Software Architecture and Detailed Design Documentation

Product Name : AlgoLab
Team Number : 3

Team Members
Muskan Sagar
Keerthan Mahesh
Swathi Selvakumaran
Barath Bhaskaran
Sachin Velmurugan

1. Preliminary Architecture Decisions

A. System name and description

Algolab is a web-based platform for boosting your expertise, broadening your understanding, and getting ready for technical interviews. Our LMS is specifically tailored to meet the unique needs of interviewees, providing a seamless and efficient experience from start to finish. It encompasses coding challenges commonly encountered during interview rounds, along with all the necessary resources and links to assist in problem-solving. Users can get interview-ready by mastering and tackling coding challenges, and They can keep track of their learning progress.

B. Feature list and description

Feature Name	**Feature Description**
Register user	User shall be able to register themselves with their details
Login	User shall be able to login
Profile Management	User shall be able to manage their profile
Create Courses	Instructors/Admin shall be able to create courses.
Modify Courses	Instructors/Admin shall be able to modify their own course details.
Create Lessons	Instructors/Admin shall be able to create the contents of lessons present in each course.
Modify Lessons	Instructors/Admin shall be able to modify the contents of lessons present in each course
View Course Feedback	Instructors shall be able to view the feedback that they receive from students for the courses they teach.
View Courses	Candidates shall be able to view the list of all interview courses.
Register for Courses	Candidates shall be able to register for courses
View Course Modules/RoadMap	Candidates shall be able to view the details of the courses such as course roadmap, course contents, course modules, etc.

View Lessons and its contents	Candidates shall be able to view the list of lessons in a course. Users shall be able to view the contents of each lesson such as video tutorials, articles and programming questions
Track Progress for Courses	Candidates shall be able to track their progress for registered courses such as challenges solved, time taken, and success rate.
Provide Course Feedback	Candidates shall be able to give feedback for the courses and programming challenges that they take.
Report Error/Inappropriate Content	Candidates shall be able to report admin for errors, inappropriate content, or bugs in the application.

C. Architectural design decisions

i. Technology Trade-off

	License	Team Experience	Performance	Community support	Compatibility	Maintainability	complexity	Total
	10	**20**	**20**	**10**	**10**	**10**	**20**	100
Front end framework								
Angular	3	4	4	3	5	4	3	370
React	5	5	5	3	5	5	5	480
Vanilla Javascript	4	3	4	5	3	4	5	400
Database management								
Postgres	4	3	3	4	3	4	3	330
MongoDB	5	5	5	5	5	5	5	500
MySQL	4	4	3	3	3	4	3	340
Back end technology								
Nodejs	3	2	3	2	2	3	3	260
spring	4	4	5	5	4	4	5	450
Deployment								
Cloud	3	3	4	3	4	4	4	360
On-premise	5	5	4	4	5	5	5	470

IDE								
IntelliJ	5	5	4	5	5	5	5	480
Eclipse	3	4	4	4	3	4	3	360
VS Code	5	5	5	5	5	4	4	470

ii. **Technologies Selected**

The following technologies have been selected based on technology tradeoffs -

1. **Front-end framework:** React
2. **Database management:** MongoDB
3. **Back-end technology:** Springboot
4. **Deployment:** On-premise
5. **IDE:** IntelliJ

Advantages of selected technologies:

1. React's architecture optimizes rendering, resulting in faster and more responsive user interfaces.
2. MongoDB is a flexible, document-based NoSQL database, which can handle unstructured or semi-structured data.
3. Spring has a large and active community, extensive documentation, and third-party integrations.
4. On-premise deployment offers full control over hardware, data, and infrastructure.
5. IntelliJ provides code completion, refactoring, and other productivity-enhancing tools.

Disadvantages of selected technologies:

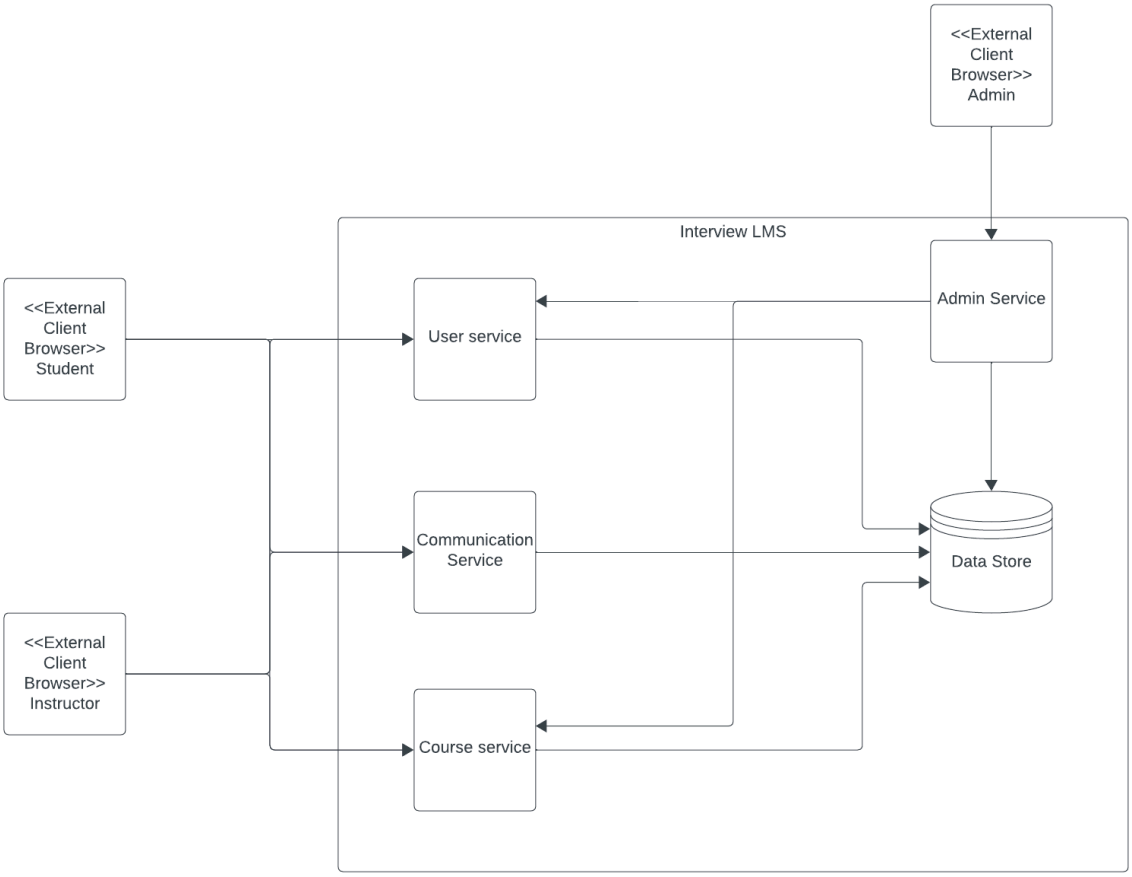
1. React evolves rapidly, which can lead to updates and potential issues for projects.
2. In MongoDB, complex queries can lead to performance issues.
3. Managing dependencies and ensuring compatibility can be a challenge with Spring.
4. On-premise applications may not be as accessible as cloud-based ones.
5. IntelliJ can be resource-intensive, leading to slower performance on less powerful machines.

1.D. Risks

Technology/tools Risk	Probability	Impact	Score(probability*impact)
Security (spring)	1	2	2
Compatibility (react)	1	2	2
Complex Queries(mongo db)	2	2	4
Resource intensive(IntelliJ)	1	1	1

Technology risks	**Prevention or Mitigation actions**
Security (spring)	conduct security testing, such as penetration testing and code reviews, to identify and fix vulnerabilities
Compatibility (react)	prepare a backup plan in case an update introduces severe compatibility issues.
Complex Queries(mongo db)	Identify and create appropriate indexes for the fields commonly used in your queries. Indexes can significantly improve query performance.
Resource intensive(IntelliJ)	Periodically clean and rebuild your projects to eliminate unnecessary cache and temporary files.

1.E



The structural architecture of your Interview Learning Management System (LMS) includes various components that interact to provide the necessary functionality. Let's provide a detailed description of each component and its role within the architecture:

1. External Client: Browser

The browser serves as the primary user interface for your Interview LMS, allowing users to access the system and interact with its features. Users can log in, access courses, view content, submit assignments, and participate in discussions through the browser.

2. User Service

The User Service is responsible for managing user-related operations within the system. It includes user registration, authentication, and authorization.

User profiles, roles, and permissions are maintained by this service, ensuring secure access and personalized experiences.

It interacts with the Data Store to access and update user-related data, such as profiles and preferences.

3. Communication Service
The Communication Service handles various communication aspects of the LMS, including sending announcements, feedback, and notifications to users.
It facilitates the dissemination of important information, updates, and reminders to students and instructors.
This service also collaborates with the Course Service to ensure timely communication regarding course-related activities.

4. Course Service
The Course Service is responsible for managing the creation, organization, and delivery of courses within the LMS.
Instructors can create and configure courses, upload course materials, schedule lessons, and manage enrollments.
It interacts with the Data Store to access and update course data, including course content and scheduling information.

5. Data Store
The Data Store is a critical component that serves as the database for storing structured data. It houses user data, course information, announcements, feedback, and other relevant data.
The User Service, Course Service, and Communication Service access and update data stored in this component.

6. Admin Service
The Admin Service provides tools and functionality for system administrators to manage the LMS effectively.
Administrators can perform operations such as user and role management, system configuration, and customization of system-wide settings.
This service ensures the smooth administration and maintenance of the LMS.

Architecture styles and tactics

2. A Driving quality attributes

The main quality attributes that drive our architecture styles and tactics are:

1. Security

Security in the context of a software system refers to the protection of information and system resources from malicious attacks, unauthorized access, alteration, theft, or damage. It involves ensuring data integrity, confidentiality, and availability.

High-priority quality scenarios include:

1. **SQL Injection Prevention**

Source	Malicious user or bot.
Stimuli	A user or bot attempts an SQL injection through the search field on the web interface.
Artifact	The web application's backend system, including the database management system.
Environment	The web application is in operation with multiple users accessing it.
Response	The system detects the attempt, prevents the SQL injection, logs the incident, and alerts the administrator.
Response measure	The system identifies and mitigates SQL injection attempts with 100% accuracy, ensuring no unauthorized data retrieval or database modification occurs.

2. **Unauthorized Access Prevention**

Source	Registered user.
Stimuli	A user attempts to access privileged content or data for which they do not have permission.
Artifact	Access control mechanisms within the software system which has the capability to edit/delete course materials.
Environment	The system is live and available to users.
Response	The system enforces access control policies, denying the request and logging the attempt.

Response measure	Unauthorized access attempts are rejected and logged 100% of the time without affecting the performance of authorized users.

2. Scalability

Scalability is the ability of a software system to handle increased load without compromising performance, such as the capacity to support more users, handle more transactions, and process more data.

1. **User Load Increase Handling - Scaling Infrastructure to Accommodate User Surges**

Source	A surge in users.
Stimuli	There is a sudden increase in the number of users attempting to use the web application due to an upcoming popular interview season.
Artifact	The software system's load balancer and resource management components.
Environment	The system is operational and under increased load.
Response	The system dynamically allocates additional resources to handle the increased load without service degradation.
Response measure	The system scales to support up to a 300% increase in active users with no more than a 1% increase in response time.

2. **Data Growth Management**

Source	A growing database.
Stimuli	The web application accumulates a vast amount of data over time due to the continuous addition of new learning materials, user data, and performance tracking.
Artifact	The database servers and storage systems.
Environment	The system is in continuous operation, with an expanding dataset.

Response	The system's database scales horizontally and vertically to accommodate the growing data without performance loss.
Response measure	The database handles data growth at a rate of 100% per year with no degradation in query response times.

3. Performance

Performance refers to the responsiveness and speed with which a software system operates. It measures how quickly a system performs under a particular workload and is crucial for user satisfaction and system efficiency.

1. **Video Content Delivery Optimization**

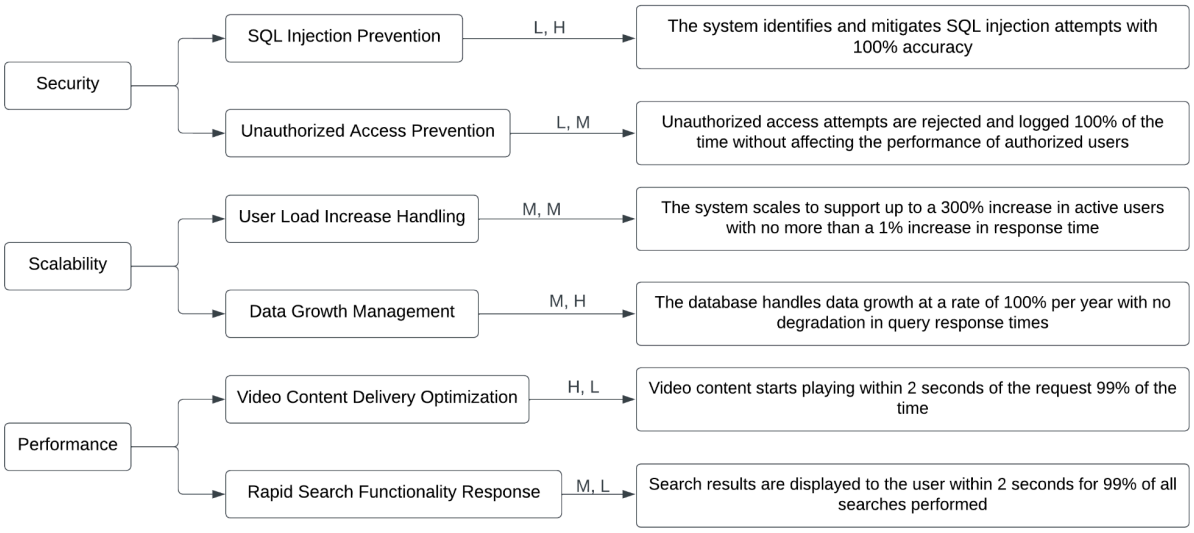
Source	Concurrent users accessing learning content.
Stimuli	Multiple users request video tutorials from the web application.
Artifact	Media player servers.
Environment	The system is in normal operation.
Response	The server serves the video content efficiently, using adaptive bitrate streaming to increase the users' video playback speed.
Response measure	Video content starts playing within 2 seconds of the request 99% of the time, and maintains a buffer rate of less than 0.5% rebuffering incidents relative to total play-time.

2. **Rapid Search Functionality Response**

Source	Individual user.
Stimuli	A user utilizes the search functionality within the web application to find specific interview preparation materials.
Artifact	The search engine and database indexing services.
Environment	The system is operational with a varying number of user requests.

Response	The system efficiently processes the search query using optimized search algorithms and indexed data to provide relevant results quickly.
Response measure	Search results are displayed to the user within 2 seconds for 99% of all searches performed, regardless of the size of the dataset.

Utility Tree



Constraints

1. The web application shall be built using specific languages and frameworks - React for front-end and Springboot for back-end and the system should be compatible with MongoDB.
2. The web application shall be fully functional on the latest versions of Chrome, Firefox, Safari, and Edge.
3. The web application shall comply with accessibility standards such as WCAG (Web Content Accessibility Guidelines) 2.1 for users with disabilities.
4. The web application shall comply with relevant data protection and privacy laws of GDPR (General Data Protection Regulation).
5. The content on the web application shall be culturally appropriate and sensitive to global audiences.

2.B. Proposed architecture styles

The architecture styles used for our system are:

1. Model View Controller (MVC)
2. Client-Server
3. REST architecture
4. Event driven architecture

MVC:

Choosing the Model-View-Controller (MVC) architectural style for an Interview Learning Management System (LMS) offers several advantages, especially when it comes to developing web applications and systems like an LMS. Here are some reasons to consider using MVC for an Interview LMS:

Separation of Concerns: MVC enforces a clear separation of concerns within the application. In the context of an LMS, this means separating the core application logic (Model), user interface (View), and user interactions (Controller). This separation makes it easier to manage, maintain, and update each component independently.

Modular and Maintainable Code: By organizing your code into separate components, it becomes more modular and maintainable. You can work on the Model, View, or Controller without significantly impacting the other parts of the system. This is essential for a complex system like an LMS, which may require ongoing updates and feature additions.

Scalability: MVC can help you scale different parts of the application independently. For example, you might need to scale the user interface to handle a large number of users while keeping the underlying logic and data layer relatively unchanged.

Testing: The separation of concerns makes unit testing and automated testing of individual components (e.g., Controllers and Models) more manageable. This is particularly important for ensuring the reliability and quality of an LMS.

Security: By clearly defining the responsibilities of each component, you can implement security measures more effectively. For example, you can secure user input handling within the Controller and protect sensitive data within the Model.

Possible drawbacks of MVC:

Learning Curve: Developers new to MVC may face a learning curve when understanding the roles and responsibilities of each component, potentially slowing down development.

Client Server:
Choosing a client-server architecture for an Interview Learning Management System (LMS) offers several advantages, especially when considering the requirements and characteristics of such a system. Here are some reasons to consider using a client-server architecture for an Interview LMS:

Centralized Control and Management: In an Interview LMS, you need centralized control over various components, such as user profiles, courses, assessments, and interview data. A client-server architecture allows you to centralize data storage and management on the server side, making it easier to maintain and update.

