DESIGN AND IMPLEMENTATION FOR

Kaiju Academy

Version 1.0

Prepared by
Group A2

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

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

March 11, 2025

Contents

Document Revision History

Version	Revised By	Revision Date	Comments
0.1	Group A2	2024-03-10	Added: –Initial document structure –Basic content outline
1.0	Group A2	2024-03-11	Updated: –Completed all sections –Added diagrams and technical details
1.1	Group A2	2024-03-12	Updated: –Reorganized content into 7 main sections –Enhanced section details and clarity

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

Based on the requirements specification, Kaiju Academy includes:

- **User Management & Authentication:** Role-based access control for students, educators, administrators, and moderators with secure authentication.
- **Course Creation & Management:** Tools for creating, updating, and organizing courses with various content types (videos, PDFs, quizzes, coding exercises).
- **Interactive Learning:** Real-time code execution, automated grading, and personalized learning dashboards.
- **Community Features:** Discussion forums with moderation tools and notification systems.
- **Administrative Tools:** User analytics, system monitoring, and compliance management.

1.2 Quality Goals

Goal	Description
Performance	- Response time <2 seconds for 95% of users. - Code execution results in <5 seconds for 99% of submissions.
Scalability	- Support 10,000 concurrent users. - Horizontal scaling via AWS Lambda and SurrealDB sharding.
Reliability	- 99.9% uptime with automatic failover. - Recovery from failures within 5 minutes.
Security	- AES-256 encryption for data at rest. - TLS 1.2+ for data in transit. - Role-based permissions and MFA.
Usability	- Intuitive UI with 90% user satisfaction. - Mobile-responsive design for screen sizes $\geq$ 320px.
Maintainability	- Modular codebase with 80% test coverage. - Infrastructure-as-Code (AWS CDK) for reproducible environments.
Compliance	- GDPR, COPPA, and accessibility law compliance. - Daily encrypted backups stored for 30 days.

Table 1.1: Quality Goals for Kaiju Academy

1.3 Stakeholders

Stakeholder	Role & Responsibilities
Administrators	- Manage system health, user roles, and backups. - Monitor performance and enforce security policies.
Students	- Enroll in courses, complete assessments, and track progress. - Participate in forums and coding competitions.
Educators	- Create and update course content. - Grade submissions and provide feedback.
Content Moderators	- Moderate discussion forums. - Enforce community guidelines and resolve disputes.
Developers	- Implement and maintain backend services (Rust/AWS) and SurrealDB schema. - Optimize performance and security.
Legal & Compliance	- Ensure adherence to GDPR, COPPA, and other regulations. - Manage data retention and privacy policies.

Table 1.2: Stakeholders and Their Roles

2. System Architecture

2.1 Overview

The Kaiju Academy platform follows a modern, cloud-native approach with a focus on scalability, reliability, and security. The system is designed as a serverless application deployed on AWS, using a microservices pattern for modularity and maintainability.

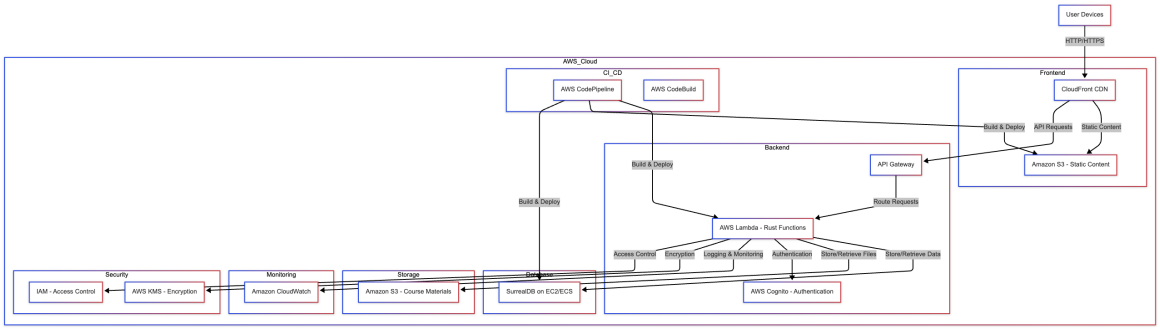

Figure 2.1: AWS Serverless Deployment Architecture

2.2 Major Components

Frontend Application: Single-page React application providing the user interface across all devices.

