Architecture Description of

Monolithic Architecture for

Learning Management System (LMS)

“Bare bones” edition version: 2.2

Template prepared by:

Rich Hilliard

r.hilliard@computer.org

Distributed under Creative Commons Attribution 3.0 Unported License.

For terms of use, see: http://creativecommons.org/licenses/by/3.0/

# Contents

Contents 2

Using the template 4

License 5

Version history 5

Editions 6

Comments or questions 6

1 Introduction 7

1.1 Identifying information 7

1.2 Supplementary information 7

1.3 Other information 7

1.3.1 Overview (optional) 7

1.3.2 Architecture evaluations 7

1.3.3 Rationale for key decisions 8

2 Stakeholders and concerns 9

2.1 Stakeholders 9

2.2 Concerns 9

2.3 Concern–Stakeholder Traceability 10

3 Viewpoints

3.1 Conceptual Viewpoint

3.1.1 Overview

3.1.2 Concerns Addressed

3.1.3 Typical Stakeholders

3.2 Execution Viewpoint

3.2.1 Overview

3.2.2 Concerns Addressed

3.2.3 Typical Stakeholders

3.3 Data Viewpoint

3.3.1 Overview

3.3.2 Concerns Addressed

3.3.3 Typical Stakeholders 15

5 Consistency and correspondences 18

5.1 Known inconsistencies 18

5.2 Correspondences in the AD 18

5.3 Correspondence rules 18

A Architecture decisions and rationale 19

A.1 Decisions 19

Bibliography 20

## 1. Introduction

### 1.1 Identifying Information

Architecture Name: Monolithic Architecture

System of Interest: Learning Management System for Course Management and Assessment

### 1.2 Supplementary Information

Version: 1.0

Date: November 2024

Status: Initial Draft

Architecture Style: Layered Architecture

### 1.3 Other Information

#### 1.3.1 Overview

The Learning Management System (LMS) is designed as a monolithic web application that provides functionalities for course management, user management, and assessment handling. The primary stakeholders interact with the system as follows: - Administrators: Manage system settings, users, and overall functionality. - Instructors: Create and manage courses, assign tasks, and assess student progress. - Students: Enroll in courses, submit assignments, and view progress reports.

Key architectural characteristics: - Centralized deployment and management: All components are deployed together, simplifying management. - Single codebase: Ensures consistency and ease of maintenance. - Shared database: A unified repository for all system data. - Integrated components: Seamless interaction between modules - Synchronized data processing: Ensures real-time data updates.

#### 1.3.2 Architecture Evaluations

The monolithic architecture was evaluated for the following quality attributes: - Scalability: Limited to vertical scaling; suitable for small to medium-scale use cases. - Maintainability: Single codebase reduces overhead for smaller teams but may challenge larger systems as they grow. - Security: Centralized control of authentication and authorization simplifies security implementation. - Performance: Efficient for integrated workflows due to the absence of inter-service communication - Reliability: Simplified testing and debugging improve reliability for smaller systems. - Modifiability: Changes in one part of the system require testing the entire system, a potential trade-off.

#### 1.3.3 Rationale for Key Decisions

The decision to adopt a monolithic architecture is supported by the following considerations: - Simplified Development and Deployment: Ideal for early-stage or small-scale applications where fast iteration is crucial. - Ease of Debugging and Testing: A single codebase facilitates debugging and end-to-end testing. - Reduced Complexity in Team Coordination: Suitable for teams with limited resources or experience in distributed systems. - Transaction Management: Monolithic designs simplify database transaction handling, avoiding the complexity of distributed systems. - Performance Suitability: Effective for tightly integrated workflows common in small to medium-scale systems

## 2. Stakeholders and Concerns

### 2.1 Stakeholders

System Users

Administrators

Instructors

Students

Development Team

Software Developers

Software Architects

### 2.2 Concerns

#### Purposes of the System-of-Interest

The primary purpose of the Learning Management System (LMS) is to provide a centralized platform for educational institutions to manage courses, students, instructors, and learning materials. It aims to streamline administrative tasks, improve learning outcomes, and offer a scalable, secure, and user-friendly experience for all stakeholders, including students, instructors, and administrators.

