

---

# **DESIGN AND IMPLEMENTATION FOR**

## **Kaiju Academy**

**Version 1.1**

**Prepared by  
Group A2**

<b>YU Ching Hei</b>	<b>1155193237</b>	<b>chyu@link.cuhk.edu.hk</b>
<b>Lei Hei Tung</b>	<b>1155194969</b>	<b>1155194969@link.cuhk.edu.hk</b>
<b>Ankhbayar Enkhtaivan</b>	<b>1155185142</b>	<b>1155185142@link.cuhk.edu.hk</b>
<b>Yum Ho Kan</b>	<b>1155195234</b>	<b>1155195234@link.cuhk.edu.hk</b>
<b>Leung Chung Wang</b>	<b>1155194650</b>	<b>1155194650@link.cuhk.edu.hk</b>

**The Chinese University of Hong Kong  
Department of Computer Science and Engineering  
CSCI3100: Software Engineering**

**May 7, 2025**

# Contents

<b>Contents</b>	<b>ii</b>
<b>Document Revision History</b>	<b>iv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Summary of Requirements . . . . .	1
1.2 Quality Goals . . . . .	1
1.3 Stakeholders . . . . .	1
<b>2 System Architecture</b>	<b>2</b>
2.1 Major Components . . . . .	2
2.2 Component Relationships . . . . .	2
2.3 Database . . . . .	2
2.4 Security . . . . .	2
2.5 User Authentication and Authorization . . . . .	3
2.6 Testability . . . . .	3
<b>3 Data Models</b>	<b>4</b>
3.1 Database Schema . . . . .	4
3.2 Request and Response Structure . . . . .	4
3.3 Data Flow Sequence . . . . .	4
<b>4 Interface Design</b>	<b>6</b>
4.1 User Registration and Authentication . . . . .	6
4.1.1 Course Management . . . . .	7
4.1.2 Course Access . . . . .	8
4.1.3 Code Assessment and Grading . . . . .	8
4.2 Service Communication . . . . .	8
4.3 Security Implementation . . . . .	8
4.4 Exception Handling . . . . .	9
4.4.1 Expected Exceptions . . . . .	9
4.4.2 Error Handling Example . . . . .	9
4.4.3 Credit Purchase and Payment Flow . . . . .	10
4.4.4 Credit Purchase and Course Enrollment Flow . . . . .	10
4.4.5 Credit Purchase and Enrollment Logic (Mermaid Diagram) . . . . .	10
<b>5 Component Design</b>	<b>11</b>
5.1 Frontend Components . . . . .	11
5.1.1 Navbar Component . . . . .	11
5.1.2 Code Editor Component . . . . .	11
5.2 Backend Components . . . . .	11
5.2.1 Authentication Service . . . . .	11
5.2.2 Course Management Service . . . . .	11
5.2.3 Assessment Service . . . . .	11
<b>6 User Interface Design</b>	<b>12</b>
6.1 Design Principles . . . . .	12
6.2 User-wise Navigation Flow . . . . .	12
6.2.1 Student . . . . .	12
6.2.2 Educator . . . . .	13
6.3 Interface Storyboards . . . . .	13
6.3.1 Authentication Flow . . . . .	13

6.3.2	Student Learning Journey . . . . .	14
6.3.3	Learning Environment . . . . .	14
6.4	Accessibility Considerations . . . . .	14
6.5	Responsive Design . . . . .	14
<b>7</b>	<b>Assumptions</b>	<b>16</b>
7.1	Technical Constraints . . . . .	16
7.1.1	Hardware Constraints . . . . .	16
7.1.2	Software Constraints . . . . .	16
7.2	Operational Assumptions . . . . .	16
7.3	Dependencies . . . . .	16
7.3.1	Third-Party Services . . . . .	16

## Document Revision History

Version	Revised By	Revision Date	Comments
0.1	Group A2	2025-02-27	Added: –Initial document structure –Basic content outline
1.0	Group A2	2025-03-11	Updated: –Final review and integration
1.1	C. W. Leung	2025-04-23	Updated: –Pruned and updated per requirements in meeting1.docx

# 1. Introduction

Kaiju Academy is a **web-based e-learning platform** designed to make learning programming accessible and engaging. It combines modern Learning Management System (LMS) capabilities with interactive coding features, enabling users to learn at their own pace.

## 1.1 Summary of Requirements

Kaiju Academy includes:

- **User Management & Authentication:** Role-based access control for users (student), teachers (educator), and admins, with secure authentication and optional MFA.
- **Course Creation & Management:** Tools for creating, updating, and organizing courses with videos, PDFs, quizzes, and coding assessments.
- **Interactive Learning:** Real-time code execution and automated grading via an online code editor.
- **Profile and Progress:** User and educator profiles, progress tracking, recommended and registered courses.
- **Monetization:** Course credit system for payment and buying credits to unlock courses.

**Removed:** Forum, dashboard, code competition, notification, calendar, data backup, keyboard shortcut, screen reader, color blindness features, and forum admin role.

## 1.2 Quality Goals

Goal	Description
<b>Performance</b>	<ul style="list-style-type: none"> <li>- Response time &lt;2 seconds for 95% of users.</li> <li>- Code execution results in &lt;5 seconds for 99% of submissions.</li> </ul>
<b>Scalability</b>	<ul style="list-style-type: none"> <li>- Support 10,000 concurrent users.</li> <li>- Horizontal scaling via AWS Lambda and SurrealDB sharding.</li> </ul>
<b>Reliability</b>	<ul style="list-style-type: none"> <li>- 99.9% uptime with automatic failover.</li> <li>- Recovery from failures within 5 minutes.</li> </ul>
<b>Security</b>	<ul style="list-style-type: none"> <li>- AES-256 encryption for data at rest.</li> <li>- TLS 1.2+ for data in transit.</li> <li>- Role-based permissions and MFA.</li> </ul>