Scalability: Client-server architectures are inherently scalable. As your Interview LMS grows and more users access it simultaneously, you can scale the server resources to handle the increased load. This is crucial for maintaining system performance during peak usage times.

Security: In an LMS, security is paramount. Storing sensitive user data, assessment results, and interview information on a centralized server allows you to implement robust security measures to protect this data. You can control access, encryption, authentication, and auditing more effectively at the server level.

Cross-Platform Compatibility: Client-server architectures allow you to build different clients for various platforms (e.g., web, mobile, desktop). Users can access the Interview LMS from their preferred devices and operating systems, enhancing accessibility and user adoption.

Backup and Recovery: Centralized data storage simplifies backup and recovery procedures. Regular server backups help safeguard against data loss, ensuring that no valuable user information is lost.

Possible drawbacks of client-server:
Latency: The client-server architecture may introduce network latency, which can impact the user experience, particularly for applications that require real-time interactions.

REST Architecture:

REST (Representational State Transfer) is an architectural style for designing and developing networked applications. It is based on the idea of using a uniform interface to access resources. This means that all resources are identified by a URI (Uniform Resource Identifier), and all

interactions with resources are performed using a set of standard HTTP methods, such as GET, POST, PUT, and DELETE.

Simplicity: REST is a simple and lightweight architecture that is easy to understand and implement. It is based on the concepts of resources, representations, and standard HTTP methods. This makes it easy to create and maintain AlgoLab that can be scalable and provide high performance.

Flexibility: REST is a very flexible architecture that supports a variety of data formats and can be used to build both simple and complex applications. This makes it a good choice to build AlgoLab to meet their specific application requirements.

Scalability: RESTful APIs are stateless and can be easily scaled horizontally by adding more servers behind a load balancer. This makes it easy to build an AlgoLab that can handle large numbers of users and large amounts of data.

Performance: RESTful APIs are typically very performant because they use standard HTTP requests and responses. This makes it easy to build AlgoLab that are responsive and can handle high volumes of traffic.

Portability: RESTful APIs are portable and can be accessed from a variety of platforms and devices. This makes it easy to build AlgoLab that can be used by users on a variety of devices, including desktops, laptops, tablets, and smartphones.

Interoperability: RESTful APIs are designed to be interoperable with other RESTful APIs. This makes it easy to integrate AlgoLab with other systems.

Maintainability: RESTful APIs are easy to maintain and update, which makes it easy to keep AlgoLab up-to-date.

Event-Driven Architecture:

Decoupling: An EDA decouples the email verification and announcement processes from the AlgoLab core, making it easier to scale and maintain these processes independently.

Scalability: An EDA can be easily scaled to handle a large volume of email notifications by adding more event handlers. AlgoLab can experience spikes in traffic during certain times of the year, such as when new announcements are made and this can be easily scaled.

Reliability: An EDA can be made more reliable by using messaging queues and other fault tolerance mechanisms. This ensures reliable mail delivery for user verification in AlgoLab. When an important announcement is made about AlgoLab, an event can be published to a

messaging queue. An event handler can then process the event and send an email to all users to inform them of the announcement.

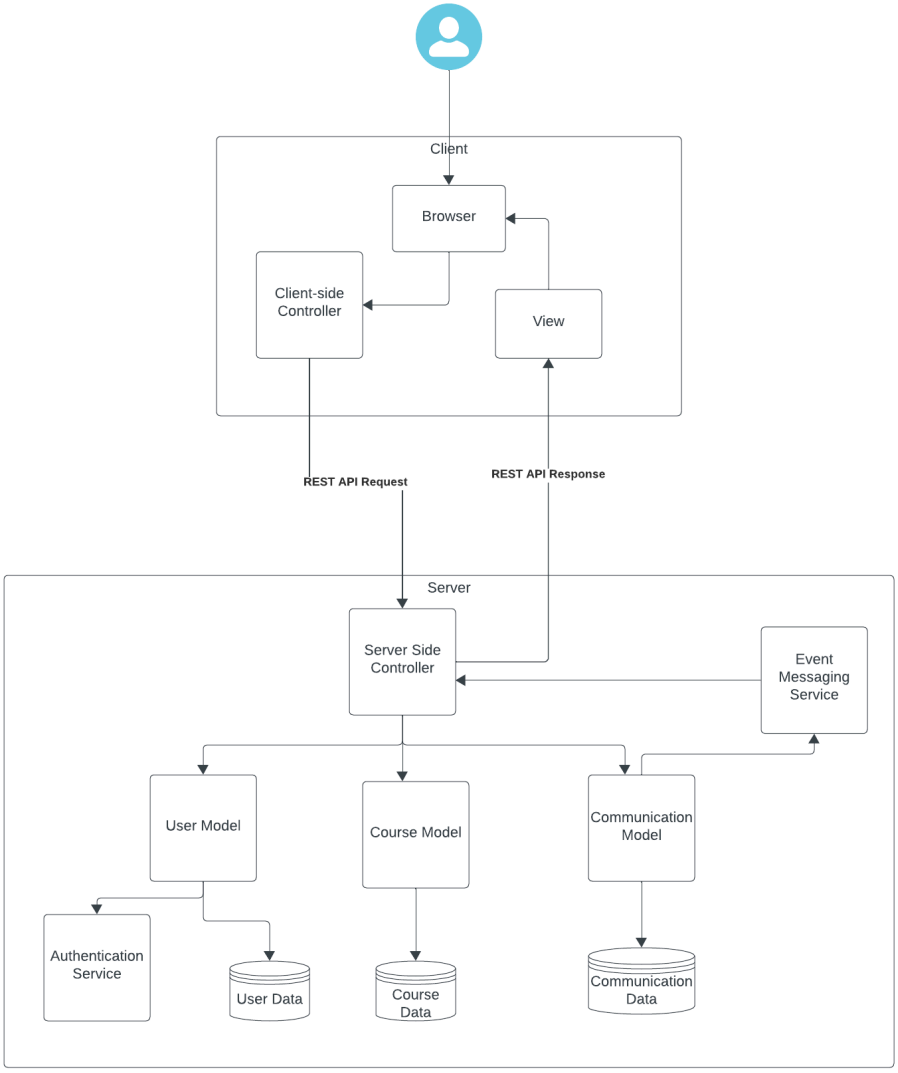
Reusability: An EDA can be used to implement other types of asynchronous processes, such as generating reports or sending push notifications. This can make it easier to develop and maintain a consistent architecture for different parts of the AlgoLab.

Security: When a new user registers for an AlgoLab account, an event can be published to a messaging queue. An event handler can then process the event and send an email to the user to verify their email address. This process helps the application in verifying the credibility of the user in a timely and secure manner.

Potential drawbacks of Event Driven Architecture:

Troubleshooting: Troubleshooting issues in Event Driven Architecture systems can be challenging due to the distributed nature of event processing and the potential for race conditions. This can be prevented with proper logging in all the required scenarios.

Refined architecture



Client-Side Components:

Browser:

The browser is the user interface through which end-users interact with your LMS. It renders the web pages and handles user input, making it the front-end of your application.
Client-Side Controller:

This component, often implemented in JavaScript, manages user interactions within the browser. It processes user input, communicates with the server through REST APIs, and updates the view based on server responses.
View:

The view represents the user interface elements and presentation of your application. It includes HTML, CSS, and JavaScript code that defines how the content is displayed to users. The view is responsible for rendering the user interface in the browser.
Server-Side Components:

Server-Side Controller:

The server-side controller receives client requests, processes them, and coordinates the interaction between the client and the server. It handles incoming HTTP requests, routes them to appropriate server-side components, and manages the server's response.
User Model:

The user model represents user-related data and business logic. It may include functions for user authentication, user data management, and user-related operations. The user model is connected to the authentication service and user data storage.

Authentication Service:

Responsible for user authentication and authorization. It ensures that users are who they claim to be and controls their access to resources based on permissions.
User Data:

Stores user-specific data, such as user profiles, preferences, and any data associated with users.

Course Model:

The course model represents course-related data and business logic. It includes functions for managing courses, course materials, and user enrollments in courses. It is connected to course data storage.

Course Data:

Stores information about courses, including course content, lessons, and other course-related data.

Communication Model:

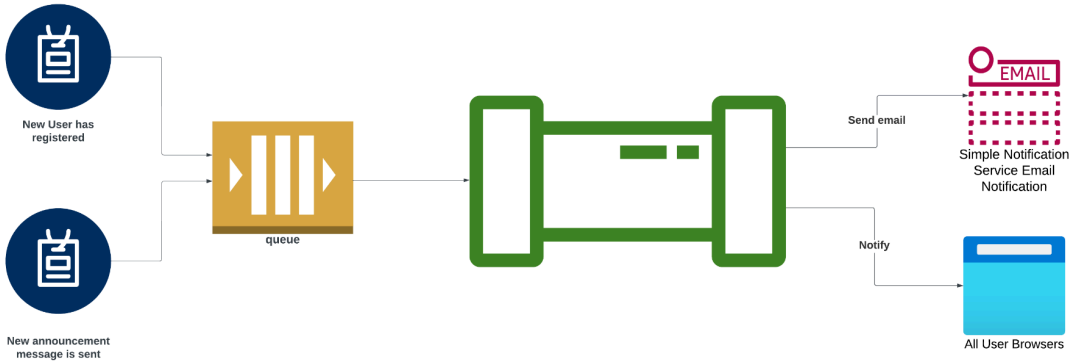
Represents the messaging and notification system used to manage announcements, issues and feedback to users. It utilizes event driven architecture to publish and subscribe to messages, ensuring efficient communication between the server and users.

In summary, your architecture employs a client-server model with MVC (Model-View-Controller) on the client-side and models on the server-side. RESTful APIs facilitate communication between the client and server, while the event messaging architecture allows for efficient message distribution and announcements. This well-structured architecture ensures the separation of concerns and scalability of your Interview LMS.

REST API:

REST API communication is a stateless way of exchanging data between a client and a server using standard HTTP methods. Client sends a request to the server using a specific HTTP method (GET, POST, PUT, DELETE) and a URI (Uniform Resource Identifier). Server receives the request, processes it, and sends back a response with the requested data using the same HTTP method and a status code.

Event-driven architecture for Messaging Model:



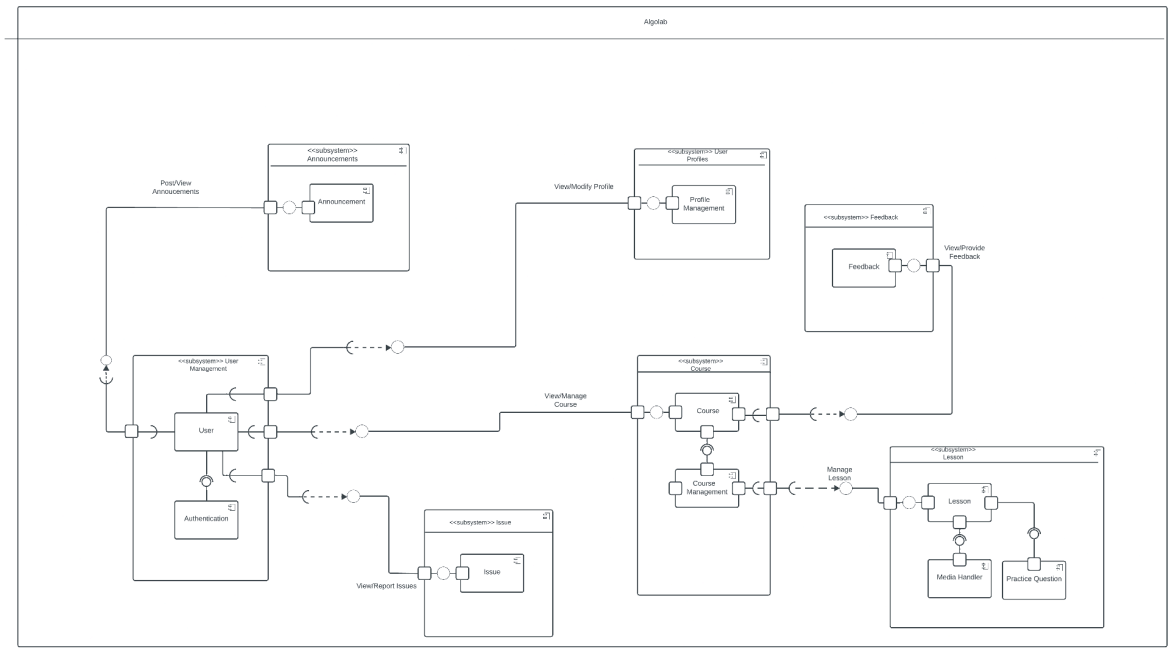
Event Producers: The event producers are the user registration which triggers the user email verification process to send the verification email to the user. The other event producer is the new announcement triggered by the system.

Event Channel: The event queue asynchronously transmits the events to the consumers.

Event Consumers: The event consumers are the users who newly registered into the application receiving an email to verify their authenticity. The event consumers are all users when the announcement is posted on the web page of all users.

Architecture views

3. A. Logical view:



Component Description

Component Name: Manage Lesson

Component Responsibilities:

This component is responsible for managing Lesson related information like lesson videos and Practice questions.

Provided Interfaces:

Manage lessons which include create , modify and delete lessons**.**

Required Interfaces: None

Interface name: Create lesson**s**

Interface responsibilities: Creates new lessons for a course.

Interface Operations: createLesson() , createLessons()

Operations description:

createLesson()	
Operation signature	createLesson(lesson: Lesson): Lesson
Operation description	Creates a new lesson and returns it.
Operation preconditions	The lesson parameter must be a valid lesson object for the createLesson operation.
Operation postconditions	A new lesson is created and returned for the createLesson operation.
createLessons()	
Operation signature	createLessons(lessons: Lesson[]): Lesson[]
Operation description	Creates a new array of lessons and returns it.
Operation preconditions	The lessons parameter must be an array of valid lesson objects for the createLessons operation.
Operation postconditions	A new array of lessons is created and returned for the createLessons operation.

Components and Interface Description:

Subsystem: User Management

Component name: User
Component Responsibilities: Performs login, registration, roles and permissions management operations for the user.

Component name: Authentication
Component Responsibilities: Performs authentication and authorization operations for the user.

Subsystem: Course

Component name: Course
Component Responsibilities: Performs operations of search, view and modification of course.

Component name: Course Management
Component Responsibilities: Performs all operations on the course repository to manage course roadmap, course contents, course modules, track course progress.

Subsystem: Issue

Component name: Issue
Component Responsibilities: Performs operations related to reporting for errors, inappropriate content, or bugs in the application.

Subsystem: Feedback

Component name: Feedback
Component Responsibilities: Performs operations that allows users to provide feedback and admin to view feedback.

Subsystem: Profiles

Component name: Profile management
Component Responsibilities: Performs operations that allows users to view and maintain their profile information.

Subsystem: Announcements

Component name: Announcements

Component Responsibilities: Performs operations that allows the system to post announcements.

Subsystem: Lesson

Component name: Lesson
Component Responsibilities: Performs operations of managing course related information like lesson videos and Practice questions.

Component name: Media Handler
Component Responsibilities: Performs operations of uploading and maintaining the video lectures of a specific lesson.

Component name: Practice Questions
Component Responsibilities: Performs operations of uploading and maintaining the video lectures of a specific lesson.

Interfaces:

Manage Lesson: Create, modify or view the lessons for a specific course based on user permissions.

View/Report Issues: Reports the issues faced by the user to the system and system administrator can view these reported issues.

Manage Courses: User creates, modifies, views or searches the course roadmap, course contents, course modules based on their user permission.

View/Modify Profile: Users can view and modify their profile.

View/Provide Feedback: Candidates of the course can provide feedback for a course and the Admin and Instructor can view the feedback.

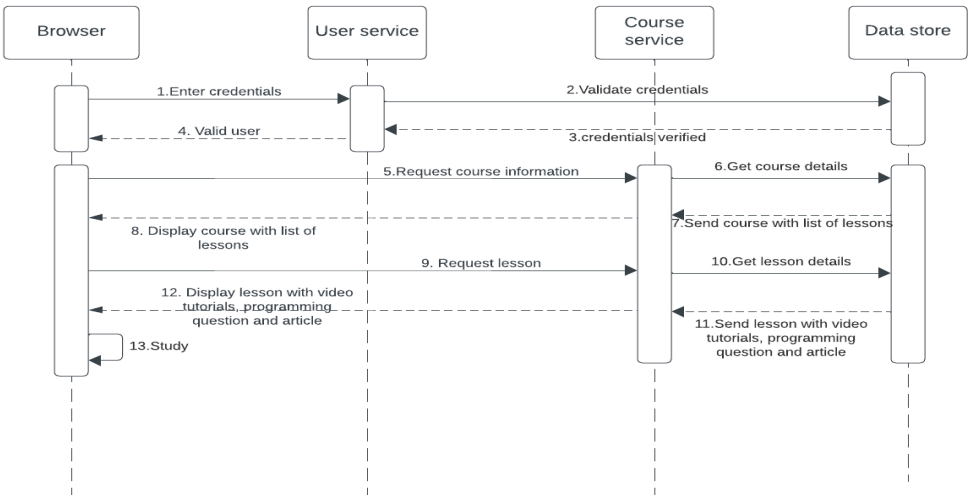
Post/View Announcements: Admin users will post the announcements and all users can view the announcements.

