                        parts[1].trim(),
                        parts[2].trim()
                    ));
                }
            }
        }

        return submissions;
    }

    public List<GradingRecord> gradeSubmissions(List<Submission> submissions) {
        List<GradingRecord> records = new ArrayList<>();

        for (Submission submission : submissions) {
            try {
                GradeResult result = client.assignments().grade()
                    .assignmentId(submission.getAssignmentId())
                    .studentSubmission(submission.getSubmissionText())
                    .gradingStyle("detailed")
                    .execute();

                records.add(new GradingRecord(
                    submission.getStudentId(),
                    result.getScore(),
                    result.getFeedback(),
                    String.join(", ", result.getSuggestions())
                ));

                System.out.printf("  Graded %s: %d/100%n",
                    submission.getStudentId(), result.getScore());

            } catch (KaiException e) {
```

```
                System.err.printf("  Failed to grade %s: %s%n",
                    submission.getStudentId(), e.getMessage());

                records.add(new GradingRecord(
                    submission.getStudentId(),
                    null,
                    "Error: " + e.getMessage(),
                    ""
                ));
            }
        }

        return records;
    }

    public void saveResults(List<GradingRecord> records, Path outputPath)
            throws IOException {
        try (PrintWriter writer = new PrintWriter(Files.newBufferedWriter(outputPath))) {
            writer.println("student_id,score,feedback,suggestions");

            for (GradingRecord record : records) {
                writer.printf("%s,%s,\"%s\",\"%s\"%n",
                    record.studentId,
                    record.score != null ? record.score : "",
                    record.feedback.replace("\"", "\"\""),
                    record.suggestions.replace("\"", "\"\"")
                );
            }
        }

        System.out.println("\n  Results saved to " + outputPath);
    }

    public static void main(String[] args) {
        try (KaiClient client = new KaiClient()) {
            AutomatedGradingPipeline pipeline = new AutomatedGradingPipeline(client);

            // Load submissions
            Path inputPath = Paths.get("submissions.csv");
            List<Submission> submissions = pipeline.loadSubmissions(inputPath);
            System.out.printf("Loaded %d submissions%n", submissions.size());

            // Grade all submissions
            List<GradingRecord> results = pipeline.gradeSubmissions(submissions);

            // Save results
            Path outputPath = Paths.get("grading_results.csv");
```

```
            pipeline.saveResults(results, outputPath);

            // Print summary
            long successful = results.stream()
                .filter(r -> r.score != null)
                .count();

            double avgScore = results.stream()
                .filter(r -> r.score != null)
                .mapToInt(r -> r.score)
                .average()
                .orElse(0.0);

            System.out.println("\n=== Summary ===");
            System.out.printf("Total submissions: %d%n", submissions.size());
            System.out.printf("Successfully graded: %d%n", successful);
            System.out.printf("Average score: %.1f/100%n", avgScore);

        } catch (Exception e) {
            System.err.println("Pipeline failed: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

24.12.2. Example 2: Quiz Generator with Export

```
import com.chi2labs.kai.KaiClient;
import com.chi2labs.kai.models.QuizResult;
import com.chi2labs.kai.models.QuizQuestion;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import java.io.*;
import java.nio.file.*;
import java.util.*;

/**
 * Generate quizzes for multiple topics and export to various formats.
 */
public class QuizGenerator {

    private final KaiClient client;
    private final Gson gson;

    public QuizGenerator(KaiClient client) {
```

```
        this.client = client;
        this.gson = new GsonBuilder().setPrettyPrinting().create();
    }

    public QuizResult generateQuiz(String topic, String difficulty, int count)
            throws Exception {
        QuizResult quiz = client.quizzes().generate()
            .topic(topic)
            .difficulty(difficulty)
            .questionCount(count)
            .questionTypes("multiple_choice", "true_false", "short_answer")
            .execute();

        System.out.printf("%n  Generated quiz: %s%n", topic);
        System.out.printf("  ID: %s%n", quiz.getQuizId());
        System.out.printf("  Questions: %d%n", quiz.getQuestionCount());
        System.out.printf("  Time: ~%d minutes%n", quiz.getEstimatedTimeMinutes());

        return quiz;
    }

    public void exportToJson(QuizResult quiz, String filename) throws IOException {
        Map<String, Object> quizData = new HashMap<>();
        quizData.put("quiz_id", quiz.getQuizId());
        quizData.put("topic", quiz.getTopic());
        quizData.put("difficulty", quiz.getDifficulty());
        quizData.put("estimated_time_minutes", quiz.getEstimatedTimeMinutes());

        List<Map<String, Object>> questions = new ArrayList<>();
        for (QuizQuestion q : quiz.getQuestions()) {
            Map<String, Object> questionData = new HashMap<>();
            questionData.put("id", q.getQuestionId());
            questionData.put("text", q.getQuestionText());
            questionData.put("type", q.getQuestionType());
            questionData.put("options", q.getOptions());
            questionData.put("correct_answer", q.getCorrectAnswer());
            questionData.put("explanation", q.getExplanation());
            questions.add(questionData);
        }
        quizData.put("questions", questions);

        String json = gson.toJson(quizData);
        Files.writeString(Paths.get(filename), json);
        System.out.println("  Exported to " + filename);
    }

    public void exportToMarkdown(QuizResult quiz, String filename) throws IOException {
```

```
        StringBuilder md = new StringBuilder();
        md.append("# ").append(quiz.getTopic()).append(" Quiz\n\n");
        md.append("**Difficulty:** ").append(quiz.getDifficulty()).append("\n");
        md.append("**Estimated Time:** ").append(quiz.getEstimatedTimeMinutes())
          .append(" minutes\n\n");
        md.append("---\n\n");

        int i = 1;
        for (QuizQuestion q : quiz.getQuestions()) {
            md.append("## Question ").append(i++).append("\n\n");
            md.append(q.getQuestionText()).append("\n\n");

            if (q.getOptions() != null) {
                for (String opt : q.getOptions()) {
                    md.append("- ").append(opt).append("\n");
                }
                md.append("\n");
            }

            if (q.getExplanation() != null) {
                md.append("**Explanation:** ").append(q.getExplanation()).append("\n\n");
            }

            md.append("---\n\n");
        }

        Files.writeString(Paths.get(filename), md.toString());
        System.out.println("  Exported to " + filename);
    }

    public static void main(String[] args) {
        try (KaiClient client = new KaiClient()) {
            QuizGenerator generator = new QuizGenerator(client);

            // Topics to generate quizzes for
            List<TopicConfig> topics = List.of(
                new TopicConfig("Java Fundamentals", "easy", 15),
                new TopicConfig("Data Structures", "medium", 20),
                new TopicConfig("Algorithm Design", "hard", 10)
            );

            for (TopicConfig config : topics) {
                // Generate quiz
                QuizResult quiz = generator.generateQuiz(
                    config.topic,
                    config.difficulty,
                    config.count
```

```
                );

                // Export to different formats
                String baseName = config.topic.toLowerCase().replace(" ", "_");
                generator.exportToJson(quiz, baseName + "_quiz.json");
                generator.exportToMarkdown(quiz, baseName + "_quiz.md");
            }

            System.out.println("\n All quizzes generated and exported successfully!");

        } catch (Exception e) {
            System.err.println("Quiz generation failed: " + e.getMessage());
            e.printStackTrace();
        }
    }

    private static class TopicConfig {
        final String topic;
        final String difficulty;
        final int count;

        TopicConfig(String topic, String difficulty, int count) {
            this.topic = topic;
            this.difficulty = difficulty;
            this.count = count;
        }
    }
}
```

24.13. Best Practices

Performance Optimization Tips

1. **Reuse KaiClient Instance**: Create once and share across your application

```
// Good: Single instance
private static final KaiClient client = new KaiClient();

// Bad: Creating new instances
new KaiClient().assignments().grade()...
```

2. **Use Batch Operations**: Grade multiple submissions in one request

```
List<GradeResult> results = client.assignments().batchGrade(submissions);
```

3. **Enable Async for Concurrent Operations**: Use `executeAsync()` for parallel requests

```
CompletableFuture<?>[] futures = submissions.stream()
    .map(s -> client.assignments().grade()
        .assignmentId(s.getId())
        .studentSubmission(s.getText())
        .executeAsync())
    .toArray(CompletableFuture[]::new);

CompletableFuture.allOf(futures).join();
```

4. **Implement Caching**: Cache frequently accessed data

```
private final LoadingCache<String, RubricData> rubricCache =
    CacheBuilder.newBuilder()
        .maximumSize(100)
        .expireAfterWrite(1, TimeUnit.HOURS)
        .build(new CacheLoader<String, RubricData>() {
            public RubricData load(String rubricId) {
                return client.rubrics().get(rubricId);
            }
        });
```

5. **Monitor Rate Limits**: Check response headers

```
GradeResult result = client.assignments().grade()...execute();
int remaining = client.getLastResponse()
    .header("X-RateLimit-Remaining")
    .map(Integer::parseInt)
    .orElse(0);
```

6. **Use Try-With-Resources**: Ensure proper cleanup

```
try (KaiClient client = new KaiClient()) {
    // Use client
} // Automatically closed
```

24.14. Links and Resources

24.14.1. Related Documentation

- API Reference - Complete REST API documentation
- Getting Started - Initial setup and configuration
- Guides - Integration guides and tutorials

24.14.2. SDK Resources

- GitHub Repository - Source code and issues
- Maven Central - Package distribution
- Javadoc - API documentation
- Changelog - Version history
- Examples - More code examples

24.14.3. Support

- API Status: status.kaitheai.com
- Developer Forum: forum.kaitheai.com
- Email: developers@chi2labs.com

> **Stay Updated**
>
> Star the GitHub repository to receive notifications about new releases and features.

25. Ruby SDK

Official Ruby SDK for Kai the AI

25.1. Overview

The official Ruby SDK for Kai the AI provides an elegant, idiomatic Ruby interface for integrating intelligent teaching assistance into your Ruby and Rails applications. Built with Ruby 2.7+ features, the SDK embraces Ruby conventions with blocks, symbols, and expressive DSLs while supporting both synchronous and asynchronous operations.

25.2. Installation

Add the SDK to your project using Bundler:

25.2.1. Gemfile

```
# Gemfile
gem 'kai-sdk', '~> 1.0'
```

Then install:

```
bundle install
```

25.2.2. Manual Installation

```
gem install kai-sdk
```

25.2.3. Rails Application

For Rails applications, add to your Gemfile and create an initializer:

```ruby
# Gemfile
gem 'kai-sdk', '~> 1.0'

# config/initializers/kai.rb
Kai.configure do |config|
  config.api_key = ENV['KAI_API_KEY']
  config.timeout = 30
  config.max_retries = 3
end
```

25.2.4. Requirements

- Ruby 2.7 or higher
- Active Kai the AI API key
- Internet connection for API requests

25.3. Quick Start

Here's a minimal example to get you started:

```ruby
require 'kai-sdk'

# Initialize the client
client = Kai::Client.new(api_key: 'your_api_key_here')

# Grade a student submission
result = client.assignments.grade(
  assignment_id: 'CS101-HW1',
  student_submission: 'The sorting algorithm works by...',
  grading_style: 'detailed'
)

puts "Score: #{result.score}/100"
puts "Feedback: #{result.feedback}"
```

25.4. Authentication

25.4.1. API Key Setup

The SDK requires an API key for authentication. You can provide the key in several ways:

25.4.2. Environment Variable

```
export KAI_API_KEY="your_api_key_here"
```

```ruby
require 'kai-sdk'

# Automatically reads from KAI_API_KEY environment variable
client = Kai::Client.new
```

25.4.3. Direct Initialization

```ruby
require 'kai-sdk'

client = Kai::Client.new(api_key: 'your_api_key_here')
```

25.4.4. Configuration Block

```ruby
require 'kai-sdk'

Kai.configure do |config|
  config.api_key = 'your_api_key_here'
  config.base_url = 'https://chi2api.com/v1'
  config.timeout = 30
end

client = Kai::Client.new
```

25.4.5. Rails Credentials

```ruby
# config/initializers/kai.rb
Kai.configure do |config|
  config.api_key = Rails.application.credentials.dig(:kai, :api_key)
end
```

> **Security Best Practice**
>
> Never hardcode API keys in your source code or commit them to version control. Use environment variables, Rails credentials, or secure secret management systems.