**Table 1.1:** Quality Goals for Kaiju Academy

## 1.3 Stakeholders

Stakeholder	Role & Responsibilities
<b>Admin</b>	<ul style="list-style-type: none"> <li>- Manage system health, user roles, and platform configurations.</li> <li>- Enforce security policies.</li> </ul>
<b>User (Student)</b>	<ul style="list-style-type: none"> <li>- Enroll in courses, complete assessments, and track progress.</li> </ul>
<b>Teacher (Educator)</b>	<ul style="list-style-type: none"> <li>- Create and update course content.</li> <li>- Grade submissions and provide feedback.</li> </ul>

**Table 1.2:** Stakeholders and Their Roles

## 2. System Architecture

### 2.1 Major Components

**Frontend Application:** A React application providing the user interface across all devices.

**Backend Services:** Rust-based AWS Lambda functions for service logic.

**SurrealDB:** Primary database for persistent data.

**AWS Services:** API Gateway, Lambda, Cognito (auth/MFA), S3 (media), CloudFront (CDN), Fargate (code execution).

### 2.2 Component Relationships

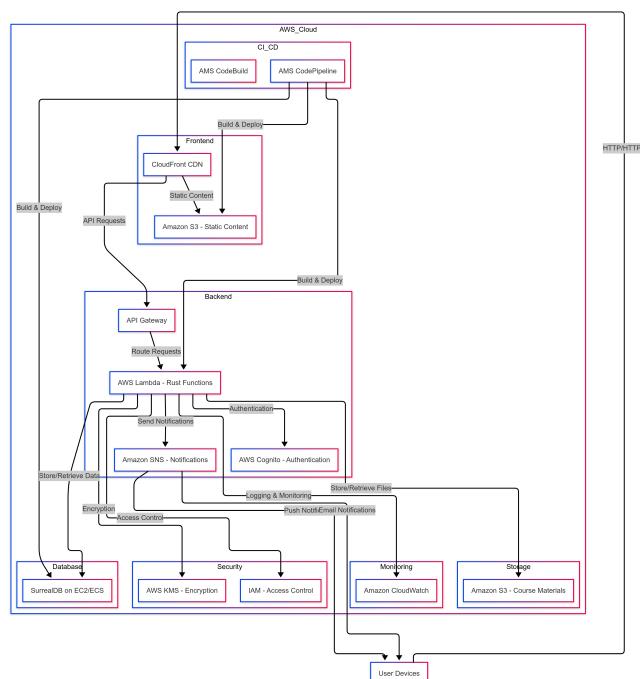


Figure 2.1: AWS Serverless Deployment Architecture

- Users access via browsers, routed through CloudFront CDN and API Gateway.
- Lambda functions handle business logic, SurrealDB stores data.
- Cognito manages authentication and optional MFA.
- S3 stores learning materials.

### 2.3 Database

SurrealDB's multi-model database supports course, user, credit/payment, progress, and assessment data.

### 2.4 Security

- AWS-managed VPC and IAM boundaries.
- AES-256 encryption at rest, TLS 1.2+ in transit.
- JWT for stateless API authentication, MFA support via Cognito.

## 2.5 User Authentication and Authorization

- Email/password registration and login, MFA if supported.
- JWT tokens for session management.
- Roles: user (student), teacher (educator), admin.

## 2.6 Testability

- Automated unit and integration tests in Rust.
- CI/CD pipeline with test coverage enforcement.