3. B. Behavior view:

In the context of algolab, the following are sequence diagrams for 5 top priority requirements.

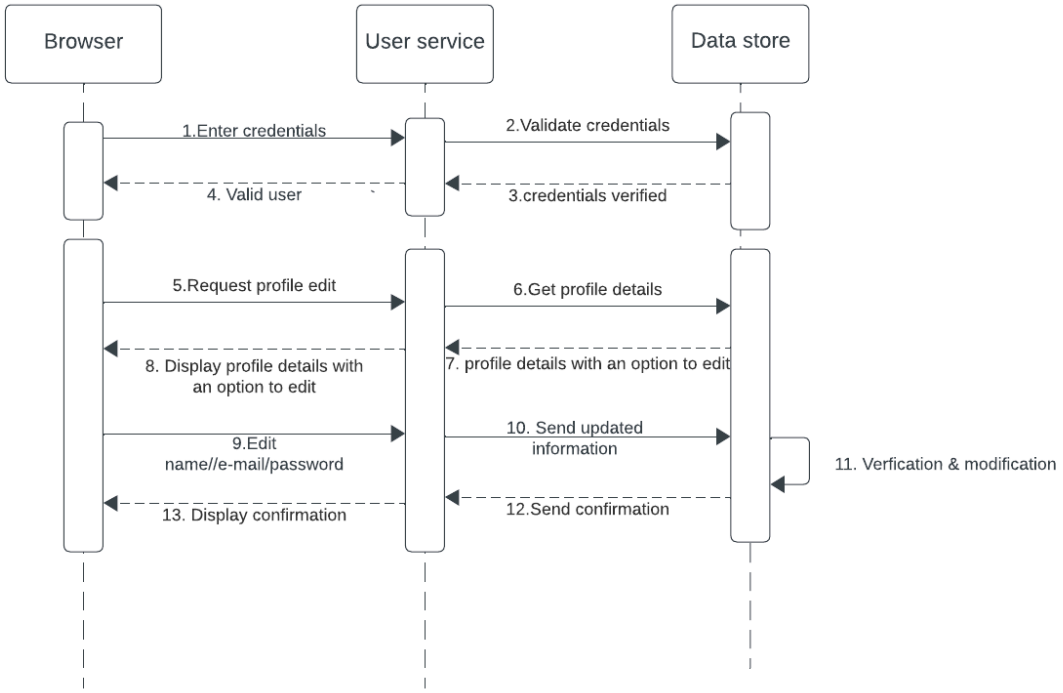
Requirement: Candidates shall be able to view the list of lessons in a course. Users shall be able to view the contents of each lesson such as video tutorials, articles and programming questions.

There shall be 4 components interacting for this requirement. Browser(external entity), user service, course service and data store. Firstly, the user will be validated through the interaction between browser, user service and data store. Next, the browser will interact with course service for getting course information and lesson information. Course service will retrieve course and lesson information from the data store and display relevant information.



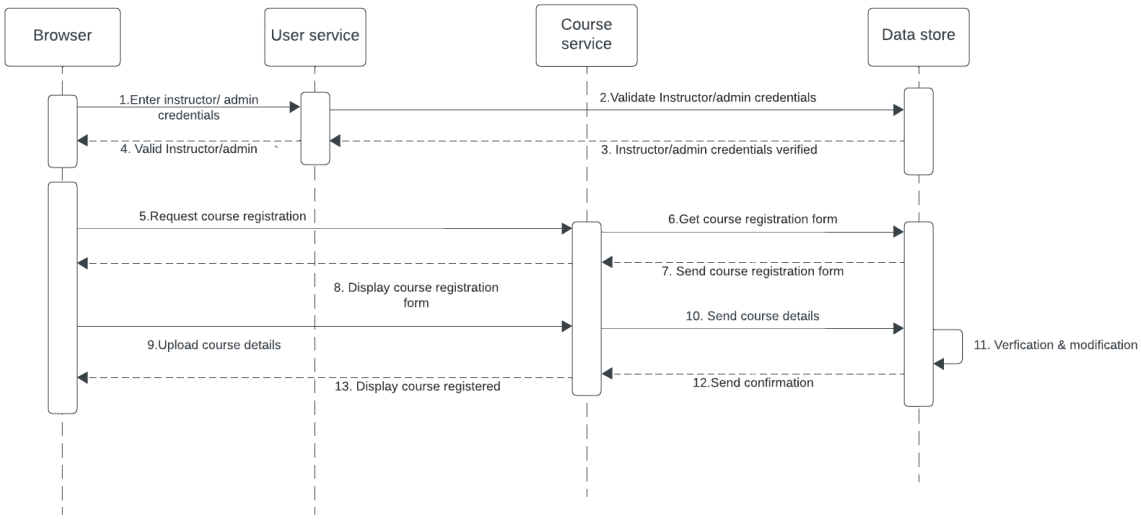
Requirement: User shall be able to manage their profile

There shall be 3 components interacting for this requirement. Browser(external entity), user service, and data store. Firstly, the user will be validated through the interaction between browser, user service and data store. Next, the browser will interact with user service for getting profile information. User service will retrieve profile information from the data store and display relevant information. Lastly, users will upload updated information. User service will receive the updated information, validate with the data store and display confirmation.



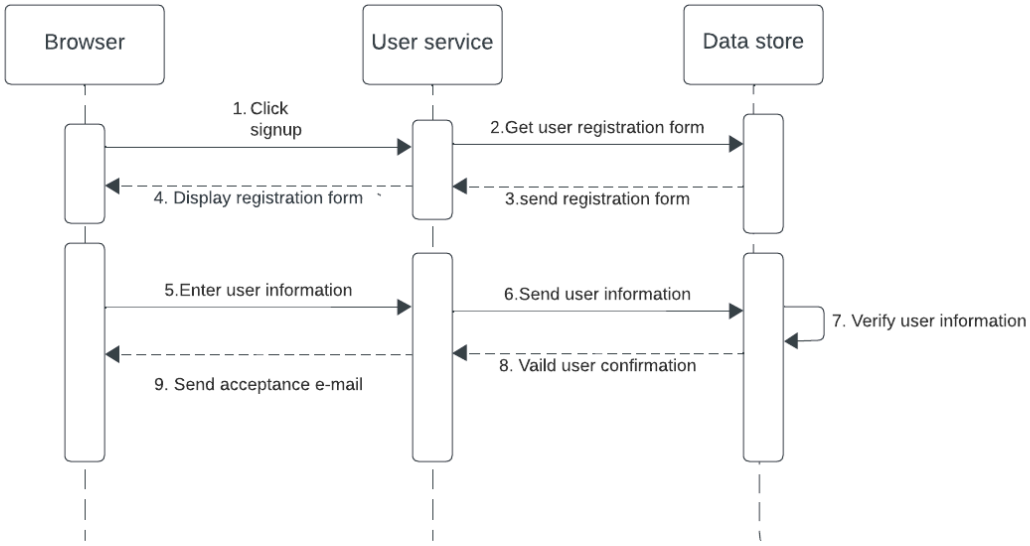
Requirement: Instructors/Admin shall be able to create courses

There shall be 4 components interacting for this requirement. Browser(external entity), user service, course service and data store. Firstly, the instructor/admin will be validated through the interaction between browser, user service and data store. Next, the browser will interact with the course service for getting the course registration form. Course service will retrieve course registration form from the data store and display relevant information.Lastly, instructor/admin will upload course information. Course service will receive the course information, validate with the data store and display confirmation for course registration.



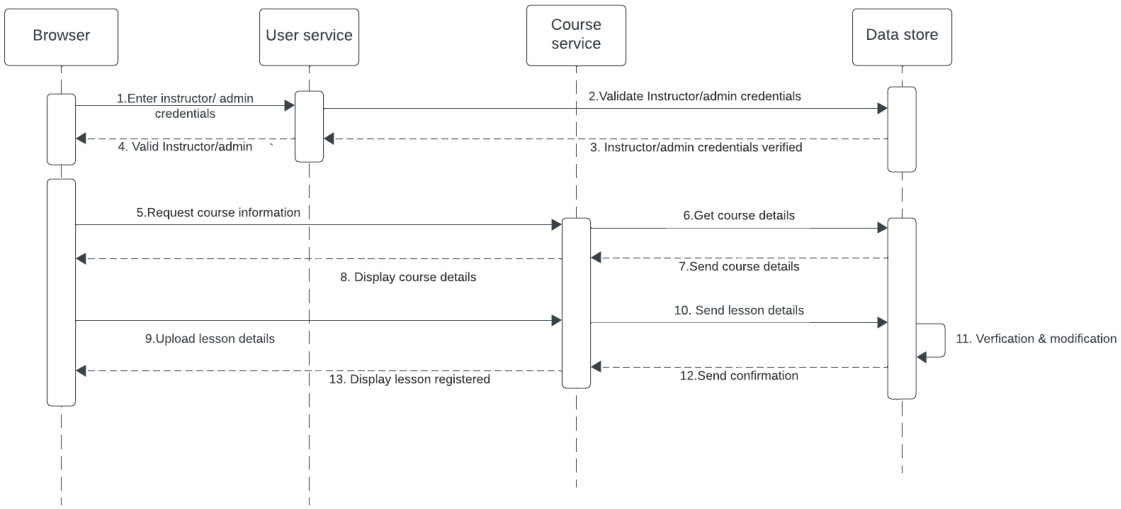
Requirement: User shall be able to register themselves with their details

There shall be 3 components interacting for this requirement. Browser(external entity), user service, and data store. Firstly, the user will request for registration form by clicking on sign up. User service will retrieve the registration form from the data store and display it. Lastly, the user will upload information. User service will receive the user information, validate with the data store and display confirmation.



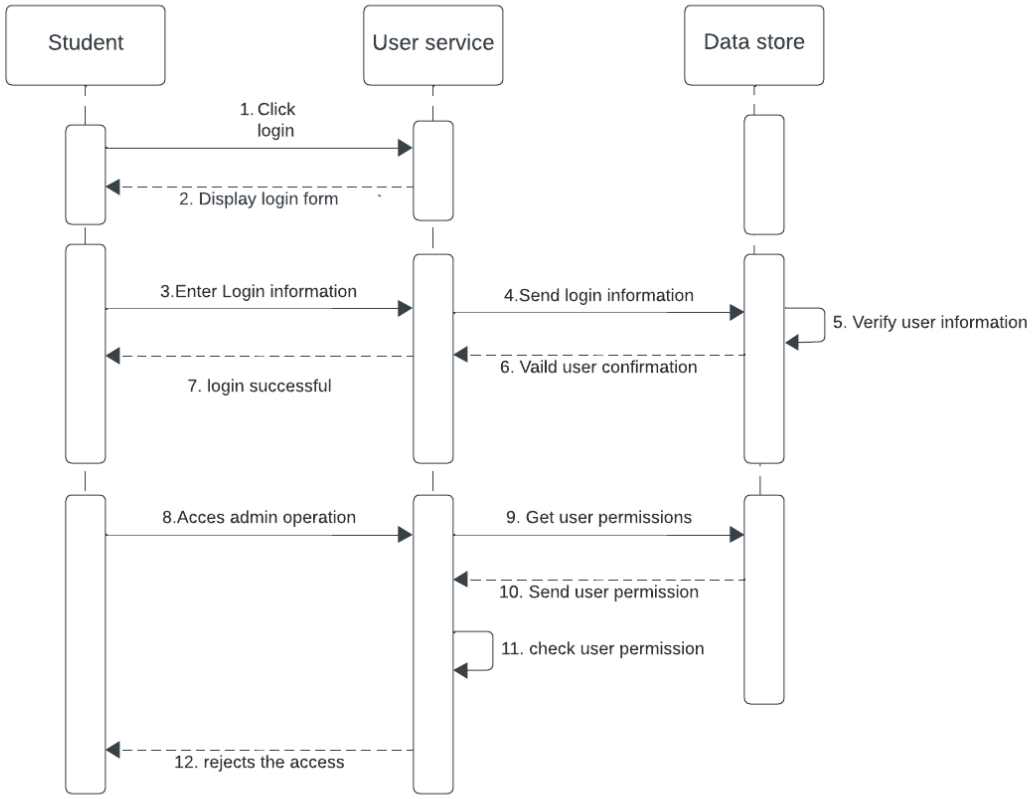
Requirement: Instructors/Admin shall be able to create the contents of lessons present in each course.

There shall be 4 components interacting for this requirement. Browser(external entity), user service, course service and data store. Firstly, the instructor/admin will be validated through the interaction between browser, user service and data store. Next, the user will interact with the course service for getting the course information. Course service will retrieve course information from the data store and display relevant information. Lastly, the instructor/admin will upload lesson information for the displayed course. Course service will receive the lesson information, validate with the data store and display confirmation for lesson registration.

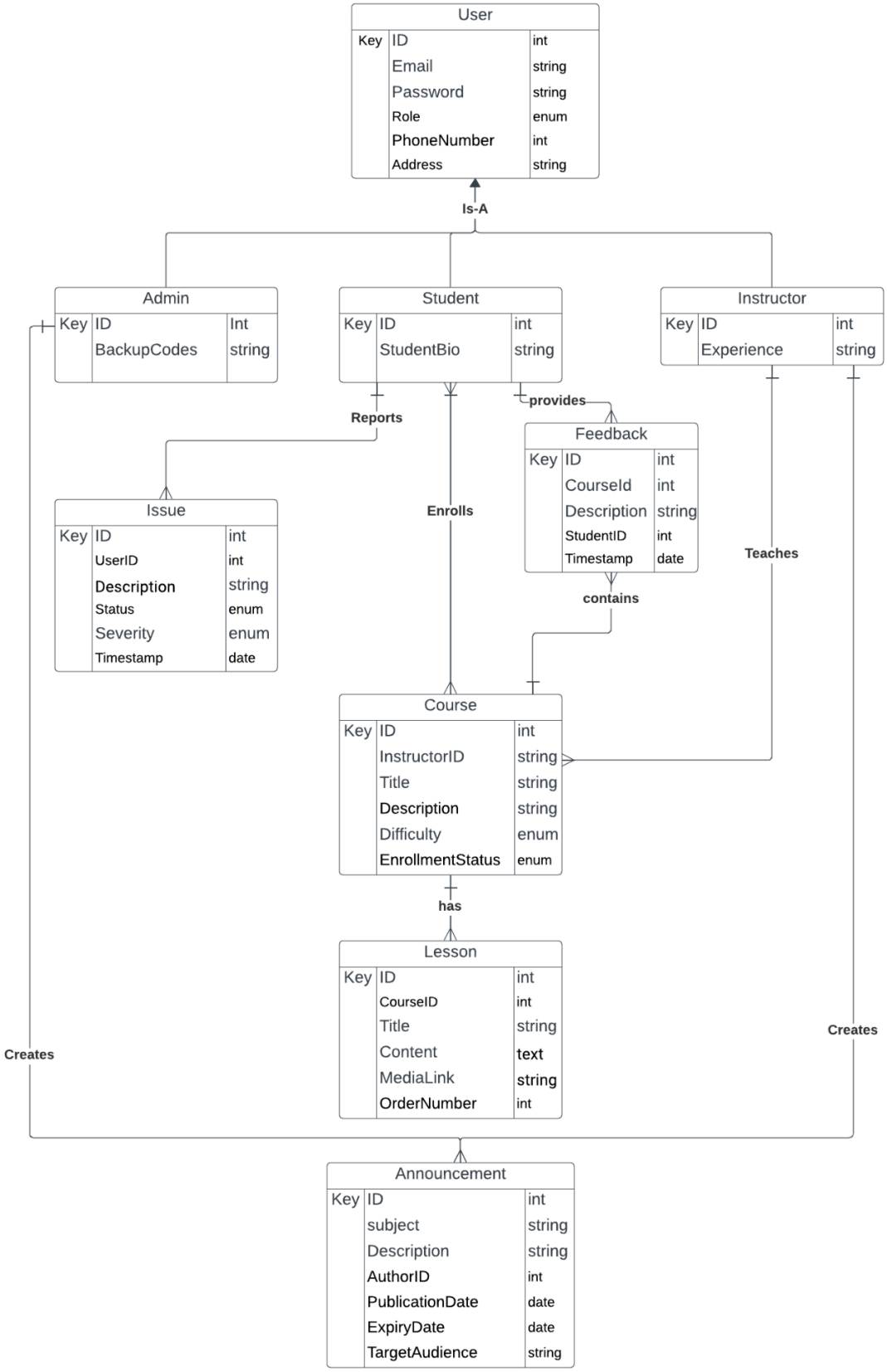


Quality scenario: The system shall reject access when another user tries to access or carry out an admin operation.

There shall be 3 components interacting for this requirement. Browser(Student), user service, data store. Firstly, the Student will login with the help of user service and data store. Next, if the Student tries to access admin operations like deleting the user, the user service will check it with the data store and verify if the student has permission to carry out the admin operation. If the user is not authorized for admin operation, the user service will reject the request.