25.5. Core Classes

25.5.1. Kai::Client

The main client class for interacting with Kai's API.

```ruby
module Kai
  class Client
    # Create a new client instance
    #
    # @param api_key [String] Your API key (defaults to ENV['KAI_API_KEY'])
    # @param base_url [String] API base URL
    # @param timeout [Integer] Request timeout in seconds
    # @param max_retries [Integer] Maximum retry attempts
    def initialize(api_key: nil, base_url: nil, timeout: 30, max_retries: 3)
      # ...
    end

    # Access assignment grading operations
    # @return [Kai::Assignments]
    def assignments
      @assignments ||= Kai::Assignments.new(self)
    end

    # Access content generation operations
    # @return [Kai::Content]
    def content
      @content ||= Kai::Content.new(self)
    end

    # Access student analytics operations
    # @return [Kai::Analytics]
    def analytics
      @analytics ||= Kai::Analytics.new(self)
    end

    # Access quiz generation operations
    # @return [Kai::Quizzes]
    def quizzes
      @quizzes ||= Kai::Quizzes.new(self)
    end
  end
end
```

25.5.2. Kai::Assignments

Handle assignment grading and feedback operations.

```ruby
module Kai
  class Assignments
    # Grade a student submission
    #
    # @param assignment_id [String] Assignment identifier
    # @param student_submission [String] Student's work
    # @param rubric_id [String, nil] Optional rubric to apply
    # @param grading_style [String] "detailed", "concise", or "points_only"
    # @return [Kai::GradeResult]
    # @raise [Kai::ValidationError] Invalid parameters
    # @raise [Kai::RateLimitError] Rate limit exceeded
    # @raise [Kai::APIError] API request failed
    def grade(assignment_id:, student_submission:, rubric_id: nil, grading_style: 'detailed')
      # ...
    end

    # Grade with a block for additional configuration
    #
    # @yield [request] Configuration block
    # @return [Kai::GradeResult]
    def grade_with(&block)
      request = GradeRequest.new
      block.call(request)
      execute_grade(request)
    end

    # Retrieve a previously generated grade
    #
    # @param grade_id [String] Grade identifier
    # @return [Kai::GradeResult]
    def get_grade(grade_id)
      # ...
    end

    # Grade multiple submissions in batch
    #
    # @param submissions [Array<Hash>] Array of submission data
    # @return [Array<Kai::GradeResult>]
    def batch_grade(submissions)
      # ...
    end
  end
end
```

25.5.3. Kai::Quizzes

Generate and manage quiz content.

```ruby
module Kai
  class Quizzes
    # Generate quiz questions
    #
    # @param topic [String] Subject matter for quiz
    # @param difficulty [String] "easy", "medium", or "hard"
    # @param question_count [Integer] Number of questions (1-50)
    # @param question_types [Array<String>] Types of questions to include
    # @return [Kai::QuizResult]
    def generate(topic:, difficulty: 'medium', question_count: 10, question_types: nil)
      # ...
    end

    # Generate quiz with configuration block
    #
    # @yield [request] Configuration block
    # @return [Kai::QuizResult]
    def generate_with(&block)
      request = QuizRequest.new
      block.call(request)
      execute_generate(request)
    end

    # Retrieve a previously generated quiz
    #
    # @param quiz_id [String] Quiz identifier
    # @return [Kai::QuizResult]
    def get_quiz(quiz_id)
      # ...
    end
  end
end
```

25.5.4. Kai::Analytics

Access student performance data and analytics.

```ruby
module Kai
  class Analytics
    # Retrieve student performance metrics
    #
    # @param student_id [String] Student identifier
    # @param start_date [String, Date, Time] Start date filter (ISO 8601)
    # @param end_date [String, Date, Time] End date filter (ISO 8601)
    # @param course_id [String, nil] Optional course filter
    # @return [Kai::PerformanceData]
    def student_performance(student_id:, start_date: nil, end_date: nil, course_id: nil)
```

```
      # ...
    end

    # Retrieve class-level analytics
    #
    # @param course_id [String] Course identifier
    # @param metric_types [Array<String>, nil] Specific metrics to retrieve
    # @return [Kai::ClassAnalytics]
    def class_analytics(course_id:, metric_types: nil)
      # ...
    end
  end
end
```

25.6. Response Models

25.6.1. Kai::GradeResult

```
module Kai
  class GradeResult
    attr_reader :grade_id, :score, :feedback, :suggestions,
                :timestamp, :rubric_breakdown

    # @param attributes [Hash] Grade result attributes
    def initialize(attributes = {})
      @grade_id = attributes[:grade_id]
      @score = attributes[:score]
      @feedback = attributes[:feedback]
      @suggestions = attributes[:suggestions] || []
      @timestamp = parse_time(attributes[:timestamp])
      @rubric_breakdown = attributes[:rubric_breakdown]
    end

    # Check if grade meets passing threshold (>= 70)
    # @return [Boolean]
    def passed?
      score >= 70
    end

    # Get letter grade based on score
    # @return [String]
    def letter_grade
      case score
      when 90..100 then 'A'
      when 80..89 then 'B'
```

```
      when 70..79 then 'C'
      when 60..69 then 'D'
      else 'F'
      end
    end

    def to_h
      {
        grade_id: grade_id,
        score: score,
        feedback: feedback,
        suggestions: suggestions,
        timestamp: timestamp,
        rubric_breakdown: rubric_breakdown
      }
    end
  end
end
```

25.6.2. Kai::QuizResult

```
module Kai
  class QuizQuestion
    attr_reader :question_id, :question_text, :question_type,
                :options, :correct_answer, :explanation

    def initialize(attributes = {})
      @question_id = attributes[:question_id]
      @question_text = attributes[:question_text]
      @question_type = attributes[:question_type]
      @options = attributes[:options]
      @correct_answer = attributes[:correct_answer]
      @explanation = attributes[:explanation]
    end

    def multiple_choice?
      question_type == 'multiple_choice'
    end

    def true_false?
      question_type == 'true_false'
    end
  end

  class QuizResult
    attr_reader :quiz_id, :topic, :difficulty, :questions,
```

```
                :estimated_time_minutes

    def initialize(attributes = {})
      @quiz_id = attributes[:quiz_id]
      @topic = attributes[:topic]
      @difficulty = attributes[:difficulty]
      @questions = parse_questions(attributes[:questions])
      @estimated_time_minutes = attributes[:estimated_time_minutes]
    end

    # Get total number of questions
    # @return [Integer]
    def question_count
      questions.size
    end

    # Iterate over questions
    # @yield [question] Each question
    def each_question(&block)
      questions.each(&block)
    end

    private

    def parse_questions(questions_data)
      (questions_data || []).map { |q| QuizQuestion.new(q) }
    end
  end
end
```

25.6.3. Kai::PerformanceData

```
module Kai
  class PerformanceData
    attr_reader :student_id, :average_score, :assignments_completed,
                :assignments_total, :strength_areas, :improvement_areas,
                :trend, :last_updated

    def initialize(attributes = {})
      @student_id = attributes[:student_id]
      @average_score = attributes[:average_score].to_f
      @assignments_completed = attributes[:assignments_completed]
      @assignments_total = attributes[:assignments_total]
      @strength_areas = attributes[:strength_areas] || []
      @improvement_areas = attributes[:improvement_areas] || []
      @trend = attributes[:trend]
```

```
      @last_updated = parse_time(attributes[:last_updated])
    end

    # Calculate completion rate as percentage
    # @return [Float] Completion rate (0-100)
    def completion_rate
      return 0.0 if assignments_total.zero?
      (assignments_completed.to_f / assignments_total * 100).round(1)
    end

    # Check if student is improving
    # @return [Boolean]
    def improving?
      trend == 'improving'
    end

    # Check if student is declining
    # @return [Boolean]
    def declining?
      trend == 'declining'
    end
  end
end
```

25.7. Complete API Methods

25.7.1. Assignment Grading

25.7.2. Basic Usage

```
require 'kai-sdk'

client = Kai::Client.new

# Grade a single submission
result = client.assignments.grade(
  assignment_id: 'CS101-HW1',
  student_submission: 'My algorithm implementation...',
  rubric_id: 'standard_coding_rubric',
  grading_style: 'detailed'
)

puts "Grade ID: #{result.grade_id}"
puts "Score: #{result.score}/100"
puts "Letter Grade: #{result.letter_grade}"
```

```ruby
puts "Feedback: #{result.feedback}"

if result.rubric_breakdown
  puts "\nRubric Breakdown:"
  result.rubric_breakdown.each do |criterion, points|
    puts "  #{criterion}: #{points}"
  end
end

puts "\nSuggestions:"
result.suggestions.each do |suggestion|
  puts "  - #{suggestion}"
end
```

25.7.3. Block Syntax

```ruby
require 'kai-sdk'

client = Kai::Client.new

# Use block for cleaner configuration
result = client.assignments.grade_with do |req|
  req.assignment_id = 'CS101-HW1'
  req.student_submission = 'My algorithm implementation...'
  req.rubric_id = 'standard_coding_rubric'
  req.grading_style = 'detailed'
end

puts "Score: #{result.score}/100"
puts "Passed: #{result.passed? ? 'Yes' : 'No'}"
```

25.7.4. Error Handling

```ruby
require 'kai-sdk'

client = Kai::Client.new

begin
  result = client.assignments.grade(
    assignment_id: 'CS101-HW1',
    student_submission: 'Student work...',
    grading_style: 'detailed'
  )

  puts "Score: #{result.score}/100"
```

```ruby
rescue Kai::ValidationError => e
  puts "Invalid parameters: #{e.message}"
  puts "Details: #{e.details}"

rescue Kai::AuthenticationError => e
  puts "Authentication failed: #{e.message}"
  puts "Please check your API key"

rescue Kai::RateLimitError => e
  puts "Rate limit exceeded: #{e.message}"
  puts "Retry after: #{e.retry_after} seconds"

rescue Kai::TimeoutError => e
  puts "Request timed out: #{e.message}"

rescue Kai::APIError => e
  puts "API error: #{e.message}"
  puts "Status code: #{e.status_code}"

rescue Kai::Error => e
  puts "SDK error: #{e.message}"
end
```

25.7.5. Batch Grading

Grade multiple submissions efficiently:

```ruby
require 'kai-sdk'

client = Kai::Client.new

submissions = [
  {
    assignment_id: 'CS101-HW1',
    student_id: 'student_001',
    student_submission: "First student's work..."
  },
  {
    assignment_id: 'CS101-HW1',
    student_id: 'student_002',
    student_submission: "Second student's work..."
  },
  {
    assignment_id: 'CS101-HW1',
    student_id: 'student_003',
    student_submission: "Third student's work..."
```

```
  }
]

results = client.assignments.batch_grade(submissions)

results.each do |result|
  puts "Student: Score #{result.score}/100"
end

# Calculate class statistics
average_score = results.sum(&:score) / results.size.to_f
puts "Class average: #{average_score.round(1)}/100"

# Filter passing students
passing = results.select(&:passed?)
puts "Passing students: #{passing.size}/#{results.size}"
```

25.7.6. Quiz Generation

25.7.7. Basic Quiz

```
require 'kai-sdk'

client = Kai::Client.new

quiz = client.quizzes.generate(
  topic: 'Ruby Fundamentals',
  difficulty: 'medium',
  question_count: 10,
  question_types: ['multiple_choice', 'true_false']
)

puts "Quiz ID: #{quiz.quiz_id}"
puts "Topic: #{quiz.topic}"
puts "Estimated time: #{quiz.estimated_time_minutes} minutes"
puts "Questions: #{quiz.question_count}"

quiz.each_question.with_index(1) do |question, i|
  puts "\nQuestion #{i}: #{question.question_text}"

  if question.options
    question.options.each do |opt|
      puts "  #{opt}"
    end
  end
end
```

25.7.8. Block Configuration

```ruby
require 'kai-sdk'

client = Kai::Client.new

quiz = client.quizzes.generate_with do |req|
  req.topic = 'Ruby on Rails'
  req.difficulty = 'hard'
  req.question_count = 15
  req.question_types = ['multiple_choice', 'short_answer']
end

# Export quiz to hash
quiz_data = {
  id: quiz.quiz_id,
  topic: quiz.topic,
  difficulty: quiz.difficulty,
  time: quiz.estimated_time_minutes,
  questions: quiz.questions.map(&:to_h)
}

# Save as JSON
File.write('quiz.json', JSON.pretty_generate(quiz_data))
puts "Quiz saved to quiz.json"
```

25.7.9. Multiple Topics

```ruby
require 'kai-sdk'

client = Kai::Client.new

topics = [
  { topic: 'Ruby Basics', difficulty: 'easy', count: 15 },
  { topic: 'Object-Oriented Programming', difficulty: 'medium', count: 20 },
  { topic: 'Metaprogramming', difficulty: 'hard', count: 10 }
]

quizzes = topics.map do |config|
  client.quizzes.generate(
    topic: config[:topic],
    difficulty: config[:difficulty],
    question_count: config[:count]
  )
end
```

```
quizzes.each do |quiz|
  puts "Generated: #{quiz.topic} (#{quiz.question_count} questions)"
end
```

25.7.10. Student Analytics

```
require 'kai-sdk'
require 'date'

client = Kai::Client.new

# Get performance for the last 30 days
end_date = Date.today
start_date = end_date - 30

performance = client.analytics.student_performance(
  student_id: 'student_12345',
  start_date: start_date.iso8601,
  end_date: end_date.iso8601,
  course_id: 'CS101'
)

puts "Average Score: #{performance.average_score.round(1)}"
puts "Completion Rate: #{performance.assignments_completed}/#{performance.assignments_total} (#{p
puts "Trend: #{performance.trend}"

puts "\nStrengths:"
performance.strength_areas.each do |area|
  puts "    #{area}"
end

puts "\nAreas for Improvement:"
performance.improvement_areas.each do |area|
  puts "  ! #{area}"
end

# Check status
if performance.declining?
  puts "\n   Student performance is declining"
elsif performance.improving?
  puts "\n  Student performance is improving"
end
```

25.8. Error Handling

The SDK provides specific exception types for different error scenarios:

```
require 'kai-sdk'

client = Kai::Client.new

begin
  result = client.assignments.grade(
    assignment_id: 'CS101-HW1',
    student_submission: 'Student work...',
    grading_style: 'detailed'
  )

  puts "Score: #{result.score}"

rescue Kai::ValidationError => e
  # Handle invalid parameters
  puts "Invalid input: #{e.message}"
  puts "Details: #{e.details.inspect}"

rescue Kai::AuthenticationError => e
  # Handle authentication failures
  puts "Authentication failed: #{e.message}"
  puts "Please check your API key"

rescue Kai::RateLimitError => e
  # Handle rate limiting
  puts "Rate limit exceeded: #{e.message}"
  puts "Retry after: #{e.retry_after} seconds"

rescue Kai::TimeoutError => e
  # Handle timeouts
  puts "Request timed out: #{e.message}"
  puts "Try increasing the timeout parameter"

rescue Kai::APIError => e
  # Handle general API errors
  puts "API error: #{e.message}"
  puts "Status code: #{e.status_code}"
  puts "Error code: #{e.error_code}"

rescue Kai::Error => e
  # Catch-all for SDK errors
  puts "SDK error: #{e.message}"
end
```

25.8.1. Exception Hierarchy

```
Kai::Error (base)
   Kai::ValidationError         # Invalid parameters
   Kai::AuthenticationError     # Invalid API key
   Kai::RateLimitError         # Rate limit exceeded
   Kai::TimeoutError           # Request timeout
   Kai::APIError               # General API error
```

Error Recovery Pattern

Implement exponential backoff for rate limit errors:

```ruby
def grade_with_retry(client, assignment_id, submission, max_retries: 3)
  retry_delay = 1

  max_retries.times do |attempt|
    begin
      return client.assignments.grade(
        assignment_id: assignment_id,
        student_submission: submission
      )
    rescue Kai::RateLimitError => e
      if attempt < max_retries - 1
        wait_time = retry_delay * (2 ** attempt)
        puts "Rate limited, waiting #{wait_time}s..."
        sleep(wait_time)
      else
        raise
      end
    end
  end
end
```