API Gateway: Manages all external requests, authentication, and routing to appropriate backends.

Lambda Functions: Serverless computing units organized by domain (authentication, courses, assessments, forums, administration).

Cognito: Handles user authentication, MFA, and role-based access control.

SurrealDB: Primary database managing all persistent data with multi-model support.

S3 Storage: Houses course materials, user uploads, and media content.

CloudFront CDN: Optimizes content delivery globally for static assets and course materials.

Code Execution Service: Sandboxed environment for running user-submitted code securely.

Notification System: Manages email, in-app, and push notifications.

2.3 Component Relationships

The following diagram illustrates the relationships between the different components of the system:
Key interactions between components include:

- Frontend communicates exclusively through the API Gateway
- Lambda functions interact with databases and other AWS services
- Cognito handles all authentication before business logic executes
- Event-driven architecture connects components asynchronously where appropriate

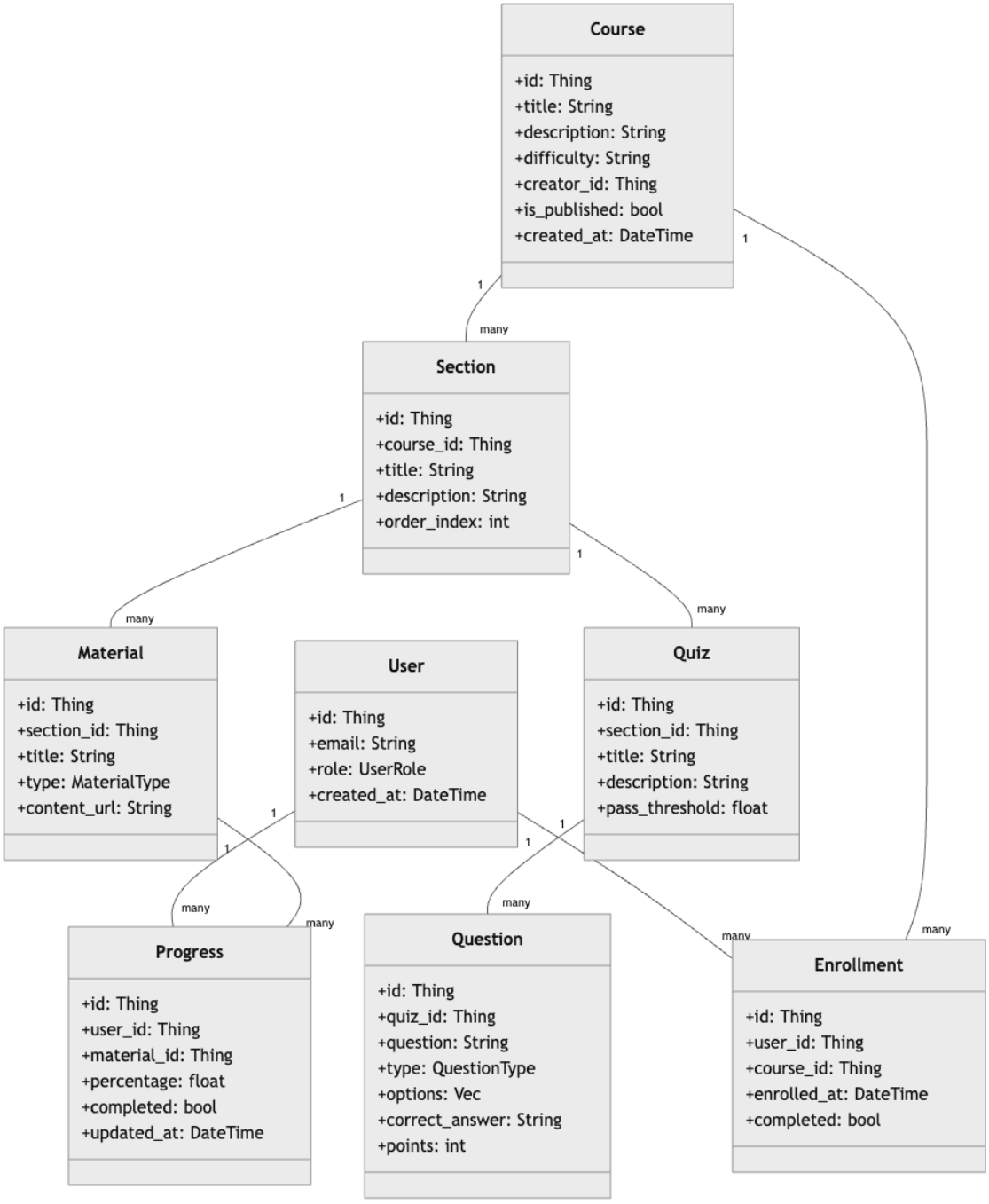

Figure 2.2: Component Relationships

3. Data Models

3.1 Database Schema

The database design incorporates relational, document, and graph capabilities through SurrealDB's multi-model approach.

