Cairo University

Advanced Software engineering CS352

2024-2025

Project Phase 1

Java-Based Learning Management System (LMS)

Team members:

ID	Name	Email
20220424	Mohamed Rabea Mohamed	mhmdrby769@gmail.com
20220503	Youssef Medhat Ahmed	yossefmedhat432@gmail.com
20220104	Habiba Alaa Eldin	habibaalaa775@gmail.com
20220378	Yasmin Mohamed Kamal	yasminm2004@gmail.com
20220531	Habiba Ali Zein El-abideen	zain.habiba0@gmail.com
20220105	Habiba Amr Abdelfattah	Habibamr440@gmail.com

Content

1. Introduction

1.1 Identifying information

Learning Management System (LMS) Architecture

This architecture description pertains to a web-based Learning Management System (LMS). The system is a software-intensive application designed to facilitate online education and training by providing a platform for course management, user interaction, assessment, and performance tracking.

1.2 Supplementary information

The LMS features are designed to provides experience for administrators, instructors, and students. It allows users to register and log in, with access determined by their roles. Administrators see the system, manage users, and handle courses, instructors focus on creating and managing course content, tracking student progress, and grading assignments and quizzes. Students can enroll in courses, access materials, and submit their work.

The platform supports course creation with uploads and lesson structuring, along with an OTP-based attendance system. Assessments include quizzes and assignments that students can submit for grading. Feedback is provided through automated quiz responses and instructor comments on assignments.

Notifications keep users informed about important updates, such as enrollment confirmations and graded submissions. Additional features like role-based access control, performance analytics, and email notifications enhance security, functionality, and communication within the system.

1.3 Other information

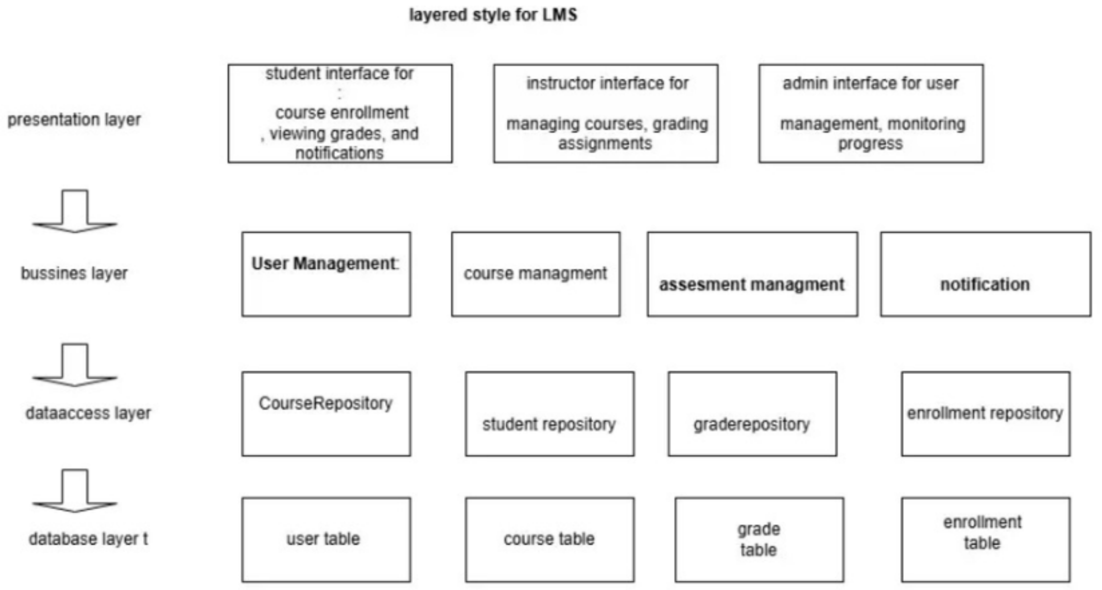
ARCHITECTURE SCENARIOS

Scenario ID	**Scenario Description**	**Expected Outcome**
Availability	The backend service for grading assignments experiences a temporary outage.	The grading requests are queued in the database until the service is restored. Once the service is back up, the grades are processed. If the service fails permanently, a fallback service is invoked to notify instructors.
Modifiability	A new user role (e.g., Teaching Assistant) needs to be added to the system	The new role is added to the user management service. The presentation

	with specific permissions.	layer and business logic layer are modified to integrate this new role seamlessly without affecting other layers or services.
Security	An instructor’s session remains inactive for 20 minutes after grading submissions.	The system automatically logs out the instructor and invalidates their session credentials to prevent unauthorized access. The instructor will need to log in again to resume their activities.
Modifiability	The MySQL database is replaced with a PostgreSQL database for better performance.	The data access layer is updated with new database connections. No changes are needed in the business logic or presentation layers, as they interact with the database through abstract data access interfaces.
Scalability	The system experiences a significant increase in concurrent user requests during final exams (peak traffic).	The application server is scaled by adding additional instances to handle the increased load. The load balancer distributes requests to ensure even traffic distribution without compromising performance.
Security	A student's session is idle for 15 minutes after submission of an assignment.	The system considers the session idle and automatically logs out the student, requiring re-authentication before accessing any sensitive data or features.

The LMS is a web-based platform designed to manage courses, user roles, assessments, and performance tracking. The architecture follows a layered style to promote separation of concerns, scalability, and maintainability. The primary layers include:

1. **Presentation Layer:** Handles user interaction and delivers a responsive interface. It focuses on rendering the user interface and communicating user inputs to the underlying layers.
2. Business Logic Layer: Processes requests, enforces rules, and coordinates interactions between components. This layer contains the core functionality of the LMS, such as handling course enrollment, quiz logic, and user role management.
3. **Data Access Layer:** Manages data persistence and retrieval. It serves as the intermediary between the business logic and the database, ensuring secure and efficient data storage and retrieval.
4. **Database Layer:** Stores all application data, including user information, course content, enrollment records, and assessment results. The database layer is optimized for reliability and scalability, ensuring the data is consistent and accessible to the data access layer.



connectors:API calls, service method calls, database queries

constraints: Separation of concerns, loose coupling, encapsulation, scalability, data integrity, and security.

How Layered Architecture Supports Non-Functional Requirements in the System:

Scalability: In Scalability Scenario 1, the system can scale by adding more application server instances to handle increased user load during peak times (e.g., exams). Layered Architecture allows this scaling without affecting the user interface or business logic.

Modifiability: In Modifiability Scenario 1, adding a new user role (e.g., Teaching Assistant) requires only changes to the business logic layer and presentation layer, while the data layer remains unaffected, showcasing how isolated changes make the system easy to modify.

Security: In Security Scenario 1 and Security Scenario 2, security measures such as session timeouts and automatic logouts are handled at the security layer, ensuring that unauthorized access is prevented, without affecting other layers.
Maintainability: In Modifiability Scenario 2, when switching from MySQL to PostgreSQL, only the data access layer needs to be modified. This separation simplifies testing and maintenance of the system.

Usability: In Security Scenario 1, the session management is handled without affecting the UI. Similarly, presentation layer updates for improved usability (e.g., adding new screens) don't require changes to the backend layers, making UI updates easy.

Performance: In Availability Scenario 1, the system queues requests during a backend outage, and once restored, the system processes them without affecting frontend interactions. Layered Architecture ensures backend issues don't disrupt user experience.

2. Stakeholders and concerns

2.1 Stakeholders

- **Students:** Need an intuitive interface for accessing courses, submitting assignments, and viewing grades.
- **Instructors:** Require efficient tools to manage courses, create quizzes, and track student performance.
- **Admins:** Oversee system operations, including user management and course approvals.
- Developers: Need modular and maintainable architecture.

Users:

- **Students:** They access courses, submit assignments, take quizzes, and view their grades.
- **Instructors:** They create courses, quizzes, assignments, and manage student performance.
- **Admins:** They manage the system's operations, user accounts, and approve course content.

Operators:

- **System Administrators:** They handle the maintenance, updates, and overall functionality of the system, ensuring that it runs smoothly.

Acquirers:

- **Institutional Management:** These are the decision-makers who purchase and implement the LMS for the institution, ensuring that it meets organizational needs.

Owners:

- **Educational Institution/University:** They own the system and its infrastructure. They oversee all activities related to the system's deployment and use.

Suppliers:

- **Third-Party Service Providers:** These could be companies or services that provide integrations for features like payment processing, external course content, or external authentication systems.

Developers:

- **Software Developers:** They design, implement, and test the system's core functionalities, including the user interface and backend components.

Builders:

- **DevOps Engineers:** They work on the deployment pipelines, server configurations, and manage the deployment of the system in various environments (e.g., development, staging, production).

Maintainers:

- **Maintenance Team:** After the system is deployed, the maintenance team monitors the system for bugs, issues, and performs routine updates and security patches.

2.2 Concerns

Concerns:

- **Students:** Usability, data privacy, and quick response times.
- **Instructors:** Efficiency in grading and quiz creation, real-time student tracking.
- **Admins:** System scalability, data integrity, and role-based access.

Purpose of the System:

- The primary purpose of the LMS is to facilitate course management, assessments, and performance tracking for students and instructors. It aims to provide an efficient, scalable, and secure platform for educational institutions to manage and deliver their courses online.