#### Suitability of the Architecture

Using a monolithic architecture, the system provides the following benefits:

Tight integration of features: All functional modules (e.g., authentication, course management, assessments) are tightly coupled, ensuring seamless communication and consistent performance.

Ease of deployment: The entire system is deployed as a single unit, reducing complexity in the deployment process.

Simplified development: The monolithic design minimizes the overhead of managing inter-service communication, making it suitable for small to medium-sized teams during initial development. While suitable for the project’s scale and initial purpose, potential limitations in scalability and flexibility for future expansions must be addressed.

#### Feasibility of Construction and Deployment

Technology stack: The choice of Java as the primary language ensures high performance, robust security, and extensive community support.

Development feasibility: Java Spring Boot framework simplifies implementing features such as authentication, course management, and notifications.

Deployment feasibility: Deployment in a monolithic architecture is straightforward, requiring only one runtime environment or container, which can simplify DevOps processes.

#### Risks and Impacts

Scalability challenges: A monolithic architecture might face bottlenecks as user demands increase.

Single point of failure: Issues in one module could potentially affect the entire system.

Stakeholder impacts: Stakeholders benefit from rapid development and deployment but may face delays or downtime during system updates, given the monolithic design.

Long-term costs: As the system evolves, maintaining a monolithic codebase might become more challenging, increasing technical debt.

#### Maintenance and Evolution

Maintainability: The use of well-structured code practices, clear module boundaries, and robust testing strategies ensures maintainability. However, significant refactoring might be required for future scalability.

Evolution: The architecture can evolve by: Refactoring individual modules into services as needed (eventual migration to microservices, if required). Implementing feature toggles to enable iterative deployment of new features. Enhancing modularization within the monolithic codebase to simplify updates and feature additions.

#### Summary of Concerns

Functional Concerns

User authentication and authorization

Profile Management

Course management

Attendance Management

Assessment & Grading management

Student Progress Tracking

Generating Reports on student performance (Performance Analytics)

Media management

Notification system

Quality Attribute Concerns

Security: Robust authentication and encryption mechanisms protect sensitive data.

Performance: The system handles concurrent users through optimized database queries and efficient threading.

Reliability: Data integrity is ensured through ACID-compliant database transactions.

Availability: Use of robust runtime environments and regular backups to maximize uptime.

Scalability: Initial focus on horizontal scaling of the monolith; modularization can ease future transitions.

Maintainability: Clean coding practices, adherence to design patterns, and a clear separation of concerns reduce maintenance overhead.

### 2.3 Concern–Stakeholder Traceability

This section associates the identified concerns from 2.2 with the stakeholders from 2.1 who have those concerns.

## #### Example showing association of stakeholders to concerns in an AD

#### Additional Details

System Users:

Administrators are concerned with functional aspects like user authentication, course management, attendance tracking, and system reliability.

Instructors focus on ease of use for course management, grading, and performance tracking.

Students prioritize accessibility, performance, and availability for their learning materials and progress tracking.

Development Team:

Software Developers are responsible for implementing features while ensuring maintainability and performance.

Software Architects focus on architecture suitability, scalability, and risk mitigation for long-term evolution.

### 3. Viewpoints

### 3.1 Conceptual Viewpoint

### Overview

### The Conceptual viewpoint describes the functional organization of the LMS, focusing on the system’s components and their responsibilities. It illustrates how the main modules interact to deliver core functionalities such as user management, course management, assessments, and notifications.