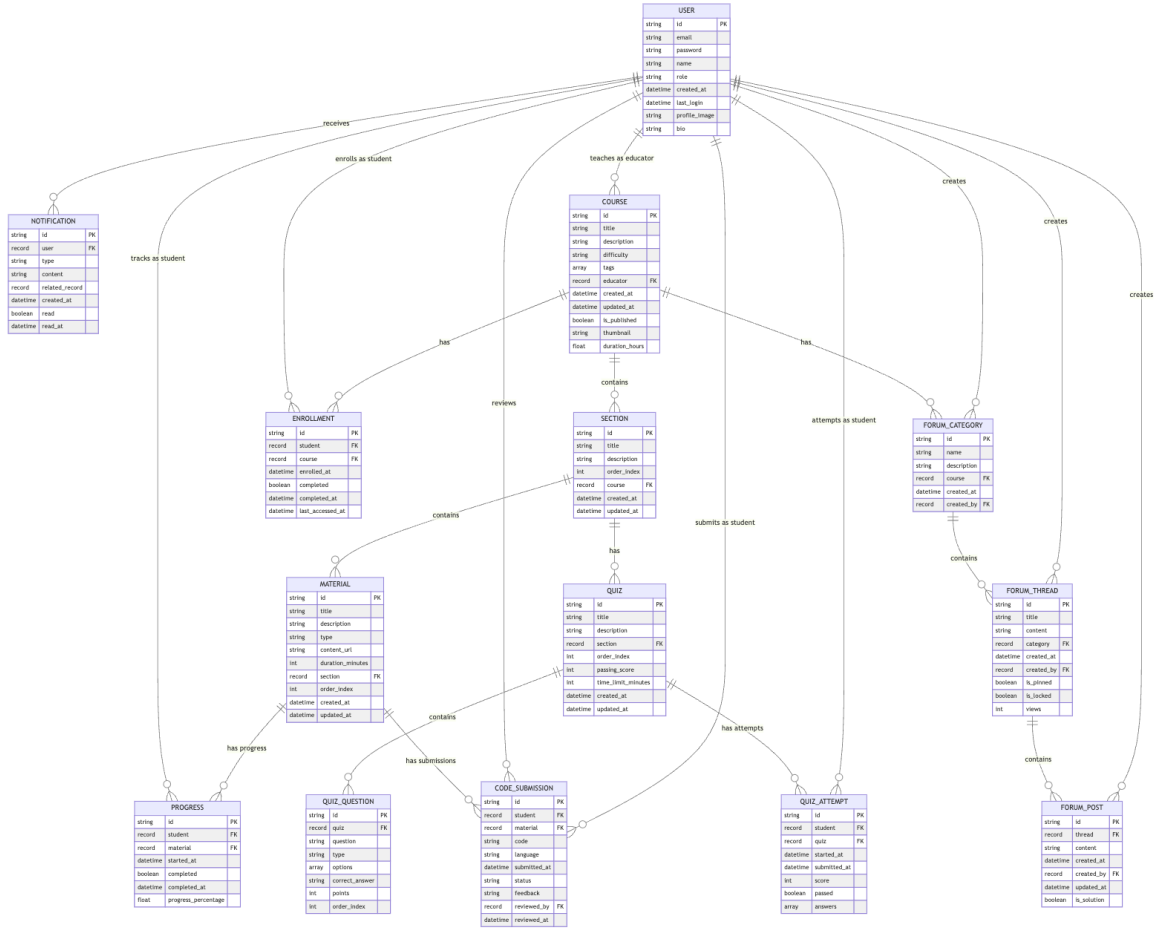

Figure 3.1: Database Schema

3.2 Core Entities

User: Contains user information, roles, and authentication details.

Course: Represents a learning unit with metadata and organizational structure.