Suitability of the Architecture:

- The chosen layered architecture is well-suited to achieving the system's purpose by providing modularity, scalability, and maintainability. It ensures separation of concerns, making it easier to scale individual components (e.g., presentation layer, business logic layer, data layer) based on user needs. The architecture also allows for quick integration of new features and easy updates.

Feasibility of Construction and Deployment:

- The architecture is designed with the latest technologies such as Spring Boot, MySQL, and UML for modeling, which are well-known and reliable in the industry. The system can be feasibly constructed and deployed on modern cloud platforms with the flexibility to scale based on usage. The modular approach ensures that each layer can be developed and tested independently, which aids in faster development cycles.

Potential Risks and Impacts:

- **Risks:**
    - Scalability Risks: The system must handle an increase in concurrent users during peak times, such as course registration or exam periods. If not properly scaled, performance could degrade.
    - Security Risks: Potential vulnerabilities in user authentication and authorization, especially in managing sensitive student data.
    - Integration Risks: Compatibility issues when integrating third-party tools or services.
- **Impacts:**
    - Positive impacts include providing students and instructors with a streamlined educational experience.

  - Negative impacts could include system downtime, data breaches, or performance bottlenecks, affecting users' trust and engagement.

Maintenance and Evolution:

- The system will be maintained and evolved using a modular and maintainable architecture. Each layer can be independently updated to incorporate new features, security patches, or technology upgrades. Routine maintenance tasks (e.g., server monitoring, database optimization) will be managed by the IT operations team. The system's architecture allows for easy adaptation to future needs, such as adding new features, expanding to mobile platforms, or supporting additional integrations.

2.3 Concern–Stakeholder Traceability

Concern	**Students**	**Instructor**	**Admins**	**Developer**	**system Administrators**	**Institutional Management**
Purpose of the System	X	X	X			X
Suitability of the Architecture			X	X	X	
Feasibility of Construction and Deployment			X	X	X	X
Potential Risks and Impacts (Scalability)	X	X	X	X	X	
Potential Risks and Impacts (Security)	X	X	X	X	X	
Potential Risks and Impacts (Integration)		X	X	X	X	
Maintenance and Evolution			X	X	X	X

3 Viewpoints+

3.1 User Management Viewpoint

A) Overview:

This viewpoint focuses on managing the lifecycle of users in the system. It includes tasks like creating, updating, and deleting user accounts while maintaining secure and structured data handling.

B) Concerns and Stakeholders

I. Concerns:

- How are user profiles created, updated, and deleted securely?
- How are user roles and permissions assigned and maintained?
- How does the system enforce data retention policies?
- How are invalid inputs or unauthorized actions handled (Robustness)?

II. Typical Stakeholders:

- Admin: Needs to manage user creation, updates, and role assignments.
- Developers: Implement and maintain user management features.
- Users: Require user-friendly interfaces to manage their profiles.

III. Anti-concerns:

- Notifications or alerts unrelated to user management.
- Low-level implementation details of the storage system.

C) Model Kinds:

1. Class Diagram:
   - Represents users, roles, and relationships
2. Sequence Diagram:
   - Describes workflows for user creation, profile updates, and deletion.
3. Activity Diagram:
   - Models the workflows of user lifecycle tasks, including account creation, updates, and deletion.

D) Operations on View

Construction Methods:

- **Process Guidance:** Define actions needed for user management tasks.
- **Workflow Guidance:** Use secure design patterns for user lifecycle operations.

Interpretation Methods:

- Diagrams clarify the flow and relationships of user management tasks.

Analysis Methods:

- **Security Testing:** Validates secure storage and handling of user data.
- **Usability Testing:** Ensures intuitive user management interfaces.

Implementation Methods:

- Use frameworks such as Spring Security or Firebase Authentication for user management.

E) Correspondence Rules:

1. Class diagrams must reflect user roles and their attributes.
2. Activity diagrams illustrate the flow of events for all operations.

3.2 Course Management Viewpoint

a) Overview:

This viewpoint focuses on managing the lifecycle of courses in the system. It includes tasks like creating, updating, and deleting courses while ensuring role-based access and structured handling of course-related data.

b) Concerns and Stakeholders

I. **Concerns:**

- How are courses created, updated, and deleted securely?
- How are course materials managed and shared with users?
- How does the system handle enrollments and role-based access to courses?
- How are invalid inputs or unauthorized actions handled (Robustness)?

II. **Typical Stakeholders:**

- Admin: Manages course creation, updates, and access settings.
- Instructors: Create and manage course content.
- Students: Enroll in courses and access content relevant to their role.

III. **Anti-concerns:**

- User profile management unrelated to courses.

Notifications or alerts unrelated to course activities.

c) Model Kinds:

- **Class Diagram:**
  Represents courses, instructors, students, and their relationships (e.g., Course ↔ Instructor ↔ Enrollment).

- **Sequence Diagram:**
  Describes workflows for creating, updating, and deleting courses, as well as enrolling users.
- **Activity Diagram:**
  Models workflows such as creating a course, enrolling students, and publishing updates.

d) Operations on View

Construction Methods:

- Process Guidance: Identify all tasks involved in managing courses.
- Workflow Guidance: Use secure design patterns to ensure proper course handling and role-based access.

Interpretation Methods:

- Diagrams provide a clear understanding of workflows and relationships in course management.

Analysis Methods:

- Security Testing: Ensures secure handling of course data and access restrictions.
- Usability Testing: Validates user-friendly interfaces for course management.

Implementation Methods:

- Use frameworks like Spring Boot for course management APIs and database handling.

e) Correspondence Rules:

- Class diagrams must capture the relationships between courses, roles, and permissions.
- Sequence diagrams should illustrate the workflows for all course management operations.
- Activity diagrams must accurately reflect the lifecycle states of a course.

3.3 Performance Tracking Viewpoint

A) Overview:

This viewpoint focuses on tracking and visualizing user performance metrics within the system. It includes monitoring grades, course completion rates, engagement levels, and overall progress, ensuring role-based access to performance data.

B) Concerns and Stakeholders

i. Concerns:

- How is user performance data collected and stored securely?
- How are performance metrics calculated and displayed?
- How does the system ensure role-based access to performance data?
- How does the system handle invalid or incomplete data inputs (Robustness)?

ii. Typical Stakeholders:

- **Admin:** Needs aggregated performance data for system evaluation.
- **Instructors:** Require detailed performance metrics for their students to adjust teaching strategies.
- **Students:** Require access to their performance metrics for self-assessment and progress tracking.

iii. Anti-concerns:

- Course creation or user management workflows unrelated to performance tracking.
- Notifications not tied to performance metrics.

C) Model Kinds:

- Class Diagram:
  Represents users, performance metrics, and their relationships (e.g., User ↔ Course ↔ Performance).
- Sequence Diagram:
  Describes workflows for calculating, updating, and retrieving performance metrics.
- Activity Diagram:
  Models workflows such as calculating grades, updating progress, and generating performance reports.

D) Operations on View

Construction Methods:

- Process Guidance: Identify all performance metrics to be tracked and associated workflows.
- Workflow Guidance: Use patterns for data collection and visualization securely and efficiently.

Interpretation Methods:

- Diagrams clarify relationships and workflows in performance tracking.

Analysis Methods:

- Accuracy Testing: Validates the correctness of performance calculations and reporting.
- Usability Testing: Ensures data is presented in an understandable and actionable format.

Implementation Methods:

- Use frameworks like Hibernate for secure data management and Chart.js or D3.js for data visualization.

E) Correspondence Rules:

- Class diagrams must include all relevant metrics and their relationships with users and courses.
- Sequence diagrams should detail workflows for calculating and displaying performance data.
- Activity diagrams must accurately represent the lifecycle of performance tracking events.

3.4 Notification System Viewpoint

A) Overview

This viewpoint focuses on managing the notification system, which delivers timely and role-based notifications to users. It includes defining how notifications are created, customized, delivered, and managed for different events and user roles.

B) Concerns and Stakeholders

i. Concerns:

- How are notifications generated and delivered securely and promptly?
- How does the system ensure role-based relevance for notifications?
- How are notification settings customized for different user preferences?

- How does the system handle failed or delayed notifications (Robustness)?

ii. Typical Stakeholders:

- **Admin:** Configures global notification settings and monitors delivery success rates.
- **Instructors:** Require notifications for student activity updates, course progress, or submission deadlines.
- **Students:** Receive notifications about course updates, deadlines, or performance feedback.

iii. Anti-concerns:

- Performance tracking workflows unrelated to notifications.
- Low-level storage or communication protocol implementations.

C) Model Kinds:

- **Class Diagram:**
  Represents notifications, users, and events (e.g., User ↔ Event ↔ Notification).
- **Sequence Diagram:**
  Describes workflows for generating, customizing, and delivering notifications.
- **Activity Diagram:**
  Models workflows such as event detection, notification creation, and delivery processes.

D) Operations on View

Construction Methods:

- Process Guidance: Identify events requiring notifications and user roles involved.
- Workflow Guidance: Use event-driven design patterns for secure and efficient notification handling.

Interpretation Methods:

- Diagrams clarify workflows and relationships in notification generation and delivery.