## 3. Data Models

### 3.1 Database Schema

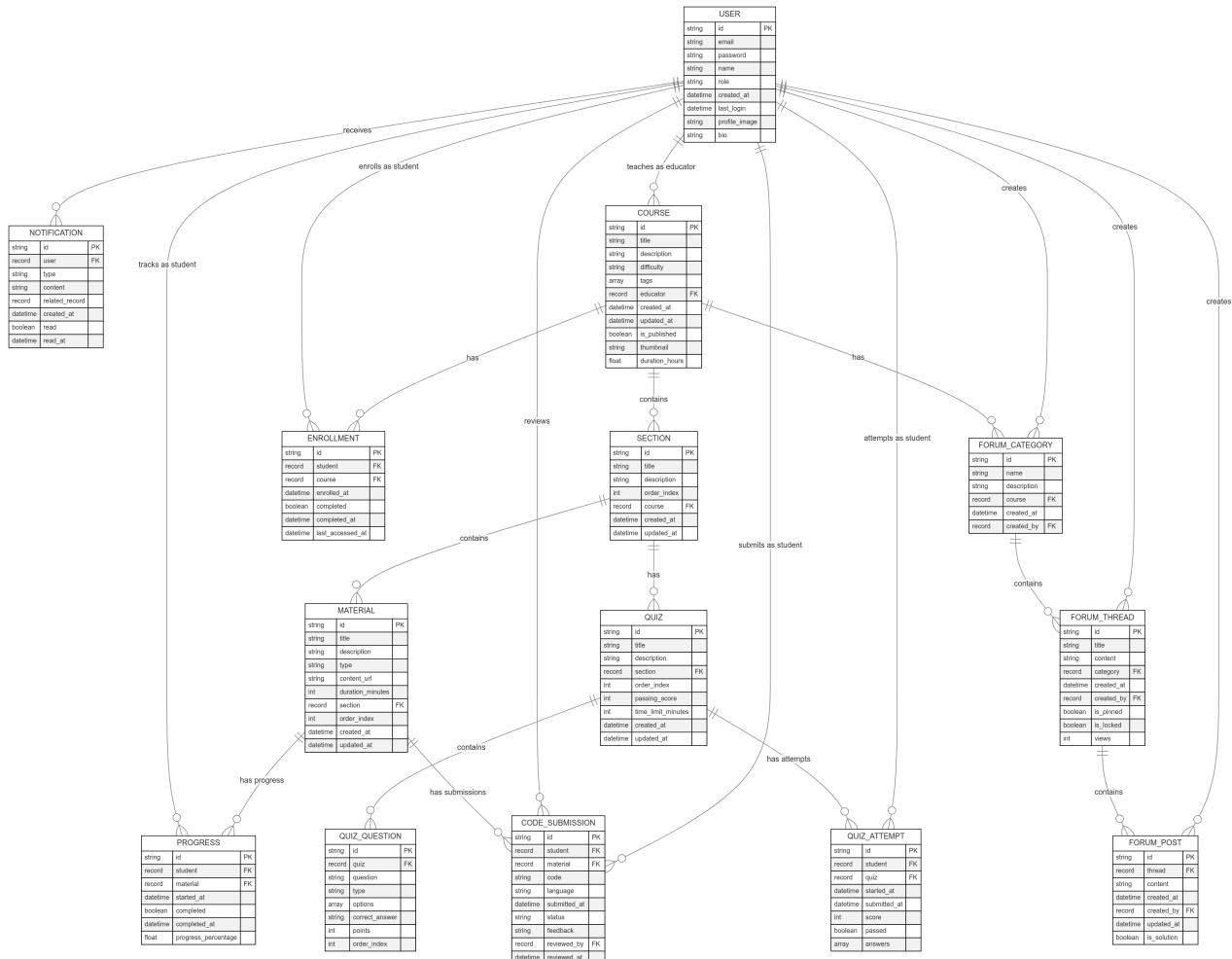


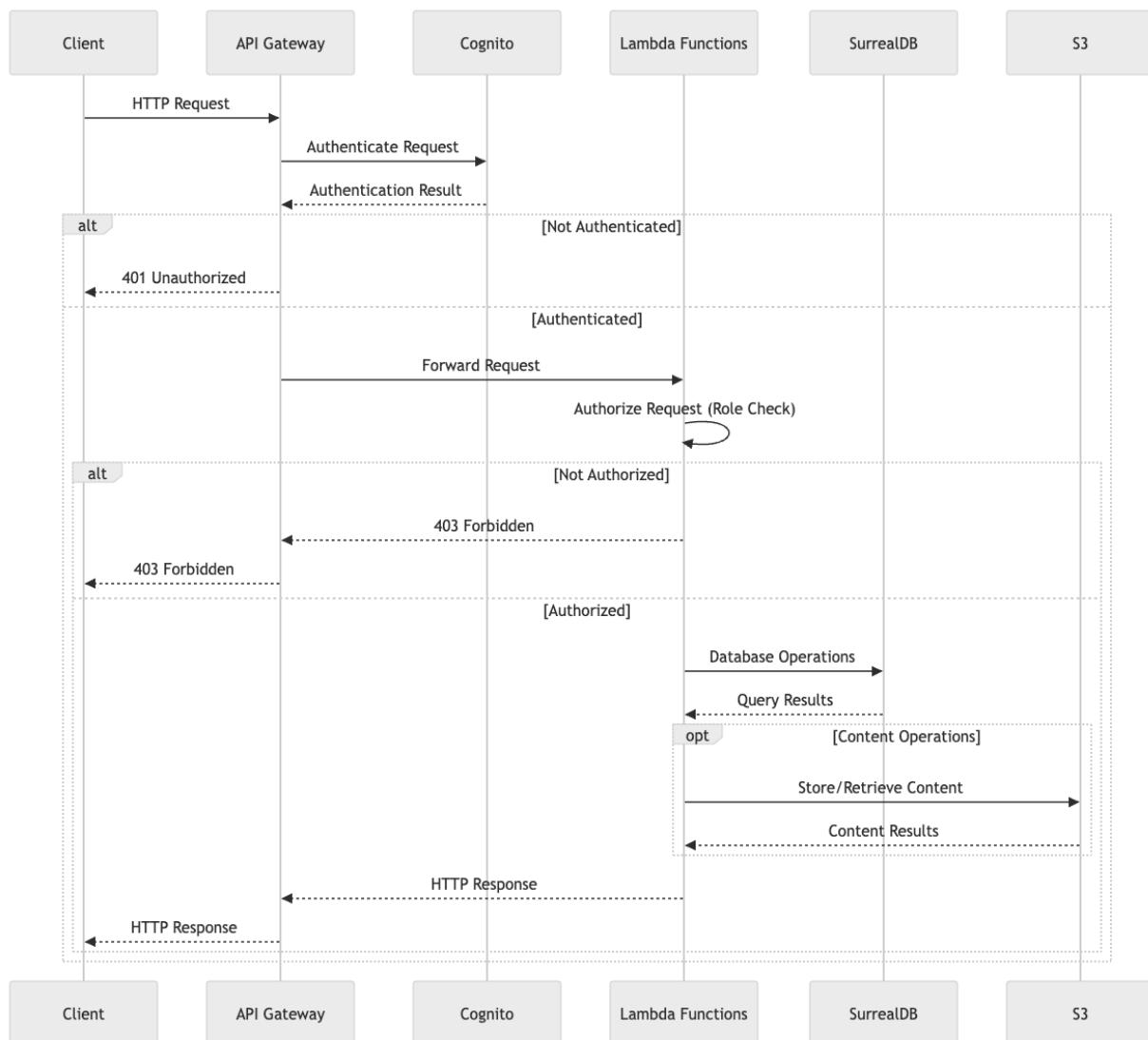
Figure 3.1: Database Schema

### 3.2 Request and Response Structure

```
let response = ApiGatewayProxyResponse {
    status_code: 200,
    headers: std::collections::HashMap::new(),
    multi_value_headers: std::collections::HashMap::new(),
    body: Some(json!({ "message": "Hello from Kaiju Academy API!" })),
    to_string(),
    is_base64_encoded: Some(false),
};

Ok(response)
```