### Concerns Addressed

### Functional decomposition of the system into modules.

### Interactions between components.

### Role-based access to functionalities.

### Data flow and process synchronization.

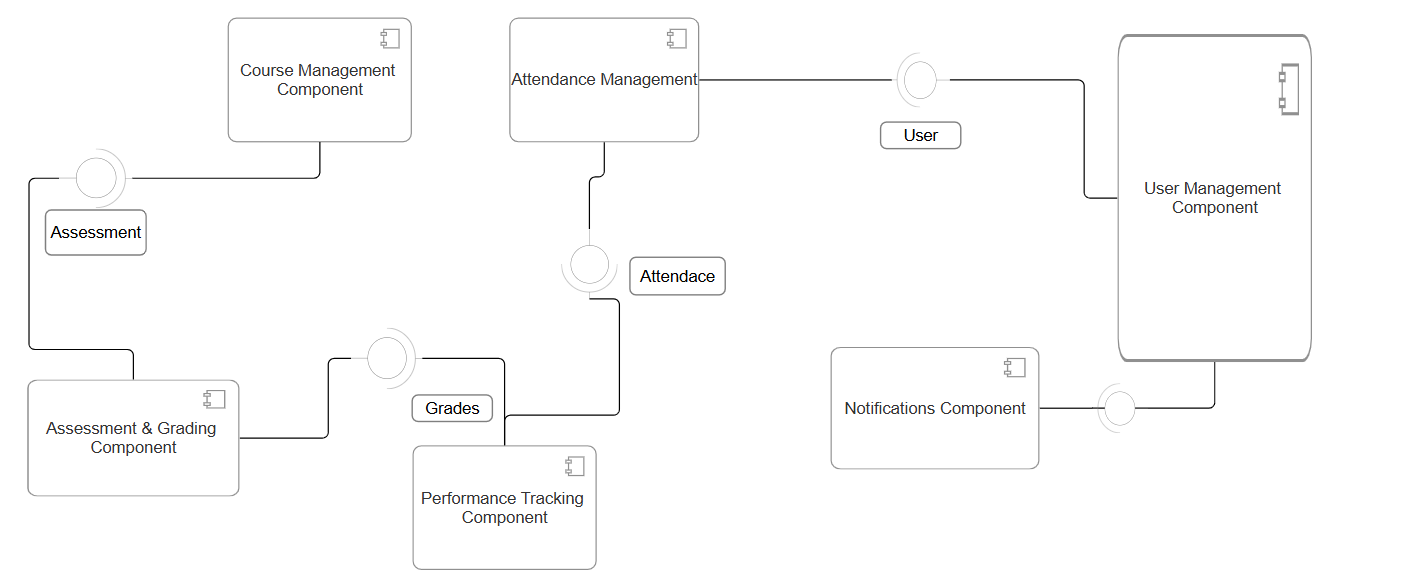
### Typical Stakeholders

### Developers: Understand functional dependencies and system organization.

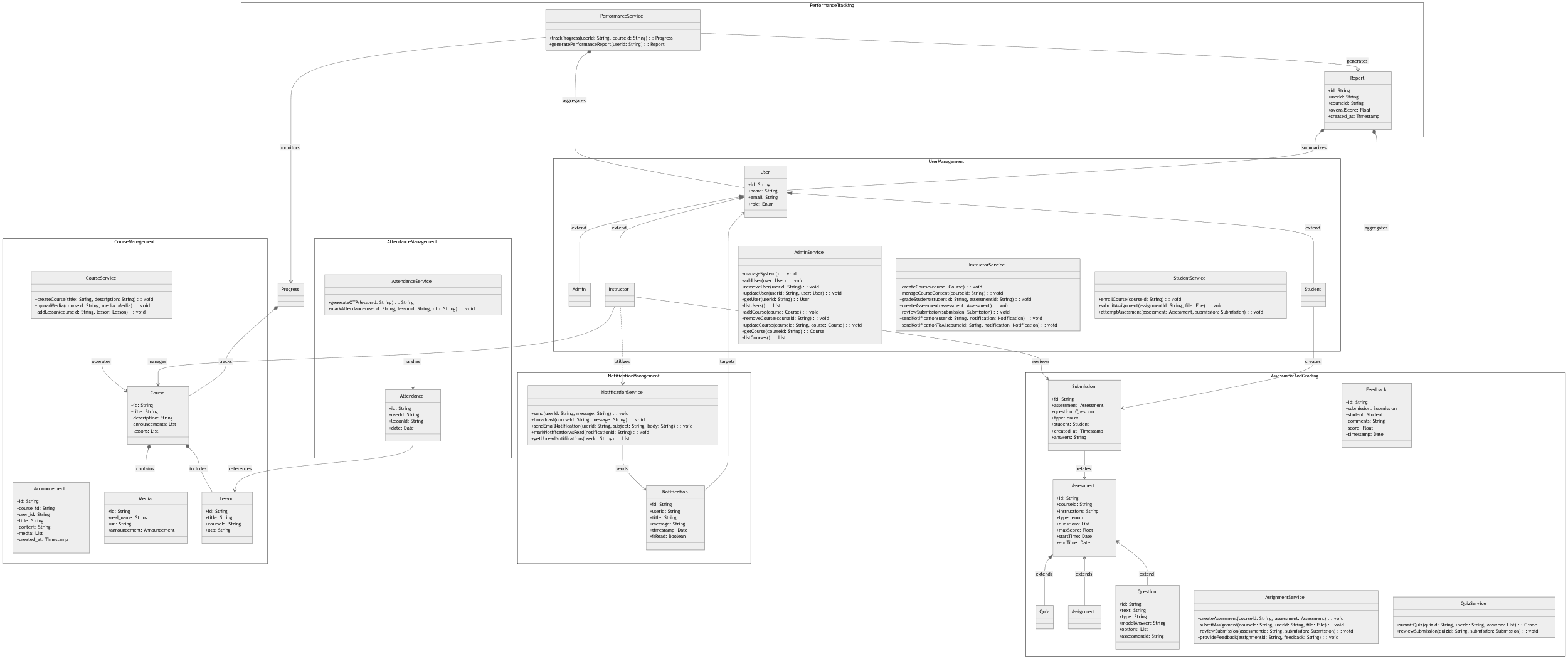
### Architects: Ensure modular and maintainable design.