Analysis Methods:

- Performance Testing: Validates timely delivery and system efficiency under high loads.

- Usability Testing: Ensures notifications are clear, relevant, and non-intrusive.

Implementation Methods:

- Use frameworks like Firebase Cloud Messaging or Amazon SNS for notification delivery and management.

E) Correspondence Rules:

- Class diagrams must include all relevant entities like users, events, and notifications with their relationships.
- Sequence diagrams should detail workflows for event-triggered notifications and user customization settings.
- Activity diagrams must represent the lifecycle of notifications from creation to resolution (delivered, failed, or dismissed).

3.5 Assessment and Grading Viewpoint

A) Overview

This viewpoint focuses on managing assessments and grading processes within the system. It includes creating, distributing, and grading assignments and exams, as well as providing feedback to users while ensuring security, accuracy, and role-based access.

B) Concerns and Stakeholders

i. Concerns:

- How are assessments created, assigned, and submitted securely?
- How are grading workflows managed and grades calculated?
- How does the system ensure role-based access to assessment and grading data?
- How does the system handle invalid submissions or grading errors (Robustness)?

ii. Typical Stakeholders:

- Admin: Manages grading policies and system-wide assessment configurations.
- Instructors: Create, distribute, and grade assessments and provide feedback.
- Students: Submit assessments and access grades and feedback.

iii. 3Anti-concerns:

- Notifications or activities unrelated to assessments or grading.
- User lifecycle management workflows outside of assessment contexts.

C) Model Kinds

- Class Diagram:
  Represents assessments, grades, and relationships (e.g., User ↔ Assessment ↔ Grade).
- Sequence Diagram:
  Describes workflows for creating assessments, submitting responses, and calculating grades.
- Activity Diagram:
  Models workflows such as assignment creation, submission handling, grading, and feedback generation.

D) Operations on View

Construction Methods:

- Process Guidance: Define tasks for creating, distributing, and grading assessments.
- Workflow Guidance: Use secure design patterns for data integrity and role-based access control.

Interpretation Methods:

- Diagrams illustrate workflows and relationships for managing assessments and grading.

Analysis Methods:

- Accuracy Testing: Validates grading calculations and feedback mechanisms.
- Usability Testing: Ensures the interface for creating and reviewing assessments is intuitive.

Implementation Methods:

- Use frameworks like Spring Boot for backend grading logic and Apache POI for handling assessment reports.

E) Correspondence Rules:

- Class diagrams must include entities such as assessments, grades, users, and their relationships.
- Sequence diagrams should detail workflows for creating, submitting, and grading assessments.
- Activity diagrams must represent the lifecycle of an assessment, from creation to archiving.

4 Views+

4.1 Conceptual View

The Conceptual View provides a high-level abstraction of the system, focusing on core entities like User, Course, Assessment, Role, and Grade and their relationships (e.g., Users enroll in Courses, Instructors create Assessments). It uses UML Class Diagrams to represent these entities and their interactions. The view helps clarify system structure and is essential for understanding core functionality, independent of technical implementation details.

The viewpoint governing the Conceptual View is the Logical Viewpoint, as it focuses on defining the system's core entities and their relationships without detailing implementation or deployment specifics. It provides a high-level abstraction that represents the system's structure and functionality, which aligns with the purpose of a logical viewpoint in architecture.

4.1.1 Models+

1. Component Diagram (Whole System)

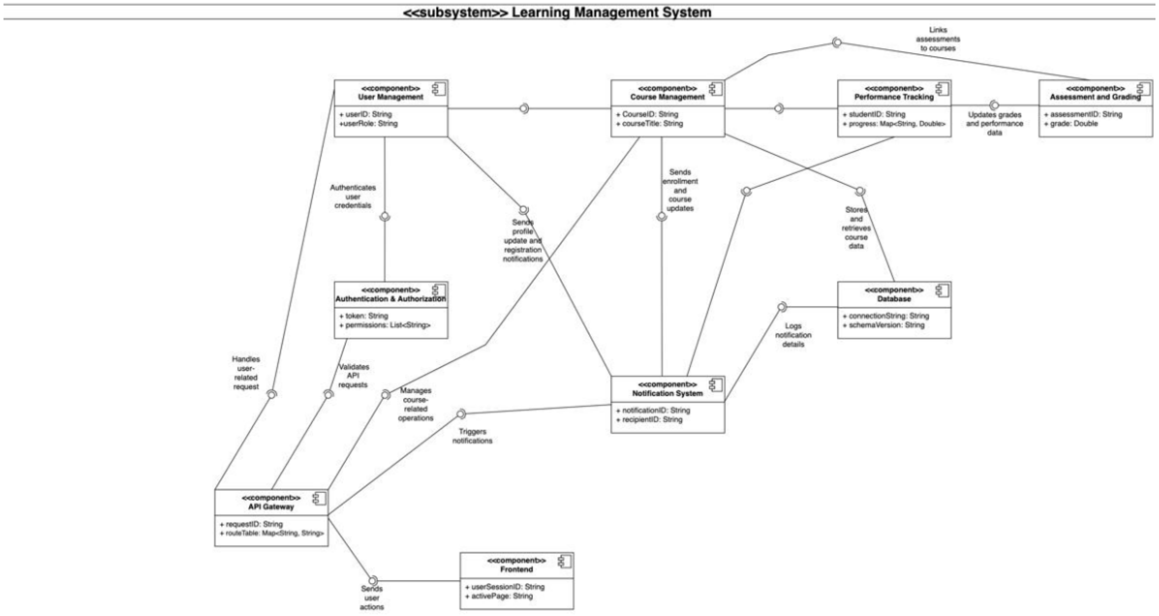
- Description: The Component Diagram provides a high-level view of the entire system's architecture, highlighting the main components (such as User Management, Course Management, Assessment Management, etc.) and how they interact. It shows how each component interacts with others and outlines the system's major modules.
- Concerns Addressed:
  - How are the core system components structured and interact? The component diagram illustrates the system's modular architecture, showing how different modules (e.g., User Management, Course Management) are organized.
  - How does the system maintain modularity and scalability? The diagram outlines independent components, which can be scaled or modified without affecting other parts of the system.

2. Class Diagram (Individual Models)

- Description: The Class Diagram provides a detailed, logical structure of the system, focusing on core entities like User, Course, Assessment, and Grade.
- Concerns Addressed:
  - entities structured the class diagram captures the relationships between entities, such as how users interact with courses or how grades are linked to assessments.
  - data integrity and consistency? By defining clear relationships and constraints between entities, the class diagram ensures proper data handling and integrity.
  - managing user roles and permissions? The class diagram includes roles as part of the User class, ensuring role-based access control within the system.

Repeat the section below for each model.

4.1.2 component diagram



4.1.3 Known Issues with View

The Conceptual View generally follows the conventions of the Logical Viewpoint, but some issues or discrepancies have been identified, which need to be addressed:

1. Inconsistency in Entity Relationships:
    - While the Class Diagram accurately depicts the relationships between core entities some entities such as Grade and Assessment may require further clarification regarding their interaction. The relationship between Grade and Assessment is not fully captured in terms of multiplicity, which could lead to confusion in understanding how multiple grades are assigned to one assessment.
    - Resolution: This needs to be revisited to specify whether a grade can be associated with multiple assessments or if one grade corresponds to a specific assessment.
2. Lack of Detailed Role Permissions:
    - The Class Diagram for User and Role entities doesn't include detailed attributes for user permissions, which is a key concern of the Logical Viewpoint. This might affect how the system's access control and role-based permissions are understood by stakeholders.
    - Resolution: Further refinement is required to clearly define user roles and their specific permissions within the Class Diagram.
3. Inadequate Representation of Security and Performance Concerns:
    - The Conceptual View lacks explicit modeling of security and performance aspects (such as encryption of user data and efficient handling of course content), which are part of the Logical Viewpoint concerns.