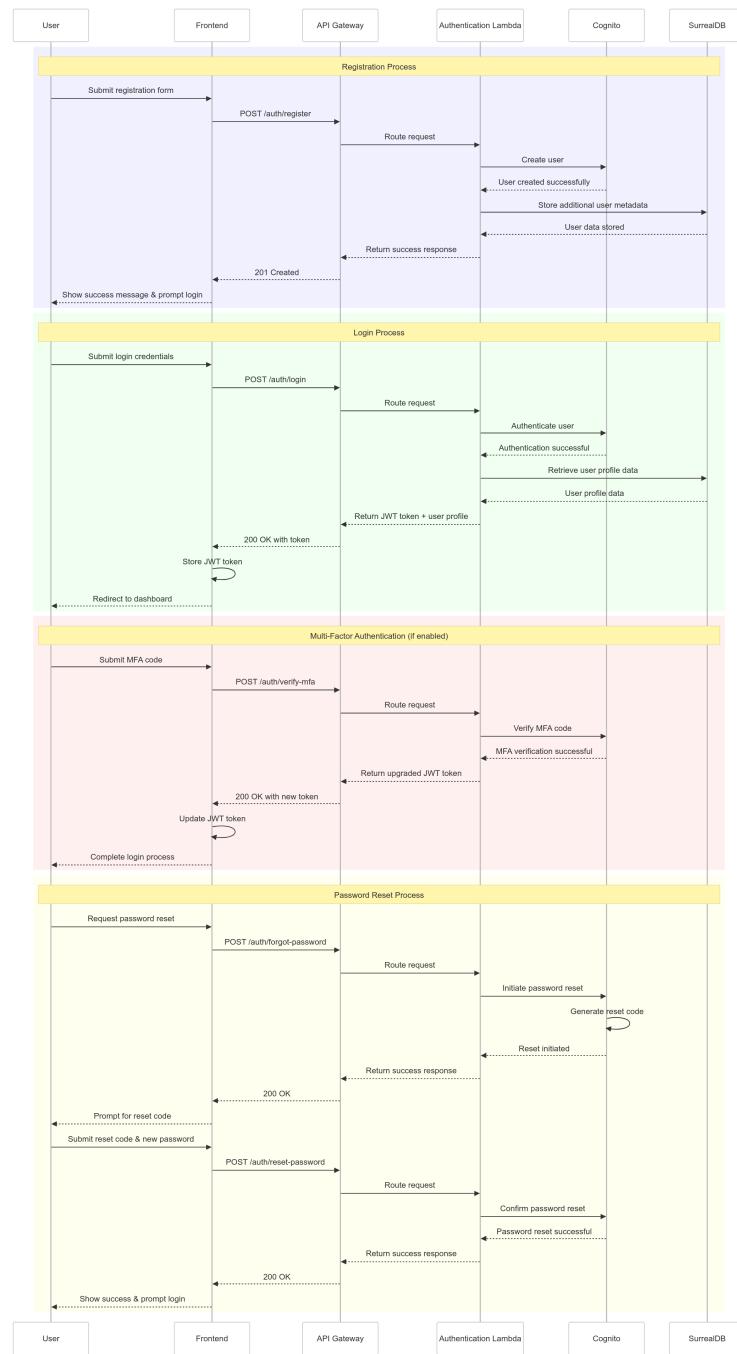
### 3.3 Data Flow Sequence



**Figure 3.2: Data Flow Sequence Diagram**

## 4. Interface Design

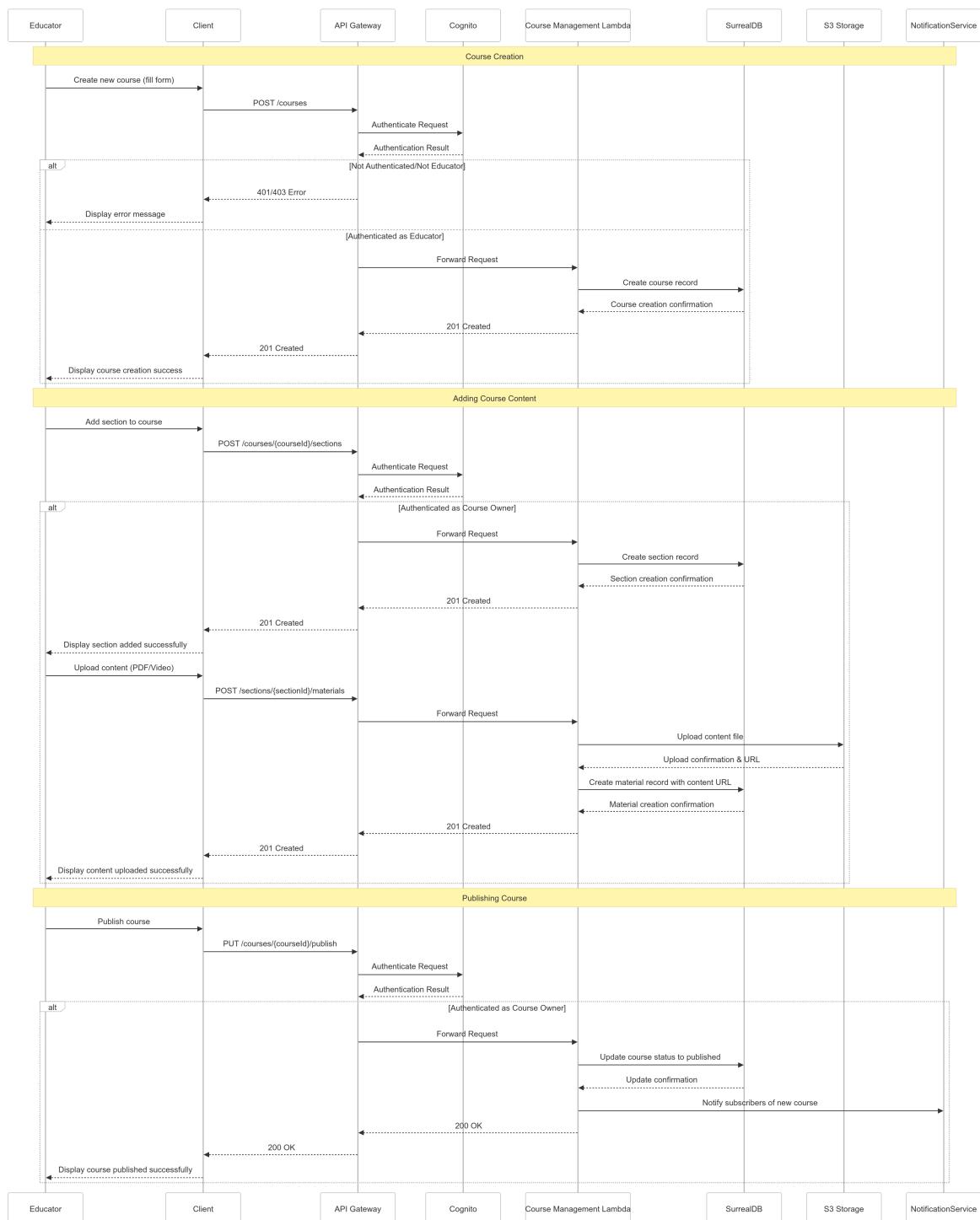
### 4.1 User Registration and Authentication