Student
User service
Data store
1. Click login
2. Display login form
3.Enter Login information
4.Send login information
5. Verify user information
6. Vaild user confirmation
7. login successful
8.Acces admin operation
9. Get user permissions
10. Send user permission
11. check user permission
12. rejects the access

3.C. Information view:



The modules of ER diagram are:

User:
ID: A unique identifier for each user.
Email: The user's email address, used for authentication and communication.
Password: The user's password for authentication in encrypted form.
Role: Specifies the role of the user (e.g., student, admin, instructor).
PhoneNumber: The user's contact phone number.
Address: The user's physical or mailing address.

Admin (inherits from User):
ID: A unique identifier for each admin user.
BackupCodes: stores a set of backup authentication codes for administrators in encrypted form.

Student (inherits from User):
ID: A unique identifier for each student.
StudentBio: Describes the biographical description of a student.

Instructor (inherits from User):
ID: A unique identifier for each instructor.
Experience: The teaching and other experience of the instructor.

Course:
ID: A unique identifier for each course.
InstructorID: The ID of the instructor or teacher responsible for the course.
Title: The title or name of the course.
Description: A brief description of the course's content and objectives.
Difficulty: This attribute quantifies the level of complexity or difficulty associated with a specific course(e.g., easy, medium, hard).
EnrollmentStatus: Indicates whether enrollment in the course is open or closed.

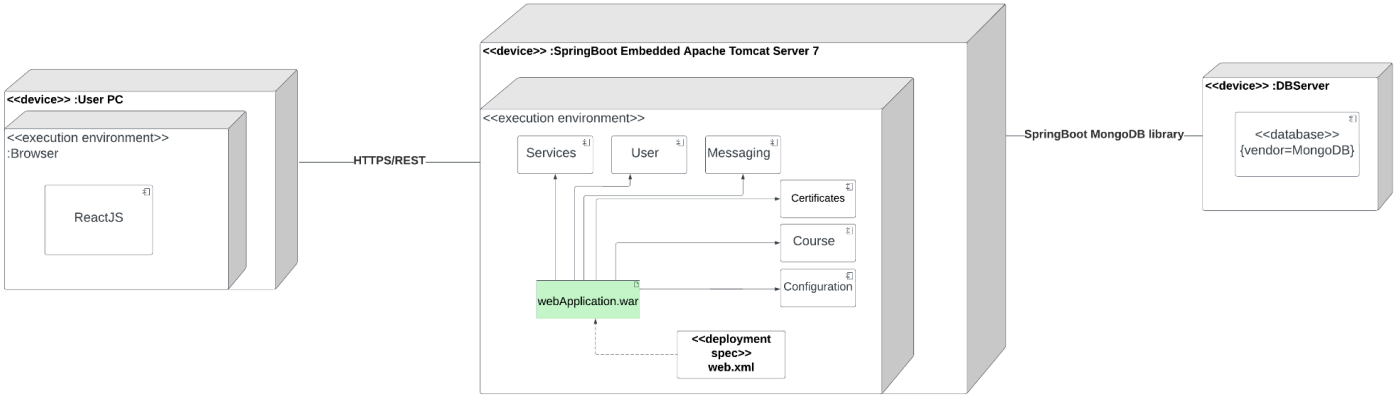
Lesson:
ID: A unique identifier for each lesson.
CourseID: The ID of the course to which the lesson belongs.
Title: The title or name of the lesson.
Content: The content or material covered in the lesson.
MediaLink: This attribute stores links or references to multimedia resources associated with a particular lesson.
OrderNumber: Specifies the order or sequence of the lesson within the course.

Issue:
ID: A unique identifier for each reported issue or support ticket.
UserID: The ID of the user who reported the issue or feedback.
Description: A detailed description of the issue provided by the user.
Status: Indicates the current status of the issue (e.g., open, in progress, resolved).
Severity: Specifies the severity level of the issue (e.g., high, medium, low).
Timestamp: Records the date and time when the issue or feedback was reported.

Announcement:
ID: A unique identifier for each announcement.
Subject: The title or subject of the announcement.
Description: The detailed content or message of the announcement.
AuthorID: The ID of the user (e.g., admin or instructor) who authored the announcement.
PublicationDate: The date when the announcement was published.
ExpiryDate: Specifies the date when the announcement is no longer valid or visible.
TargetAudience: Identifies the intended audience or recipients of the announcement (e.g., all users, specific courses).

Feedback Module:
ID: A unique identifier for each feedback or user comment.
CourseID: A unique identifier of the course for which the student provides the feedback.
StudentID: The ID of the student providing the feedback.
Description: The content or description of the feedback provided by the user.
Timestamp: Records the date and time when the feedback was submitted.

3. D. Deployment View:



The deployment view describes the relation between the hardware and software components required to deploy the algolab application. We will be hosting our environment on an on-premise server. The Browser of the User device interacts with the web server using HTTPS request and response based on the REST services. The web server uses the Spring boot embedded Apache tomcat server. The Database server uses MongoDB to store the application data.The web server uses the spring boot MongoDB library to interact with the database.

Components, Artifacts and Specifications:

User: Consists of the code of the User Model and Controller required for the functional and quality requirements for the user.

Course: Consists of the code of the Course Model and Controller required for the functional and quality requirements for the courses.

Configuration: Contains the configuration files of the application such as properties file handling the database information.

Certificates: Contains the certificates required by the application such as SSL/TLS certificates.

Messaging: Consists of the code of the Event driven messaging functional and quality requirements for the user email verification and the announcements.

Services: Includes the services required for all the components and additional services like authentication and authorization.

Deployment Specification (web.xml): Deployment specification is a document that defines how a software application or system is deployed to a production environment. It provides detailed instructions for installing, configuring, and managing the application or system on the target hardware and software infrastructure.

webApplication.war: The WAR file is a Java archive file format used to package and deploy web applications on web servers. It contains all the necessary files and resources for a web application, including the compiled Java classes, web pages, images, and other resources.

Database: MongoDB is used by the application to store the application data and the interaction between the web server and db server is through the Spring Boot MongoDB library.

Allocation view:

Feature Name/ Components	User	Authentication	Profile Management	Course	Course Management	Issues	Lesson	Media Handler	Announcements	Practice Questions	Feedback
Create Courses				X							
Modify Courses				X							
Create Lessons							X				
Modify Lessons							X				
View Course Feedback											X
View Courses				X							
Register for Courses					X						
View Course Modules/RoadMap					X						
View Lessons and its contents							X	X	X	X	

Track Progress for Courses					X						
Provide Course Feedback											X
Report Error/Inappropriate Content						X					
Login	X	X									
Registration	X	X									
Profile Management			X								

Mapping of components to team members:

Developers of each component worked on the respective detailed designs.

Components /Members	**Sachin Velmurugan**	**Swathi Selvakumaran**	**Keerthan Mahesh**	**Muskan Sagar**	**Barath Bhaskaran**
User	Front-end Development	Testing			Back-end development
Feedback	Front-end Development	Back-end development	Testing		
Authentication	Front-end Development	Back-end development	Testing		
Profile Management	Testing		Back-end development		Front-end development
Course			Front-end Development	Back-end development	Testing
Course Management	Front-end Development			Back-end development	Testing
Issues		Back-end	Front-end	Testing	

		development	Development		
Lesson	Front-end Development			Testing	Back-end development
Media Handler	Front-end Development	Testing			Back-end development
Announcement	Testing			Front-end development	Back-end development
Practice Question	Front-end Development	Back-end development	Testing		

4. Detailed Design Key Decisions and Rationale

SOLID Principle:
All components in our system adhere to the SOLID principles, ensuring a well-organized and maintainable design. Each class has a single responsibility, the system is open for extension but closed for modification, subclasses are substitutable for their base types, interfaces are tailored to specific needs, and high-level modules are not dependent on low-level details. This commitment to SOLID principles enhances the system's flexibility, scalability, and ease of maintenance.

Single Responsibility Principle:
Description: A software entity (module, class) should have one and only one reason to change.
Application: All the classes of all the components are focussing on a single responsibility.

Open Closed Principle:
Description: Software entities (classes, modules, etc.) should be open for extension but closed for modification
Application: Interfaces are a mechanism for defining the behavior of classes. New functionality can be added to existing classes by the implemented interfaces.

Liskov Substitution Principle:
Description: Derived classes must be substitutable for their base classes.
Application: There are no derived classes present in our architecture.

Interface Segregation Principle:
Description: Make client-specific fine-grained interfaces.

Application: Implemented specific interfaces for every component by segregating them separately. In all scenarios, an interface is present between the service and repository and another interface is present between the controller and service.

Dependency Inversion Principle:
Description: High-level modules should not depend on low-level modules. Both should depend on abstractions.
Application: Service layer implementation of all the components have followed the Dependency Inversion principle.

MVC Design pattern:

The structure of controllers, services, and repositories, is being used in our application.

Controller:
Responsibility: Handles incoming requests, processes input, and interacts with the appropriate services.
Interface: This may not have a formal interface in some implementations, as it primarily deals with request handling and forwarding to services.

Service:
Responsibility: Contains business logic, performs operations requested by the controller, and interacts with repositories.
Interface: Typically defined using an interface to abstract the implementation details. This allows for easier testing and future changes without affecting the controller.

Repository:
Responsibility: Manages data access and storage. Performs CRUD (Create, Read, Update, Delete) operations on the database or any data storage.
Interface: Usually defined using an interface to abstract data access operations. Similar to the service, this allows for interchangeable implementations (e.g., different databases) without affecting the service or controller.

Basic Flow in MVC pattern:
The Controller receives a request from the client or another part of the system.
The Controller processes the input, sometimes performs basic validation, and then delegates the request to the appropriate Service.
The Service contains the core business logic and interacts with one or more Repositories to perform data operations.

The Repository deals with data storage and retrieval, encapsulating the details of how data is stored.

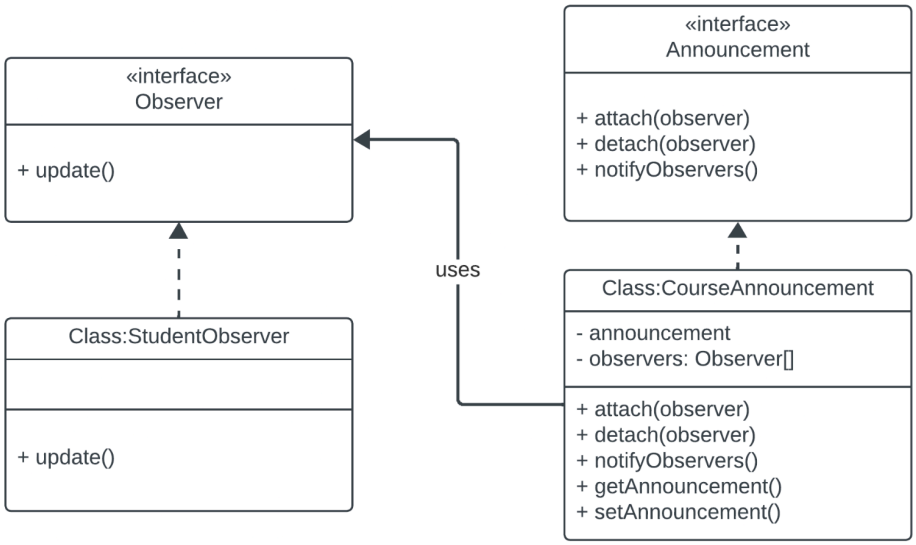
Benefits:
Separation of Concerns: Each component has a distinct responsibility, promoting maintainability and modularity.
Testability: Interfaces make it easier to create and run unit tests, as they allow for the use of mock or stub implementations.
Flexibility: The use of interfaces facilitates the swapping of implementations without affecting other parts of the system.

Observer Pattern:
Defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. An observer pattern can be implemented for an announcement when new courses are implemented. This pattern allows you to add or modify handlers without changing existing ones, providing a flexible way to process requests with different requirements. Each handler decides whether it can handle the request or should pass it to the next handler in the chain.

Chain of Responsibility Pattern:
In the context of processing a request, which involves authentication, authorization, and validation, the Chain of Responsibility pattern can be applied. This pattern allows multiple handlers to process a request in a sequential manner until it is successfully processed or rejected.

5. Detailed Design Structure

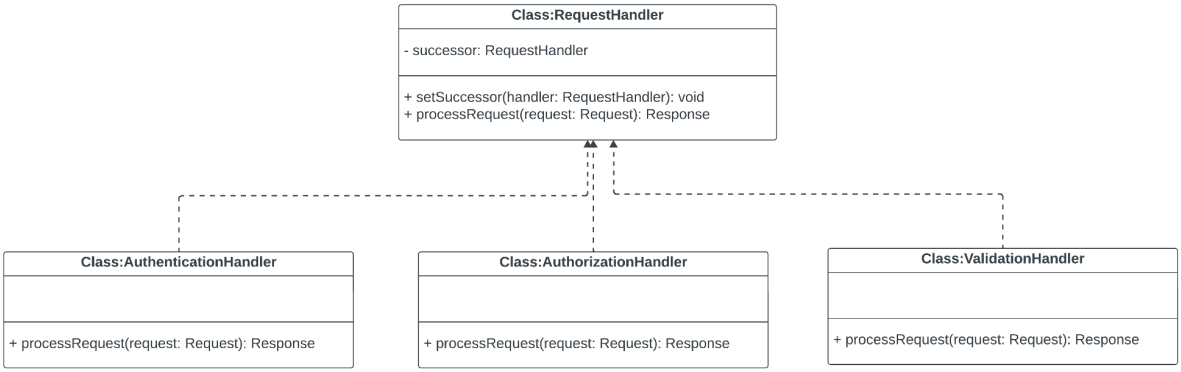
Observer Pattern Class Diagram:
The Observer Pattern for Announcement is implemented to facilitate communication between an Announcement subject and its observers, such as StudentObservers. The Announcement class serves as both a subject and observer, while StudentObserver instances act as concrete observers. When a new announcement is made using the setAnnouncement() method, attached observers are notified through the update() method. Concrete observers, like StudentObserver, can customize their reactions to the announcement. This pattern enables a flexible and decoupled way of handling announcements and allows dynamic addition and removal of observers.



Chain of Responsibility Pattern for Request Processing:

RequestHandler: An abstract class representing the base request handler. It contains a reference to the next handler in the chain (successor) and a method to set the successor and process requests.

AuthenticationHandler, AuthorizationHandler, ValidationHandler: Concrete implementations of request handlers. Each handler processes a specific aspect of the request (authentication, authorization, validation) and decides whether it can handle it or should pass it to the next handler.



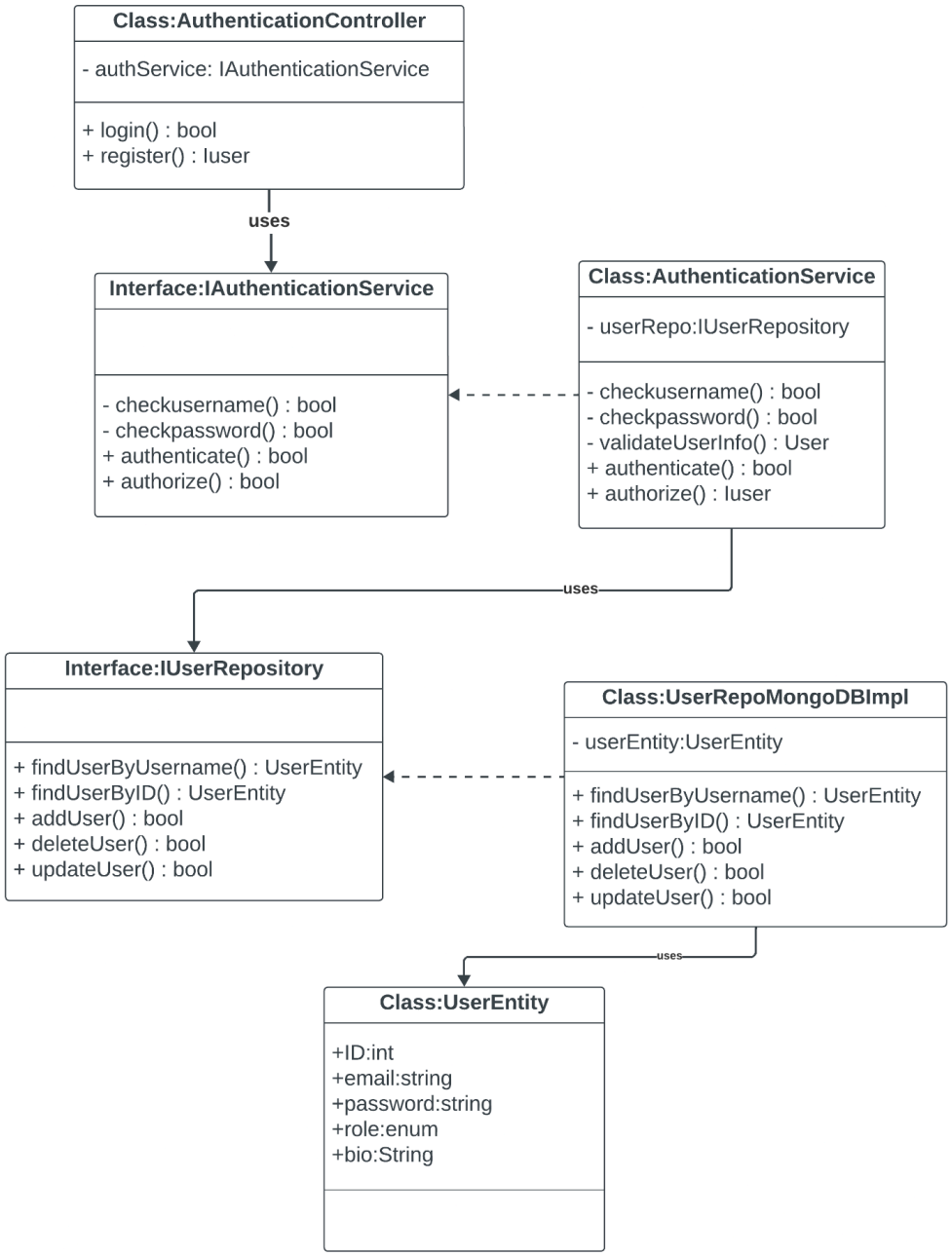
Component: Authentication

The Authentication component is a crucial part of the Learning Management System (LMS), responsible for managing user authentication and authorization. It comprises three main layers: the Authentication Controller, Authentication Service, and Authentication Repository.