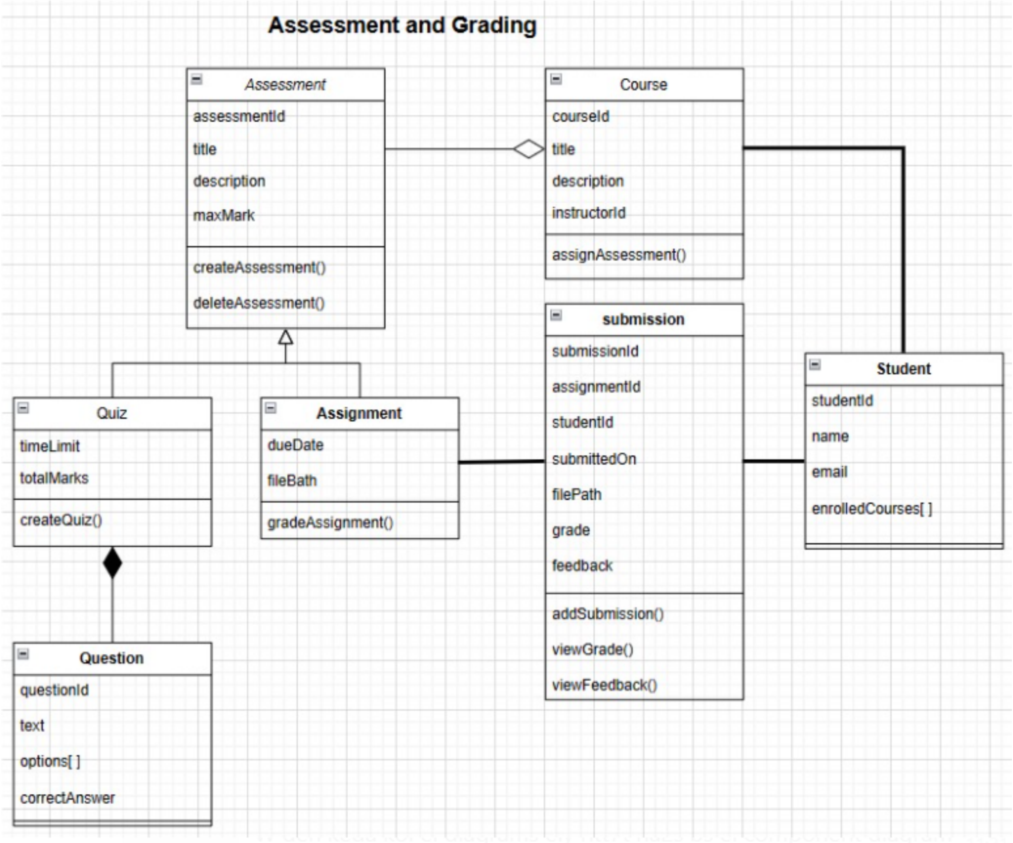
   - Resolution: Security models and performance benchmarks should be introduced in future iterations, possibly as annotations or additional views within the architecture description.

4. Unresolved Issues in Modularization and Component Dependency:
   - The Component Diagram broadly defines system modules but does not fully capture dependencies or modularization at a fine-grained level. For example, the interaction between the User Management and Course Management components should be detailed to show dependency chains and communication protocols.
   - Resolution: Additional details regarding component interdependencies and message flows should be added to ensure a clearer understanding of how components interact.
5. Open Decision Regarding Data Storage Model:
   - The Class Diagram does not specify the underlying data storage model (e.g., relational or NoSQL) for the system, which could influence decisions around scalability and performance.
   - Resolution: A decision regarding the data storage model needs to be made, considering factors such as query efficiency, scalability, and data consistency requirements.

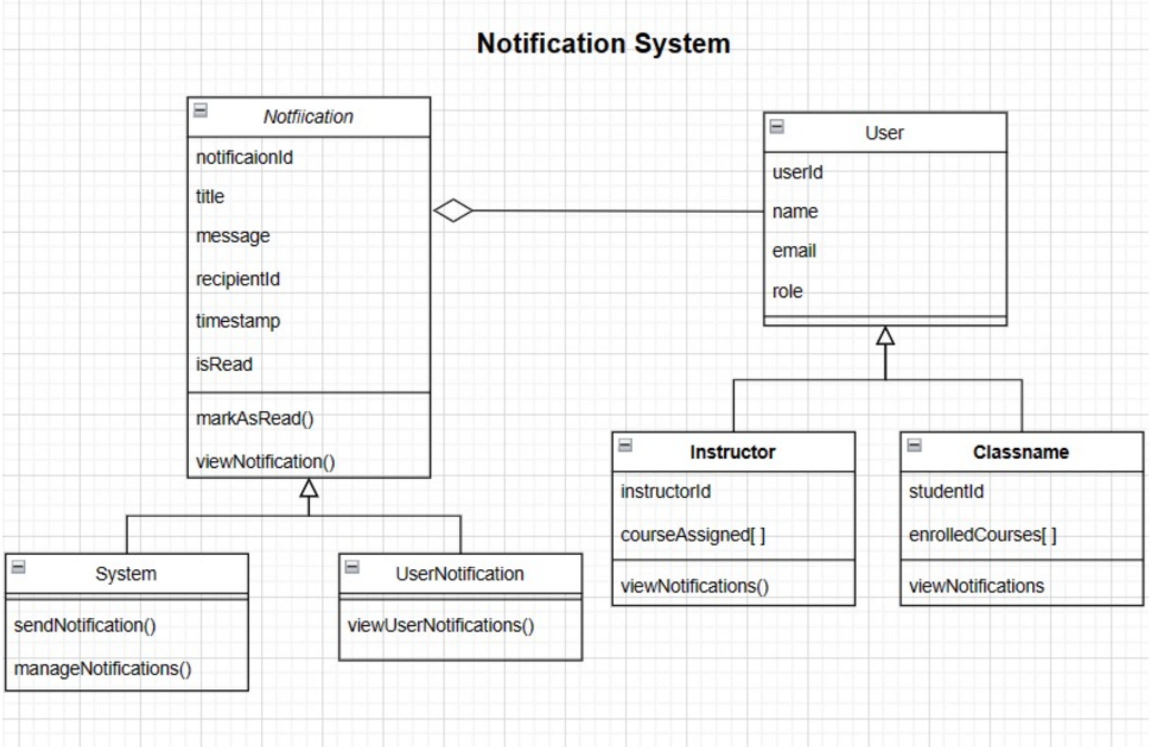
4.1.4 class diagrams

here are the class diagrams using submodule where each class diagram describes a subsystem in the large module:

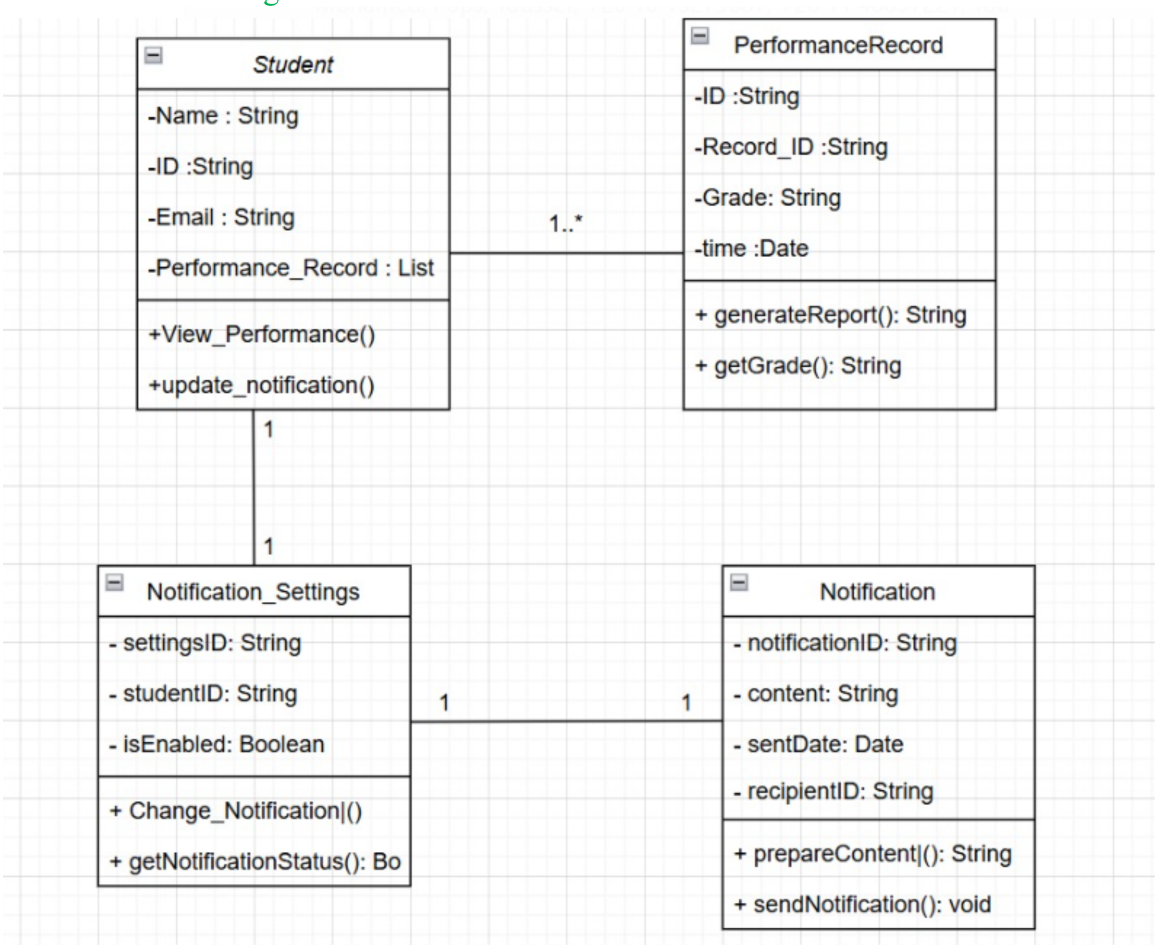
Assessment and grading:



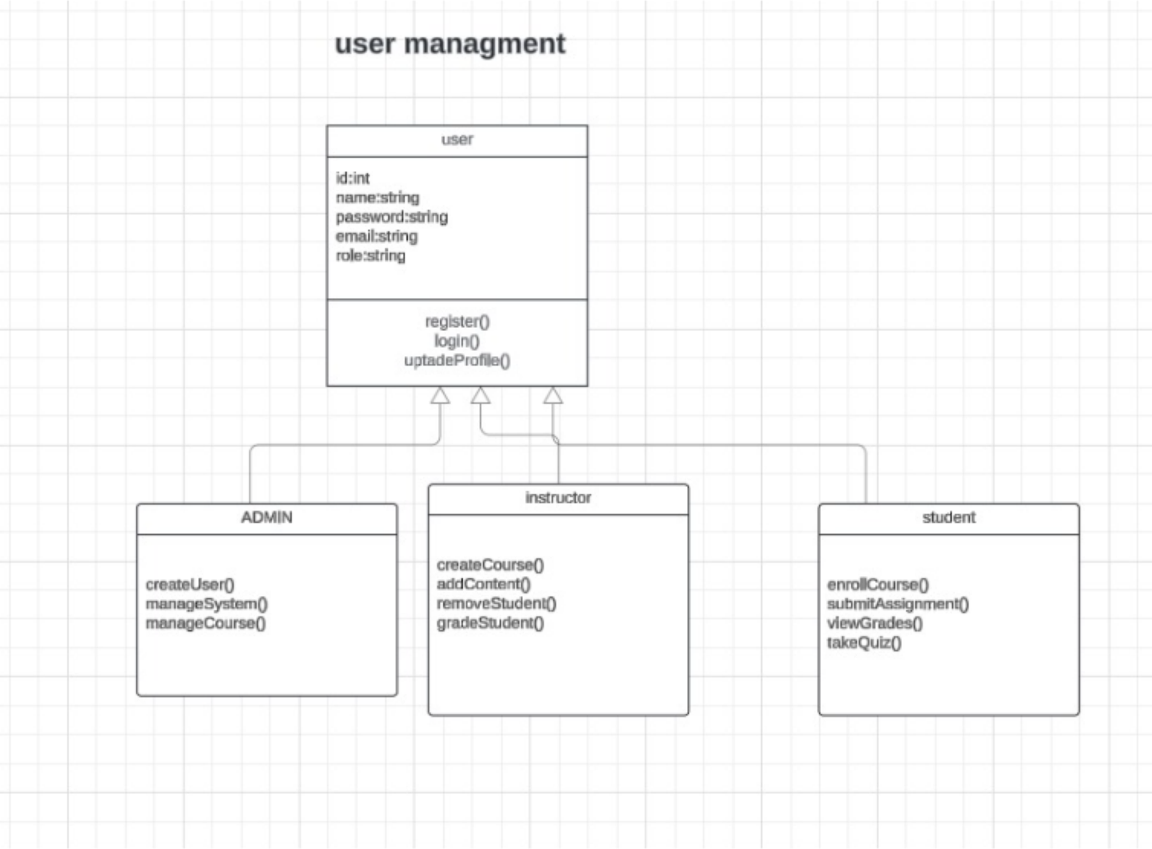
Notification system:



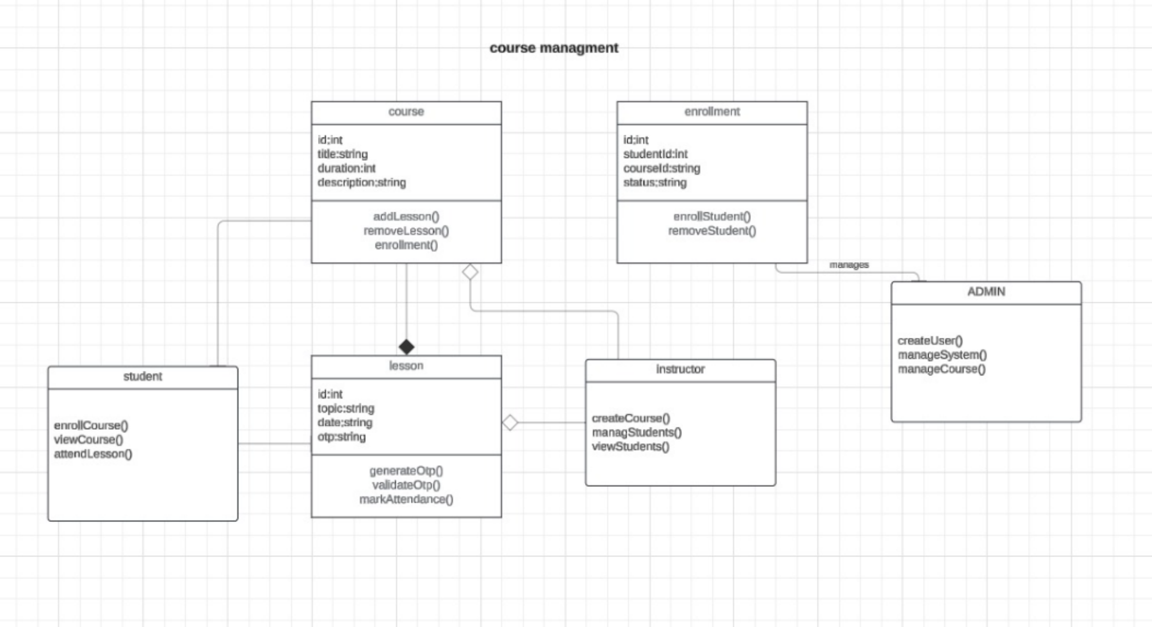
Performance tracking:



User management:



Course management:



4.2 Execution View

The Execution View highlights the system's runtime behavior, focusing on component interactions and workflows. It uses Sequence, Activity, and State Diagrams to illustrate request processing, data flow, and event handling.

The governing viewpoint is the Process Viewpoint, emphasizing runtime interactions, concurrency, and communication, ensuring functional requirements are dynamically met.

4.2.1 Models+

Activity Diagram (System Workflows)

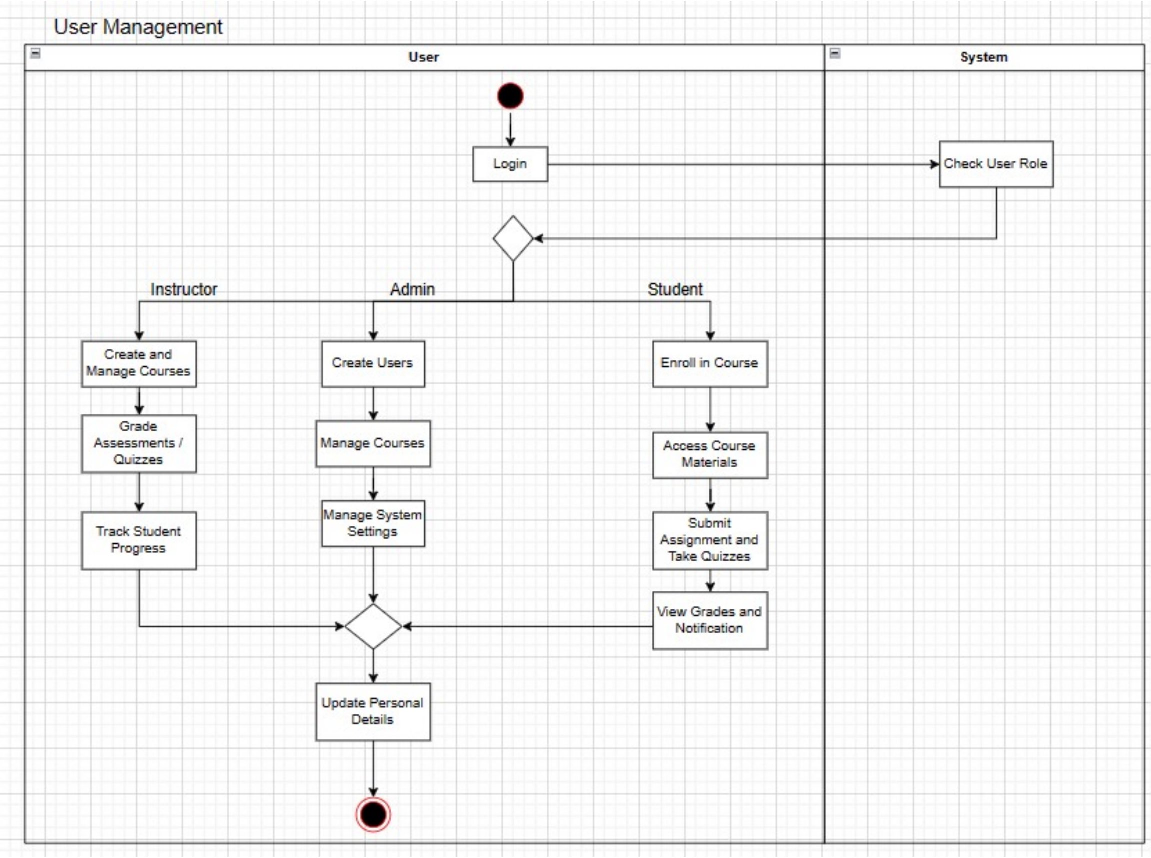
Description: The Activity Diagram represents the dynamic workflows and processes within the system, focusing on actions and decision flows for functionalities like user authentication, course enrollment, and assessment grading.