### System Administrators: Manage configurations and dependencies.

Component Diagram:



### class diagram:



### 3.2 Execution Viewpoint

### Overview

### The deployment viewpoint describes the physical deployment of the LMS system, including servers, databases, and network infrastructure.

### Concerns Addressed

### Physical placement of system components (frontend, backend, database).

### Scalability and availability considerations.

### Security of network communication and data storage.

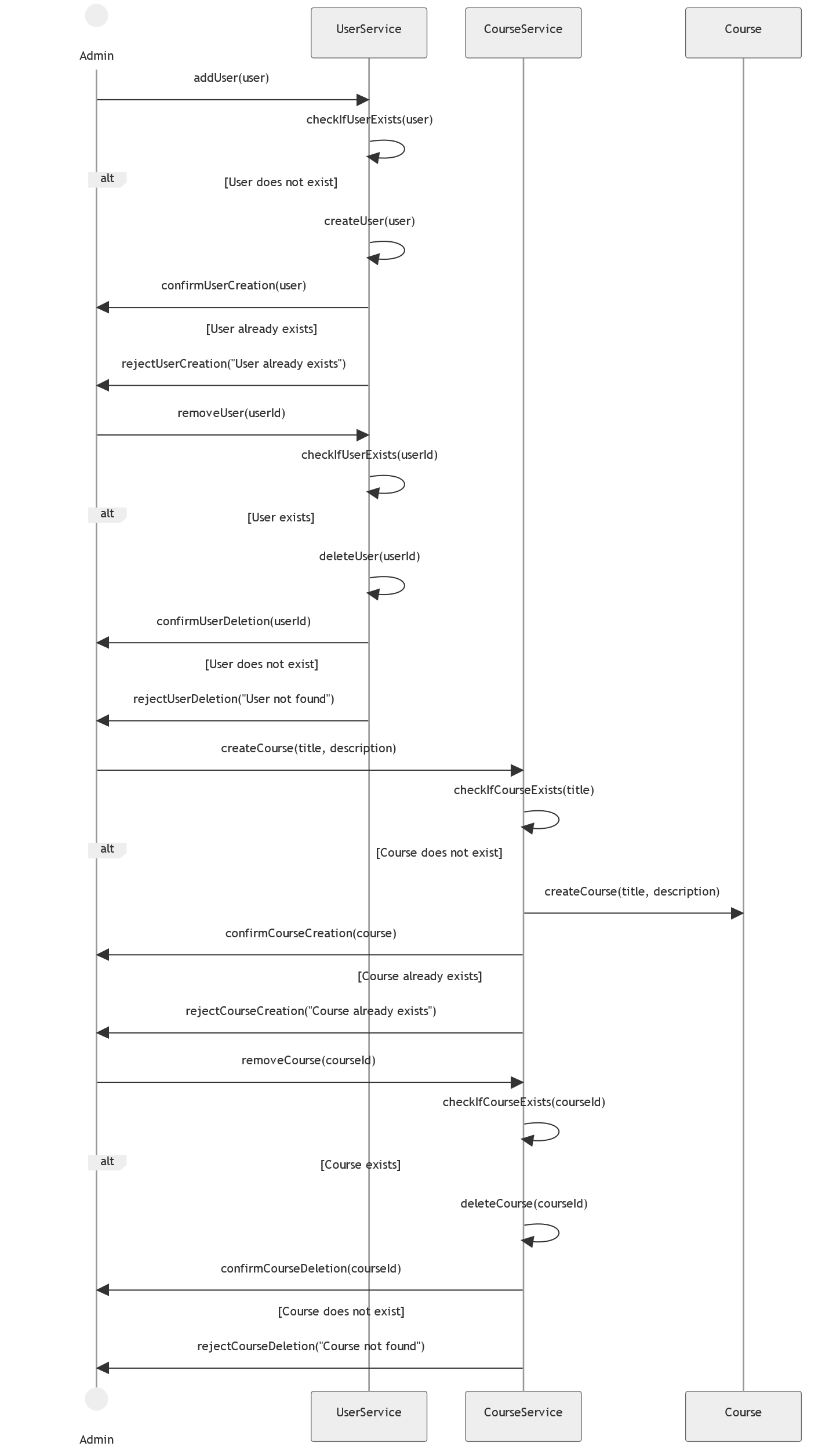
### Typical Stakeholders

### Developers: Understand functional dependencies and system organization.

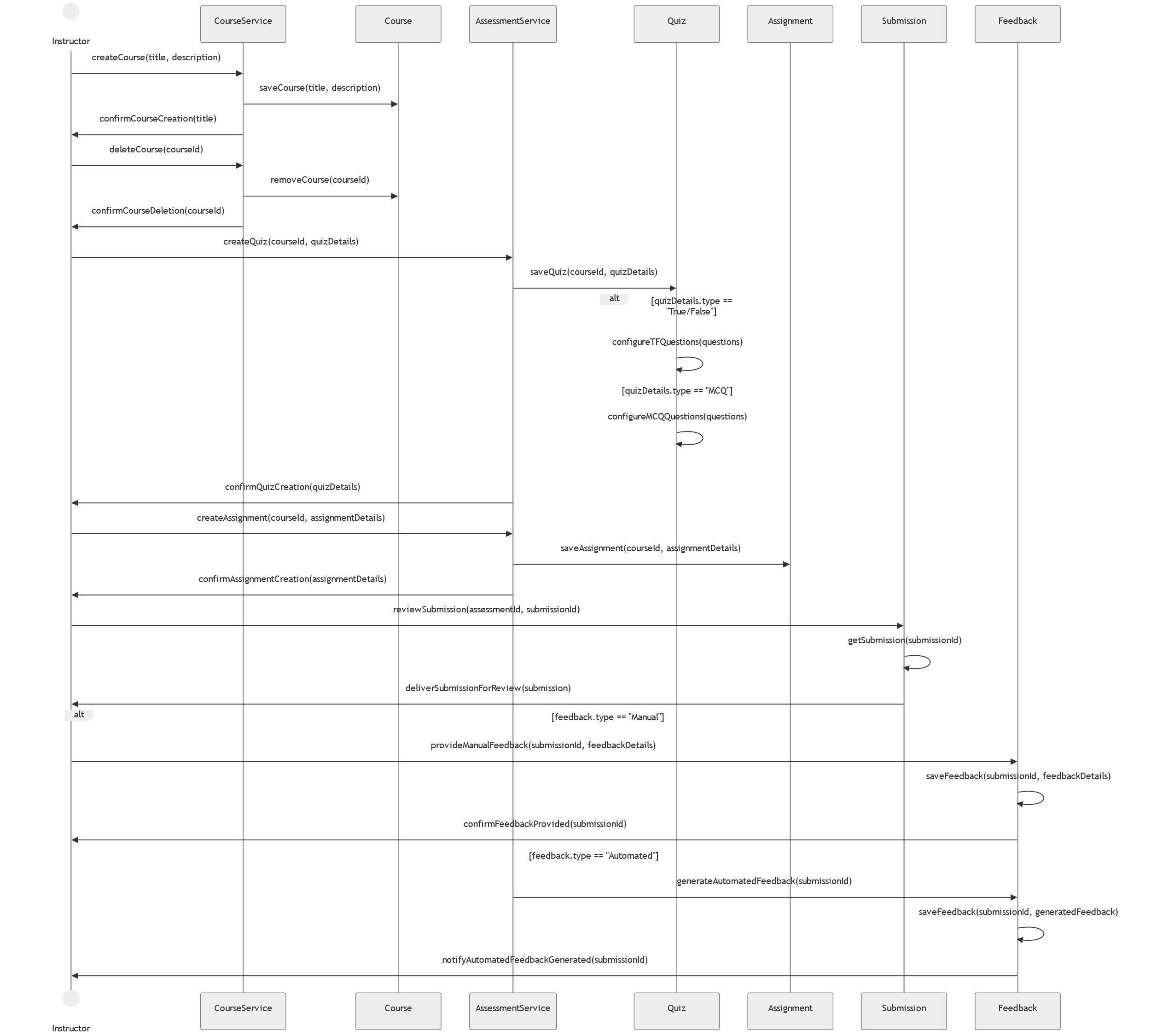
### Architects: Ensure modular and maintainable design.