Authentication Controller:
The Authentication Controller handles user input, delegates requests to the Authentication Service, and manages presentation logic for login, registration, and authorization processes.

Authentication Service:
The Authentication Service orchestrates authentication workflows, implements core logic, and interacts with the Authentication Repository for user validation, registration, and authorization.

User Repository:
The User Repository stores and retrieves user information, ensuring secure data storage, managing user credentials, and maintaining data integrity for authentication processes.


Class:AuthenticationController
- authService: IAuthenticationService
+ login() : bool
+ register() : Iuser
uses
Interface:IAuthenticationService
- checkusername() : bool
- checkpassword() : bool
+ authenticate() : bool
+ authorize() : bool
Class:AuthenticationService
- userRepo:IUserRepository
- checkusername() : bool
- checkpassword() : bool
- validateUserInfo() : User
+ authenticate() : bool
+ authorize() : Iuser
uses
Interface:IUserRepository
+ findUserByUsername() : UserEntity
+ findUserByID() : UserEntity
+ addUser() : bool
+ deleteUser() : bool
+ updateUser() : bool
Class:UserRepoMongoDBImpl
- userEntity:UserEntity
+ findUserByUsername() : UserEntity
+ findUserByID() : UserEntity
+ addUser() : bool
+ deleteUser() : bool
+ updateUser() : bool
uses
Class:UserEntity
+ID:int
+email:string
+password:string
+role:enum
+bio:String

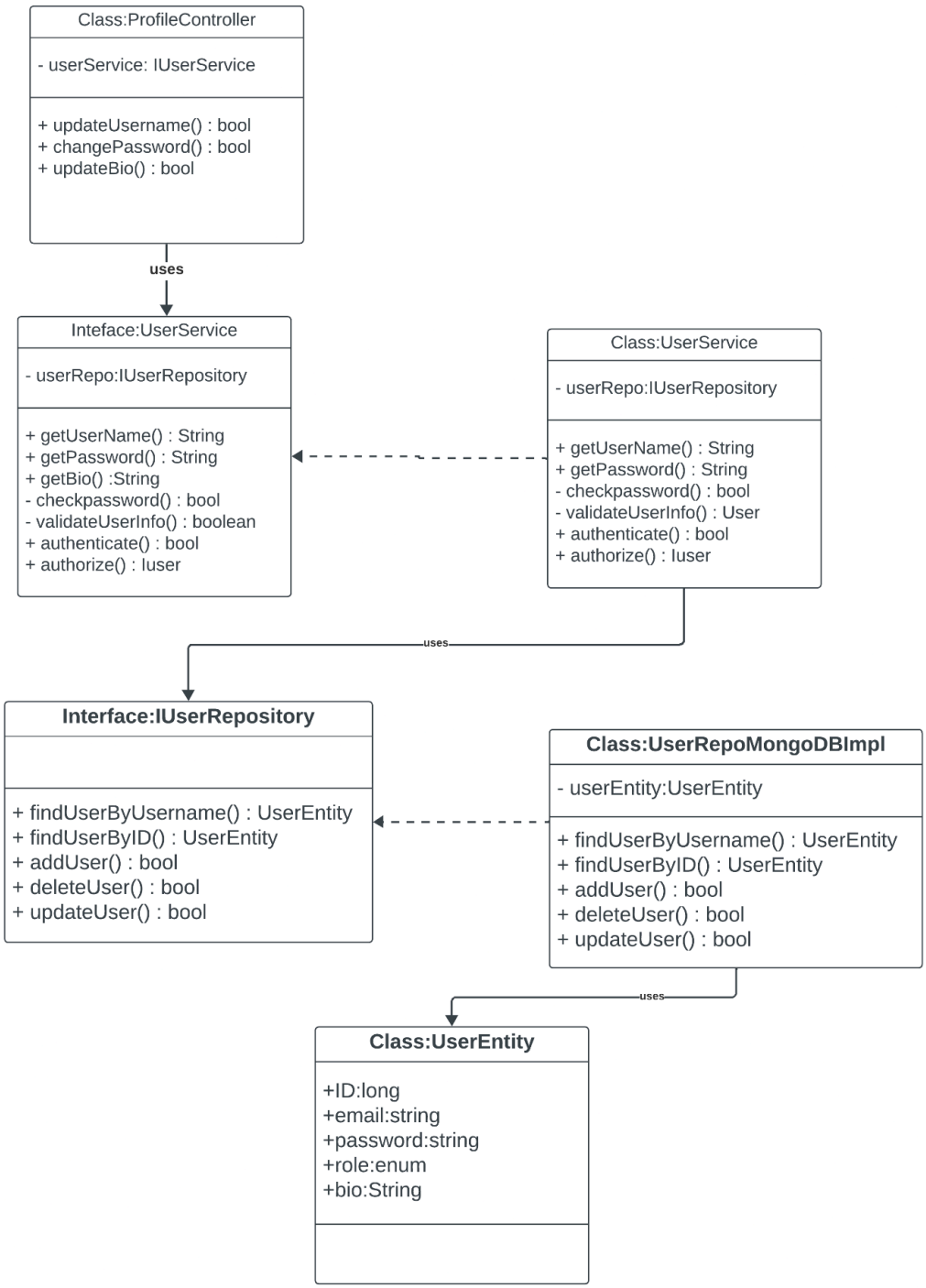
Component: Profile Management

The Authentication component is responsible for managing profile information in the Learning Management System (LMS). It comprises three main layers: the Profile Controller, Authentication Service, and Authentication Repository.

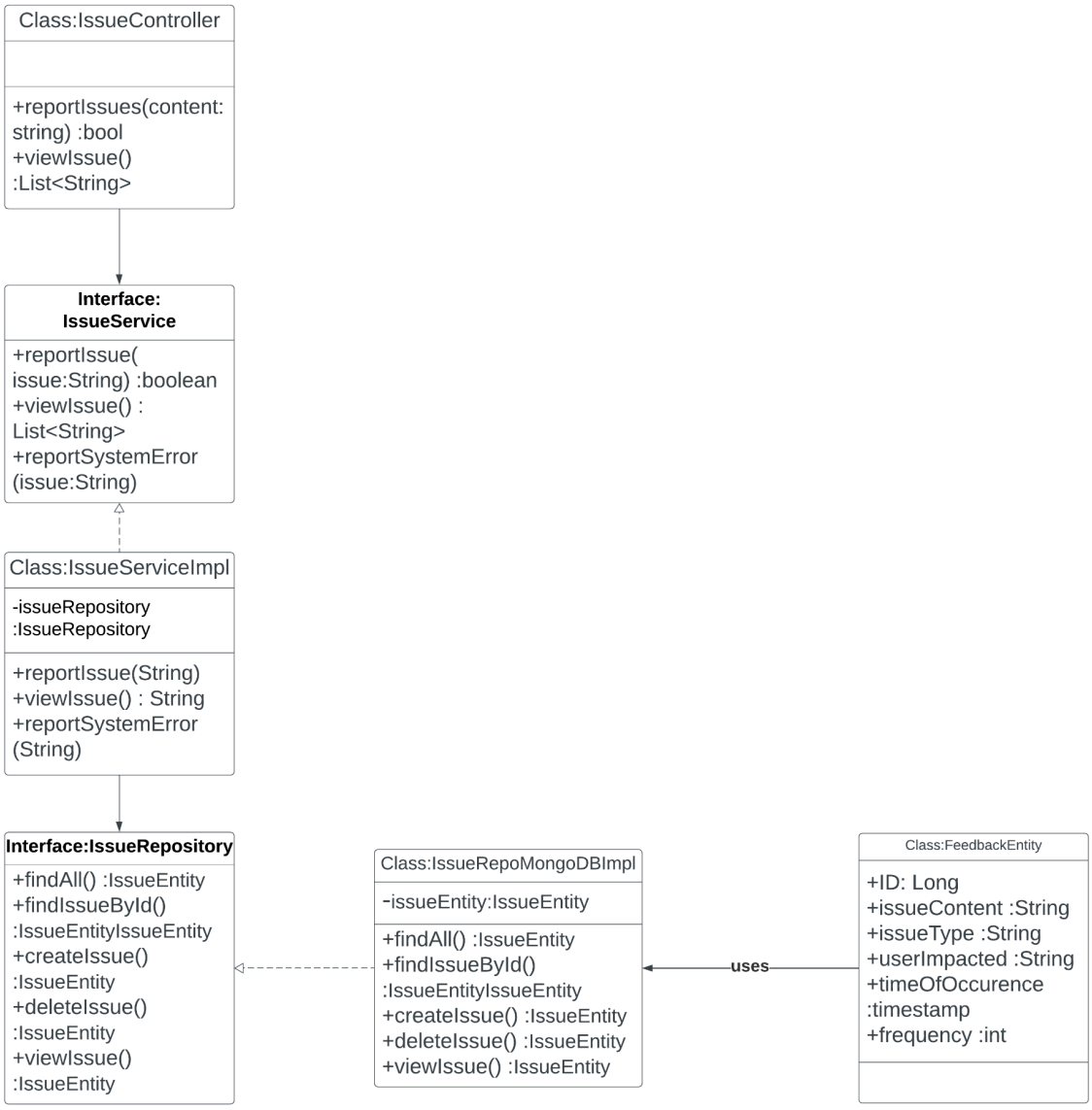
Profile Controller:
The Profile Controller acts as an interface between the user service and the backend services, specifically focusing on user profile-related interactions. It handles incoming requests related to user profiles and communicates with the User Service to execute these actions.

User Service:
The User Service is responsible for handling operations related to user management within the system. It encapsulates functionalities such as retrieval and update of user profiles.

User Repository:
The User Repository stores and retrieves user information, ensuring secure data storage, managing user credentials, and maintaining data integrity for authentication processes.


Class:ProfileController
- userService: IUserService
+ updateUsername() : bool
+ changePassword() : bool
+ updateBio() : bool
uses
Inteface:UserService
- userRepo:IUserRepository
+ getUserName() : String
+ getPassword() : String
+ getBio() :String
- checkpassword() : bool
- validateUserInfo() : boolean
+ authenticate() : bool
+ authorize() : Iuser
Class:UserService
- userRepo:IUserRepository
+ getUserName() : String
+ getPassword() : String
- checkpassword() : bool
- validateUserInfo() : User
+ authenticate() : bool
+ authorize() : Iuser
uses
Interface:IUserRepository
+ findUserByUsername() : UserEntity
+ findUserByID() : UserEntity
+ addUser() : bool
+ deleteUser() : bool
+ updateUser() : bool
Class:UserRepoMongoDBImpl
- userEntity:UserEntity
+ findUserByUsername() : UserEntity
+ findUserByID() : UserEntity
+ addUser() : bool
+ deleteUser() : bool
+ updateUser() : bool
uses
Class:UserEntity
+ID:long
+email:string
+password:string
+role:enum
+bio:String

Component: Issues



Issues Controller:
The Issues controller is responsible for receiving error reports from various sources, such as the LMS core or external applications.

Issues Service:
The Issues service is responsible for processing error reports, including validating the data, filtering out irrelevant errors, and classifying errors based on their severity or type.

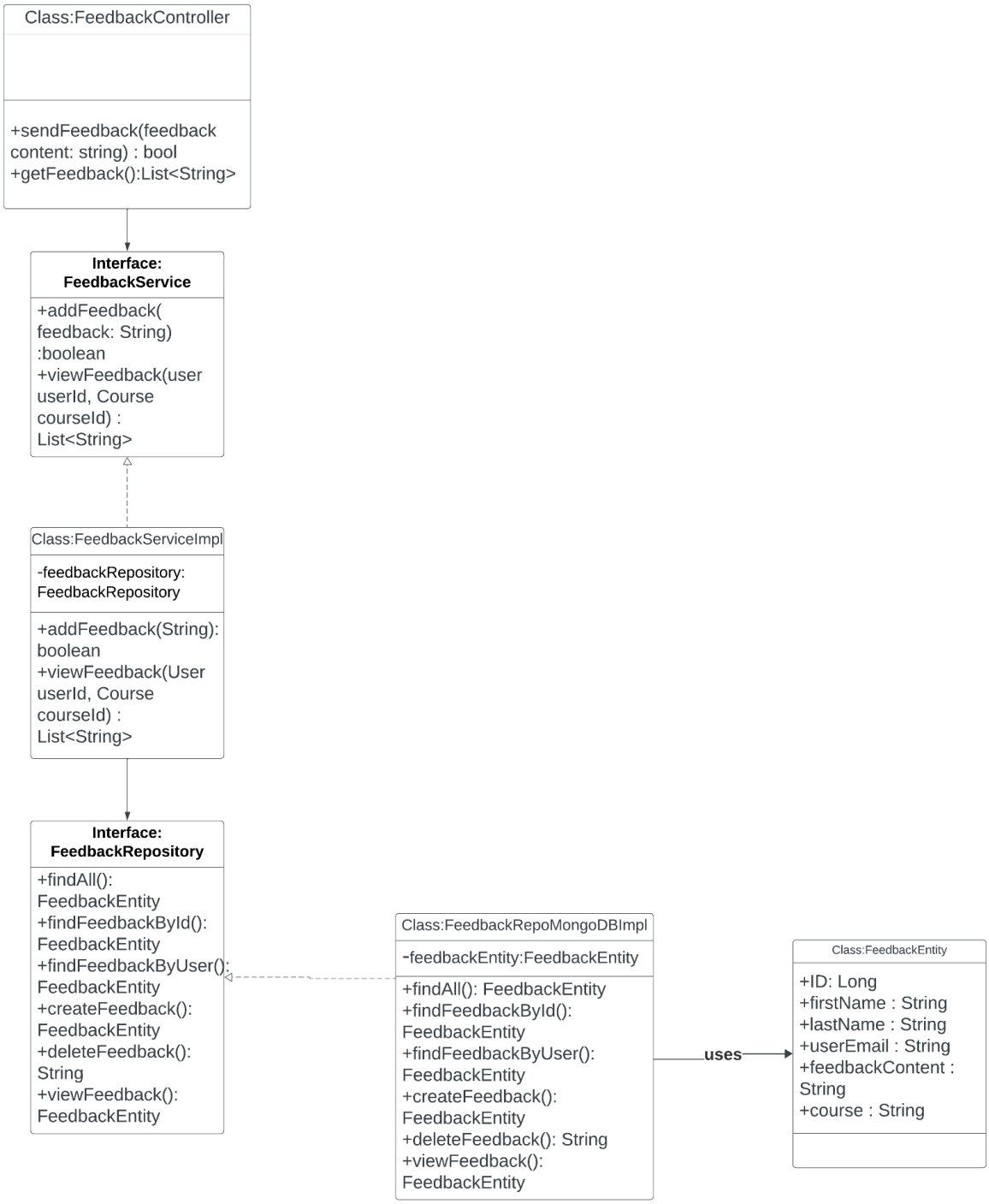
Issues Repository:
The error repository is responsible for providing an abstraction layer for accessing and managing error data.