Concerns Addressed:

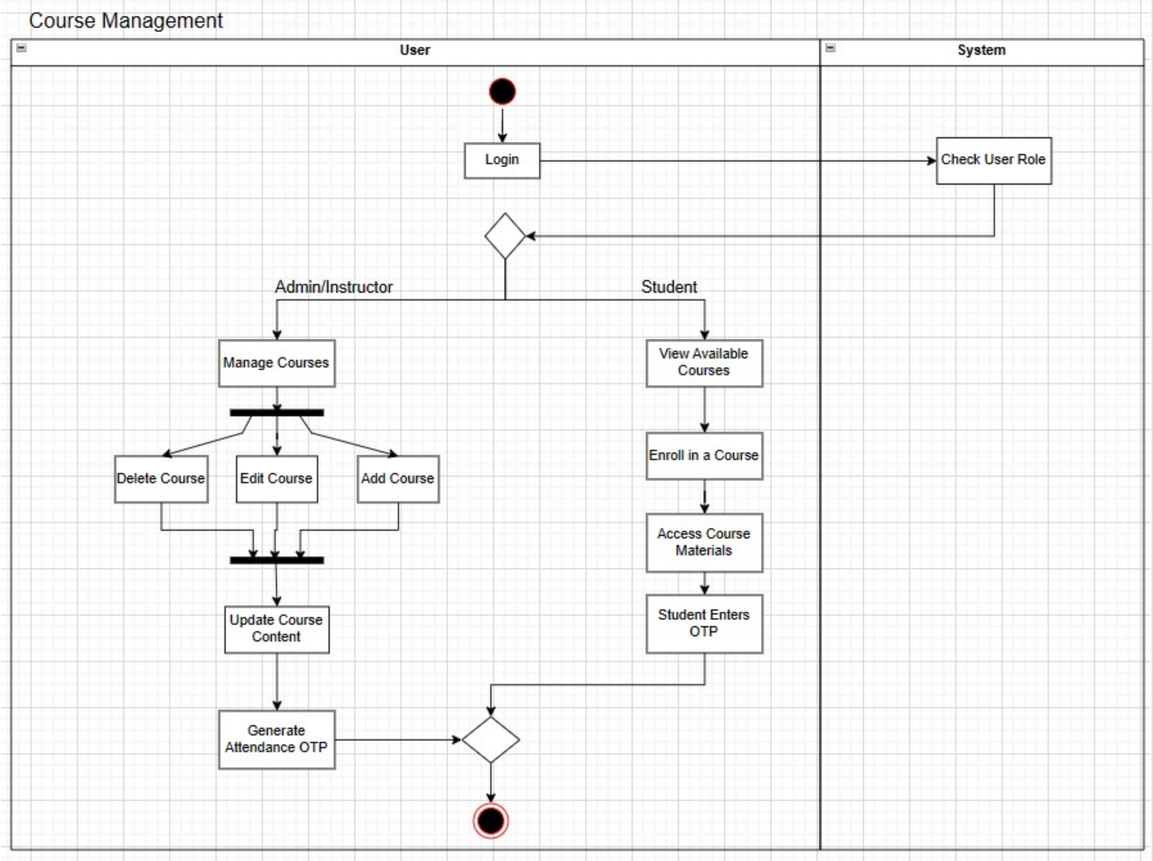
- System Processes Structured: The activity diagram illustrates the sequence of actions and decision points, ensuring clarity in system workflows.
- Handling Conditional Flows: By incorporating decisions and branches, the activity diagram ensures that workflows adapt to various scenarios, such as different user roles and system states.
- Role-Specific Workflows: The activity diagram highlights actions unique to each user role (e.g., admin, instructor, student), ensuring proper execution of role-based functionalities.

4.2.2 Activity Diagrams

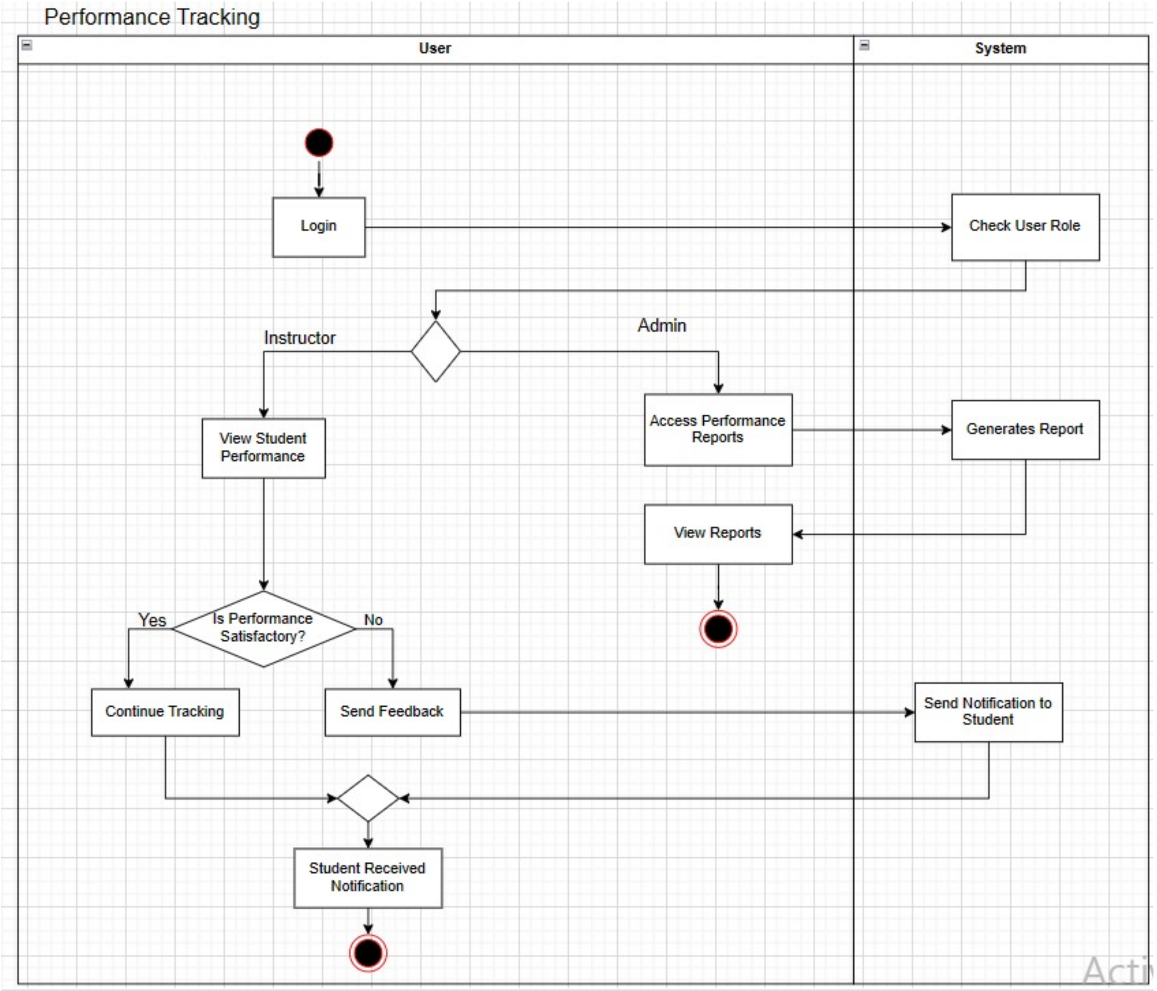
User management:


User Management
User
System
Login
Check User Role
Instructor
Admin
Student
Create and Manage Courses
Grade Assessments / Quizzes
Track Student Progress
Create Users
Manage Courses
Manage System Settings
Enroll in Course
Access Course Materials
Submit Assignment and Take Quizzes
View Grades and Notification
Update Personal Details