25.9. Rails Integration

25.9.1. Configuration

```ruby
# config/initializers/kai.rb
Kai.configure do |config|
  config.api_key = Rails.application.credentials.dig(:kai, :api_key)
  config.timeout = 30
  config.max_retries = 3
```

```
  config.logger = Rails.logger
end
```

25.9.2. Model Integration

```
# app/models/assignment.rb
class Assignment < ApplicationRecord
  has_many :submissions

  def grade_submission(submission)
    client = Kai::Client.new

    result = client.assignments.grade(
      assignment_id: id.to_s,
      student_submission: submission.content,
      rubric_id: rubric_id,
      grading_style: 'detailed'
    )

    submission.update!(
      score: result.score,
      feedback: result.feedback,
      graded_at: Time.current
    )

    result
  rescue Kai::Error => e
    Rails.logger.error("Grading failed: #{e.message}")
    nil
  end
end
```

25.9.3. Service Object Pattern

```
# app/services/grading_service.rb
class GradingService
  def initialize(client: nil)
    @client = client || Kai::Client.new
  end

  def grade_submission(assignment, submission)
    result = @client.assignments.grade(
      assignment_id: assignment.id.to_s,
      student_submission: submission.content,
```

```
      grading_style: 'detailed'
    )

    update_submission(submission, result)
    notify_student(submission)

    result
  end

  def batch_grade(assignment, submissions)
    submission_data = submissions.map do |sub|
      {
        assignment_id: assignment.id.to_s,
        student_id: sub.student_id.to_s,
        student_submission: sub.content
      }
    end

    results = @client.assignments.batch_grade(submission_data)

    submissions.zip(results).each do |submission, result|
      update_submission(submission, result)
    end

    results
  end

  private

  def update_submission(submission, result)
    submission.update!(
      score: result.score,
      feedback: result.feedback,
      suggestions: result.suggestions,
      graded_at: Time.current
    )
  end

  def notify_student(submission)
    StudentMailer.grade_notification(submission).deliver_later
  end
end
```

25.9.4. Background Job

```
# app/jobs/grade_submission_job.rb
class GradeSubmissionJob < ApplicationJob
  queue_as :default
  retry_on Kai::RateLimitError, wait: :exponentially_longer, attempts: 5

  def perform(submission_id)
    submission = Submission.find(submission_id)
    assignment = submission.assignment

    client = Kai::Client.new

    result = client.assignments.grade(
      assignment_id: assignment.id.to_s,
      student_submission: submission.content,
      grading_style: 'detailed'
    )

    submission.update!(
      score: result.score,
      feedback: result.feedback,
      graded_at: Time.current
    )

    StudentMailer.grade_notification(submission).deliver_now

  rescue Kai::Error => e
    Rails.logger.error("Grading failed for submission #{submission_id}: #{e.message}")
    submission.update!(grading_error: e.message)
    raise
  end
end

# Usage in controller
class SubmissionsController < ApplicationController
  def create
    @submission = current_student.submissions.build(submission_params)

    if @submission.save
      GradeSubmissionJob.perform_later(@submission.id)
      redirect_to @submission, notice: 'Submission received. Grading in progress.'
    else
      render :new
    end
  end
end
```

25.9.5. Controller Integration

```ruby
# app/controllers/api/v1/grading_controller.rb
module Api
  module V1
    class GradingController < ApplicationController
      before_action :authenticate_user!

      def create
        service = GradingService.new

        result = service.grade_submission(
          assignment,
          submission
        )

        if result
          render json: {
            grade_id: result.grade_id,
            score: result.score,
            feedback: result.feedback,
            suggestions: result.suggestions
          }, status: :created
        else
          render json: { error: 'Grading failed' }, status: :unprocessable_entity
        end
      end

      def batch_create
        service = GradingService.new
        results = service.batch_grade(assignment, submissions)

        render json: results.map { |r|
          {
            grade_id: r.grade_id,
            score: r.score,
            feedback: r.feedback
          }
        }, status: :created
      end

      private

      def assignment
        @assignment ||= Assignment.find(params[:assignment_id])
      end
```

```
      def submission
        @submission ||= assignment.submissions.find(params[:submission_id])
      end

      def submissions
        @submissions ||= assignment.submissions.where(id: params[:submission_ids])
      end
    end
  end
end
```

25.10. Complete Working Examples

25.10.1. Example 1: Automated Grading Pipeline

```
#!/usr/bin/env ruby
# frozen_string_literal: true

require 'kai-sdk'
require 'csv'

# Automated grading pipeline for processing student submissions from CSV
class AutomatedGradingPipeline
  def initialize(client)
    @client = client
  end

  def load_submissions(csv_path)
    submissions = []

    CSV.foreach(csv_path, headers: true) do |row|
      submissions << {
        student_id: row['student_id'],
        assignment_id: row['assignment_id'],
        submission_text: row['submission_text']
      }
    end

    submissions
  end

  def grade_submissions(submissions)
    results = []

    submissions.each do |submission|
```

```ruby
    begin
      result = @client.assignments.grade(
        assignment_id: submission[:assignment_id],
        student_submission: submission[:submission_text],
        grading_style: 'detailed'
      )

      results << {
        student_id: submission[:student_id],
        score: result.score,
        feedback: result.feedback,
        suggestions: result.suggestions.join(', ')
      }

      puts "  Graded #{submission[:student_id]}: #{result.score}/100"

    rescue Kai::Error => e
      puts "  Failed to grade #{submission[:student_id]}: #{e.message}"

      results << {
        student_id: submission[:student_id],
        score: nil,
        feedback: "Error: #{e.message}",
        suggestions: ''
      }
    end
  end

  results
end

def save_results(results, output_path)
  CSV.open(output_path, 'w') do |csv|
    csv << %w[student_id score feedback suggestions]

    results.each do |result|
      csv << [
        result[:student_id],
        result[:score],
        result[:feedback],
        result[:suggestions]
      ]
    end
  end

  puts "\n  Results saved to #{output_path}"
end
```

```ruby
  def print_summary(results)
    successful = results.count { |r| r[:score] }
    scores = results.map { |r| r[:score] }.compact
    avg_score = scores.sum / scores.size.to_f

    puts "\n=== Summary ==="
    puts "Total submissions: #{results.size}"
    puts "Successfully graded: #{successful}"
    puts "Average score: #{avg_score.round(1)}/100"
  end

  def run(input_path, output_path)
    submissions = load_submissions(input_path)
    puts "Loaded #{submissions.size} submissions"

    results = grade_submissions(submissions)
    save_results(results, output_path)
    print_summary(results)
  end
end

# Run the pipeline
if __FILE__ == $PROGRAM_NAME
  client = Kai::Client.new
  pipeline = AutomatedGradingPipeline.new(client)

  pipeline.run('submissions.csv', 'grading_results.csv')
end
```

25.10.2. Example 2: Quiz Generator with Export

```ruby
#!/usr/bin/env ruby
# frozen_string_literal: true

require 'kai-sdk'
require 'json'

# Generate quizzes for multiple topics and export to various formats
class QuizGenerator
  def initialize(client)
    @client = client
  end

  def generate_quiz(topic, difficulty: 'medium', count: 10)
    quiz = @client.quizzes.generate(
```

```
    topic: topic,
    difficulty: difficulty,
    question_count: count,
    question_types: ['multiple_choice', 'true_false', 'short_answer']
  )

  puts "\n Generated quiz: #{topic}"
  puts "  ID: #{quiz.quiz_id}"
  puts "  Questions: #{quiz.question_count}"
  puts "  Time: ~#{quiz.estimated_time_minutes} minutes"

  quiz
end

def export_to_json(quiz, filename)
  quiz_data = {
    quiz_id: quiz.quiz_id,
    topic: quiz.topic,
    difficulty: quiz.difficulty,
    estimated_time_minutes: quiz.estimated_time_minutes,
    questions: quiz.questions.map do |q|
      {
        id: q.question_id,
        text: q.question_text,
        type: q.question_type,
        options: q.options,
        correct_answer: q.correct_answer,
        explanation: q.explanation
      }
    end
  }

  File.write(filename, JSON.pretty_generate(quiz_data))
  puts "  Exported to #{filename}"
end

def export_to_markdown(quiz, filename)
  File.open(filename, 'w') do |f|
    f.puts "# #{quiz.topic} Quiz\n\n"
    f.puts "**Difficulty:** #{quiz.difficulty}\n"
    f.puts "**Estimated Time:** #{quiz.estimated_time_minutes} minutes\n\n"
    f.puts "---\n\n"

    quiz.each_question.with_index(1) do |question, i|
      f.puts "## Question #{i}\n\n"
      f.puts "#{question.question_text}\n\n"
```

```ruby
        if question.options
          question.options.each do |opt|
            f.puts "- #{opt}\n"
          end
          f.puts "\n"
        end

        if question.explanation
          f.puts "**Explanation:** #{question.explanation}\n\n"
        end

        f.puts "---\n\n"
      end
    end

    puts "  Exported to #{filename}"
  end

  def generate_all(topics)
    topics.each do |config|
      quiz = generate_quiz(
        config[:topic],
        difficulty: config[:difficulty],
        count: config[:count]
      )

      base_name = config[:topic].downcase.gsub(' ', '_')
      export_to_json(quiz, "#{base_name}_quiz.json")
      export_to_markdown(quiz, "#{base_name}_quiz.md")
    end

    puts "\n  All quizzes generated and exported successfully!"
  end
end

# Run the generator
if __FILE__ == $PROGRAM_NAME
  client = Kai::Client.new
  generator = QuizGenerator.new(client)

  topics = [
    { topic: 'Ruby Fundamentals', difficulty: 'easy', count: 15 },
    { topic: 'Rails Development', difficulty: 'medium', count: 20 },
    { topic: 'Ruby Metaprogramming', difficulty: 'hard', count: 10 }
  ]

  generator.generate_all(topics)
```

```
end
```

25.10.3. Example 3: Student Analytics Dashboard

```ruby
#!/usr/bin/env ruby
# frozen_string_literal: true

require 'kai-sdk'
require 'csv'
require 'date'

# Generate comprehensive student analytics reports
class StudentAnalyticsDashboard
  def initialize(client)
    @client = client
  end

  def load_roster(csv_path)
    student_ids = []

    CSV.foreach(csv_path, headers: true) do |row|
      student_ids << row['student_id']
    end

    student_ids
  end

  def analyze_student(student_id, course_id)
    end_date = Date.today
    start_date = end_date - 90 # Last 90 days

    performance = @client.analytics.student_performance(
      student_id: student_id,
      start_date: start_date.iso8601,
      end_date: end_date.iso8601,
      course_id: course_id
    )

    {
      student_id: student_id,
      average_score: performance.average_score,
      completion_rate: performance.completion_rate,
      trend: performance.trend,
      strengths: performance.strength_areas,
      improvements: performance.improvement_areas
```

```ruby
  }
end

def analyze_class(student_ids, course_id)
  analytics_data = []

  student_ids.each do |student_id|
    begin
      data = analyze_student(student_id, course_id)
      analytics_data << data
      puts "  Analyzed #{student_id}"
    rescue Kai::Error => e
      puts "  Failed to analyze #{student_id}: #{e.message}"
    end
  end

  analytics_data
end

def generate_report(analytics_data, output_dir)
  Dir.mkdir(output_dir) unless Dir.exist?(output_dir)

  # Save detailed CSV
  csv_path = File.join(output_dir, 'class_analytics.csv')
  CSV.open(csv_path, 'w') do |csv|
    csv << %w[student_id average_score completion_rate trend strengths improvements]

    analytics_data.each do |data|
      csv << [
        data[:student_id],
        data[:average_score].round(1),
        data[:completion_rate].round(1),
        data[:trend],
        data[:strengths].join('; '),
        data[:improvements].join('; ')
      ]
    end
  end

  # Generate summary report
  summary_path = File.join(output_dir, 'class_summary.txt')
  File.open(summary_path, 'w') do |f|
    write_summary(f, analytics_data)
  end

  puts "\n Reports saved to #{output_dir}/"
  puts "  - #{csv_path}"
```

```
    puts "  - #{summary_path}"
  end

  def write_summary(file, analytics_data)
    scores = analytics_data.map { |d| d[:average_score] }
    completion_rates = analytics_data.map { |d| d[:completion_rate] }

    file.puts "CLASS ANALYTICS REPORT"
    file.puts "=" * 50
    file.puts

    file.puts "Total Students: #{analytics_data.size}"
    file.puts "Class Average: #{(scores.sum / scores.size).round(1)}"
    file.puts "Average Completion Rate: #{(completion_rates.sum / completion_rates.size).round(1)
    file.puts

    # Students by trend
    improving = analytics_data.count { |d| d[:trend] == 'improving' }
    stable = analytics_data.count { |d| d[:trend] == 'stable' }
    declining = analytics_data.count { |d| d[:trend] == 'declining' }

    file.puts "Student Trends:"
    file.puts "  Improving: #{improving} (#{(improving.to_f / analytics_data.size * 100).round(1)
    file.puts "  Stable: #{stable} (#{(stable.to_f / analytics_data.size * 100).round(1)}%)"
    file.puts "  Declining: #{declining} (#{(declining.to_f / analytics_data.size * 100).round(1)
    file.puts

    # Students needing attention
    file.puts "Students Needing Attention:"
    analytics_data.each do |data|
      if data[:average_score] < 70 || data[:trend] == 'declining'
        file.puts "  - #{data[:student_id]}: Score #{data[:average_score].round(1)}, #{data[:tren
      end
    end
  end

  def run(roster_path, course_id, output_dir)
    student_ids = load_roster(roster_path)
    puts "Analyzing #{student_ids.size} students..."

    analytics_data = analyze_class(student_ids, course_id)
    generate_report(analytics_data, output_dir)

    puts "\n Analytics complete!"
  end
end
```

```
# Run the dashboard
if __FILE__ == $PROGRAM_NAME
  client = Kai::Client.new
  dashboard = StudentAnalyticsDashboard.new(client)

  dashboard.run('roster.csv', 'CS101', 'analytics_report')
end
```