Component: Feedback

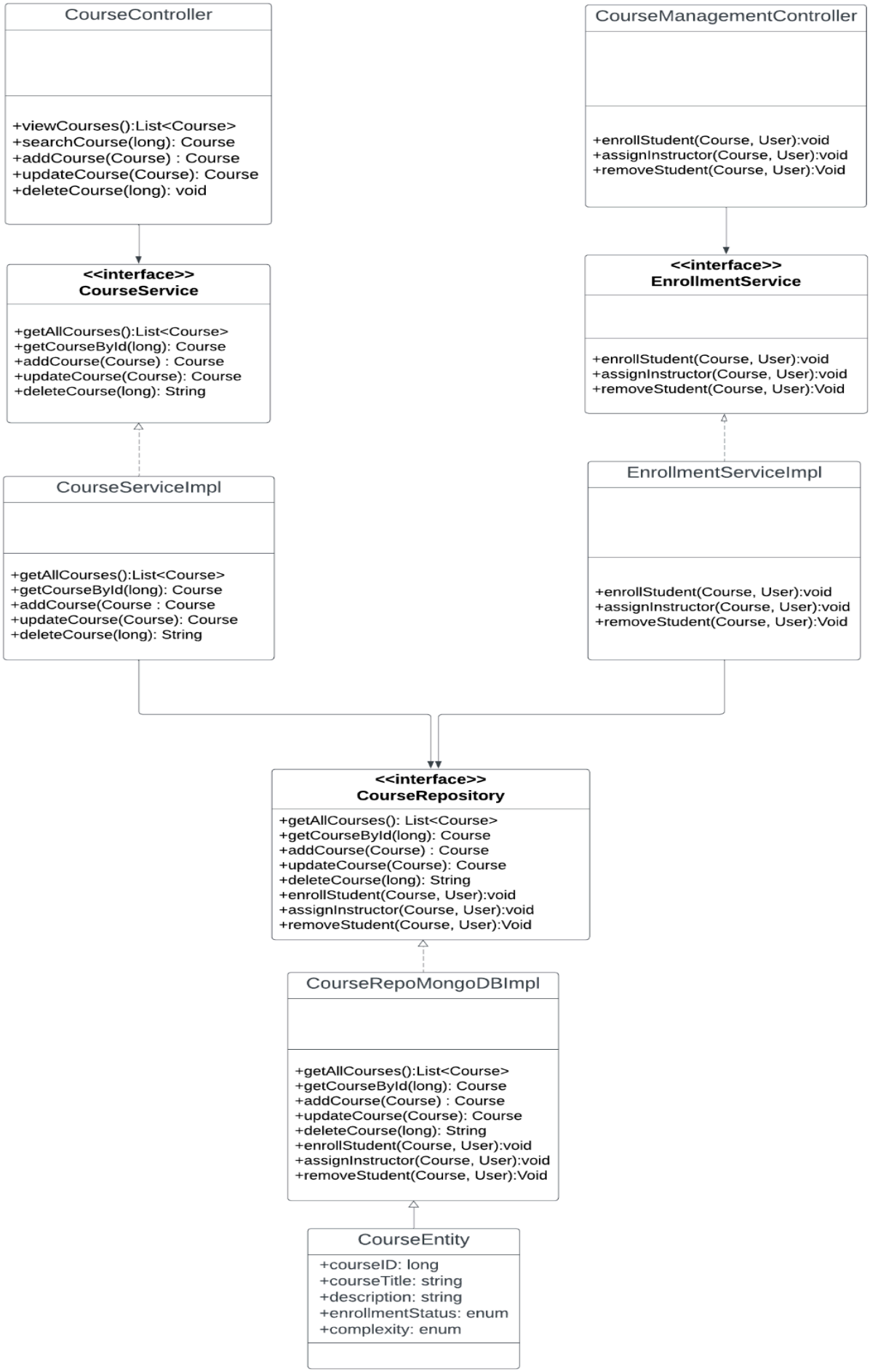
Feedback Controller:
The feedback controller is responsible for handling feedback requests from learners and instructors.

Feedback Service:
The feedback service is responsible for collecting feedback from learners, storing feedback data, and providing feedback to instructors.

Feedback Repository:
The feedback repository is responsible for storing feedback data in a persistent manner.



Component: Course and Course Management


CourseController
+viewCourses():List<Course>
+searchCourse(long): Course
+addCourse(Course) : Course
+updateCourse(Course): Course
+deleteCourse(long): void
CourseManagementController
+enrollStudent(Course, User):void
+assignInstructor(Course, User):void
+removeStudent(Course, User):Void
<<interface>>
CourseService
+getAllCourses():List<Course>
+getCourseById(long): Course
+addCourse(Course) : Course
+updateCourse(Course): Course
+deleteCourse(long): String
<<interface>>
EnrollmentService
+enrollStudent(Course, User):void
+assignInstructor(Course, User):void
+removeStudent(Course, User):Void
CourseServiceImpl
+getAllCourses():List<Course>
+getCourseById(long): Course
+addCourse(Course : Course
+updateCourse(Course): Course
+deleteCourse(long): String
EnrollmentServiceImpl
+enrollStudent(Course, User):void
+assignInstructor(Course, User):void
+removeStudent(Course, User):Void
<<interface>>
CourseRepository
+getAllCourses(): List<Course>
+getCourseById(long): Course
+addCourse(Course) : Course
+updateCourse(Course): Course
+deleteCourse(long): String
+enrollStudent(Course, User):void
+assignInstructor(Course, User):void
+removeStudent(Course, User):Void
CourseRepoMongoDBImpl
+getAllCourses():List<Course>
+getCourseById(long): Course
+addCourse(Course) : Course
+updateCourse(Course): Course
+deleteCourse(long): String
+enrollStudent(Course, User):void
+assignInstructor(Course, User):void
+removeStudent(Course, User):Void
CourseEntity
+courseID: long
+courseTitle: string
+description: string
+enrollmentStatus: enum
+complexity: enum

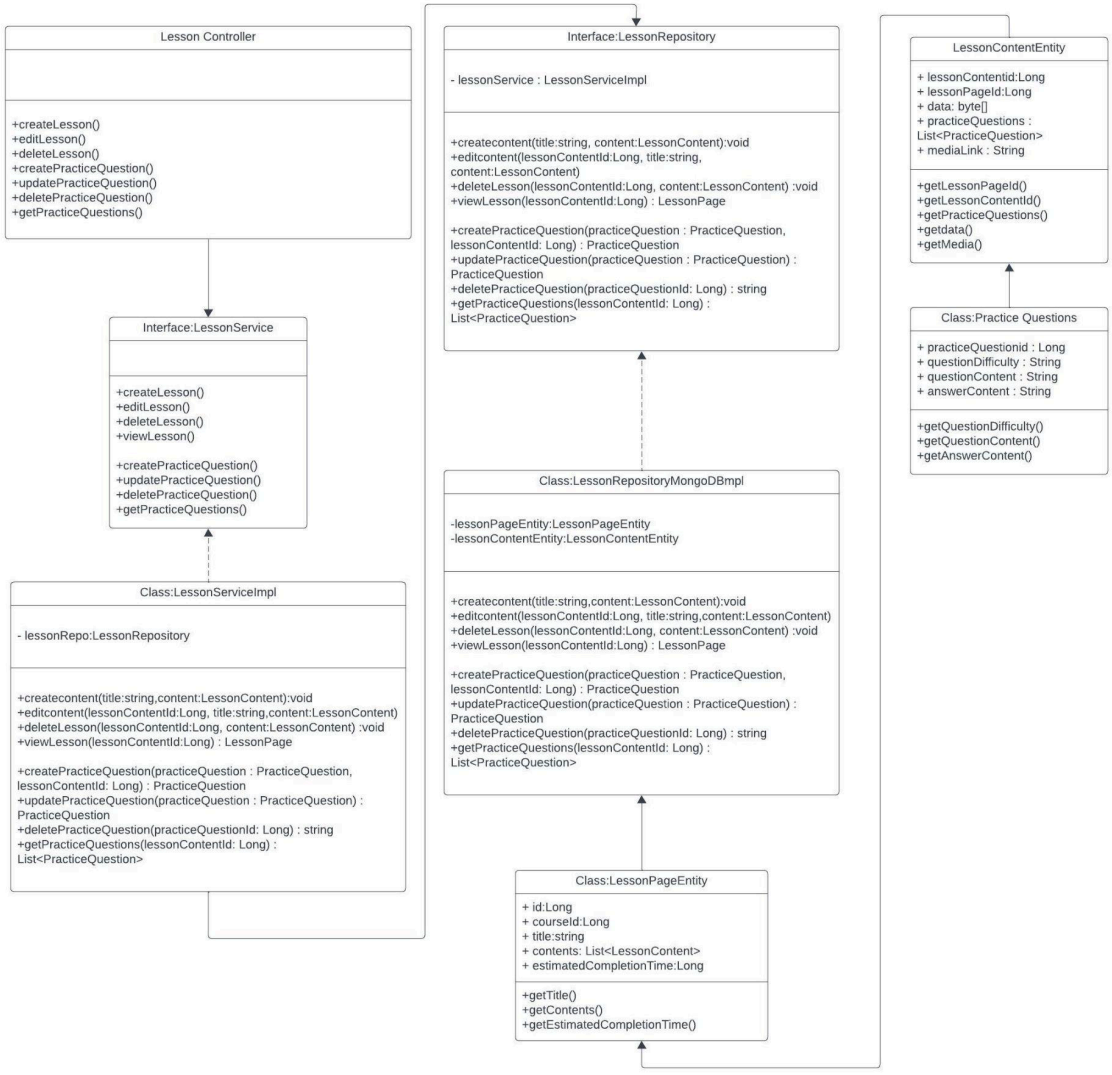
The **CourseController** & **CourseManagementController** receive user requests and invoke corresponding methods in the CourseService & EnrollmentService respectively.

The **CourseService** & **EnrollmentService** contain business logic and orchestrate the interaction between the CourseRepository and the Course objects.

The **CourseRepository** is responsible for data access and storage.

The **CourseEntity** represents individual courses and stores information such as the title, description, enrollment status and complexity.

Component: Lesson



The UML class diagram above represents the structure of the Lesson component in our application, detailing the interactions between various classes for managing lessons.

LessonController: This class serves as a controller in MVC architecture. It contains methods to manage lessons, including createLesson(), editLesson(), deleteLesson(), viewLesson(), and methods related to practice questions such as createPracticeQuestion(), editPracticeQuestion(), deletePracticeQuestion(), and getPracticeQuestions().

LessonService: This interface defines the contract for the service layer with methods that correspond to those in the LessonController. The service layer contains the business logic for handling lesson-related requests.

LessonServiceImpl: This class implements the LessonService interface, providing the logic for the service methods. It will handle business rules, validations, and invoke methods from the repository layer to access and manipulate lesson data.

LessonRepository: This interface defines data access operations that need to be performed on lessons. It has methods for creating content, adding and editing practice questions, and retrieving lesson pages.

LessonRepositoryMongoDBImpl: This class provides a concrete implementation of the LessonRepository interface for our MongoDB database. It contains the actual code to interact with the database and carry out CRUD operations for lessons.

LessonPageEntity: This class represents a lesson page, with fields for identifiers and lesson content, as well as an estimatedCompletionTime. It provides getter methods for these fields.

LessonContentEntity: This class models the data for lesson content, with fields such as lessonContentId, lessonPageId, and list to hold PracticeQuestions and mediaLink. It has getter methods to retrieve these values, which encapsulates the properties.

PracticeQuestions: This class represents the practice questions associated with a lesson. It includes properties such as practiceQuestionId, questionDifficulty, questionContent, and answerContent, with getters for each.

The diagram describes a system where the LessonController interacts with the LessonService to process requests. The service layer uses the LessonRepository to interact with the database, and the data is modeled by entities like LessonPageEntity, LessonContentEntity, and PracticeQuestions. This layered approach ensures a separation of concerns, where each layer has a specific role in the handling of lessons within the system.

Component: Announcement

The above UML class diagram describes the structure of the Announcement component in the application, detailing how the system's classes interact with one another to manage all the announcements.

AnnouncementController: This is a class that acts as a controller in the MVC (Model-View-Controller) pattern. It exposes methods such as createAnnouncement(), getAnnouncement(), updateAnnouncement(), and deleteAnnouncement(). These methods handle HTTP requests to perform CRUD (Create, Read, Update, Delete) operations on announcements.

AnnouncementService: This is an interface that declares the methods the service layer must implement. The service layer contains the business logic and calls the repository layer to persist data. It has similar methods to the controller such as createAnnouncement(), getAnnouncement(), updateAnnouncement(), and deleteAnnouncement().

AnnouncementServiceImpl: This class implements the AnnouncementService interface. It implements all the functions such as createAnnouncement(), getAnnouncement(), updateAnnouncement(), and deleteAnnouncement(). It defines the logic for the operations declared in the service interface. This layer usually handles business logic, validation, and calls the repository layer for data access.

AnnouncementRepository: This interface defines the CRUD operations that the data access layer will perform on the announcement data. It is part of the repository layer, which directly interacts with our MongoDB database.

AnnouncementRepoMongoDBImpl: This class implements the AnnouncementRepository interface and provides the MongoDB specific implementation for the repository operations. It implements functions such as createAnnouncement(), getAnnouncement(), updateAnnouncement(), and deleteAnnouncement(). It's the data access object (DAO) that performs the actual database operations.

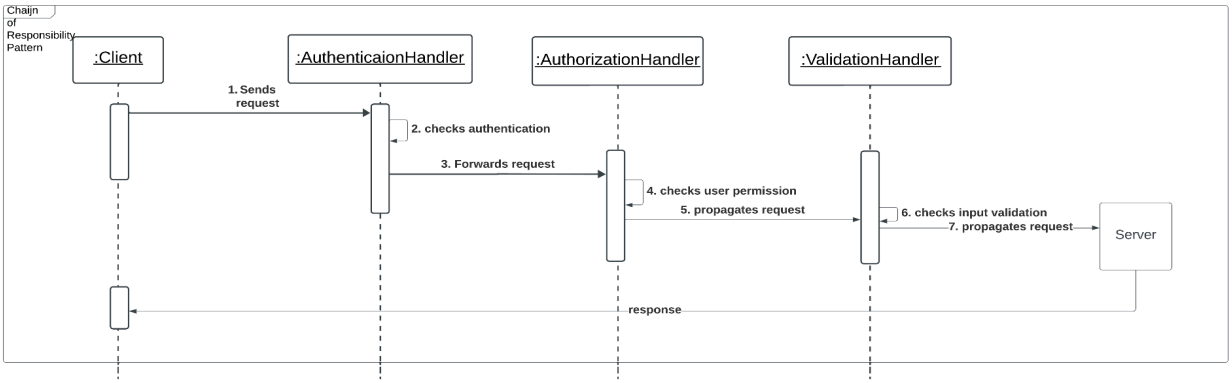
AnnouncementEntity: This class represents the data model for an announcement. It contains fields like id, title, content, author, createdAt, and updatedAt. There are also methods to get these field values, suggesting it follows the Java Bean pattern with private fields and public getter methods**.**

The diagram shows a clear separation of concerns, with a controller handling HTTP requests, a service layer managing business logic, and a repository layer interacting with a MongoDB database to persist announcements.