**Figure 4.1: User Authentication Workflow**

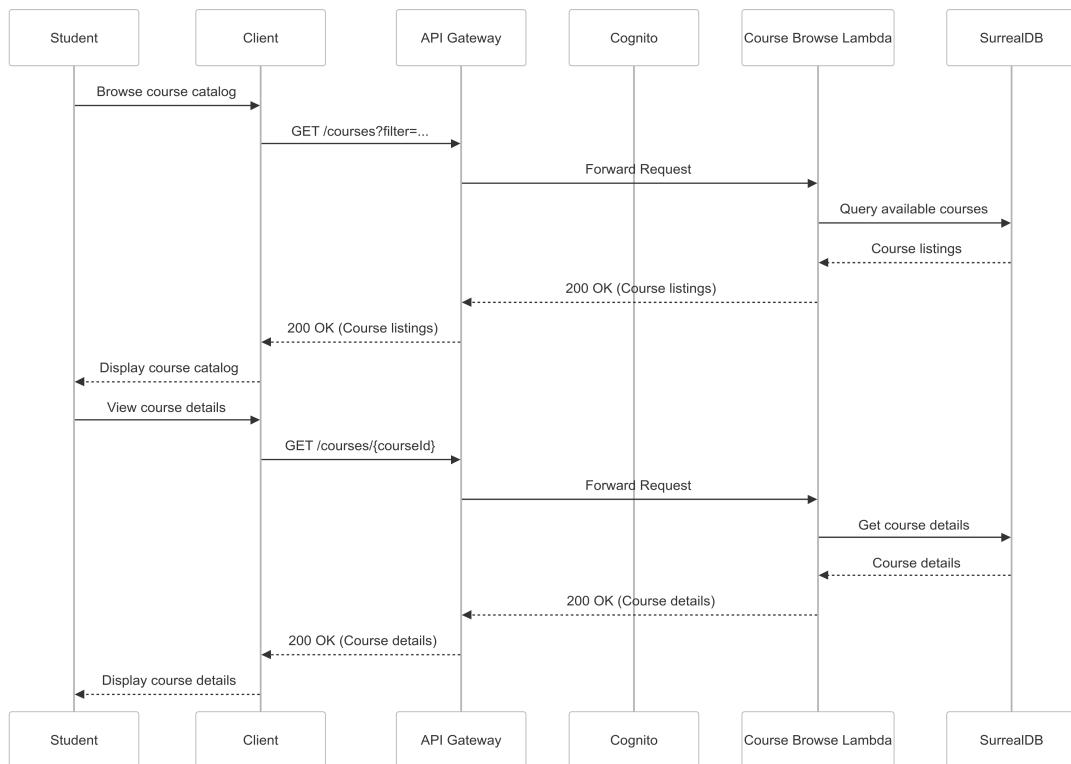
1. User submits registration/login with optional MFA.
2. API Gateway routes to Authentication Lambda.
3. Lambda creates/validates user in Cognito and SurrealDB.
4. JWT token returned for session.

### 4.1.1 Course Management



**Figure 4.2: Course Management Workflow**

1. Educator manages course structure and modules.
2. Materials uploaded to S3.
3. Changes stored in SurrealDB.

**Figure 4.3:** Course Access Workflow

#### 4.1.2 Course Access

#### 4.1.3 Code Assessment and Grading

**Sample code assessment editor (simplified):**

```
// Code Assessment Editor (conceptual)
fn CodeEditor() {
    let code = "// Write your code here";
    fn run_code() {
        // send code to backend, display result
    }
    // Renders a textarea and Run button
}
```

## 4.2 Service Communication

- RESTful APIs with JSON payloads.
- Rate limiting per user/IP.
- Batch operations and pagination.

## 4.3 Security Implementation

- Multi-factor Authentication (MFA) optional for all users.
- OAuth2 integration for social login.
- JWT for API authentication.
- AES-256 encryption for sensitive data.

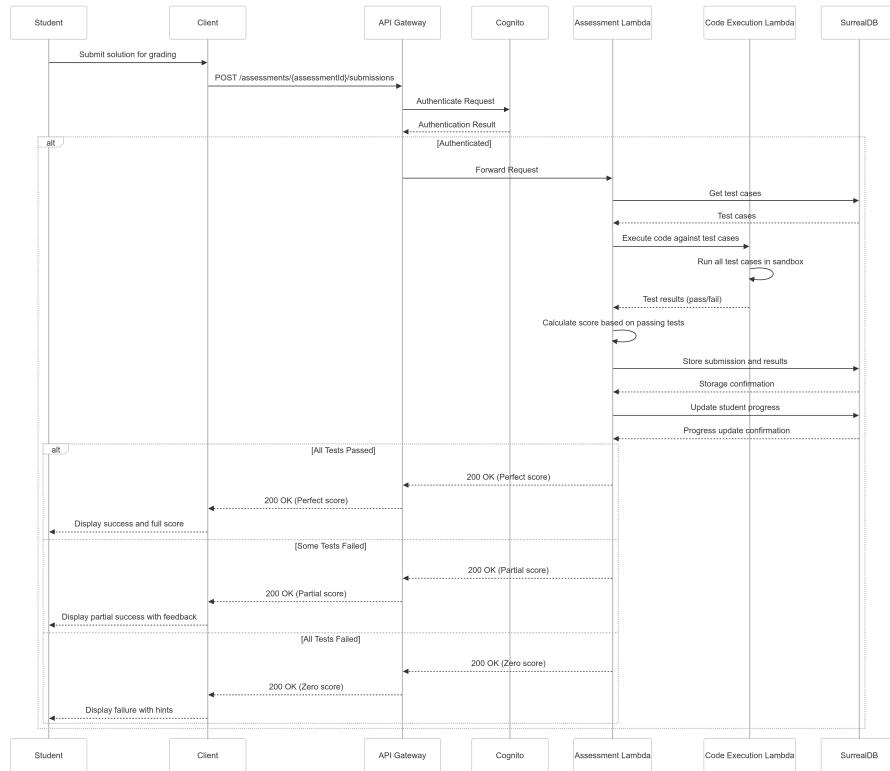


Figure 4.4: Code Auto-grading Workflow

## 4.4 Exception Handling

### 4.4.1 Expected Exceptions

- Authentication errors (401)
- Authorization errors (403)
- Validation errors (400)
- Not found errors (404)
- Database errors (500)
- Internal server errors (500)
- External service errors (502)
- Rate limit errors (429)