25.11. Rake Tasks

Create custom rake tasks for common operations:

```
# lib/tasks/kai.rake
namespace :kai do
  desc 'Grade all pending submissions'
  task grade_pending: :environment do
    client = Kai::Client.new

    Submission.pending.find_each do |submission|
      begin
        result = client.assignments.grade(
          assignment_id: submission.assignment_id.to_s,
          student_submission: submission.content
        )

        submission.update!(
          score: result.score,
          feedback: result.feedback,
          graded_at: Time.current
        )

        puts "  Graded submission #{submission.id}"
      rescue Kai::Error => e
        puts "  Failed to grade submission #{submission.id}: #{e.message}"
      end
    end
  end

  desc 'Generate weekly analytics report'
  task weekly_report: :environment do
    client = Kai::Client.new

    Course.find_each do |course|
      analytics = client.analytics.class_analytics(course_id: course.id.to_s)

      WeeklyReport.create!(
```

```
        course: course,
        data: analytics.to_h,
        generated_at: Time.current
      )

      puts "  Generated report for #{course.name}"
    end
  end
end
```

25.12. Best Practices

Ruby Idioms and Best Practices

1. **Use Symbols for Keys**: Follow Ruby conventions

```
# Good: Using symbols
client.assignments.grade(
  assignment_id: 'CS101',
  student_submission: 'Work...'
)

# Avoid: Using strings
client.assignments.grade(
  'assignment_id' => 'CS101',
  'student_submission' => 'Work...'
)
```

2. **Leverage Blocks**: Use Ruby's block syntax for configuration

```
# Good: Block-based configuration
Kai.configure do |config|
  config.api_key = ENV['KAI_API_KEY']
  config.timeout = 30
end

# Also good: Chaining with blocks
quiz = client.quizzes.generate_with do |req|
  req.topic = 'Ruby'
  req.difficulty = 'medium'
  req.question_count = 10
end
```

3. **Use Safe Navigation**: Handle nil values gracefully

```ruby
# Good: Safe navigation
score = result&.score || 0
feedback = result&.feedback&.truncate(100)

# Use presence for nil/empty checks
api_key = ENV['KAI_API_KEY'].presence || 'default_key'
```

4. **Follow Rails Patterns**: Use service objects and jobs

```ruby
# Good: Service object
class GradingService
  def call(submission)
    # Grading logic
  end
end

# Good: Background job
GradeSubmissionJob.perform_later(submission.id)
```

5. **Handle Errors Idiomatically**: Use rescue with specific exceptions

```ruby
# Good: Specific exception handling
begin
  result = client.assignments.grade(...)
rescue Kai::RateLimitError => e
  sleep(e.retry_after)
  retry
rescue Kai::Error => e
  Rails.logger.error("Grading failed: #{e.message}")
  nil
end
```

6. **Use Enumerable Methods**: Leverage Ruby's powerful enumerables

```ruby
# Good: Functional style
passing = results.select(&:passed?)
scores = results.map(&:score)
average = scores.sum / scores.size.to_f

# Chain operations
top_performers = results
  .select { |r| r.score >= 90 }
  .sort_by(&:score)
  .reverse
  .first(5)
```

25.13. Testing

25.13.1. RSpec Examples

```
# spec/services/grading_service_spec.rb
require 'rails_helper'

RSpec.describe GradingService do
  let(:client) { instance_double(Kai::Client) }
  let(:service) { described_class.new(client: client) }

  describe '#grade_submission' do
    let(:assignment) { create(:assignment) }
    let(:submission) { create(:submission, assignment: assignment) }

    let(:kai_result) do
      Kai::GradeResult.new(
        grade_id: 'grade_123',
        score: 85,
        feedback: 'Good work',
        suggestions: ['Add more comments']
      )
    end

    before do
      allow(client.assignments).to receive(:grade).and_return(kai_result)
    end

    it 'grades the submission successfully' do
      result = service.grade_submission(assignment, submission)

      expect(result.score).to eq(85)
      expect(submission.reload.score).to eq(85)
      expect(submission.feedback).to eq('Good work')
    end

    context 'when grading fails' do
      before do
        allow(client.assignments).to receive(:grade)
          .and_raise(Kai::APIError.new('API error'))
      end

      it 'handles the error gracefully' do
        expect {
          service.grade_submission(assignment, submission)
        }.not_to raise_error
```

```
        expect(submission.reload.graded_at).to be_nil
      end
    end
  end
end
```

25.14. Links and Resources

25.14.1. Related Documentation

- API Reference - Complete REST API documentation
- Getting Started - Initial setup and configuration
- Guides - Integration guides and tutorials

25.14.2. SDK Resources

- GitHub Repository - Source code and issues
- RubyGems - Package distribution
- Documentation - API documentation
- Changelog - Version history
- Examples - More code examples

25.14.3. Support

- API Status: status.kaitheai.com
- Developer Forum: forum.kaitheai.com
- Email: developers@chi2labs.com

> **Stay Updated**
>
> Star the GitHub repository to receive notifications about new releases and features.

Part VI.

Reference

26. Content Templates Guide

Create reusable templates for assignments, quizzes, and assessments

26.1. Overview

Content templates in Kai allow you to create reusable structures for assignments, quizzes, and assessments. Build once, use many times, and maintain consistency across your courses while saving significant preparation time.

26.2. Benefits of Templates

- **Time Savings**: Create assignments in minutes instead of hours
- **Consistency**: Maintain uniform structure and quality
- **Reusability**: Use across multiple courses and semesters
- **Collaboration**: Share templates with colleagues
- **Continuous Improvement**: Refine templates based on student performance

26.3. Quick Start

26.3.1. Creating Your First Template

The fastest way to create a template:

1. **Start with existing content**: Convert a successful assignment into a template
2. **Identify variables**: Mark what changes between uses (dates, topics, etc.)
3. **Test the template**: Generate one instance and review
4. **Refine and save**: Adjust and save for future use

> Template Strategy
>
> Start with 2-3 core templates for your most common assignment types. Expand your library as you identify patterns in your teaching.

26.4. Assignment Templates

26.4.1. Basic Structure

Every assignment template includes:

- **Metadata**: Title, description, type, duration
- **Instructions**: Static and variable content
- **Deliverables**: What students must submit
- **Grading**: Associated rubric and point value
- **Resources**: Supporting materials and links
- **Variables**: Customizable elements (topics, dates, readings)

26.4.2. Creating an Assignment Template

Via Dashboard:

1. **Navigate**: Dashboard → Content → Templates → "Create Assignment Template"
2. **Configure**:
    - **Template Name**: "Weekly Reading Response"
    - **Category**: "Discussion", "Essay", "Problem Set", etc.
    - **Reusability**: "High" (use weekly), "Medium" (use monthly), "Low" (occasional)

Example: Reading Response Template

```
template:
  name: "Weekly Reading Response"
  category: "discussion"

  variables:
    - name: "week_number"
      type: "integer"
      description: "Week of the semester"

    - name: "reading_title"
      type: "text"
      description: "Title of assigned reading"

    - name: "reading_author"
      type: "text"
      description: "Author of reading"

    - name: "reading_pages"
      type: "text"
      description: "Page range"

    - name: "focus_question"
```

```
      type: "text"
      description: "Main analytical question"

    - name: "due_date"
      type: "date"
      description: "Submission deadline"

  content:
    title: "Week {week_number}: Reading Response - {reading_title}"

    instructions: |
      Read **{reading_title}** by {reading_author} (pages {reading_pages}).

      Write a 500-word response addressing the following question:

      {focus_question}

      Your response should:
      - Include at least two direct quotations with page numbers
      - Connect the reading to concepts from previous weeks
      - Raise one thoughtful question for class discussion

      **Due**: {due_date} by 11:59 PM

    deliverables:
      - type: "text"
        min_words: 450
        max_words: 600
        format: "PDF or Word document"

    grading:
      rubric_id: "reading_response_rubric"
      points: 20

    resources:
      - title: "How to Write an Effective Response"
        url: "https://course-resources.edu/response-guide"
      - title: "Citation Guide"
        url: "https://course-resources.edu/citations"
```

Using the Template:

```
# Generate assignment from template
assignment = kai.assignments.create_from_template(
    template_id="reading_response_template",
    course_id="course_abc",
    variables={
```

```
        "week_number": 3,
        "reading_title": "The Great Gatsby",
        "reading_author": "F. Scott Fitzgerald",
        "reading_pages": "1-50",
        "focus_question": "How does Fitzgerald use symbolism in the opening chapters to establish
        "due_date": "2024-09-20"
    }
)

print(f"Created: {assignment.title}")
```

26.4.3. Advanced Assignment Templates

Research Paper Template with Milestones:

```
research_paper_template = {
    "name": "Research Paper - Multi-Stage",
    "category": "major_assignment",

    "stages": [
        {
            "name": "Topic Proposal",
            "duration_days": 7,
            "deliverable": "1-page proposal",
            "points": 10,
            "instructions": """
                Submit a one-page proposal including:
                1. Research question
                2. Preliminary thesis
                3. Three potential sources
                4. Expected argument
            """,
            "auto_feedback": True
        },
        {
            "name": "Annotated Bibliography",
            "duration_days": 7,
            "deliverable": "5-source bibliography",
            "points": 15,
            "instructions": """
                Compile annotated bibliography with:
                - 5 scholarly sources
                - 150-word annotation per source
                - Proper {citation_style} format
            """,
            "check_citations": True
```

```
        },
        {
            "name": "Outline",
            "duration_days": 7,
            "deliverable": "detailed outline",
            "points": 10,
            "instructions": """
                Create detailed outline with:
                - Thesis statement
                - Main sections with subsections
                - Key evidence for each section
                - Anticipated counterarguments
            """
        },
        {
            "name": "Draft",
            "duration_days": 14,
            "deliverable": "full draft",
            "points": 15,
            "instructions": """
                Submit complete draft (minimum {min_words} words)
                including all sections.
            """,
            "peer_review": {
                "enabled": True,
                "reviewers_per_paper": 2
            }
        },
        {
            "name": "Final Paper",
            "duration_days": 7,
            "deliverable": "revised paper",
            "points": 50,
            "instructions": """
                Submit final paper incorporating feedback from:
                - Instructor comments on draft
                - Peer review suggestions

                Requirements:
                - {min_words}-{max_words} words
                - {min_sources}+ scholarly sources
                - {citation_style} format
            """,
            "rubric_id": "research_paper_rubric"
        }
    ],
```

```
    "variables": {
        "topic_area": "text",
        "citation_style": "choice: MLA, APA, Chicago",
        "min_words": "integer",
        "max_words": "integer",
        "min_sources": "integer",
        "final_due_date": "date"
    }
}

# Create multi-stage assignment
kai.assignments.create_from_template(
    template_id="research_paper_multistage",
    course_id="course_abc",
    variables={
        "topic_area": "American Literature 1920-1940",
        "citation_style": "MLA",
        "min_words": 2500,
        "max_words": 3500,
        "min_sources": 8,
        "final_due_date": "2024-12-10"
    },
    auto_schedule=True  # Automatically calculate intermediate deadlines
)
```

26.5. Quiz Templates

26.5.1. Question Bank Templates

Create reusable question structures:

Multiple Choice Template:

```
mc_template = {
    "type": "multiple_choice",
    "structure": {
        "stem": "{question_text}",
        "options": [
            "{correct_answer}",
            "{distractor_1}",
            "{distractor_2}",
            "{distractor_3}"
        ],
        "explanation": "{explanation_text}"
    },
    "settings": {
```

```
        "randomize_options": True,
        "points": 1,
        "partial_credit": False
    }
}

# Use template to generate questions
question = kai.questions.create_from_template(
    template="mc_template",
    variables={
        "question_text": "Which statistical test is appropriate for comparing two independent gro
        "correct_answer": "Independent samples t-test",
        "distractor_1": "Paired samples t-test",
        "distractor_2": "One-way ANOVA",
        "distractor_3": "Chi-square test",
        "explanation": "Independent samples t-test compares means of two unrelated groups. Paired
    }
)
```

Application Problem Template:

```
application_template = {
    "type": "application",
    "structure": {
        "scenario": "{scenario_description}",
        "data": "{data_table}",
        "question": "{question_text}",
        "answer_format": "{format_specification}",
        "solution": "{solution_with_steps}"
    },
    "variables": {
        "scenario_type": "category",  # research, business, medical, etc.
        "difficulty": "range: 1-5",
        "concepts_tested": "list"
    }
}

# AI-generated variations
questions = kai.questions.generate_variations(
    template="application_template",
    count=5,
    variables={
        "scenario_type": "medical_research",
        "difficulty": 3,
        "concepts_tested": ["hypothesis_testing", "p_values", "effect_size"]
    }
)
```

26.5.2. Complete Quiz Templates

Weekly Concept Check Template:

```
concept_check_template = {
    "name": "Weekly Concept Check",
    "purpose": "formative_assessment",

    "settings": {
        "time_limit_minutes": 15,
        "attempts_allowed": 2,
        "show_correct_answers": "after_due_date",
        "randomize_questions": True,
        "question_pool_size": 15,
        "questions_shown": 10
    },

    "structure": [
        {
            "section": "Definitions",
            "question_type": "multiple_choice",
            "count": 3,
            "difficulty": "easy",
            "topics": "{week_topics}"
        },
        {
            "section": "Concepts",
            "question_type": "true_false",
            "count": 3,
            "difficulty": "medium",
            "topics": "{week_topics}"
        },
        {
            "section": "Application",
            "question_type": "short_answer",
            "count": 2,
            "difficulty": "medium",
            "topics": "{week_topics}"
        },
        {
            "section": "Analysis",
            "question_type": "scenario_based",
            "count": 2,
            "difficulty": "hard",
            "topics": "{week_topics}"
        }
    ],
```

```
    "grading": {
        "total_points": 20,
        "auto_grade": True,
        "feedback_style": "detailed",
        "show_explanations": True
    },

    "variables": {
        "week_number": "integer",
        "week_topics": "list",
        "available_date": "datetime",
        "due_date": "datetime"
    }
}

# Generate quiz for specific week
quiz = kai.quizzes.create_from_template(
    template_id="concept_check_template",
    course_id="course_abc",
    variables={
        "week_number": 5,
        "week_topics": ["Standard Error", "Confidence Intervals", "Margin of Error"],
        "available_date": "2024-10-01 00:00",
        "due_date": "2024-10-03 23:59"
    }
)
```