### System Administrators: Responsible for managing server infrastructure and deployments.

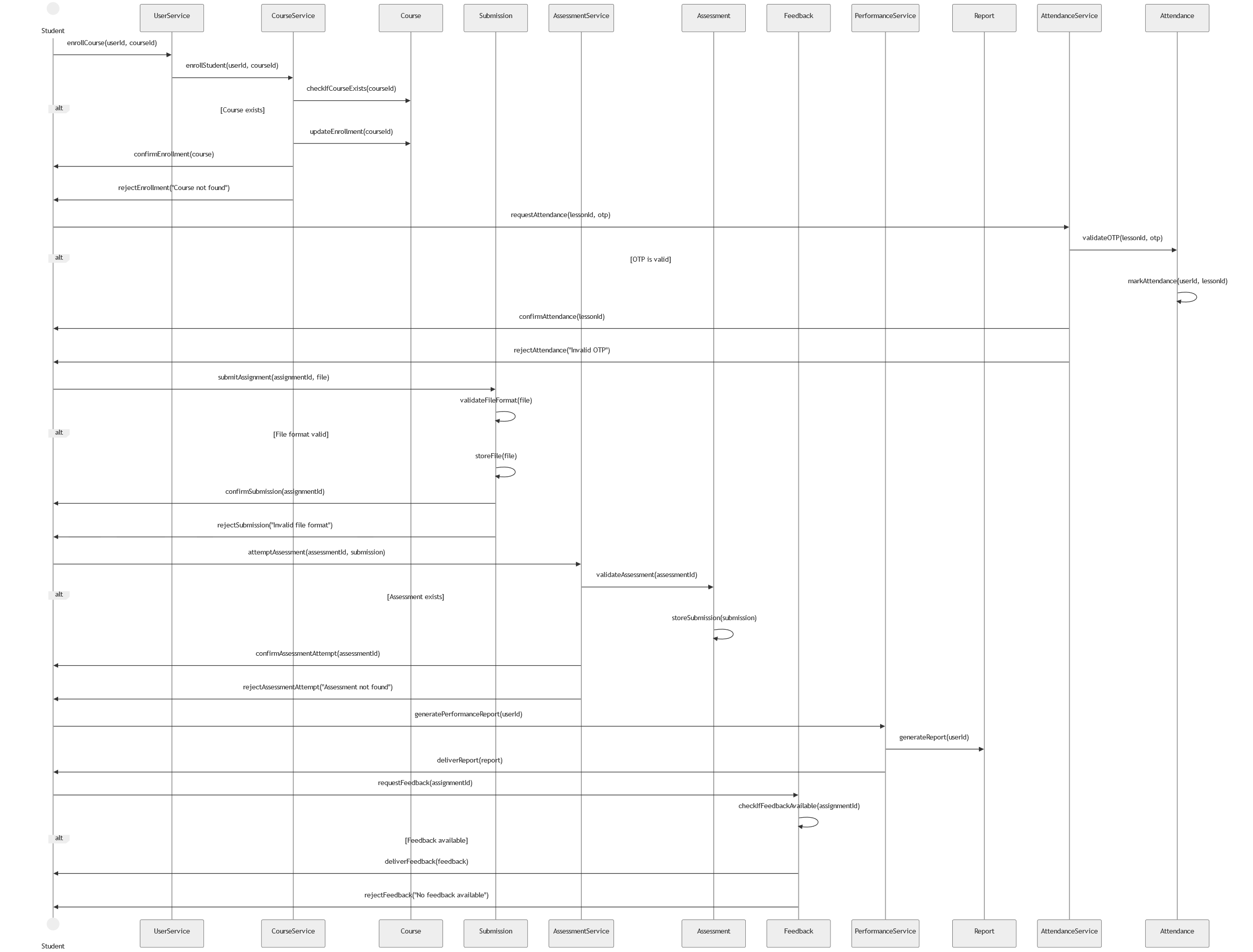
### Users: Indirect stakeholders relying on a reliable and secure system.