### 4.4.2 Error Handling Example

```

impl From<AppError> for ApiGatewayProxyResponse {
    fn from(error: AppError) -> Self {
        let (status_code, error_type) = match &error {
            AppError::Authentication(_) => (401, "Authentication Error"),
            AppError::Authorization(_) => (403, "Authorization Error"),
            AppError::NotFound(_) => (404, "Not Found"),
            AppError::Validation(_) => (400, "Validation Error"),
            AppError::Database(_) => (500, "Database Error"),
            AppError::Internal(_) => (500, "Internal Server Error"),
            AppError::ExternalService(_) => (502, "External Service Error"),
            AppError::RateLimit(_) => (429, "Rate Limit Exceeded"),
        };
        // Response body omitted for brevity
        ApiGatewayProxyResponse { ... }
    }
}

```

#### 4.4.3 Credit Purchase and Payment Flow

##### Overview:

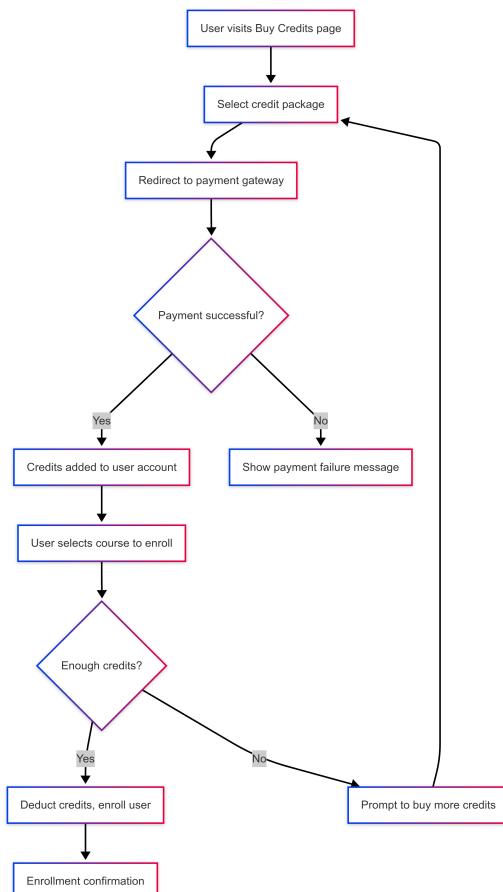
Kaiju Academy uses a credit-based payment system. Users buy credits through an integrated payment gateway and spend credits to unlock/register for courses. This system decouples payment from course registration and supports flexible monetization.

#### 4.4.4 Credit Purchase and Course Enrollment Flow

1. User navigates to the “Buy Credits” page from the profile or course page.
2. User selects a credit package (e.g., 100 credits, 200 credits).
3. User is redirected to a third-party payment gateway (e.g., Stripe) to complete the transaction.
4. Upon successful payment, the backend updates the user’s credit balance.
5. User can now browse courses; when enrolling in a paid course, the platform checks credit balance:
  - If sufficient, deducts credits and confirms registration.
  - If insufficient, prompts the user to buy more credits.
6. All transactions are logged for auditing and user transparency.

**Security Note:** All payment processing is handled via a PCI-compliant gateway. No sensitive payment data is stored on Kaiju Academy servers.

#### 4.4.5 Credit Purchase and Enrollment Logic (Mermaid Diagram)



**Figure 4.5:** Credit Purchase and Course Enrollment Flow

## 5. Component Design

### 5.1 Frontend Components

#### 5.1.1 Navbar Component

- **Intention:** Provides navigation links and user profile access.
- **Input:** User interactions (clicks), authentication state.
- **Output:** Navigation actions, profile dropdown.

#### 5.1.2 Code Editor Component

- **Intention:** Provides syntax highlighting, code execution, and result display.
- **Input:** User code, language, test cases.
- **Output:** Code submission to backend, execution result.

### 5.2 Backend Components

#### 5.2.1 Authentication Service

- **Intention:** Registration, login, MFA, token management.
- **Input:** Credentials, MFA codes.
- **Output:** JWT tokens, authentication status, error messages.

#### 5.2.2 Course Management Service

- **Intention:** Course/module creation, update, and deletion.
- **Input:** Course objects, materials.
- **Output:** Course IDs, content structures.

#### 5.2.3 Assessment Service

- **Intention:** Executes user code, grades submissions, provides feedback.
- **Input:** User code, language, test cases.
- **Output:** Execution results, scores, feedback.

## 6. User Interface Design

### 6.1 Design Principles

- **Consistency:** Unified design language
- **Accessibility:** WCAG 2.1 AA compliance
- **Responsiveness:** Mobile friendly
- **Feedback:** Clear system status for all actions