Section: Organizational unit within courses containing materials.

Material: Content items (videos, PDFs, code exercises) within sections.

Enrollment: Maps relationships between users and courses.

Progress: Tracks user advancement through materials.

Quiz/Assessment: Contains questions and evaluation criteria.

Submission: Stores user responses to assessments.

ForumPost: User discussions and questions within the community.

3.3 Data Flow

The following diagram illustrates the flow of data through the system:

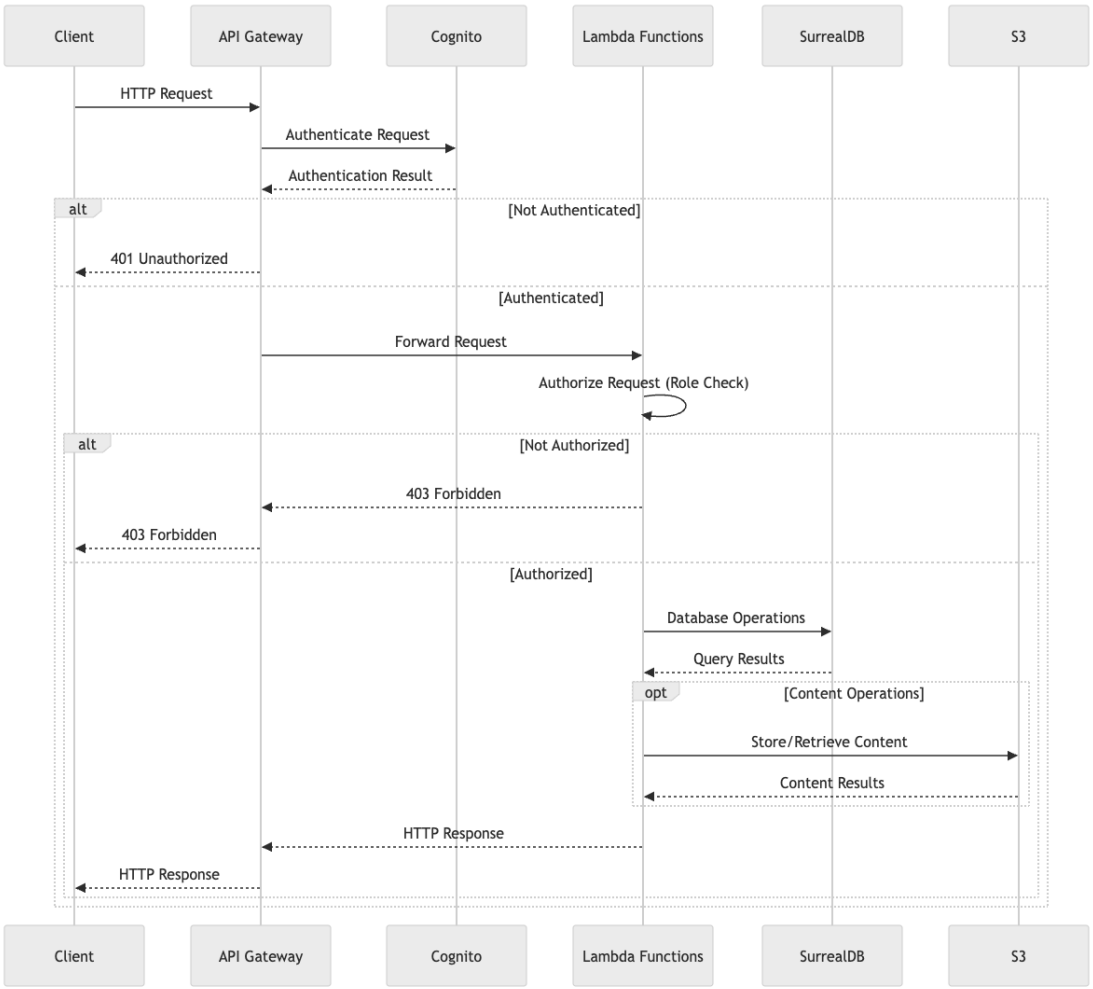

Figure 3.2: Data Flow Diagram

3.4 Key Data Flows

3.4.1 User Registration and Authentication

1. User submits registration data through frontend
2. API Gateway routes to Authentication Lambda
3. Lambda creates user in Cognito and SurrealDB
4. JWT token returned for authenticated session