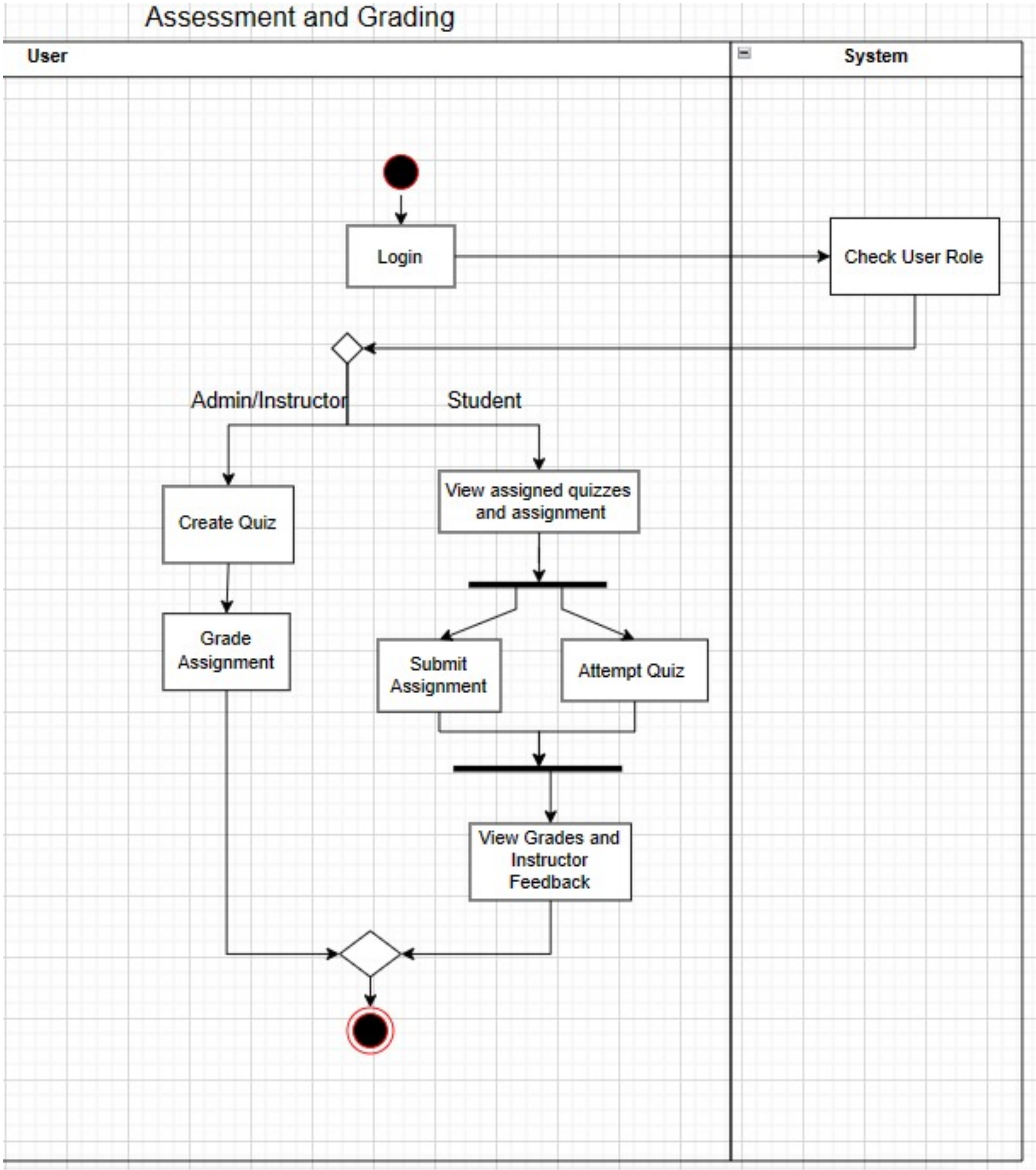
Course management:



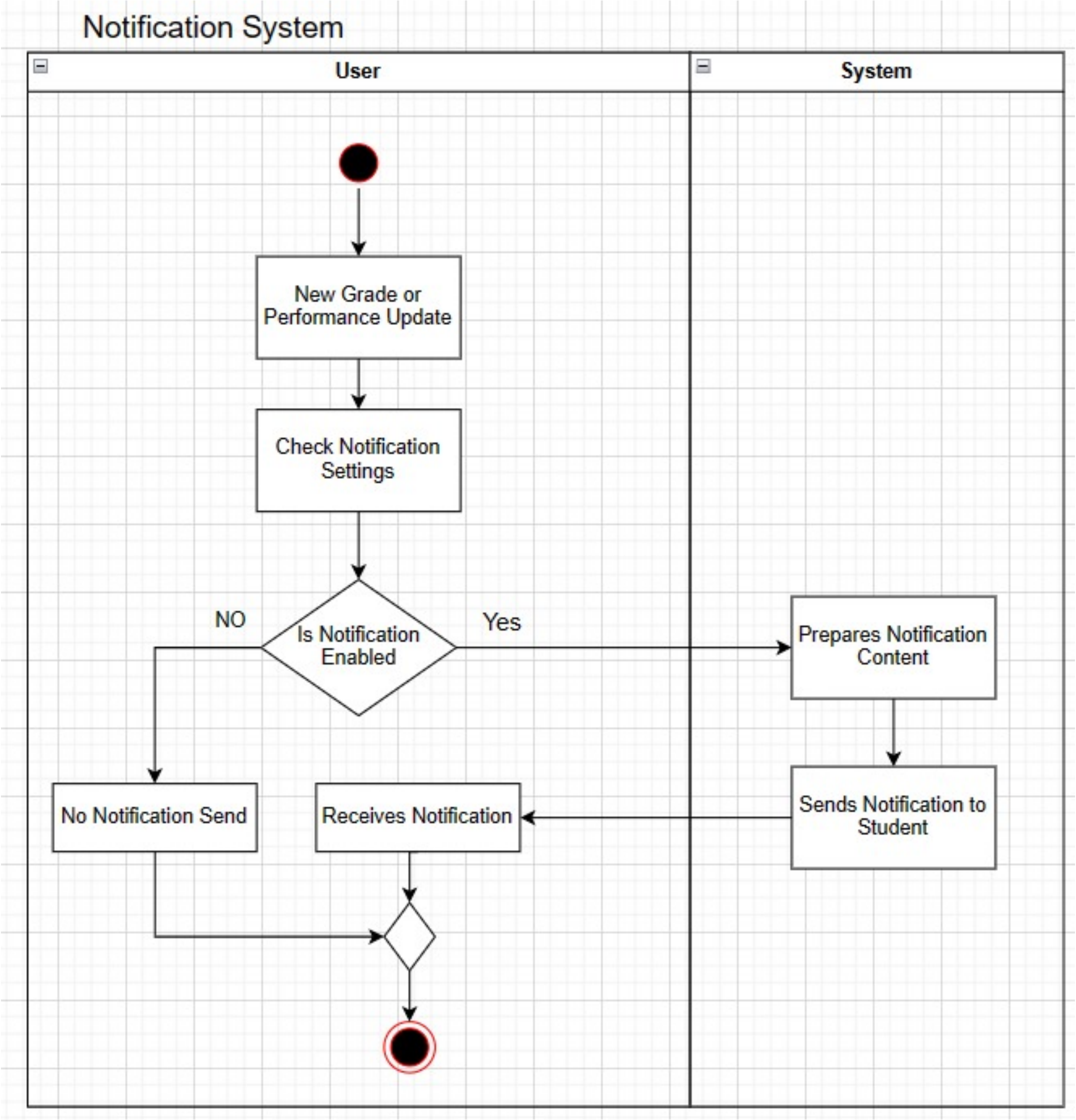
Performance tracking:



Assessment and grading:



Notification System:


Notification System
User
System
New Grade or Performance Update
Check Notification Settings
NO
Is Notification Enabled
Yes
Prepares Notification Content
No Notification Send
Receives Notification
Sends Notification to Student

4.3 Use Case View

The Use Case View focuses on the functional requirements of the system, representing the interactions between the system and its users (stakeholders). It helps to identify key use cases that describe the system’s functionalities and how the system meets the needs of users.

The governing viewpoint for the Use Case View is the Functional Viewpoint, as it focuses on the system's functionalities and interactions between users (stakeholders) and the system.

4.3.1 Models+

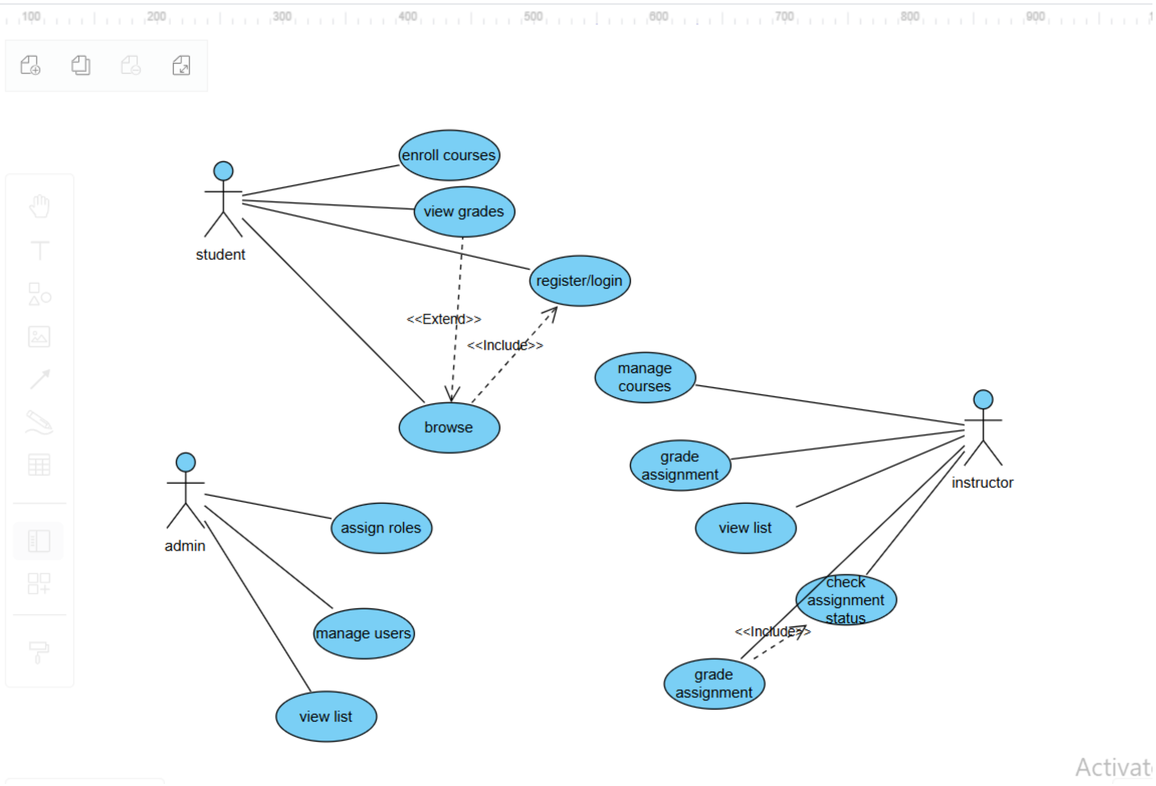
Use Case Diagram (System Workflows)

Description: The Use Case Diagram illustrates how users (actors) interact with the system to perform specific tasks (use cases). It defines the system's functional requirements by showing user actions like enrolling in courses, managing users, and grading assignments.