Adaptive Practice Quiz Template:

```
adaptive_template = {
    "name": "Adaptive Practice Quiz",
    "type": "adaptive",

    "algorithm": {
        "start_difficulty": "medium",
        "adjust_based_on": "previous_answer",
        "adjustment_rules": {
            "correct_answer": "increase_difficulty",
            "incorrect_answer": "decrease_difficulty",
            "mastery_threshold": 0.8
        }
    },

    "question_selection": {
        "min_questions": 5,
        "max_questions": 20,
        "stop_when": [
```

```
            "mastery_achieved",
            "max_questions_reached",
            "time_limit_reached"
        ]
    },

    "difficulty_levels": {
        "easy": {
            "description": "Basic recall and recognition",
            "points": 1
        },
        "medium": {
            "description": "Application and analysis",
            "points": 2
        },
        "hard": {
            "description": "Synthesis and evaluation",
            "points": 3
        }
    },

    "feedback": {
        "immediate": True,
        "show_explanation": True,
        "suggest_resources": True,
        "track_weak_areas": True
    }
}
```

26.6. Exam Templates

26.6.1. Midterm/Final Exam Template

```
exam_template = {
    "name": "Comprehensive Exam Template",
    "type": "high_stakes",

    "sections": [
        {
            "name": "Multiple Choice",
            "points": 40,
            "questions": 40,
            "time_suggestion_minutes": 40,
            "instructions": """
                Choose the best answer for each question.
```

```
                Each question is worth 1 point.
            """
        },
        {
            "name": "Short Answer",
            "points": 30,
            "questions": 6,
            "time_suggestion_minutes": 30,
            "instructions": """
                Answer in 2-3 sentences. Be specific and concise.
                Each question is worth 5 points.
            """
        },
        {
            "name": "Essay",
            "points": 30,
            "questions": 2,
            "time_suggestion_minutes": 50,
            "instructions": """
                Choose 2 of 3 essay prompts.
                Write 500-750 words per essay.
                Each essay is worth 15 points.
            """,
            "choice": {
                "prompts_provided": 3,
                "student_selects": 2
            }
        }
    ],

    "settings": {
        "time_limit_minutes": 120,
        "attempts_allowed": 1,
        "require_lockdown_browser": True,
        "randomize_question_order": True,
        "one_question_at_a_time": False,
        "allow_backtracking": True
    },

    "content_coverage": {
        "weeks_covered": "{weeks_covered}",
        "emphasis_weeks": "{emphasis_weeks}",
        "cumulative": True
    },

    "grading": {
        "auto_grade": {
```

```
            "multiple_choice": True,
            "short_answer": False,
            "essay": False
        },
        "rubric_ids": {
            "short_answer": "short_answer_rubric",
            "essay": "essay_rubric"
        },
        "review_required": True,
        "post_delay_hours": 48
    },

    "accommodations": {
        "extended_time": {
            "multiplier": 1.5,
            "eligible_students": "from_disability_services"
        },
        "separate_location": {
            "enabled": True
        }
    }
}

# Create midterm
midterm = kai.exams.create_from_template(
    template_id="comprehensive_exam",
    course_id="course_abc",
    exam_type="midterm",
    variables={
        "weeks_covered": "1-7",
        "emphasis_weeks": "5-7",
        "exam_date": "2024-10-15",
        "available_start": "2024-10-15 09:00",
        "available_end": "2024-10-15 11:30"
    }
)
```

26.7. Discussion Templates

26.7.1. Structured Discussion Template

```
discussion_template = {
    "name": "Weekly Structured Discussion",
    "type": "threaded_discussion",
```

```
    "phases": [
        {
            "name": "Initial Post",
            "duration_days": 3,
            "requirements": {
                "min_words": 250,
                "required_elements": [
                    "answer_to_prompt",
                    "evidence_from_reading",
                    "personal_analysis",
                    "question_for_peers"
                ]
            },
            "points": 10
        },
        {
            "name": "Peer Responses",
            "duration_days": 3,
            "requirements": {
                "min_responses": 2,
                "min_words_per_response": 100,
                "substantive_engagement": True
            },
            "points": 10
        },
        {
            "name": "Follow-up",
            "duration_days": 1,
            "requirements": {
                "respond_to_replies": True,
                "optional": False
            },
            "points": 5
        }
    ],

    "prompt_structure": {
        "introduction": "{context_setting}",
        "main_question": "{discussion_question}",
        "sub_questions": [
            "{sub_question_1}",
            "{sub_question_2}"
        ],
        "instructions": """
            In your initial post:
            1. Answer the main question with specific evidence
            2. Address at least one sub-question
```

```
            3. Pose a thoughtful question for your peers

            In peer responses:
            1. Engage substantively with peers' ideas
            2. Provide additional perspectives or evidence
            3. Answer questions posed by peers
        """
    },

    "grading": {
        "rubric_id": "discussion_rubric",
        "auto_grade_elements": [
            "word_count",
            "number_of_posts",
            "timeliness"
        ],
        "human_grade_elements": [
            "quality_of_analysis",
            "engagement_depth"
        ]
    },

    "safestream": {
        "enabled": True,
        "monitor_for": [
            "respect_boundaries",
            "academic_integrity",
            "constructive_feedback"
        ]
    }
}

# Create discussion for specific week
discussion = kai.discussions.create_from_template(
    template_id="structured_discussion",
    course_id="course_abc",
    variables={
        "week_number": 4,
        "context_setting": "After reading Chapter 3 on social constructivism...",
        "discussion_question": "How do social factors influence individual learning?",
        "sub_question_1": "Provide an example from your own educational experience.",
        "sub_question_2": "How might this theory apply to online learning environments?",
        "initial_post_due": "2024-09-25 23:59",
        "responses_due": "2024-09-28 23:59",
        "followup_due": "2024-09-29 23:59"
    }
)
```

26.8. Lab/Practical Templates

26.8.1. Science Lab Report Template

```
lab_template = {
    "name": "Lab Report Template",
    "type": "lab_report",

    "sections": [
        {
            "name": "Title & Abstract",
            "required": True,
            "word_limit": 250,
            "elements": ["title", "objective", "methods_summary", "results_summary"]
        },
        {
            "name": "Introduction",
            "required": True,
            "word_range": [300, 500],
            "elements": ["background", "hypothesis", "predictions"]
        },
        {
            "name": "Methods",
            "required": True,
            "word_range": [400, 600],
            "elements": ["materials", "procedure", "data_collection"]
        },
        {
            "name": "Results",
            "required": True,
            "required_components": [
                "data_table",
                "graph_or_figure",
                "statistical_analysis"
            ],
            "word_range": [300, 500]
        },
        {
            "name": "Discussion",
            "required": True,
            "word_range": [500, 750],
            "elements": [
                "interpretation",
                "comparison_to_hypothesis",
                "sources_of_error",
                "future_directions"
```

```
            ]
        },
        {
            "name": "References",
            "required": True,
            "min_sources": 5,
            "citation_style": "{citation_style}"
        }
    ],

    "data_files": {
        "required": True,
        "accepted_formats": [".xlsx", ".csv", ".txt"],
        "data_validation": {
            "check_completeness": True,
            "check_units": True,
            "flag_anomalies": True
        }
    },

    "grading": {
        "rubric_id": "lab_report_rubric",
        "section_weights": {
            "title_abstract": 0.10,
            "introduction": 0.15,
            "methods": 0.15,
            "results": 0.25,
            "discussion": 0.25,
            "references": 0.10
        }
    },

    "variables": {
        "experiment_name": "text",
        "lab_date": "date",
        "citation_style": "choice: APA, CSE, AMA",
        "due_date": "date"
    }
}
```

26.9. Template Management

26.9.1. Organizing Templates

Category System:

```
categories = {
    "by_type": [
        "assignments",
        "quizzes",
        "exams",
        "discussions",
        "labs"
    ],
    "by_frequency": [
        "daily",
        "weekly",
        "monthly",
        "once_per_semester"
    ],
    "by_discipline": [
        "humanities",
        "social_sciences",
        "STEM",
        "professional"
    ],
    "by_purpose": [
        "formative",
        "summative",
        "practice",
        "engagement"
    ]
}

# Tag templates for easy retrieval
kai.templates.update(
    template_id="reading_response",
    tags=["weekly", "humanities", "formative", "discussion"]
)

# Search templates
templates = kai.templates.search(
    tags=["weekly", "formative"],
    course_level="undergraduate"
)
```

26.9.2. Sharing Templates

Within Institution:

```
# Share with department
kai.templates.share(
```

```
    template_id="research_paper_multistage",
    share_with="department",
    permissions={
        "view": True,
        "use": True,
        "edit": False,  # Can customize when using, but can't edit original
        "share": False
    }
)

# Share with specific colleagues
kai.templates.share(
    template_id="adaptive_quiz",
    share_with=["instructor_123", "instructor_456"],
    permissions={"view": True, "use": True, "edit": True}
)
```

Public Library:

```
# Publish to Kai community library
kai.templates.publish(
    template_id="concept_check",
    public=True,
    license="CC-BY",
    metadata={
        "subject": "Statistics",
        "level": "Undergraduate",
        "description": "Weekly formative assessment for statistics concepts",
        "usage_notes": "Customize week_topics for your specific curriculum"
    }
)

# Browse community templates
community_templates = kai.templates.browse_community(
    subject="Statistics",
    type="quiz",
    rating_min=4.0
)
```

26.9.3. Version Control

Track changes to templates over time:

```
# Create new version
kai.templates.create_version(
    template_id="reading_response",
```

```
    changes="Added requirement for peer question; increased word count to 500",
    version_notes="Based on student feedback from Fall 2024"
)

# View version history
versions = kai.templates.get_versions(template_id="reading_response")

for version in versions:
    print(f"Version {version.number}: {version.date}")
    print(f"  Changes: {version.changes}")
    print(f"  Used {version.usage_count} times")

# Revert to previous version if needed
kai.templates.revert(
    template_id="reading_response",
    to_version=3
)
```

26.10. AI-Assisted Template Creation

26.10.1. Generate Templates from Examples

```
# Upload example assignment
kai.templates.generate_from_example(
    example_file="my_successful_assignment.pdf",
    template_name="Problem Set Template",
    instructions="""
        Create a reusable template that:
        - Maintains the general structure
        - Identifies variable elements (topics, due dates)
        - Preserves the grading criteria
        - Allows for different difficulty levels
    """
)

# Review generated template
generated = kai.templates.get_pending_review()

# Approve or edit
kai.templates.approve(template_id=generated.id, edits={...})
```

26.10.2. AI Template Suggestions

```
# Get AI suggestions for improvement
suggestions = kai.templates.analyze(
    template_id="reading_response",
    analyze_for=[
        "clarity",
        "student_engagement",
        "grading_efficiency",
        "learning_outcomes_alignment"
    ]
)

print("Improvement Suggestions:")
for suggestion in suggestions:
    print(f"- {suggestion.area}: {suggestion.recommendation}")
    print(f"  Impact: {suggestion.estimated_impact}")
    print(f"  Effort: {suggestion.implementation_effort}")
```

26.11. Template Analytics

26.11.1. Usage Tracking

```
# View template performance
analytics = kai.templates.get_analytics(
    template_id="concept_check",
    timeframe="semester"
)

print(f"Used {analytics.usage_count} times")
print(f"Average student score: {analytics.avg_score}")
print(f"Average completion rate: {analytics.completion_rate}%")
print(f"Student feedback rating: {analytics.student_rating}/5")
print(f"Time saved: {analytics.estimated_time_saved} hours")

# Identify improvements
if analytics.avg_score < 0.70:
    print("Suggestion: Questions may be too difficult")
if analytics.completion_rate < 0.85:
    print("Suggestion: Consider reducing length or extending deadline")
```

26.11.2. A/B Testing Templates

```
# Test two versions
kai.templates.ab_test(
    template_a="reading_response_v1",
    template_b="reading_response_v2",
    course_id="course_abc",
    assignments_count=4,  # Test over 4 weeks
    metrics=["student_engagement", "avg_score", "completion_rate"]
)

# Review results
results = kai.templates.get_ab_test_results(test_id="test_123")

print(f"Version A: {results.version_a.avg_engagement}")
print(f"Version B: {results.version_b.avg_engagement}")
print(f"Winner: {results.recommended_version}")
```

26.12. Best Practices

26.12.1. Template Design Principles

> **Flexibility vs. Consistency**
>
> Design templates to be flexible enough for different contexts while maintaining core structure and quality standards.

Checklist:

- Clear instructions that work across variations
- Appropriate number of variables (not too many)
- Realistic time expectations
- Aligned with learning objectives
- Built-in assessment criteria
- Student-friendly formatting

26.12.2. Maintenance Schedule

Regular Review:

```
maintenance_schedule = {
    "after_each_use": [
        "Check student feedback",
        "Note any confusion points",
```

```
        "Track completion rates"
    ],
    "end_of_semester": [
        "Review overall effectiveness",
        "Update based on outcomes",
        "Revise difficulty if needed",
        "Update resources/links"
    ],
    "annually": [
        "Major content update",
        "Alignment with curriculum changes",
        "Rubric refinement",
        "Archive outdated versions"
    ]
}
```

26.13. Troubleshooting

26.13.1. Template Variables Not Working

Problem: Variables not substituting correctly

Solutions:

```
# Check variable syntax
# Correct: {variable_name}
# Incorrect: {{variable_name}}, $variable_name

# Verify all variables are defined
missing_vars = kai.templates.validate(template_id="template_123")
print(f"Missing variables: {missing_vars}")

# Provide defaults for optional variables
kai.templates.update(
    template_id="template_123",
    variable_defaults={
        "optional_reading": "No additional reading this week"
    }
)
```

26.13.2. Students Confused by Instructions

Problem: Template-generated instructions unclear

Solutions: - Review generated instance before publishing - Add context-specific clarification - Include examples in template - Test with colleague before using with students

26.13.3. Template Too Rigid

Problem: Template doesn't allow needed customization

Solutions: - Add more variables for flexibility - Create template variants - Use "optional sections" feature - Consider if template is too specific

26.14. Support Resources

- **Template Library**: Browse pre-built templates
- **Workshops**: Template creation workshop
- **Email**: templates@chi2labs.com
- **Community**: Share and discuss templates

Next Steps: - Set up grading rubrics to use with templates - Personalize learning experience using template data - Review best practices for template usage

27. Emoji Style Guide

Standardized Emoji Usage for Kai Documentation

27.1. Overview

This guide establishes standardized emoji usage across all Kai the AI documentation, diagrams, and user interfaces. Consistent emoji use enhances visual communication while maintaining professional appearance.