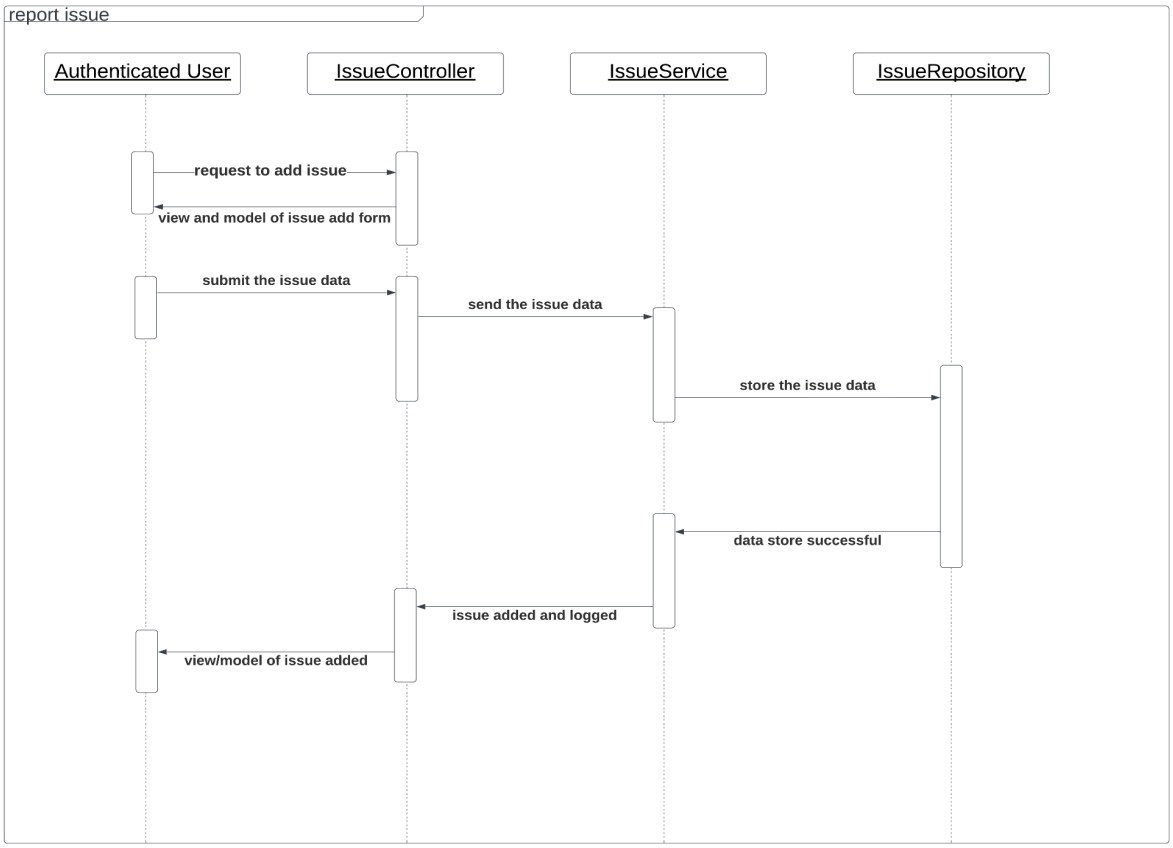
6. Detailed Design Behavior

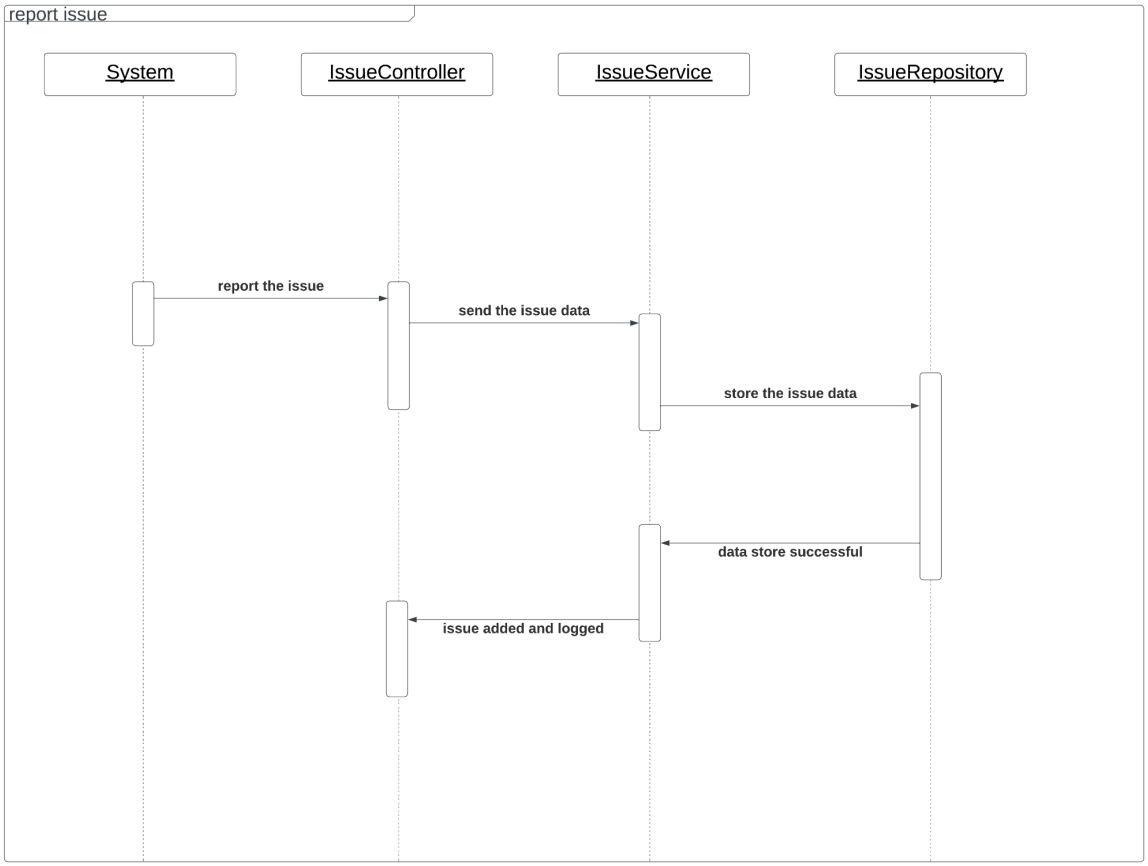
Chain of Responsibility Pattern for Request Processing:

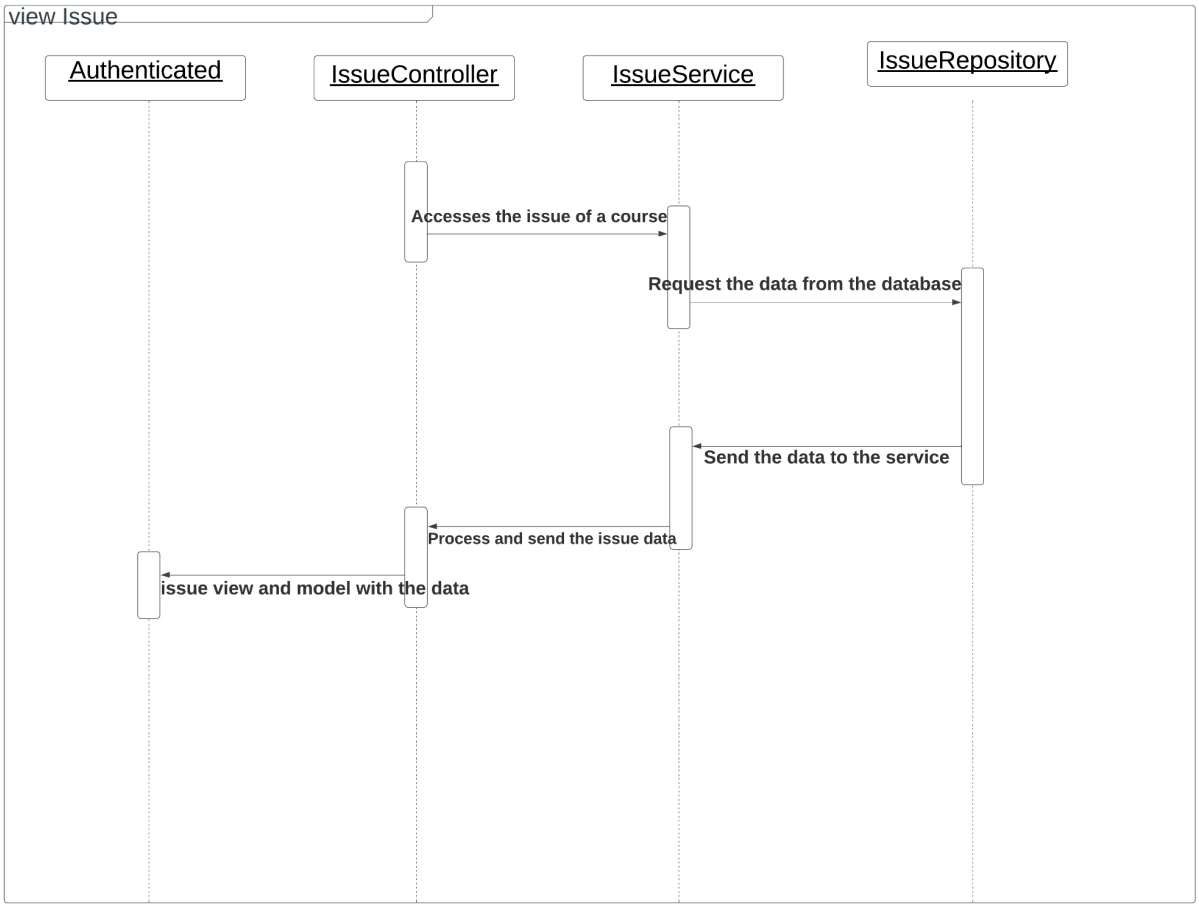
1. The client initiates a request that needs to go through authentication, authorization, and validation.
2. The request is initially passed to the AuthenticationHandler. If authentication is successful, it continues to the next handler. Otherwise, the process stops, and a response is generated.
3. The AuthorizationHandler processes the request if needed. If authorization is successful, it continues to the next handler. Otherwise, the process stops, and a response is generated.
4. The ValidationHandler processes the request if needed. If validation is successful, it continues to the next handler. Otherwise, the process stops, and a response is generated.
5. The final response is returned to the client.



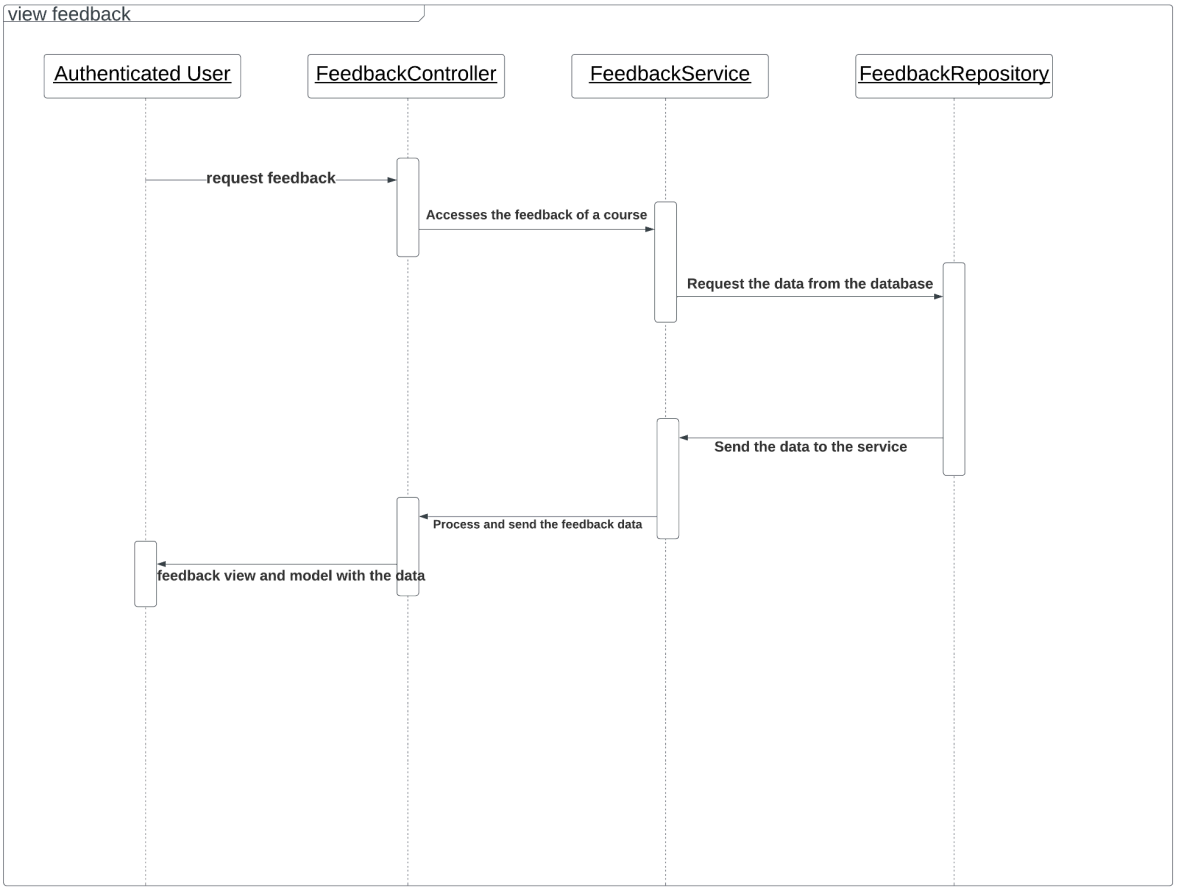
Component: Issues

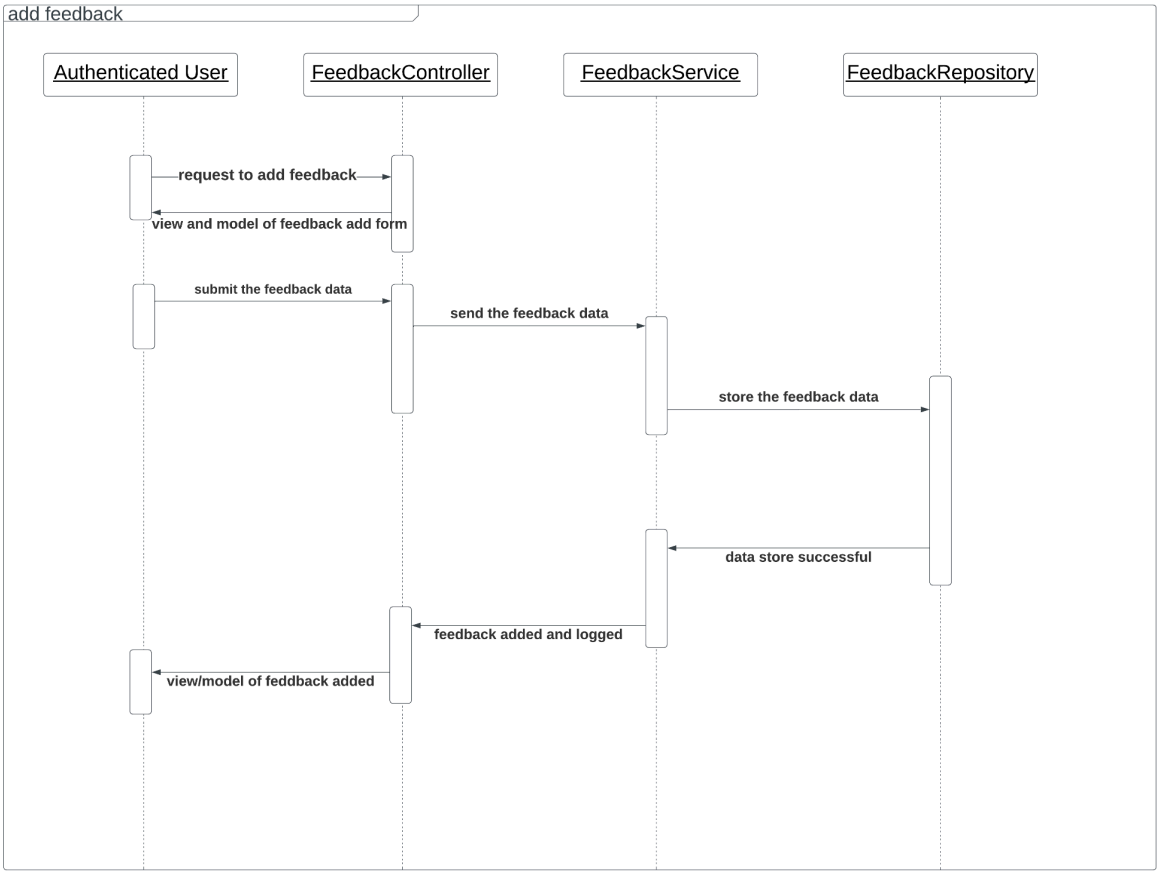



report issue
System
IssueController
IssueService
IssueRepository
report the issue
send the issue data
store the issue data
data store successful
issue added and logged

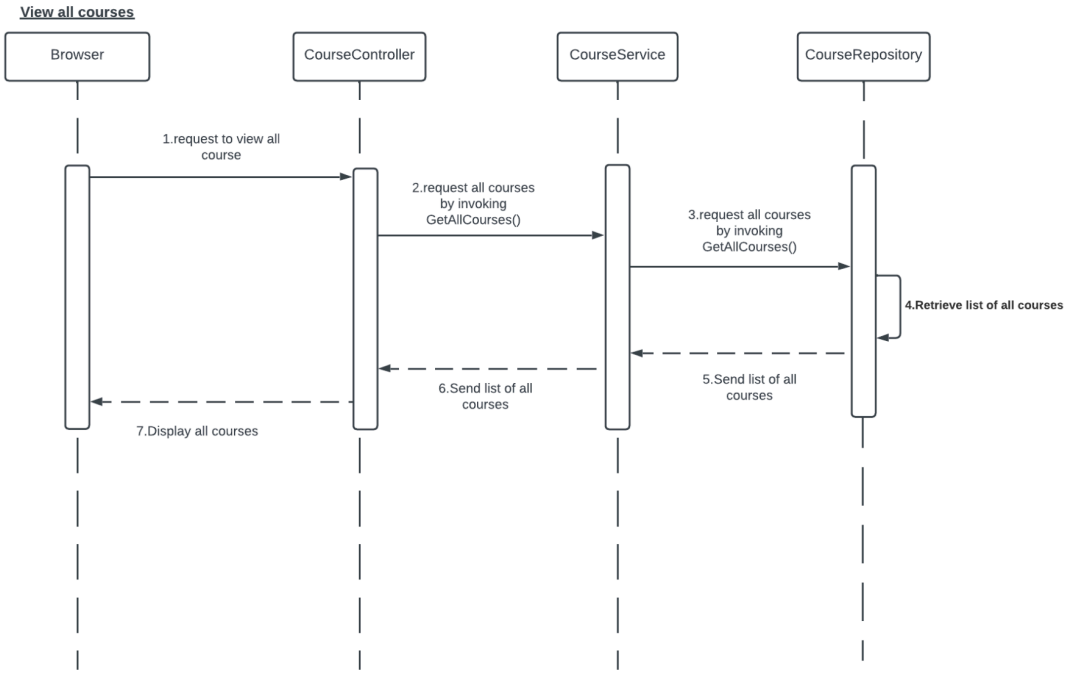

view Issue
Authenticated
IssueController
IssueService
IssueRepository
Accesses the issue of a course
Request the data from the database
Send the data to the service
Process and send the issue data
issue view and model with the data