Admin Sequence Diagram

instructor Sequence Diagram



Student Sequence Diagram



### 3.3 Data Viewpoint

### Overview

### The data viewpoint focuses on how data is structured, stored, and exchanged within the system. It defines the relationships between entities such as users, courses, and assessments, ensuring data consistency and integrity.

### Concerns Addressed

### Structure of data and database schemas.

### Relationships between entities (e.g., users, courses, assessments).

### Data flow between modules for various operations like enrollment, grading, and notifications.

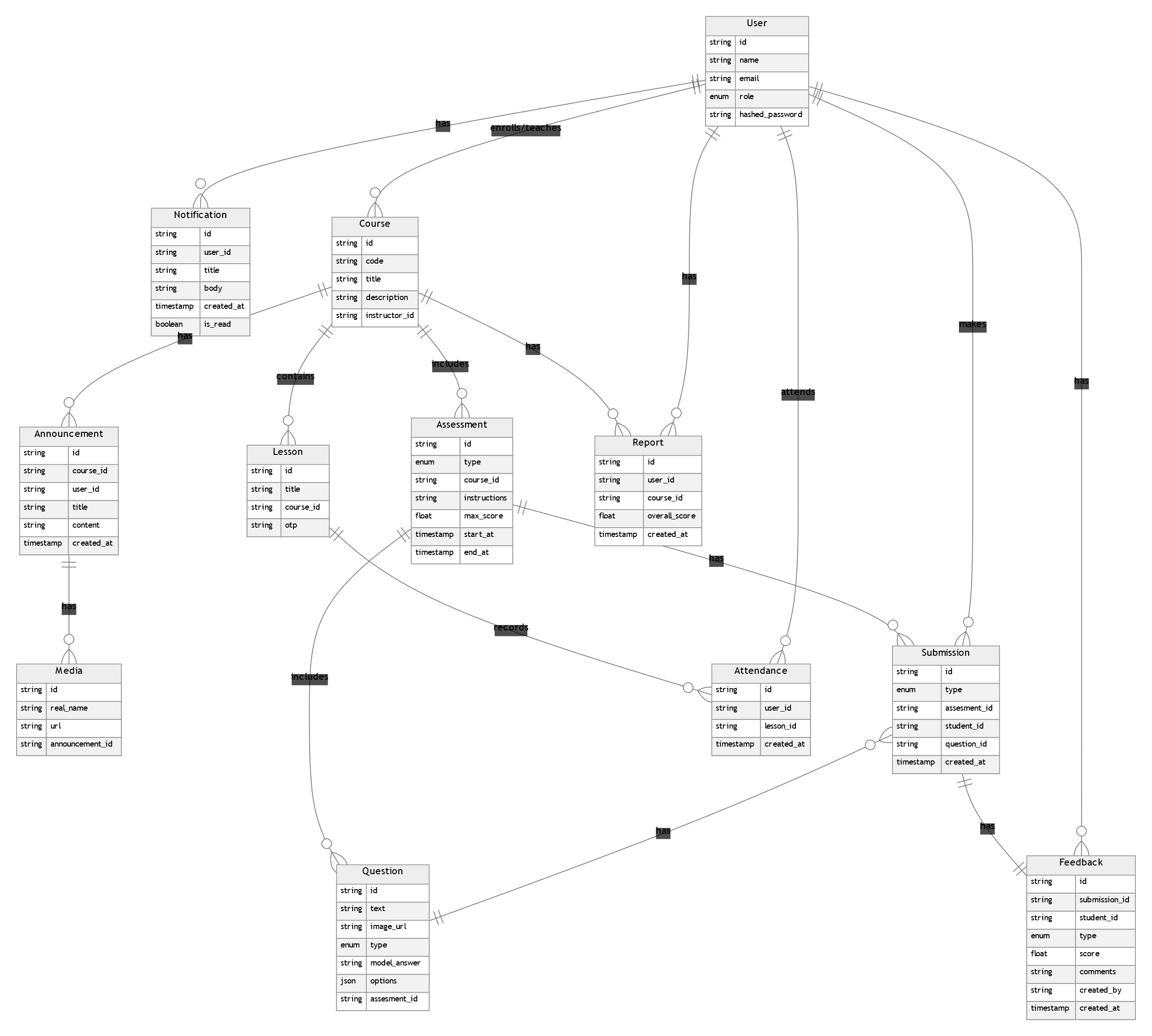
### Typical Stakeholders

### Database Administrators: Optimize data schemas and manage integrity.

### Developers: Interact with data through APIs and ensure proper CRUD operations.

### System Architects: Verify that data structures meet functional and performance requirements.

ERD



### 5.1 Known Inconsistencies

Current Status: No known inconsistencies have been identified at this stage of the architecture.

Future Management:

Any detected inconsistencies will be documented, including:

The affected components or models.

The reasons for the inconsistency (e.g., limited time, evolving requirements).

Proposed resolutions and their timelines.

### 5.2 Correspondences

Correspondences ensure consistency between the various elements of the AD and their implementation. Below are the key correspondences for the LMS:

Logical Components ↔ Spring Boot Components:

Each logical feature (e.g., authentication, course management) is implemented using specific Spring Boot components (e.g., controllers, services, repositories).

Example: AuthenticationService ↔ AuthenticationController.

Database Entities ↔ Domain Models:

Database tables are directly mapped to domain model classes for data representation.

Example: users table ↔ User entity.

REST Endpoints ↔ Service Operations:

Each REST API endpoint corresponds to a defined service operation.