### 6.2 User-wise Navigation Flow

#### 6.2.1 Student

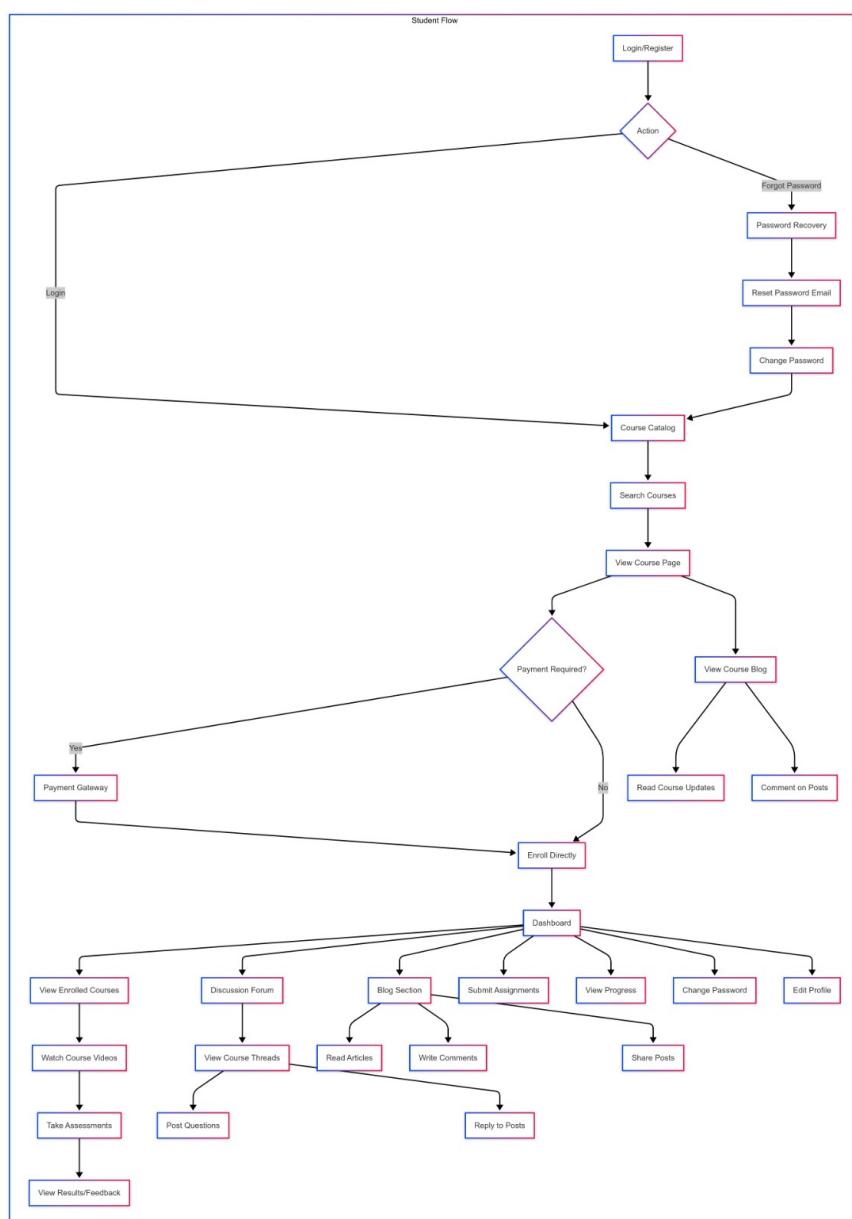
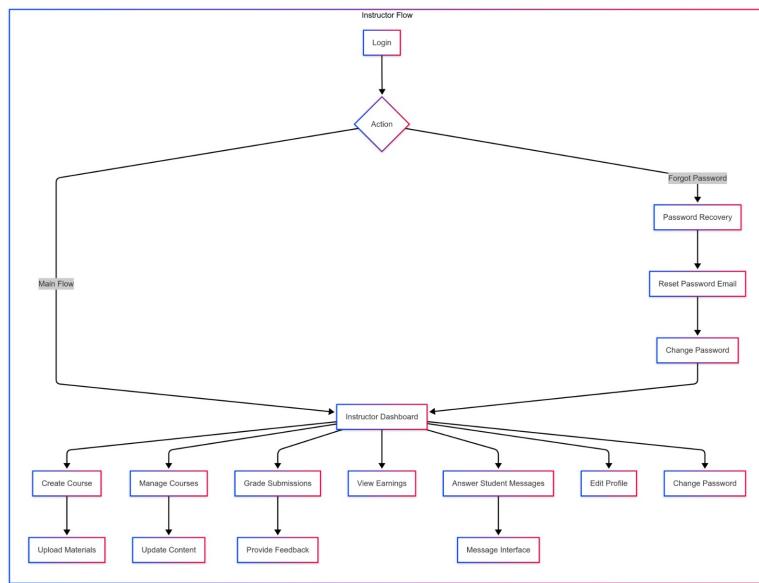