27.2. Core Principles

1. **Consistency**: Use the same emoji for the same concept throughout all documentation
2. **Clarity**: Choose emojis that are universally understood
3. **Accessibility**: Ensure emojis supplement, not replace, text descriptions
4. **Cross-platform**: Select emojis that render well on all platforms

27.3. Emoji Dictionary

27.3.1. Educational Components

Emoji	Concept	Usage Context	Alternative Text
	Textbook/Learning Materials	Course content, reading materials	`[Textbook]`
	Individual Book/Document	Single resource, manual	`[Book]`
	Document/File	Generic documents, notes	`[Document]`
	Writing/Notes	Note-taking, assignments	`[Notes]`
	Presentation/Slides	PowerPoint, slides	`[Slides]`
	Graduation/Achievement	Completion, success	`[Achievement]`
	Clipboard/Assessment	Tests, evaluations	`[Assessment]`

27.3.2. People & Roles

Emoji	Concept	Usage Context	Alternative Text
	Instructor (Male)	Teacher, professor	`[Instructor]`
	Instructor (Female)	Teacher, professor	`[Instructor]`
	Instructor (Neutral)	Teacher, professor	`[Instructor]`
	Students/Group	Student body, learners	`[Students]`
	Individual User	Single student/user	`[User]`
	Student (Male)	Learner	`[Student]`
	Student (Female)	Learner	`[Student]`
	Student (Neutral)	Learner	`[Student]`

27.3.3. Technology & AI

Emoji	Concept	Usage Context	Alternative Text
	AI/Bot/Kai	Artificial intelligence, Kai	`[AI]`
	Computer/System	Computing systems	`[System]`
	Mobile/App	Mobile application	`[App]`
	Desktop/Monitor	Desktop interface	`[Desktop]`
	Settings/Configuration	System settings	`[Settings]`
	Tools/Utilities	Development tools	`[Tools]`
	Web/Internet	Online resources	`[Web]`
	Storage/Database	Data storage	`[Storage]`

27.3.4. Communication

Emoji	Concept	Usage Context	Alternative Text
	Chat/Conversation	Chatbot, messaging	`[Chat]`
	Email	Email communication	`[Email]`
	Announcement	Notifications, alerts	`[Announcement]`
	Notification	Alerts, reminders	`[Notification]`
	Thought/Idea	Concepts, thinking	`[Idea]`
	Insight/Content	Ideas, solutions	`[Insight]`
	Question	Help, queries	`[Question]`
	Confirmed/Success	Completion, approval	`[Success]`

27.3.5. Data & Analytics

Emoji	Concept	Usage Context	Alternative Text
	Analytics/Growth	Performance metrics	`[Analytics]`

Emoji	Concept	Usage Context	Alternative Text
	Decline/Issues	Problem indicators	`[Decline]`
	Charts/Reports	Data visualization	`[Charts]`
	Target/Goals	Objectives, aims	`[Goal]`
	Location/Point	Specific items	`[Point]`
	Achievement/Success	Top performance	`[Achievement]`
	Rating/Quality	Excellence, favorites	`[Rating]`

27.3.6. Media & Content

Emoji	Concept	Usage Context	Alternative Text
	Video/Recording	Lecture capture	`[Video]`
	Audio/Listening	Audio content, podcasts	`[Audio]`
	Microphone/Speaking	Voice input	`[Microphone]`
	Camera/Photo	Images, screenshots	`[Camera]`
	Image/Picture	Graphics, diagrams	`[Image]`
	Production/Creation	Content creation	`[Production]`

27.3.7. Processes & Actions

Emoji	Concept	Usage Context	Alternative Text
	Sync/Feedback	Refresh, updates	`[Sync]`
	Return/Back	Navigation	`[Back]`
	Forward/Next	Progression	`[Next]`
	Upload/Increase	Upload, improve	`[Upload]`
	Download/Decrease	Download, reduce	`[Download]`
	Search/Find	Discovery, search	`[Search]`
	Link/Connection	Relationships, URLs	`[Link]`
	Pin/Important	Priority items	`[Important]`

27.3.8. Status & Alerts

Emoji	Concept	Usage Context	Alternative Text
	Success/Complete	Task done	`[Complete]`
	Error/Failed	Problems, issues	`[Error]`
	Warning/Caution	Attention needed	`[Warning]`
	Information	General info	`[Info]`
	Launch/Deploy	Go live, release	`[Launch]`
	Security/Private	Protected content	`[Secure]`
	Unlocked/Public	Open access	`[Public]`
	Protection/Safety	Security features	`[Protected]`

27.3.9. Time & Scheduling

Emoji	Concept	Usage Context	Alternative Text
	Calendar/Schedule	Timetables, dates	`[Calendar]`
	Time/Deadline	Due dates, timing	`[Time]`
	Timer/Duration	Time tracking	`[Timer]`
	Clock/Specific Time	Timestamps	`[Clock]`
	Date/Day	Specific dates	`[Date]`

27.4. Usage Guidelines

27.4.1. In Mermaid Diagrams

```
graph LR
    A[ <br/>Instructor] --> B[ <br/>KAI]
    B --> C[ <br/>Analytics]
    B --> D[ <br/>Chatbot]
```

27.4.2. In Documentation Text

Do: - Use emojis to enhance section headers: `##  How Kai Works` - Add emojis to bullet points for visual hierarchy - Include emojis in status messages: `Setup complete`

Don't: - Overuse emojis (maximum 1 per heading, 1 per bullet point) - Use emojis in formal technical specifications - Replace critical text with emojis alone

27.4.3. In User Interface

- **Navigation Icons**: Use consistently across all pages
- **Status Indicators**: for success/error/warning
- **Action Buttons**: Pair with text labels for accessibility

27.5. Accessibility Considerations

1. **Always provide alternative text** for emojis in critical contexts
2. **Use aria-label** attributes in HTML when emojis convey meaning
3. **Don't rely solely on emojis** for important information
4. **Test with screen readers** to ensure content is accessible

27.6. Platform Compatibility

27.6.1. Well-Supported Emojis (Use These)

- Basic symbols:
- Common objects:
- Standard faces:
- Arrows:

27.6.2. Avoid These (Poor Support)

- Complex combined emojis
- Newer Unicode additions
- Platform-specific emojis
- Skin tone modifiers in technical docs

27.7. Implementation Examples

27.7.1. Kai Component Mapping

Based on the core Kai architecture diagram:

```
Components:
  Textbook:
  Instructor:
  AV-Feed:
  KAI:
  Chatbot:
  Formative/Evaluative:
  Content:
  Feedback:
  Analytics:
  Group Chat:
  Students:
  Lecture Notes:
```

27.7.2. Status Messages

```
Success: "  Operation completed successfully"
Warning: "  Please review the following items"
Error: "  Failed to process request"
Info: "  Additional information available"
```

27.8. Maintenance

This guide should be updated when: - New components are added to the system - User feedback indicates confusion with emoji choices - Platform support for emojis changes - New use cases emerge in documentation

27.9. References

- Unicode Emoji List
- Emojipedia - Cross-platform emoji reference
- WCAG Guidelines for Icons

Last Updated: 2025-11-17

28. Analytics Guide

Track student performance, identify trends, and make data-informed decisions

28.1. Overview

Kai's analytics system provides comprehensive insights into student learning, engagement patterns, and class-wide trends. From individual student dashboards to sophisticated performance metrics, these tools help you understand what's working and where students need support.

28.2. Why Analytics Matter

Benefits for Instructors: - Identify struggling students early - Track learning progress over time - Measure intervention effectiveness - Optimize teaching strategies with data - Allocate support resources efficiently - Demonstrate learning outcomes

Benefits for Students: - Visualize their own progress - Understand strengths and weaknesses - Set data-informed goals - Track improvement over time - Stay motivated with tangible metrics

28.3. Quick Start

28.3.1. Accessing Analytics

1. **Navigate**: Dashboard → Analytics
2. **Select scope**: Individual, class, or course-level
3. **Choose timeframe**: Week, month, semester, or custom range
4. **View insights**: Pre-configured dashboards or custom reports

> Start with Overview Dashboard
>
> Begin with the Course Overview dashboard to get a high-level understanding before diving into detailed analytics.

28.4. Student Performance Analytics

28.4.1. Individual Student Dashboards

Comprehensive view of each student's learning journey.

Key Metrics:

```
student_analytics = {
    "student_id": "student_123",
    "course_id": "course_abc",

    "performance_summary": {
        "current_grade": 87.5,
        "grade_trend": "improving",  # improving, stable, declining
        "percentile_in_class": 72,  # Optional: compare to peers

        "by_assessment_type": {
            "quizzes": {"avg": 85.2, "count": 12, "trend": "stable"},
            "assignments": {"avg": 88.3, "count": 8, "trend": "improving"},
            "exams": {"avg": 82.0, "count": 2, "trend": "improving"},
            "participation": {"avg": 92.0, "trend": "stable"}
        },

        "by_topic": {
            "descriptive_statistics": {
                "mastery_level": 0.92,
                "status": "mastered",
                "attempts": 15,
                "time_to_mastery_hours": 4.5
            },
            "inferential_statistics": {
                "mastery_level": 0.78,
                "status": "proficient",
                "attempts": 20,
                "recent_performance": "improving"
            },
            "regression_analysis": {
                "mastery_level": 0.55,
                "status": "developing",
                "attempts": 8,
                "needs_attention": True
            }
        }
    },

    "engagement_metrics": {
        "participation_rate": 0.88,  # % of assignments completed
```

```
        "avg_time_per_assignment_minutes": 45,
        "resource_usage": "high",  # high, medium, low
        "help_seeking_frequency": 3.2,  # times per week
        "peer_interaction_score": 0.75,
        "login_frequency_per_week": 5.2,
        "total_time_on_platform_hours": 32.5
    },

    "learning_patterns": {
        "optimal_study_time": "evening",  # morning, afternoon, evening, night
        "preferred_learning_mode": "visual",
        "common_error_types": [
            "rushing_through_problems",
            "skipping_assumption_checks"
        ],
        "strength_areas": [
            "calculation_accuracy",
            "following_procedures"
        ],
        "growth_areas": [
            "conceptual_interpretation",
            "assumption_verification"
        ]
    },

    "trajectory": {
        "predicted_final_grade": 88.5,
        "confidence_interval": [85.0, 92.0],
        "risk_level": "low",  # low, medium, high
        "intervention_recommended": False,
        "growth_velocity": 0.8  # grade points per week
    }
}

# Access student analytics
analytics = kai.analytics.student.get(
    student_id="student_123",
    course_id="course_abc",
    timeframe="semester"
)

print(f"Current Grade: {analytics.current_grade}")
print(f"Trend: {analytics.grade_trend}")
print(f"At-Risk: {analytics.at_risk}")
```

Dashboard Visualization:

```
STUDENT PERFORMANCE DASHBOARD: Jane Smith
Course: Statistics 101 | Period: Fall 2024


OVERALL PERFORMANCE


Current Grade: 87.5% (B+)        Trend:   Improving (+2.3 pts this month)
Predicted Final: 88.5%           Risk Level:   Low

Grade History:
Week 1-4:          80.2%
Week 5-8:          85.1%
Week 9-12:         87.5%


ASSESSMENT BREAKDOWN


Quizzes:         85.2% (12 completed)
Assignments:     88.3% (8 completed)
Exams:           82.0% (2 completed)
Participation:   92.0%


TOPIC MASTERY


  Descriptive Statistics                92% Mastered
  Inferential Statistics                78% Proficient
  Regression Analysis                   55% Developing
   Hypothesis Testing                    68% Proficient


ENGAGEMENT


Participation Rate:     88%
Time on Platform:       32.5h
Resource Usage:         High
Help Seeking:           3.2/wk


INSIGHTS & RECOMMENDATIONS


Strengths:
• Strong calculation accuracy
```

```
• Excellent participation and engagement
• Consistent upward trajectory

Areas for Growth:
  Regression analysis needs attention (55% mastery)
→ Recommended: Review Module 6, Practice Set 3
→ Suggested: Office hours this week

• Interpretation skills developing
→ Focus on connecting calculations to context
```

28.4.2. Class-Level Performance Analytics

Aggregate metrics for entire class sections.