Concerns Addressed:

- Functionality: Identifies the core actions users can perform in the system (e.g., enrolling in courses, grading assignments).
- User Interaction: Focuses on how different users (students, instructors, admins) interact with the system.
- System Purpose: Demonstrates how the system meets the needs of its stakeholders by providing essential services and tasks.
- Security & Role Management: Ensures only authorized users can access specific functionalities, like user management and grading.
- Usability: Ensures ease of use by clearly defining user actions and interactions.

4.3.2 Use Case Diagram



5 Architecture Evaluation

5.1 Scenarios

ID	Scenario	Priority	Quality Attributes Addressed
S1	A student can enroll in a course and access materials immediately after registration.	High	Usability, Security
S2	An instructor can create a quiz with randomized questions from a question bank.	Medium	Usability, Modifiability
S3	Admins can track system usage statistics to plan for scalability.	Medium	Scalability, Maintainability
S4	The system supports 1,000 concurrent users during peak times without performance degradation.	High	Scalability
S5	Unauthorized users cannot access any course materials.	High	Security
S6	A new user role (e.g., Teaching Assistant) can be added to the system without affecting other roles.	Medium	Modifiability

The following architecture scenarios are defined to evaluate the system based on key quality attributes:

5.2 Evaluation Using ATAM Method

The architecture is evaluated using the ATAM (Architecture Tradeoff Analysis Method):

1. Scalability:
   - Concern: The system should handle a growing number of users (students, instructors, admins), especially during peak periods such as enrollment times.
   - Evaluation: The system uses a layered architecture, allowing independent scaling of components, particularly those that handle user requests and database load.
   - Risk: The architecture must ensure scalability by considering horizontal scaling for specific layers (course management, user management).
2. Security:
   - Concern: Ensuring the protection of sensitive data (personal information, grades) and preventing unauthorized access.
   - Evaluation: Spring Security is implemented for user authentication and role-based access control to protect system resources and ensure secure access.
   - Risk: There may be potential vulnerabilities in third-party components, such as database connections or user authentication, which could expose sensitive data if not secured properly.
3. Usability:
   - Concern: The system should provide an intuitive interface for all users (students, instructors, admins) to interact with their respective features.
   - Evaluation: The system provides role-based views for each type of user, ensuring that all interactions are simplified and intuitive.
   - Risk: The complexity of managing different user roles and permissions might introduce user confusion, especially if the user interface is not designed clearly.
4. Maintainability:
   - Concern: The system should be easy to maintain, with a clear separation of concerns between components and ease of debugging and upgrading.
   - Evaluation: The layered architecture provides modularity, ensuring each layer (user management, course management, assessments, etc.) is loosely coupled, making maintenance easier.
   - Risk: The growing complexity of interactions between layers may introduce challenges in system maintenance and debugging.
5. Modifiability:
   - Concern: The system should be flexible enough to accommodate new features and modifications without affecting existing functionality.
   - Evaluation: The layered architecture allows for modular changes, especially in areas such as course management, user management, or adding new features like reporting or notifications.
   - Risk: Tight coupling between layers could hinder the ability to modify one layer without affecting others, limiting the system’s flexibility.

5.3 Risks and Trade-offs

- Risk 1: Security Vulnerabilities: Potential security breaches if encryption and authentication mechanisms are not implemented or maintained correctly. There is a risk of unauthorized access to sensitive user data.
  - Trade-off: Implementing enhanced security measures could increase system complexity, reduce performance slightly, and require additional testing to avoid vulnerabilities.
- Risk 2: Scalability Bottleneck: Although the system's layered architecture supports scalability, some components (such as database management or the course management layer) may become bottlenecks under heavy load.
  - Trade-off: Focusing on scaling one layer (e.g., the course management system) may require additional hardware or cloud resources, increasing infrastructure costs.
- Risk 3: Maintenance Overhead: The more modular the architecture, the easier it is to maintain. However, managing dependencies between various layers and components could become complex as the system grows.
  - Trade-off: To ensure long-term maintainability, the system may need regular refactoring to keep components decoupled and reduce technical debt.
  -

6. Architecture decisions and rationale

Decisions

A) Decision 1: Use of Layered Architecture Style

- Unique Identifier: Decision 001
- Decision Statement: The architecture will use a layered architecture style, with clear separation of concerns across components such as user management, course management, and assessments.
- Concerns Addressed: Modifiability, Scalability, Maintainability
- Owner of Decision: Architecture team
- Affected AD Elements: Conceptual View, Execution View, Component Diagram
- Rationale: The layered architecture ensures modularity, scalability, and separation of concerns, making it easier to scale individual components (e.g., user authentication, course management) and maintain the system over time. This style aligns with the system's goals of flexibility and modularity, allowing for future enhancements and easy debugging.
- Forces and Constraints:
  - Forces: Need for scalability and ease of maintenance.
  - Constraints: Complexity of managing interactions between layers.
- Assumptions: System load may increase, and modularity will facilitate easier handling of scaling issues.
- Alternatives Considered:
  - Monolithic Architecture: Not chosen due to the lack of scalability and modularity, making it harder to manage large user bases or frequent feature changes.
  - Microservices Architecture: Considered but rejected due to complexity and high overhead in managing multiple independent services for this system size.

B) Decision 2: Use of Spring Boot and MySQL for Backend

- Unique Identifier: Decision 002
- Decision Statement: The system will use Spring Boot for backend development and MySQL as the relational database management system (RDBMS).
- Concerns Addressed: Performance, Maintainability, Scalability, Security
- Owner of Decision: Development team
- Affected AD Elements: Execution View, Component Diagram
- Rationale: Spring Boot provides a robust and scalable framework for building web applications, while MySQL is a widely used, secure, and reliable database system that integrates seamlessly with Spring Boot. This combination is well-suited for handling the expected volume of data and traffic in the LMS, supporting future scalability needs.
- Forces and Constraints:
    - Forces: The need for secure, fast, and scalable backend development.
    - Constraints: Database management complexity.
- Assumptions: The team has prior experience with Spring Boot and MySQL, ensuring faster development and deployment.
- Alternatives Considered:
    - Node.js with MongoDB: Rejected due to lack of experience and concerns about consistency in managing complex relationships in non-relational databases.
    - Java EE (Enterprise Edition): Chosen against due to its heavier setup and configuration requirements.

C) Decision 3: Role-Based Access Control (RBAC)

- Unique Identifier: Decision 003
- Decision Statement: The system will implement Role-Based Access Control (RBAC) to manage permissions and ensure secure access to features based on user roles (Admin, Instructor, Student).
- Concerns Addressed: Security, Usability
- Owner of Decision: Security team
- Affected AD Elements: Conceptual View, Execution View, Security Model
- Rationale: RBAC provides a structured and scalable way of handling user permissions, ensuring that users only have access to resources appropriate to their roles. This decision meets the security requirements of the system and reduces the complexity of managing user permissions dynamically.
- Forces and Constraints:
    - Forces: Need for fine-grained security control and ease of management.
    - Constraints: Complexity in managing multiple roles and associated permissions.
- Assumptions: The role structure will remain relatively stable, with only minor adjustments needed as new roles or permissions are added.
- Alternatives Considered:
    - Attribute-Based Access Control (ABAC): Rejected due to increased complexity and the lack of clearly defined attributes across roles in the LMS system.
    - Discretionary Access Control (DAC): Rejected due to concerns about potential security risks and difficulties in managing user-specific access rights.

D) Decision 4: Use of UML for Documentation

- Unique Identifier: Decision 004

- Decision Statement: The architecture documentation will use UML (Unified Modeling Language) for designing and visualizing system components, interactions, and data flow.
- Concerns Addressed: Usability, Maintainability
- Owner of Decision: Documentation team
- Affected AD Elements: Conceptual View, Execution View, Class Diagram, Component Diagram, Sequence Diagrams
- Rationale: UML provides a standardized way to represent system designs and interactions. It ensures clarity in communicating the system structure and behavior to various stakeholders (developers, project managers, etc.), helping to reduce misunderstandings and development errors.
- Forces and Constraints:
  - Forces: Need for a clear, standardized approach to documenting the architecture.
  - Constraints: The potential overhead of maintaining detailed diagrams as the system evolves.
- Assumptions: UML will be regularly updated as the system evolves and new features are added.
- Alternatives Considered:
  - Flowchart Diagrams: Not chosen due to their limited ability to represent complex interactions and system components.

Entity-Relationship Diagrams (ERD): Rejected for system-wide documentation due to its focus on data modeling rather than behavior or interactions.