3.4.2 Course Creation and Publishing

1. Educator creates course structure through UI
2. Course metadata and structure stored in SurrealDB
3. Materials uploaded to S3 with references stored in database
4. Publishing workflow updates visibility and notifies subscribers

3.4.3 Code Assessment and Grading

1. Student submits code solution
2. Code executed in isolated sandbox
3. Results compared against test cases
4. Grading results stored in database and displayed to student
5. Progress records updated automatically

3.4.4 Forum Moderation

1. User creates or flags post
2. Post stored in database with metadata
3. Moderators review flagged content
4. Moderation actions update content status and notify relevant users

4. Interface Design

4.1 API Structure

The API follows RESTful principles organized into domain-specific endpoints. All external communication occurs through a standardized API Gateway.

Figure 4.1: API Domains and Organization

4.2 Authentication Protocol

- OAuth2 for third-party authentication (Google, GitHub)
- JWT tokens for session management and API authentication
- MFA via TOTP (Time-based One-Time Password) for sensitive operations

4.3 Service Communication

4.3.1 External Interfaces

- RESTful APIs with JSON payloads
- Rate limiting (10 requests/second per user)
- Batch operations for efficiency
- Pagination for large result sets

4.3.2 Internal Communication

- AWS SNS/SQS for asynchronous event processing
- AWS Step Functions for complex workflows
- Direct Lambda invocations for synchronous operations

4.4 Error Handling

- Standardized error responses with HTTP status codes and descriptive messages
- Comprehensive logging with correlation IDs
- Global and resource-specific rate limiting
- Circuit breakers for downstream service failures

4.5 Security Implementation

4.5.1 Authentication and Authorization

- Multi-factor Authentication (MFA): Optional for all users, required for administrators
- OAuth2 Integration: Support for Google, GitHub, and Microsoft accounts
- Role-based Access Control (RBAC): Granular permissions based on user roles
- JWT for API Authentication: Secure, stateless authentication for API requests

4.5.2 Data Protection

- Encryption at Rest: AES-256 encryption for all sensitive data
- Encryption in Transit: TLS 1.2+ for all network communications

- PII Protection: Minimized collection of personally identifiable information
- Data Retention Policies: Automated purging of unnecessary data