Figure 6.1: Student Navigation Flow

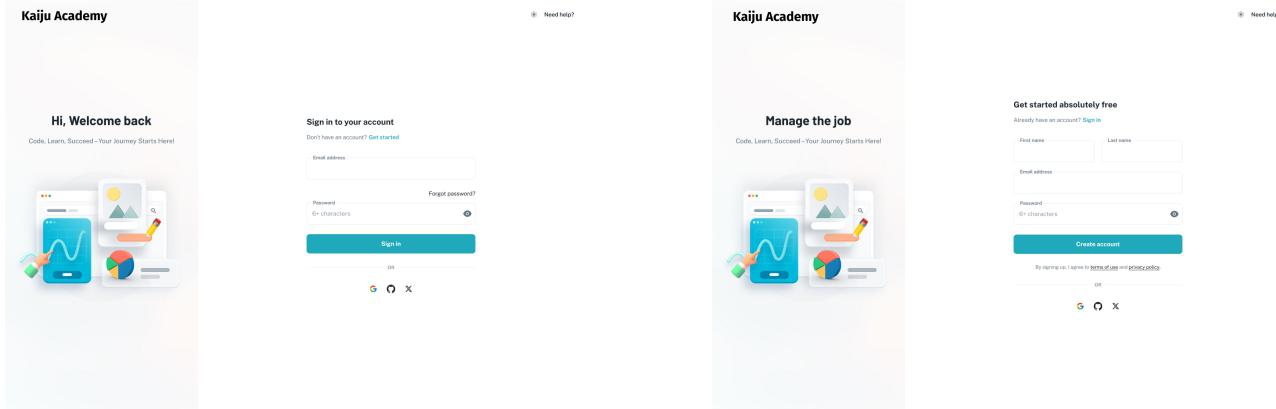
## 6.2.2 Educator



**Figure 6.2:** Educator Navigation Flow

## 6.3 Interface Storyboards

### 6.3.1 Authentication Flow



**Figure 6.3:** Authentication Screens

### 6.3.2 Student Learning Journey

**(a) Student Dashboard**

**(b) Course Catalog**

Figure 6.4: Student courses view

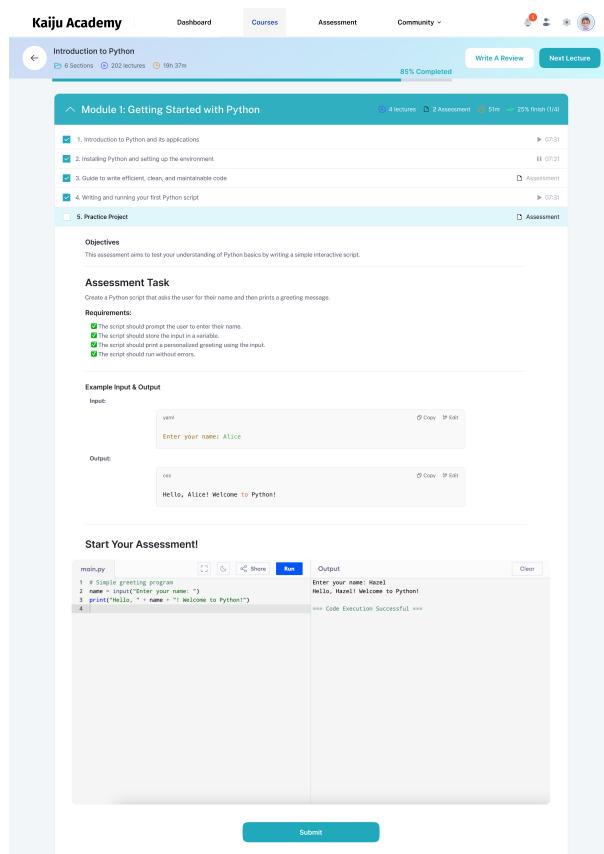
### 6.3.3 Learning Environment

## 6.4 Accessibility Considerations

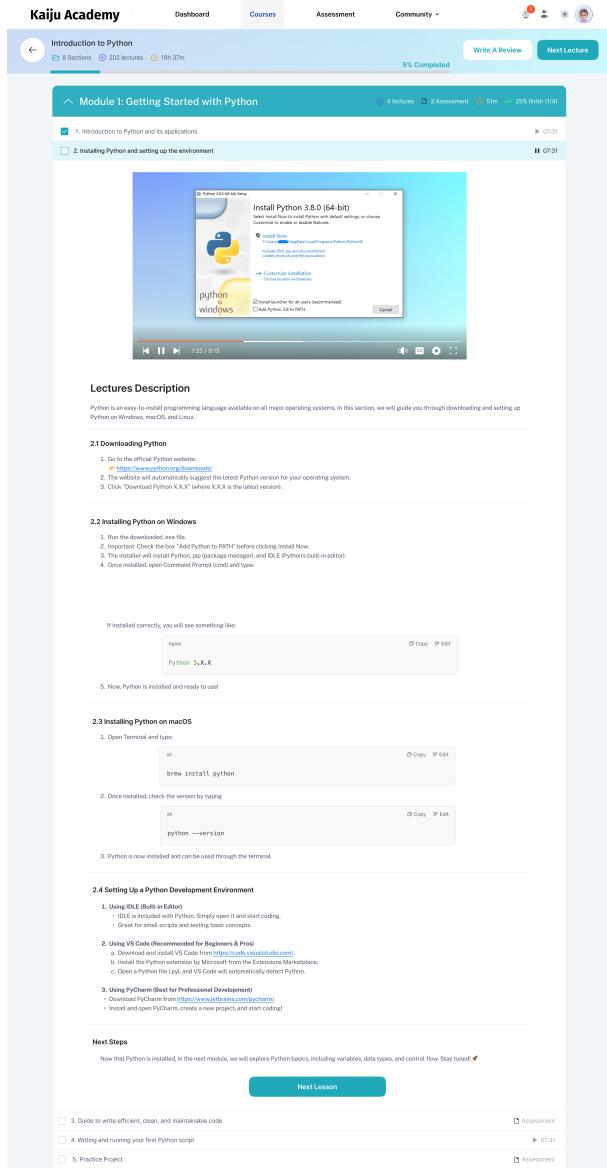
- **Screen Reader Support:** Proper ARIA labels and semantic HTML
- **Text Resize:** Interface remains functional when text is enlarged 200%
- **Motion Control:** Animations can be disabled
- **Alternative Text:** All images include descriptive alt text
- **Light/Dark Mode:** Option to switch between light and dark themes
- **Font Size Adjustment:** Users can easily change font size

## 6.5 Responsive Design

- **Breakpoints:** 480px, 768px, 1024px, and 1440px
- **Layout:**
  - Single column on mobile
  - Sidebar overlay on small screens
  - Multi-column on larger screens
- **Touch Targets:** Minimum 44×44px for all interactives



**(a) Assessment Interface**



**(b) During Course Interface**

Figure 6.5: Learning and Assessment Interface

## 7. Assumptions

### 7.1 Technical Constraints

#### 7.1.1 Hardware Constraints

- At least 4GB RAM, 320px width, 5 Mbps internet
- AWS cloud deployment, serverless Lambda, horizontal scaling

#### 7.1.2 Software Constraints

- Modern browsers (Chrome, Firefox, Safari, Edge, latest 2 versions)
- HTML5 and JavaScript required
- Responsive web (no native app initially)

### 7.2 Operational Assumptions

- Peak: 10,000 code submissions/minute
- 80/20 read/write database
- Code execution max 5 seconds
- CI/CD pipeline with automated testing
- 99.9% uptime excluding maintenance, daily backups

### 7.3 Dependencies

#### 7.3.1 Third-Party Services

- Authentication via OAuth providers (Google, GitHub, Microsoft)
- Payment processing for course credits
- Email delivery through Amazon SNS
- CDN via CloudFront
- Media transcoding via AWS MediaConvert