Class Overview:

```python
class_analytics = {
    "course_id": "course_abc",
    "section": "Section 001",
    "enrollment": 32,
    "timeframe": "current_semester",

    "performance_summary": {
        "class_average": 82.3,
        "median": 84.0,
        "std_dev": 8.5,
        "distribution": {
            "A (90-100)": 8,   # 25%
            "B (80-89)": 14,   # 44%
            "C (70-79)": 7,    # 22%
            "D (60-69)": 2,    # 6%
            "F (0-59)": 1      # 3%
        },
        "trend": "improving",  # +1.2 points vs. last period

        "by_assessment_type": {
            "quizzes": {"avg": 85.5, "completion": 0.94},
            "assignments": {"avg": 83.2, "completion": 0.89},
            "exams": {"avg": 78.9, "completion": 0.97},
            "participation": {"avg": 88.5, "completion": 0.85}
        }
    },

    "engagement_summary": {
        "avg_participation_rate": 0.85,
        "active_users_this_week": 29,  # out of 32
```

```
        "avg_time_per_student_hours": 28.3,
        "resource_access_rate": 0.92,
        "help_requests_per_week": 45,
        "peer_interaction_level": "high"
    },

    "topic_mastery_distribution": {
        "descriptive_statistics": {
            "avg_mastery": 0.87,
            "students_mastered": 28,  # 88%
            "students_struggling": 1   # 3%
        },
        "inferential_statistics": {
            "avg_mastery": 0.75,
            "students_mastered": 20,  # 63%
            "students_struggling": 4   # 13%
        },
        "regression_analysis": {
            "avg_mastery": 0.68,
            "students_mastered": 15,  # 47%
            "students_struggling": 8   # 25%
        }
    },

    "at_risk_students": [
        {
            "student_id": "student_456",
            "risk_level": "high",
            "current_grade": 58.5,
            "primary_issues": ["low_engagement", "multiple_topic_gaps"],
            "last_login": "8_days_ago",
            "intervention_suggested": "immediate_outreach"
        },
        {
            "student_id": "student_789",
            "risk_level": "medium",
            "current_grade": 72.3,
            "primary_issues": ["declining_performance", "reduced_participation"],
            "recent_trend": "declining",
            "intervention_suggested": "check_in_meeting"
        }
    ],

    "high_achievers": [
        {
            "student_id": "student_101",
            "current_grade": 98.2,
```

```
            "mastery_across_topics": 0.96,
            "engagement_level": "very_high",
            "recommendation": "enrichment_opportunities"
        }
    ]
}

# Access class analytics
analytics = kai.analytics.class_summary.get(
    course_id="course_abc",
    section="Section 001",
    timeframe="semester"
)

# Export class report
report = kai.analytics.reports.generate(
    type="class_performance",
    course_id="course_abc",
    format="pdf",  # or "csv", "xlsx"
    include_charts=True
)
```

28.4.3. Cohort Comparison Analytics

Compare performance across sections, semesters, or years.

```
cohort_comparison = kai.analytics.cohorts.compare(
    courses=["course_abc_section_001", "course_abc_section_002"],
    metrics=[
        "average_grade",
        "mastery_rates",
        "engagement_levels",
        "completion_rates"
    ],
    timeframe="semester"
)

"""
COHORT COMPARISON REPORT


                    Section 001    Section 002    Difference
Average Grade       82.3%          79.8%          +2.5%
Median Grade        84.0%          81.5%          +2.5%
Std Dev             8.5            11.2           -2.7 (more consistent)
```

```
Completion Rate     89%             84%             +5%
Participation Rate  85%             78%             +7%

Topic Mastery:
- Descriptive       87%             82%             +5%
- Inferential       75%             71%             +4%
- Regression        68%             65%             +3%

At-Risk Students    2 (6%)          5 (16%)         -10%
High Achievers      8 (25%)         6 (19%)         +6%

Key Differences:
• Section 001 shows higher engagement across all metrics
• Section 001 has more consistent performance (lower std dev)
• Both sections struggle similarly with regression analysis
"""
```

28.5. Learning Trend Analysis

28.5.1. Progress Over Time

Track how students develop throughout the course.

Longitudinal Analysis:

```
trend_analysis = {
    "course_id": "course_abc",
    "analysis_type": "longitudinal",
    "timeframe": "full_semester",

    "class_wide_trends": {
        "grade_trajectory": {
            "week_1_4": {"avg": 78.5, "std_dev": 12.3},
            "week_5_8": {"avg": 81.2, "std_dev": 10.8},
            "week_9_12": {"avg": 83.7, "std_dev": 9.2},
            "week_13_16": {"avg": 84.5, "std_dev": 8.5},
            "overall_improvement": 6.0,  # points
            "trend_consistency": 0.88  # how smooth the improvement
        },

        "mastery_progression": {
            "early_concepts": {
                "time_to_mastery_avg_days": 12,
                "mastery_rate": 0.94
            },
            "mid_concepts": {
```

```
                "time_to_mastery_avg_days": 15,
                "mastery_rate": 0.81
            },
            "advanced_concepts": {
                "time_to_mastery_avg_days": 18,
                "mastery_rate": 0.72
            }
        },

        "engagement_trends": {
            "early_semester": {"participation": 0.92, "time_per_week_hrs": 5.2},
            "mid_semester": {"participation": 0.85, "time_per_week_hrs": 4.8},
            "late_semester": {"participation": 0.78, "time_per_week_hrs": 4.5},
            "trend": "declining_engagement",  # Common pattern
            "intervention_points": ["week_10", "week_14"]
        }
    },

    "student_trajectories": {
        "steady_improvers": {
            "count": 18,  # 56%
            "pattern": "consistent_upward_trajectory",
            "avg_gain": 8.5,  # points
            "success_rate": 0.94
        },
        "quick_starters": {
            "count": 7,  # 22%
            "pattern": "high_early_maintain",
            "avg_starting_grade": 88.5,
            "avg_final_grade": 90.2
        },
        "late_bloomers": {
            "count": 4,  # 13%
            "pattern": "struggle_then_improve",
            "avg_starting_grade": 68.2,
            "avg_final_grade": 82.5,
            "turning_point": "week_7"
        },
        "concerning_patterns": {
            "count": 3,  # 9%
            "pattern": "declining_performance",
            "needs_intervention": True
        }
    },

    "topic_difficulty_profile": {
        "easiest_topics": [
```

```
            {"topic": "descriptive_statistics", "avg_mastery": 0.92, "time_to_master": 10},
            {"topic": "data_visualization", "avg_mastery": 0.89, "time_to_master": 8}
        ],
        "moderate_topics": [
            {"topic": "probability", "avg_mastery": 0.78, "time_to_master": 14},
            {"topic": "sampling", "avg_mastery": 0.75, "time_to_master": 13}
        ],
        "challenging_topics": [
            {"topic": "hypothesis_testing", "avg_mastery": 0.70, "time_to_master": 18},
            {"topic": "regression_analysis", "avg_mastery": 0.68, "time_to_master": 20}
        ]
    }
}

# Visualize trends
trends = kai.analytics.trends.analyze(
    course_id="course_abc",
    dimension="grade_over_time",
    group_by="student_trajectory_type"
)

# Generate insight report
insights = kai.analytics.insights.generate(
    course_id="course_abc",
    focus_areas=["struggling_students", "topic_difficulties", "engagement_patterns"]
)
```

28.5.2. Predictive Analytics

Identify students who may need support before they fall behind.

```
predictive_model = {
    "model_type": "early_warning_system",
    "prediction_window": "4_weeks_ahead",

    "risk_factors": [
        {
            "factor": "declining_quiz_scores",
            "weight": 0.25,
            "threshold": "3_consecutive_declines"
        },
        {
            "factor": "reduced_engagement",
            "weight": 0.20,
            "threshold": "below_60%_participation_2_weeks"
        },
```

```
        {
            "factor": "incomplete_assignments",
            "weight": 0.20,
            "threshold": "2_missed_in_row"
        },
        {
            "factor": "low_resource_usage",
            "weight": 0.15,
            "threshold": "below_25th_percentile"
        },
        {
            "factor": "help_seeking_absence",
            "weight": 0.10,
            "threshold": "no_help_requests_3_weeks"
        },
        {
            "factor": "peer_interaction_low",
            "weight": 0.10,
            "threshold": "minimal_collaboration"
        }
    ],

    "predictions": [
        {
            "student_id": "student_456",
            "current_grade": 75.2,
            "predicted_grade_4_weeks": 68.5,
            "risk_level": "high",
            "confidence": 0.87,
            "primary_risk_factors": [
                "declining_quiz_scores",
                "reduced_engagement"
            ],
            "recommended_actions": [
                "immediate_outreach",
                "suggest_tutoring",
                "adjust_workload_if_needed"
            ]
        },
        {
            "student_id": "student_789",
            "current_grade": 82.0,
            "predicted_grade_4_weeks": 79.3,
            "risk_level": "medium",
            "confidence": 0.72,
            "primary_risk_factors": [
                "incomplete_assignments"
```

```
            ],
            "recommended_actions": [
                "check_in_email",
                "review_assignment_schedule"
            ]
        }
    ],

    "model_performance": {
        "accuracy": 0.83,  # % of predictions within 5 points
        "precision": 0.79,  # % of flagged students who actually needed help
        "recall": 0.88,     # % of struggling students successfully identified
        "false_positive_rate": 0.12
    }
}

# Get predictions for your class
predictions = kai.analytics.predictions.get(
    course_id="course_abc",
    risk_levels=["high", "medium"],
    confidence_threshold=0.70
)

# Configure alerts
kai.analytics.predictions.configure_alerts(
    course_id="course_abc",
    alert_when="high_risk_detected",
    notification_method="email",
    frequency="daily_digest"
)
```

28.6. Engagement Metrics

28.6.1. Participation Analytics

Track how students interact with course materials and activities.

```
engagement_analytics = {
    "course_id": "course_abc",
    "timeframe": "current_semester",

    "overall_engagement": {
        "active_users_today": 28,
        "active_users_this_week": 31,
        "active_users_this_month": 32,
```

```
        "avg_sessions_per_week": 4.2,
        "avg_session_duration_minutes": 38.5,
        "avg_time_on_platform_per_week_hours": 3.2,

        "peak_usage_times": {
            "day_of_week": "Tuesday",
            "time_of_day": "7-9 PM",
            "secondary_peak": "Sunday 2-4 PM"
        }
    },

    "activity_breakdown": {
        "quiz_completion_rate": 0.94,
        "assignment_submission_rate": 0.89,
        "resource_access_rate": 0.87,
        "discussion_participation_rate": 0.72,
        "office_hours_attendance_rate": 0.45,

        "by_activity_type": {
            "watching_videos": {"avg_time_min": 45, "completion_rate": 0.88},
            "reading_materials": {"avg_time_min": 32, "completion_rate": 0.82},
            "practice_problems": {"avg_time_min": 55, "completion_rate": 0.91},
            "interactive_tutorials": {"avg_time_min": 28, "completion_rate": 0.85}
        }
    },

    "engagement_segments": {
        "highly_engaged": {
            "count": 12,  # 38%
            "characteristics": {
                "participation_rate": 0.95,
                "time_per_week_hours": 5.2,
                "resource_usage": "comprehensive",
                "avg_grade": 89.5
            }
        },
        "moderately_engaged": {
            "count": 16,  # 50%
            "characteristics": {
                "participation_rate": 0.85,
                "time_per_week_hours": 3.8,
                "resource_usage": "selective",
                "avg_grade": 82.3
            }
        },
        "low_engagement": {
            "count": 4,  # 13%
```

```
            "characteristics": {
                "participation_rate": 0.58,
                "time_per_week_hours": 1.9,
                "resource_usage": "minimal",
                "avg_grade": 71.2
            },
            "needs_attention": True
        }
    },

    "engagement_correlation": {
        "participation_vs_grade": {
            "correlation": 0.68,
            "p_value": 0.001,
            "interpretation": "strong_positive_relationship"
        },
        "time_on_platform_vs_grade": {
            "correlation": 0.52,
            "p_value": 0.012,
            "interpretation": "moderate_positive_relationship"
        },
        "resource_usage_vs_grade": {
            "correlation": 0.61,
            "p_value": 0.003,
            "interpretation": "moderate_positive_relationship"
        }
    }
}

# Generate engagement report
engagement_report = kai.analytics.engagement.report(
    course_id="course_abc",
    include_recommendations=True
)
```

28.6.2. Resource Usage Analytics

Understand which resources students find most valuable.

```
resource_analytics = {
    "resource_usage": {
        "lecture_videos": {
            "total_views": 1234,
            "unique_viewers": 30,
            "avg_watch_time_pct": 0.87,  # 87% of video watched on average
            "completion_rate": 0.82,
```

```
            "most_rewatched_sections": [
                {"timestamp": "12:30", "topic": "p_value_interpretation", "rewatch_count": 45},
                {"timestamp": "18:15", "topic": "confidence_intervals", "rewatch_count": 38}
            ]
        },

        "readings": {
            "total_opens": 892,
            "unique_readers": 29,
            "avg_time_spent_minutes": 18.5,
            "completion_rate": 0.78,
            "most_highlighted": [
                {"page": 42, "topic": "hypothesis_testing", "highlights": 52},
                {"page": 67, "topic": "type_I_II_errors", "highlights": 48}
            ]
        },

        "practice_problems": {
            "total_attempts": 2156,
            "unique_users": 32,
            "avg_attempts_per_problem": 2.3,
            "avg_completion_time_minutes": 12.5,
            "most_attempted": [
                {"problem_id": "prob_45", "topic": "regression", "attempts": 78},
                {"problem_id": "prob_32", "topic": "hypothesis_test", "attempts": 65}
            ]
        },

        "supplementary_materials": {
            "access_rate": 0.67,
            "most_accessed": [
                {"resource": "formula_sheet", "accesses": 245, "users": 31},
                {"resource": "worked_examples", "accesses": 189, "users": 28},
                {"resource": "tutorial_videos", "accesses": 156, "users": 24}
            ]
        }
    },

    "resource_effectiveness": {
        "high_impact": [
            {
                "resource": "worked_examples_regression",
                "usage_by_struggling_students": 0.95,
                "avg_improvement_after_use": 12.5,  # grade points
                "effectiveness_score": 0.89
            },
            {
```

```
                "resource": "interactive_hypothesis_testing_simulator",
                "usage_rate": 0.78,
                "avg_improvement_after_use": 8.3,
                "effectiveness_score": 0.82
            }
        ],

        "moderate_impact": [
            {
                "resource": "textbook_chapter_summaries",
                "usage_rate": 0.72,
                "avg_improvement_after_use": 4.5,
                "effectiveness_score": 0.68
            }
        ],

        "low_impact": [
            {
                "resource": "additional_reading_articles",
                "usage_rate": 0.23,
                "avg_improvement_after_use": 1.2,
                "effectiveness_score": 0.34,
                "recommendation": "consider_removing_or_replacing"
            }
        ]
    }
}

# Find most effective resources
effective_resources = kai.analytics.resources.effectiveness(
    course_id="course_abc",
    metric="improvement_after_use",
    min_usage_threshold=10  # Must be used by at least 10 students
)
```

28.7. Individual Student Dashboards

28.7.1. Student-Facing Analytics

Empower students to track their own progress.