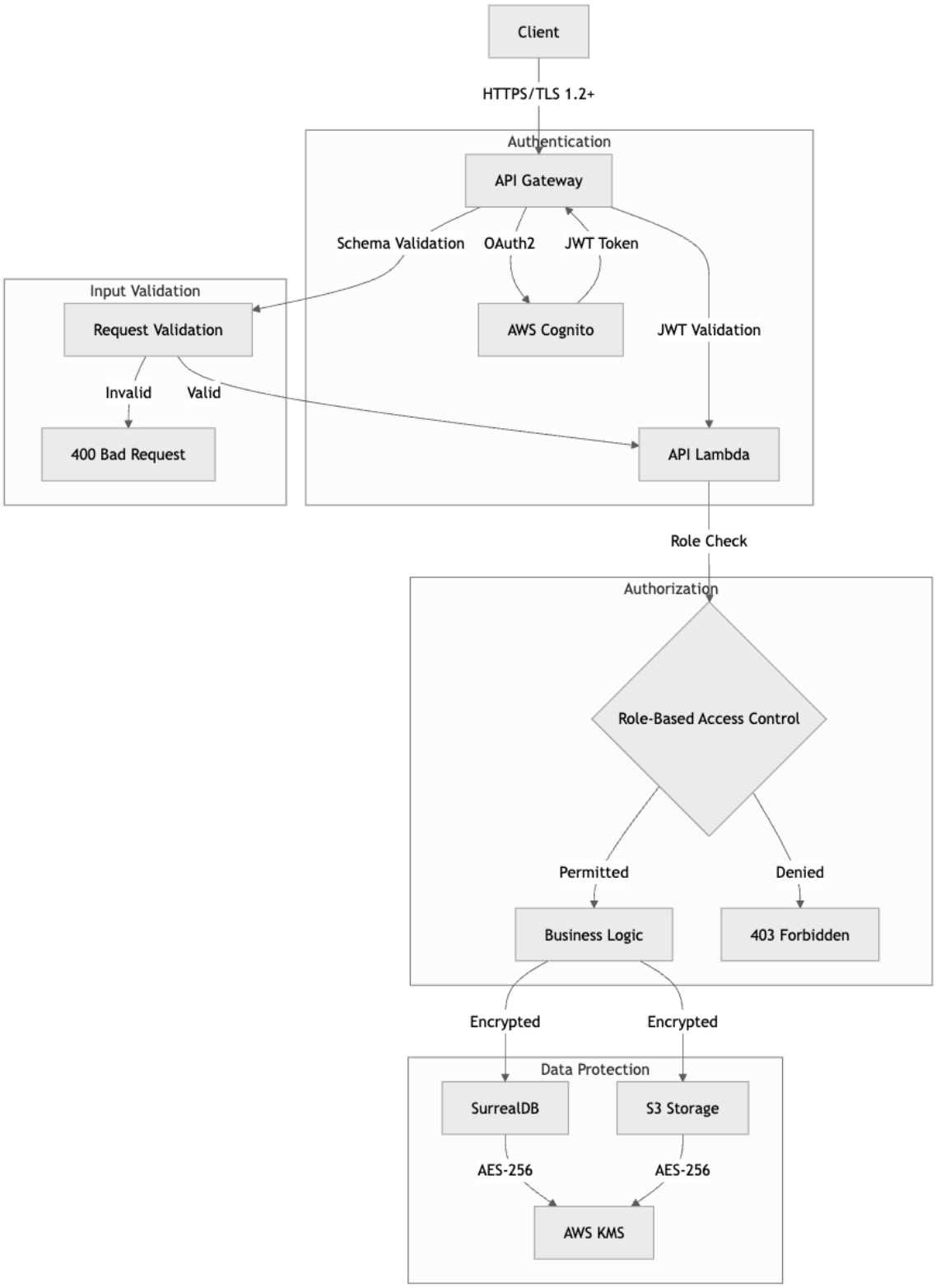

Figure 4.2: Security Implementation Diagram

5. Component Design

5.1 Frontend Components

5.1.1 Layout Components

Navbar Top navigation component with:

- Logo and site navigation
- Search functionality
- Notification center
- User profile dropdown

Sidebar Navigation component with:

- Course navigation
- Quick links to user dashboard
- Role-specific menus
- Collapsible sections

Footer Bottom component with:

- Terms and privacy links
- Contact information
- Copyright notices

5.1.2 Page Components

Dashboard Main user interface with:

- Welcome section with personalized greeting
- Statistics cards showing progress and achievements
- Visualization charts for activity and progress
- Upcoming deadlines and calendar

Course Browser Course discovery component with:

- Filterable/sortable course grid
- Category navigation
- Featured course carousel
- Course preview cards

Code Editor Interactive coding environment with:

- Syntax highlighting for multiple languages
- Resizable panels for instructions/tests
- Run/Submit controls
- Result visualization

5.2 Backend Components

5.2.1 Authentication Service

Responsibilities:

- User registration and authentication
- Token generation and validation
- Role management and authorization
- MFA enrollment and verification

Input/Output:

- Input: User credentials, registration data

- Output: JWT tokens, user profile information

Key Functions:
- `register_user(user_data) -> Result<UserProfile, AuthError>`
- `authenticate_user(credentials) -> Result<AuthToken, AuthError>`
- `validate_token(token) -> Result<TokenClaims, AuthError>`
- `change_user_role(user_id, role) -> Result<(), AuthError>`

5.2.2 Course Management Service

Responsibilities:
- Course CRUD operations
- Content organization and delivery
- Enrollment management
- Progress tracking

Input/Output:
- Input: Course data, enrollment requests, content updates
- Output: Course structures, material access, progress reports

Key Functions:
- `create_course(course_data) -> Result<CourseId, CourseError>`
- `update_course(course_id, updates) -> Result<(), CourseError>`
- `enroll_user(course_id, user_id) -> Result<Enrollment, EnrollmentError>`
- `update_progress(user_id, material_id, progress) -> Result<Progress, ProgressError>`

5.2.3 Assessment Service

Responsibilities:
- Quiz and assessment creation
- Code execution in sandboxed environment
- Automated grading
- Submission management

Input/Output:
- Input: Assessment definitions, code submissions, quiz responses
- Output: Execution results, grades, feedback