Example: GET /api/courses ↔ CourseService.getAllCourses().

Stakeholder Needs ↔ Functional Modules:

Stakeholder concerns are addressed through corresponding functional modules.

Example: “Course Management” (Instructor need) ↔ CourseModule.

Views ↔ Code Implementation:

The functional and logical views of the system map directly to modules in the codebase.

Example: “Assessment View” ↔ AssessmentService + AssessmentController.

### 5.3 Correspondence Rules

Layered Rule:

Each layer (Presentation, Business Logic, Data) must only interact with its adjacent layer.

Example: Controllers can call services, but not directly access repositories.

Service Consistency Rule:

Every service method should align with a specific business requirement and be documented.

Example: registerStudent() ↔ Registration requirements.

Database-Model Rule:

Every database table must map to a domain model class.

Example: grades table ↔ Grade class.

Traceability Rule:

All implemented features must trace back to documented stakeholder needs and concerns.

Example: “Progress Tracking” (Student need) ↔ ProgressService.

Deployment Rule:

Deployment artifacts (e.g., JAR file) must include all modules defined in the architecture.

Violation Handling: - Any violation of correspondence rules will be documented with: - A description of the inconsistency. - The affected AD elements. - Proposed mitigation strategies and timelines.

# A. Architecture Decisions and Rationale

## A.1 Decisions

### Key Architectural Decisions for LMS

The following decisions are considered key to the architecture of the LMS:

Choice of Monolithic Architecture:

A monolithic architecture was chosen for simplicity in development and deployment.

Technology Stack (Java with Spring Boot):

The decision to use Java and Spring Boot ensures a solid, widely-used framework with a large development community.

Relational Database (PostgreSQL):

A relational database like PostgreSQL was selected to manage the structured data for courses, assessments, and user profiles.