Student View Configuration:

```
student_dashboard_config = {
    "visible_to_students": True,
```

```
    "components": [
        {
            "component": "progress_overview",
            "display": {
                "current_grade": True,
                "grade_trend": True,
                "goal_comparison": True,
                "class_percentile": False  # Avoid social comparison
            }
        },
        {
            "component": "topic_mastery",
            "display": {
                "mastery_levels": True,
                "progress_bars": True,
                "recommended_focus": True,
                "time_to_mastery_estimates": True
            }
        },
        {
            "component": "engagement_insights",
            "display": {
                "participation_rate": True,
                "time_on_platform": True,
                "resource_usage": True,
                "study_pattern_insights": True
            }
        },
        {
            "component": "personalized_recommendations",
            "display": {
                "next_steps": True,
                "resource_suggestions": True,
                "study_plan": True,
                "predicted_performance": False  # Optional: some students find stressful
            }
        },
        {
            "component": "achievements",
            "display": {
                "milestones_reached": True,
                "improvement_highlights": True,
                "consistency_badges": True
            }
        }
    ],
```

```
    "privacy_controls": {
        "student_can_hide_components": True,
        "student_can_export_data": True,
        "data_retention_visible": True
    },

    "motivational_features": {
        "show_improvement_over_time": True,
        "celebrate_milestones": True,
        "growth_mindset_framing": True,
        "avoid_peer_comparison": True
    }
}

# Enable student dashboards
kai.analytics.student_dashboards.enable(
    course_id="course_abc",
    config=student_dashboard_config
)

"""
STUDENT VIEW EXAMPLE

Your Progress Dashboard


Current Grade: 87.5% (B+)
Trend:  Up 2.3 points this month - Great progress!

Your Goals:
• Target for course: 90% (A-)
• Gap to close: 2.5 points
• On track: Yes
• Estimated weeks to goal: 3 weeks at current pace


What You've Mastered


 Descriptive Statistics       92% Excellent!
 Data Visualization           88% Strong
 Inferential Statistics       78% Good - keep practicing
 Regression Analysis          55% Developing - focus here


Recommended Next Steps
```

```
1. Focus on Regression Analysis (55% mastery)
   → Review Module 6: Linear Regression
   → Complete Practice Set 3 (15 problems)
   → Estimated time: 45 minutes

2. Strengthen Interpretation Skills
   → Watch: "Connecting Calculations to Context" (8 min)
   → Try: Interpretation Quiz (10 questions)

3. Prepare for Upcoming Exam
   → Your exam is in 6 days
   → Personalized study plan ready
   → View study plan →


Recent Achievements


  Perfect Quiz Score - Quiz 12
  3-Week Improvement Streak
  All Assignments On Time This Month
"""
```

28.7.2. Growth Tracking

Help students visualize learning progress.

```python
growth_tracking = {
    "student_id": "student_123",
    "timeframe": "semester",

    "grade_trajectory": [
        {"week": 1, "grade": 75.2, "milestone": ""},
        {"week": 2, "grade": 76.8, "milestone": ""},
        {"week": 3, "grade": 78.5, "milestone": ""},
        {"week": 4, "grade": 80.1, "milestone": "First quiz mastery"},
        {"week": 5, "grade": 81.3, "milestone": ""},
        {"week": 6, "grade": 82.8, "milestone": ""},
        {"week": 7, "grade": 84.2, "milestone": "Midterm success"},
        {"week": 8, "grade": 85.0, "milestone": ""},
        {"week": 9, "grade": 86.3, "milestone": ""},
        {"week": 10, "grade": 87.5, "milestone": "Current"},
    ],

    "growth_metrics": {
        "total_improvement": 12.3,  # grade points
```

```
        "avg_weekly_growth": 1.4,
        "growth_consistency": 0.91,  # How steady the improvement
        "best_growth_period": "weeks_6_8",
        "growth_velocity": "accelerating"
    },

    "skill_development": {
        "calculation_skills": {
            "start": 0.68,
            "current": 0.92,
            "growth": 0.24,
            "status": "mastered"
        },
        "interpretation_skills": {
            "start": 0.45,
            "current": 0.78,
            "growth": 0.33,
            "status": "significant_improvement"
        },
        "application_skills": {
            "start": 0.52,
            "current": 0.71,
            "growth": 0.19,
            "status": "steady_progress"
        }
    }
}
```

28.8. Data Export Capabilities

28.8.1. Export Formats

Generate reports in various formats for different needs.

```
# Export student performance data
student_export = kai.analytics.export(
    type="student_performance",
    course_id="course_abc",
    format="csv",  # csv, xlsx, json, pdf
    students="all",  # or list of student IDs
    fields=[
        "student_id",
        "current_grade",
        "quiz_avg",
        "assignment_avg",
        "participation_rate",
```

```
        "mastery_by_topic"
    ],
    timeframe="semester"
)

# Export class summary
class_export = kai.analytics.export(
    type="class_summary",
    course_id="course_abc",
    format="pdf",
    include_charts=True,
    sections=[
        "performance_overview",
        "grade_distribution",
        "topic_mastery",
        "engagement_metrics",
        "at_risk_students"
    ]
)

# Export for LMS integration
lms_export = kai.analytics.export(
    type="gradebook",
    course_id="course_abc",
    format="csv",
    lms_compatible="canvas",  # canvas, blackboard, moodle, etc.
    include_categories=True
)

# Export detailed analytics
detailed_export = kai.analytics.export(
    type="comprehensive",
    course_id="course_abc",
    format="xlsx",
    sheets=[
        "student_roster",
        "grade_history",
        "assignment_details",
        "quiz_results",
        "engagement_logs",
        "resource_usage",
        "topic_mastery"
    ],
    timeframe="semester"
)

# Custom query export
```

```
custom_export = kai.analytics.export(
    type="custom_query",
    query={
        "metrics": ["grade", "participation", "time_on_platform"],
        "filters": {
            "grade_below": 75,
            "participation_below": 0.70
        },
        "group_by": "week",
        "aggregation": "average"
    },
    format="json"
)
```

28.8.2. Scheduled Reports

Automate regular analytics delivery.

```
scheduled_report = kai.analytics.schedules.create(
    name="Weekly Performance Summary",
    report_type="class_performance",
    course_id="course_abc",

    schedule={
        "frequency": "weekly",
        "day": "Monday",
        "time": "08:00",
        "timezone": "America/New_York"
    },

    delivery={
        "method": "email",
        "recipients": ["instructor@university.edu"],
        "subject": "Weekly Analytics: Statistics 101",
        "include_attachments": True
    },

    content={
        "sections": [
            "class_average_trend",
            "at_risk_students",
            "topic_difficulties",
            "engagement_summary"
        ],
        "format": "pdf",
        "include_visualizations": True
```

```
    },

    alerts={
        "highlight_if": {
            "at_risk_count_above": 3,
            "class_average_below": 75,
            "participation_below": 0.70
        }
    }
)

# List active schedules
schedules = kai.analytics.schedules.list(course_id="course_abc")

# Pause schedule temporarily
kai.analytics.schedules.pause(schedule_id="schedule_123")
```

28.9. Integration with LMS Analytics

28.9.1. LMS Sync

Combine Kai analytics with your institution's LMS data.

```
lms_integration = {
    "lms_type": "canvas",  # canvas, blackboard, moodle, etc.
    "sync_enabled": True,

    "data_flow": {
        "from_lms_to_kai": [
            "student_roster",
            "assignment_due_dates",
            "external_grades",
            "course_structure"
        ],
        "from_kai_to_lms": [
            "kai_assignment_grades",
            "quiz_scores",
            "participation_scores"
        ]
    },

    "sync_frequency": "daily",

    "grade_mapping": {
        "kai_quizzes": {
```

```
            "lms_category": "Quizzes",
            "weight": 0.30
        },
        "kai_assignments": {
            "lms_category": "Assignments",
            "weight": 0.40
        },
        "kai_participation": {
            "lms_category": "Participation",
            "weight": 0.10
        }
    },

    "analytics_integration": {
        "combine_kai_and_lms_data": True,
        "unified_dashboard": True,
        "cross_platform_insights": True
    }
}

# Configure LMS integration
kai.integrations.lms.configure(
    course_id="course_abc",
    lms_course_id="canvas_12345",
    config=lms_integration
)

# Fetch combined analytics
combined_analytics = kai.analytics.integrated.get(
    course_id="course_abc",
    include_lms_data=True,
    timeframe="semester"
)
```

28.9.2. Cross-Platform Reports

Generate reports that combine multiple data sources.

```
unified_report = kai.analytics.unified.generate(
    course_id="course_abc",

    data_sources=[
        "kai_analytics",
        "lms_gradebook",
        "attendance_system",
        "library_usage"
```

```
    ],

    report_sections=[
        {
            "section": "comprehensive_performance",
            "includes": {
                "kai_quiz_scores": True,
                "lms_assignment_scores": True,
                "attendance_rate": True,
                "combined_average": True
            }
        },
        {
            "section": "engagement_composite",
            "includes": {
                "kai_participation": True,
                "lms_discussion_posts": True,
                "attendance": True,
                "library_resource_usage": True
            }
        }
    ],

    format="pdf",
    include_recommendations=True
)
```

28.10. Privacy and Data Handling

28.10.1. Data Privacy Compliance

Ensure analytics comply with privacy regulations.

> **!** Privacy First
>
> All analytics features comply with FERPA, GDPR, and institutional privacy policies. Student data is never shared without proper consent.

Privacy Configuration:

```
privacy_settings = {
    "data_collection": {
        "collect_performance_data": True,
        "collect_engagement_data": True,
        "collect_behavioral_data": True,
        "student_consent_required": True,
```

```
        "allow_opt_out": True
    },

    "data_retention": {
        "active_course_data": "duration_of_course_plus_1_year",
        "historical_data": "5_years",
        "anonymize_after": "course_completion",
        "student_can_request_deletion": True
    },

    "data_sharing": {
        "share_with_instructor": True,
        "share_with_institution": "aggregated_only",
        "share_with_third_parties": False,
        "student_controls_visibility": True
    },

    "access_controls": {
        "instructor_access": "own_courses_only",
        "admin_access": "aggregated_reports_only",
        "student_access": "own_data_only",
        "audit_logging": True
    },

    "transparency": {
        "privacy_policy_visible": True,
        "data_usage_explained": True,
        "student_dashboard_shows_what_data_collected": True,
        "student_can_export_own_data": True
    }
}

# Apply privacy settings
kai.privacy.configure(
    course_id="course_abc",
    settings=privacy_settings
)

# Generate privacy compliance report
compliance = kai.privacy.audit(
    course_id="course_abc",
    frameworks=["FERPA", "GDPR"]
)
```

28.10.2. Student Data Rights

Support student rights to access and control their data.

```
# Student data export (self-service)
student_data = kai.privacy.export_student_data(
    student_id="student_123",
    format="json",
    include=[
        "profile_data",
        "performance_history",
        "engagement_logs",
        "resource_usage",
        "system_interactions"
    ]
)

# Student data deletion request
kai.privacy.delete_student_data(
    student_id="student_123",
    course_id="course_abc",
    retention_override="delete_immediately",
    keep_anonymized_aggregate=True  # For institutional research
)

# Student consent management
consent = kai.privacy.consent.update(
    student_id="student_123",
    consents={
        "basic_analytics": True,
        "predictive_analytics": True,
        "research_participation": False
    }
)
```

28.11. Best Practices

28.11.1. Effective Analytics Use

> **Start with Questions**
>
> Begin with specific questions you want to answer, then find the relevant analytics. Avoid drowning in data without clear purpose.

Recommended Workflow:

1. **Identify Questions**:
   - "Which students are struggling?"
   - "Which topics are most challenging?"

- "Is engagement declining over time?"
- "Are interventions working?"

2. **Select Relevant Metrics**:
   - Match analytics to questions
   - Focus on actionable insights
   - Avoid vanity metrics
3. **Set Regular Review Times**:
   - Weekly: At-risk student check
   - Biweekly: Topic difficulty review
   - Monthly: Overall trend analysis
   - End-of-semester: Comprehensive evaluation
4. **Act on Insights**:
   - Don't just observe - intervene
   - Document what works
   - Adjust strategies based on data
   - Close the feedback loop

28.11.2. Interpreting Analytics Wisely

Guidelines:

- **Context Matters**: Consider external factors (holidays, exam weeks, etc.)
- **Trends Over Snapshots**: Look for patterns, not single data points
- **Combine Quantitative and Qualitative**: Numbers + student feedback
- **Respect Privacy**: Never publicly compare students
- **Avoid**: Over-relying on predictions without human judgment
- **Avoid**: Labeling students based on early data
- **Avoid**: Using analytics punitively

28.11.3. Data-Informed, Not Data-Driven

Analytics inform decisions but don't replace professional judgment:

```
decision_framework = {
    "analytics_indicates": "Student X predicted to fail",

    "appropriate_response": [
        "Reach out with support offer",
        "Discuss challenges and barriers",
        "Provide resources and options",
        "Collaborate on success plan"
    ],
```

```
    "inappropriate_response": [
        "Give up on student",
        "Reduce expectations",
        "Label as 'lost cause'",
        "Share prediction with student publicly"
    ]
}
```

28.12. Troubleshooting

28.12.1. Missing Data

Problem: Analytics showing incomplete or missing data

Common Causes: - Recent course setup (insufficient data collected) - Student privacy settings blocking data collection - Integration issues with LMS - Students not engaging with platform

Solutions:

```
# Diagnose missing data
diagnosis = kai.analytics.diagnose(course_id="course_abc")

# Check data completeness
completeness = kai.analytics.data_quality.check(
    course_id="course_abc",
    expected_students=32,
    expected_activities=15
)

# Verify integrations
kai.integrations.status(course_id="course_abc")
```

28.12.2. Inaccurate Predictions

Problem: Predictive analytics not matching reality

Solutions: - Allow 3-4 weeks for model calibration - Ensure sufficient historical data - Review and adjust risk factor weights - Validate predictions against outcomes - Consider class-specific factors

28.12.3. Overwhelming Amount of Data

Problem: Too many metrics to track effectively

Solutions:

```
# Create focused dashboard
kai.analytics.dashboards.create_custom(
    name="Essential Metrics Only",
    course_id="course_abc",
    widgets=[
        "at_risk_students",
        "class_average_trend",
        "topic_difficulties_top_3"
    ]
)
```

28.12.4. Privacy Concerns

Problem: Students or institution worried about data collection

Solutions: - Share privacy policy clearly - Enable student data dashboards (transparency) - Allow opt-out options - Use aggregated data when possible - Conduct privacy audit

28.13. Support Resources

- **Analytics Training**: Schedule workshop
- **Research Library**: Evidence base for analytics
- **Email Support**: analytics@chi2labs.com
- **Community**: Analytics best practices forum

Next Steps: - Set up personalization features informed by analytics - Configure grading to track performance metrics - Review best practices for data-informed teaching - Explore integration options to combine analytics sources