Key Functions:
- `create_assessment(assessment_data) -> Result<AssessmentId, AssessmentError>`
- `execute_code(code, language, inputs) -> Result<ExecutionResult, ExecutionError>`
- `grade_submission(submission) -> Result<Grade, GradingError>`
- `get_submission_history(user_id, assessment_id) -> Result<Vec<Submission>, SubmissionErr`

5.2.4 Forum Service

Responsibilities:
- Discussion thread management
- Post moderation
- Content filtering
- Notification dispatching

Input/Output:
- Input: Forum posts, moderation actions, thread creation
- Output: Thread listings, post content, moderation results

Key Functions:
- `create_post(post_data) -> Result<PostId, ForumError>`
- `moderate_post(post_id, action) -> Result<(), ModerationError>`
- `get_thread(thread_id) -> Result<Thread, ForumError>`
- `get_flagged_posts() -> Result<Vec<FlaggedPost>, ForumError>`

5.2.5 Class Structure

The following class diagram represents the core domain model:

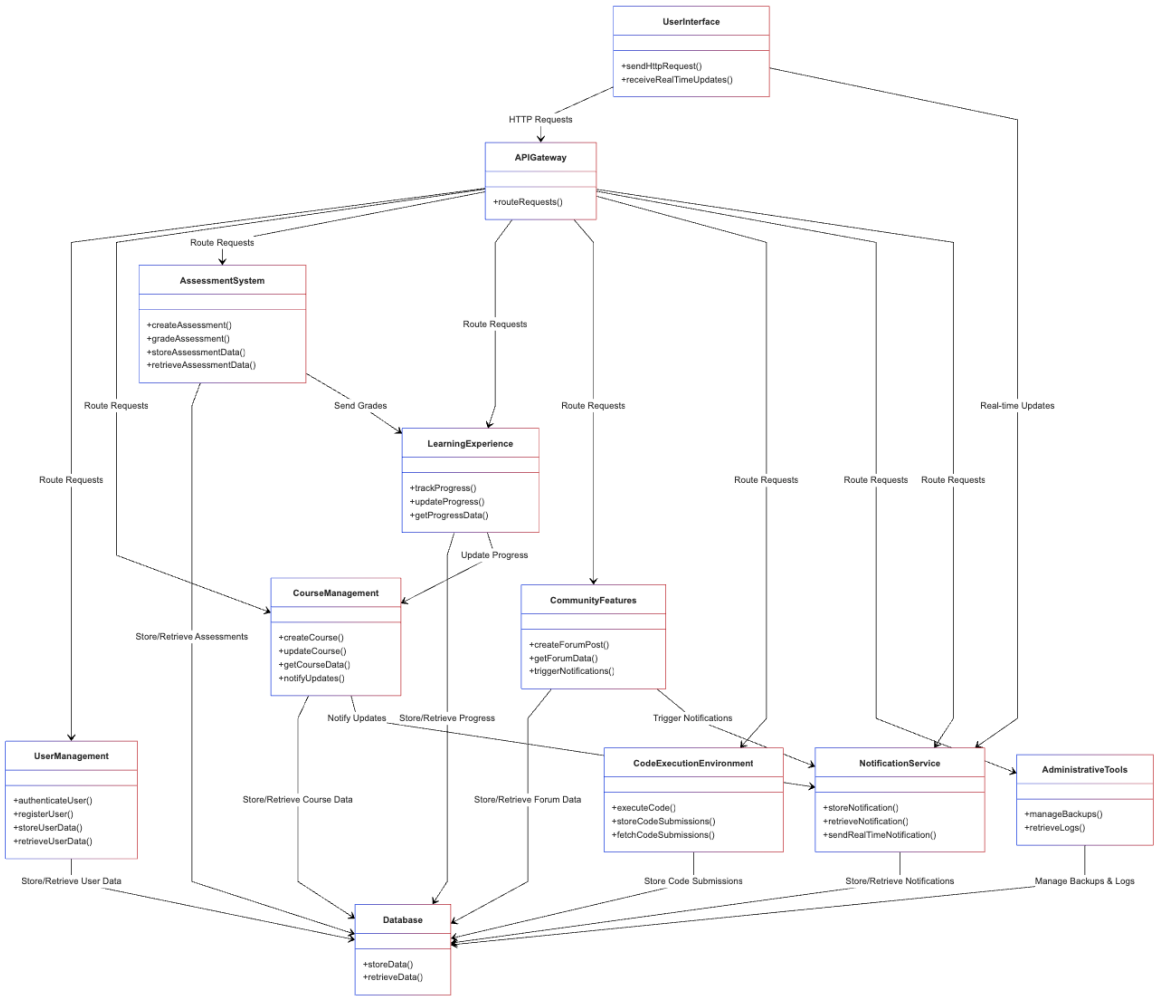

Figure 5.1: Class Diagram

6. User Interface Design

6.1 Design Principles

The UI design follows these core principles:

- **Consistency:** Unified design language across all pages
- **Accessibility:** WCAG 2.1 AA compliance for all users
- **Responsiveness:** Optimal experience across devices (320px and up)
- **Progressive Enhancement:** Core functionality works without JavaScript
- **Feedback:** Clear indication of system status and actions

6.2 Site Map and Navigation Flow

Figure 6.1: Site Map and Navigation Flow

6.3 Interface Storyboards

6.3.1 Authentication Flow

Figure 6.2: User Authentication Storyboard

6.3.2 Course Enrollment and Participation

Figure 6.3: Course Enrollment and Learning Storyboard

6.3.3 Assessment Completion

6.4 Accessibility Considerations

- **Keyboard Navigation:** Full functionality available without mouse
- **Screen Reader Support:** Proper ARIA labels and semantic HTML
- **Color Contrast:** WCAG AA (4.5:1) minimum contrast ratios
- **Text Resize:** Interface remains functional when text is enlarged 200%
- **Motion Control:** Animations can be disabled via prefers-reduced-motion
- **Assistive Technology:** Regular testing with NVDA, JAWS, and VoiceOver
- **Alternative Text:** All images include descriptive alt text

6.5 Responsive Design

- **Mobile First:** Designed primarily for mobile, then enhanced for larger screens
- **Breakpoints:** Key breakpoints at 480px, 768px, 1024px, and 1440px
- **Layout Adaptations:**
  - Single column layout on mobile
  - Sidebar appears as overlay on small screens
  - Multi-column layout on tablets and larger

Figure 6.4: Assessment Completion Storyboard

  - Expanded dashboard visualizations on desktop
- **Touch Targets:** Minimum 44×44px for all interactive elements

7. Assumptions

7.1 Technical Constraints

7.1.1 Hardware Constraints

- **Minimum Device Specifications:** Users require devices with:
  - At least 4GB RAM for smooth performance
  - Screen resolution of 320px width minimum
  - Internet connection with 5 Mbps minimum bandwidth
- **Server Environment:** Development and deployment assumes:
  - AWS cloud infrastructure
  - Serverless architecture with AWS Lambda
  - No dedicated hardware requirements
  - Horizontal scaling capabilities

7.1.2 Software Constraints

- **Browser Support:** Application designed for:
  - Modern browsers (Chrome, Firefox, Safari, Edge - latest 2 versions)
  - HTML5 and JavaScript required
  - No Internet Explorer support
- **Development Environment:**
  - Standardized Docker containers for development
  - Rust 1.58+ for backend development
  - Node.js 16+ for frontend development
  - AWS CDK for infrastructure definition
- **Mobile Support:**
  - Responsive web design (no native app initially)
  - Mobile browser focused (iOS Safari, Android Chrome)

7.2 Operational Assumptions

7.2.1 Load and Performance

- Maximum concurrent users: 10,000
- Peak submission rate: 10,000 code submissions per minute
- Average session duration: 30 minutes
- 80/20 read/write ratio for database operations
- Code execution takes maximum 5 seconds per submission

7.2.2 Development Process

- Agile methodology with 2-week sprints
- CI/CD pipeline with automated testing
- Feature branching development workflow
- Containerized development environments
- Terraform for infrastructure provisioning

7.2.3 Maintenance and Support

- 99.9% uptime requirement excluding planned maintenance
- Scheduled maintenance windows during off-peak hours

- Automated backups performed daily and retained for 30 days
- Customer support available during business hours
- Issue response time within 24 hours

7.3 Dependencies

7.3.1 Third-Party Services

- Authentication via OAuth providers (Google, GitHub, Microsoft)
- Payment processing through Stripe
- Email delivery through Amazon SES
- CDN services through AWS CloudFront
- Media transcoding through AWS MediaConvert

7.3.2 External Systems

- LMS integration via LTI standard
- External code repositories (GitHub, GitLab)
- Social media sharing APIs
- Analytics integrations (Google Analytics, Mixpanel)