Component: Feedback



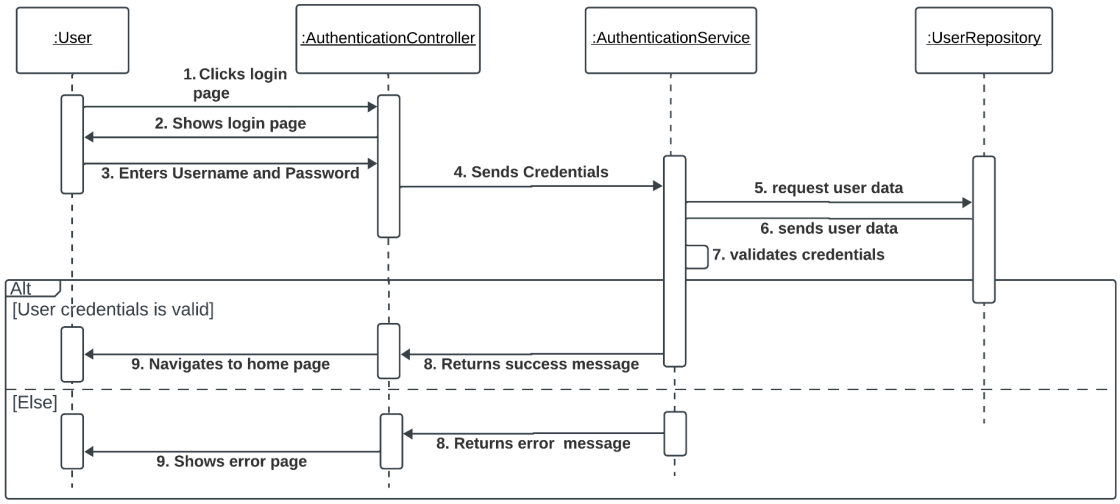

add feedback
Authenticated User
FeedbackController
FeedbackService
FeedbackRepository
request to add feedback
view and model of feedback add form
submit the feedback data
send the feedback data
store the feedback data
data store successful
feedback added and logged
view/model of feddback added

Component: Course



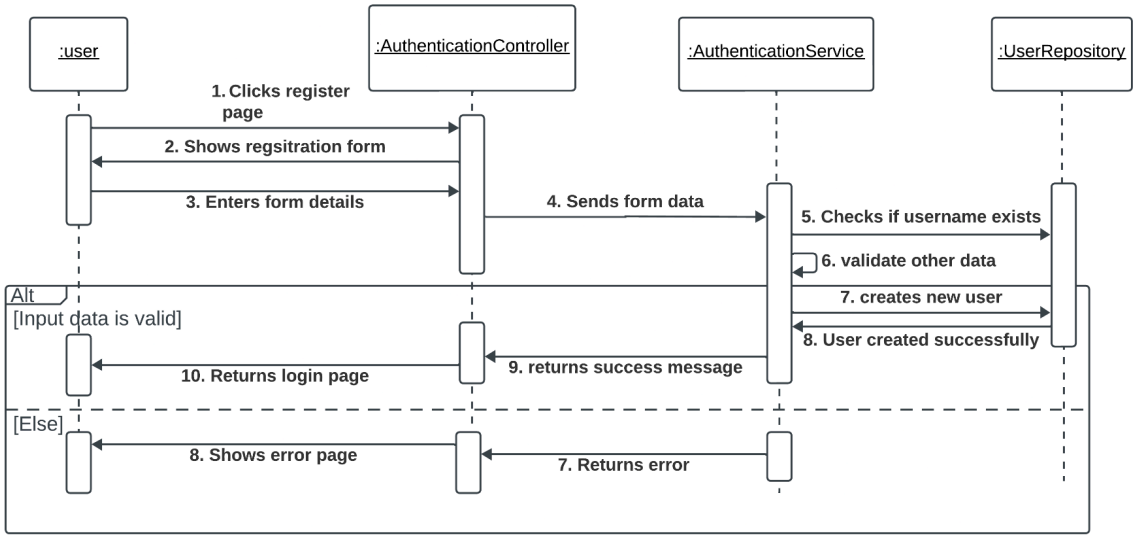
Component: Authentication

1. **Login**
   In this activity, The user logs in before carrying out any other activity. This activity involves an authentication component**.**



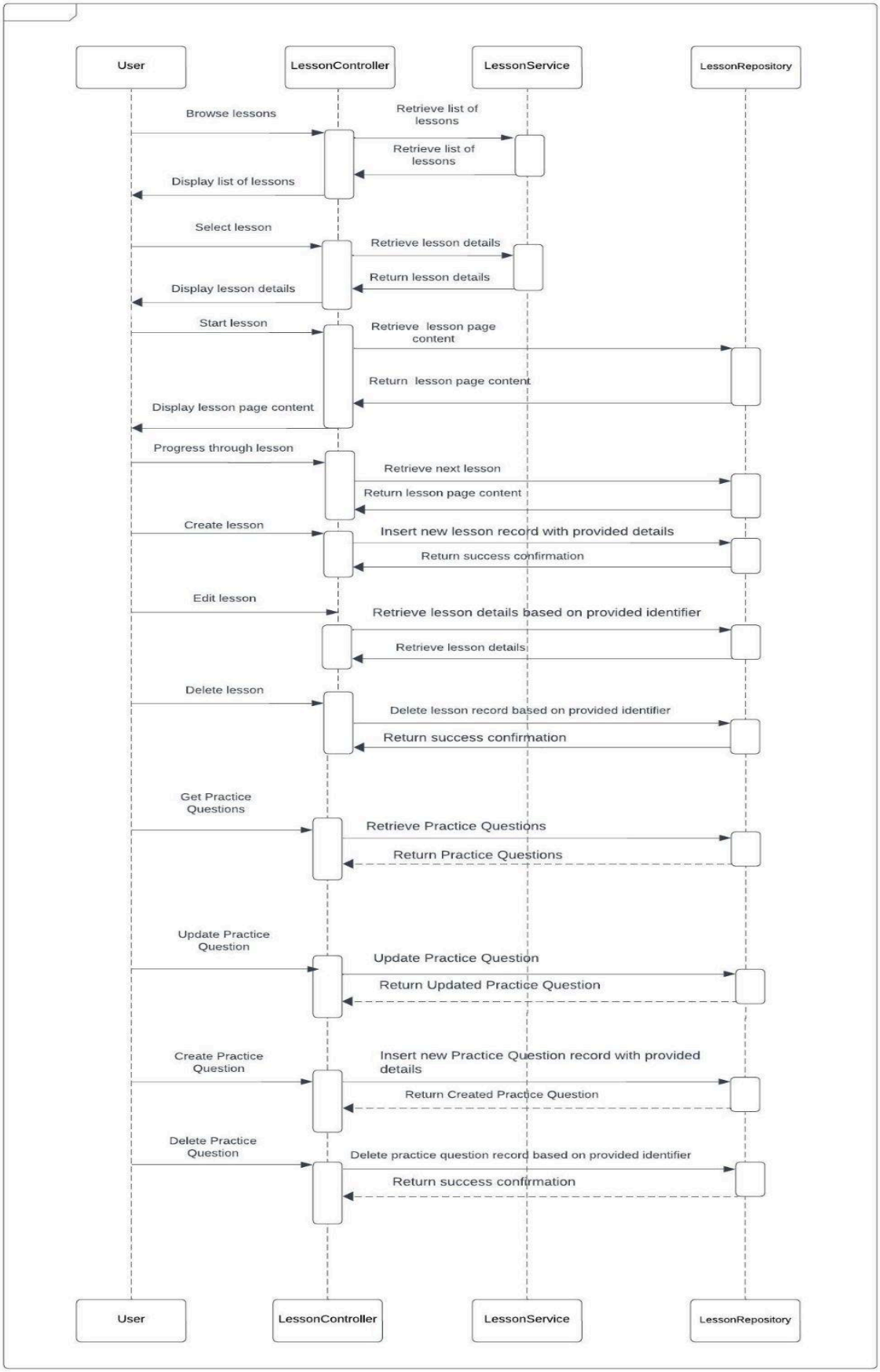
2. Register

In this activity, The user registers as a new user before carrying out any other activity. This activity involves an authentication component**.**



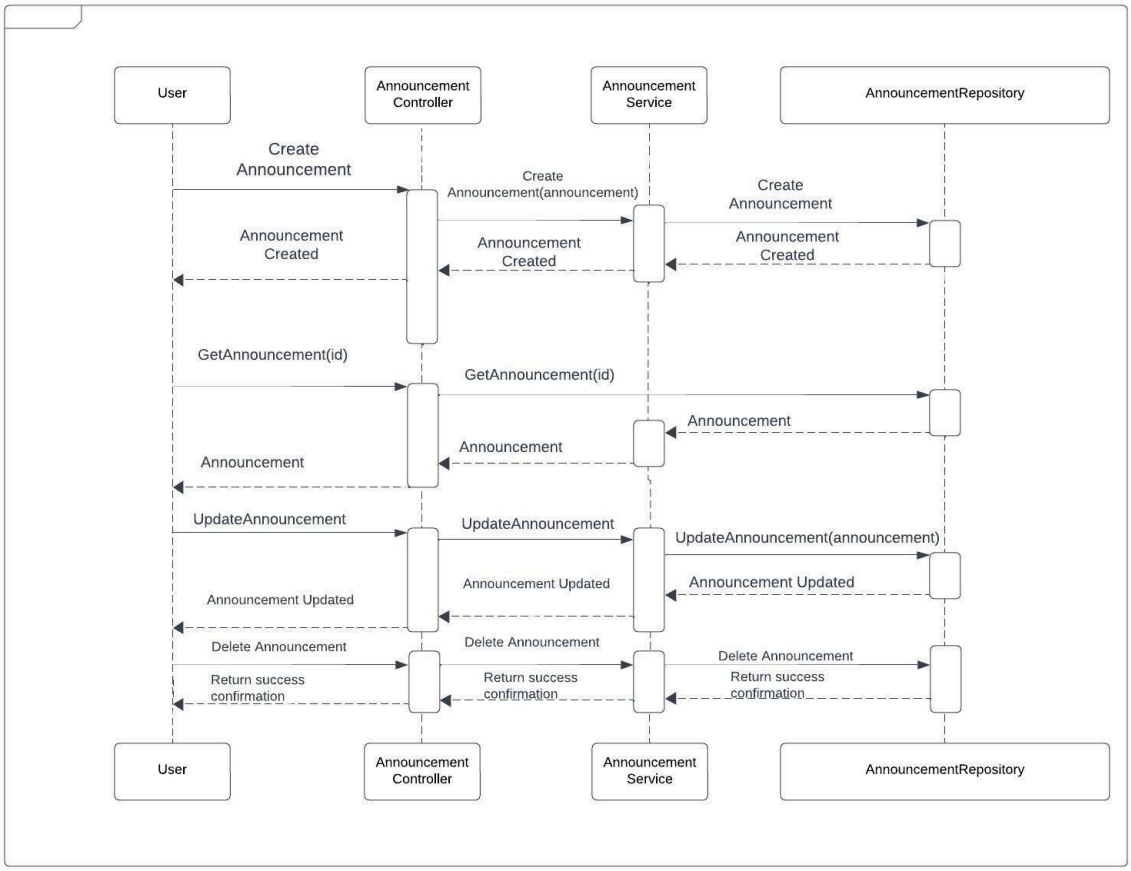
Component: Lesson

The sequence diagram below outlines the interactions between different components when a user performs operations related to lessons.


User
LessonController
LessonService
LessonRepository
Browse lessons
Retrieve list of lessons
Retrieve list of lessons
Display list of lessons
Select lesson
Retrieve lesson details
Return lesson details
Display lesson details
Start lesson
Retrieve lesson page content
Return lesson page content
Display lesson page content
Progress through lesson
Retrieve next lesson
Return lesson page content
Create lesson
Insert new lesson record with provided details
Return success confirmation
Edit lesson
Retrieve lesson details based on provided identifier
Retrieve lesson details
Delete lesson
Delete lesson record based on provided identifier
Return success confirmation
Get Practice Questions
Retrieve Practice Questions
Return Practice Questions
Update Practice Question
Update Practice Question
Return Updated Practice Question
Create Practice Question
Insert new Practice Question record with provided details
Return Created Practice Question
Delete Practice Question
Delete practice question record based on provided identifier
Return success confirmation
User
LessonController
LessonService
LessonRepository

Component: Announcement

The sequence diagram below outlines the interactions between different components when a user performs operations related to announcements.



7. Physical Data Model

User

Primary Key	ID	Long
	Email	string
Physical Data Model	Password	string

	Role	enum
	Bio	string

ID: A unique identifier for each user.
Email: The user's email address is used for authentication and communication.
Password: The user's password for authentication in encrypted form.
Role: Specifies the role of the user (e.g., student, admin, instructor).
Bio: Describes the biographical description of a user.

Feedback

Primary Key	ID	Long
	FirstName	string
	LastName	string
Foreign Key	UserEmail	string
	FeedbackContent	string
Foreign Key	CourseTitle	string

Feedback Module:
ID: A unique identifier for each feedback or user comment.
FirstName: First name of the user providing the feedback.
LastName: Last name of the user providing the feedback.
UserEmail: The email ID of the user providing the feedback.
FeedbackContent: Attribute specifying the content of the feedback.
CourseTitle: attribute for the course feedback is given.

Issue

Primary Key	ID	Long
	IssueContent	string
	IssueType	string
	UserImpacted	string
	TimeOfOccurence	dateTime

	Frequency	Integer

Issue:
ID: A unique identifier for each reported issue or support ticket.
IssueContent: An attribute to specify the issue.
IssueType: An identifier for the type of an issue.
UserImpacted: The user impacted with the issue.
TimeOfOccurence: The time issue occurred.
Frequency: Number of times the issue happened.

Course

Primary Key	CourseID	Long
	Title	string
	Description	string
	Complexity	enum
	EnrollmentStatus	enum

Course:
ID: A unique identifier for each course.
Title: The title or name of the course.
Description: A brief description of the course's content and objectives.
Complexity: This attribute quantifies the level of complexity or difficulty associated with a specific course (e.g., easy, medium, hard).
EnrollmentStatus: Indicates whether enrollment in the course is open or closed.

Lesson

Lesson Page Entity

Primary Key	ID	Long
Foreign Key	CourseID	Long
	Title	string
	Content	List<LessonContent>

	EstimatedCompletionTime	Long

Lesson page entity:
ID: A unique identifier for each lesson.
CourseID: The ID of the course to which the lesson belongs.
Title: The title or name of the lesson.
Content: The content or material covered in the lesson.
EstimatedCompletionTime: Specifies the time in which the lesson should be completed

Lesson Content Entity

Primary Key	LessonContentId	Long
	LessonPageId	Long
	Data	string
	PracticeQuestions	List<PracticeQuestions>
	MediaLink	string

Lesson content entity:
LessonContentId: A unique identifier for each lesson content.
LessonPageId: The ID of the lesson page.
Data: The content or material covered in the lesson.
PracticeQuestions: List of all the practice questions in a lesson.
MediaLink: This attribute stores links or references to multimedia resources associated with a particular lesson.

Practice questions

Primary Key	PracticeQuestionId	Long
	QuestionDifficulty	string
	QuestionContent	string
	AnswerContent	string

Practice Questions:
PracticeQuestionId: A unique identifier for each practice question in a lesson.
QuestionDifficulty: The complexity level of the question.

QuestionContent: The attribute for the question.
AnswerContent: Attribute for the solution of the question.

Announcement

Primary Key	ID	Long
	Title	string
	Content	string
	Author	string
	CreatedAt	date
	UpdatedAt	date

Announcement:
ID: A unique identifier for each announcement.
Title: The title or subject of the announcement.
Content: The detailed content or message of the announcement.
Author: The ID of the user (e.g., admin or instructor) who authored the announcement.
CreatedAt: The date when the announcement was published.
UpdatedAt: Specifies the date when the announcement is updated.

8. Algorithms

An efficient algorithm for searching terms in courses is the Trie (Prefix Tree) algorithm. A Trie is a tree-like data structure that is used to store a dynamic set of strings, where the keys usually represent words. It allows for efficient insertion, deletion, and lookup operations associated with strings.

Here's a simplified version of a Trie-based search algorithm for terms in courses:

Build the Trie:

Create a Trie data structure to store the terms from the courses.
Each node in the Trie represents a character, and the edges represent the characters' relationships.

Insert Terms:

For each term in the courses, insert it into the Trie by adding nodes for each character in the term.

Search Terms:

To search for a term:
Start at the root of the Trie.
Traverse down the Trie following the characters of the term.
If the term is found in the Trie, return the corresponding information (course details, IDs, etc.).

9. Architecture to Detailed Design Tracing

Architecture/Detailed Design	**CourseController**	**CourseManagementController**	**CourseService**	**Enrollement Service**	**Course Repository**	**CourseEntity**
Course	X		X		X	X
Course Management		**X**		**X**	X	X
Manage Courses	X	**X**	**X**	**X**	X	X

Architecture/Detailed Design	**AuthenticationController**	**ProfileController**	**AuthenticationService**	**UserService**	**UserRepository**	**UserEntity**
User	X	**X**	X	X	X	X
Profile Management		**X**		**X**	X	X
Authentication	X		**X**		X	X
view/modify Profile	X	**X**	**X**	**X**	X	X

Architecture/ Detailed Design	**Feedback Controller**	**Feedback Service**	**Feedback Repository**	**Feedback Entity**	**Issue Controller**	**IssueServiceImpl**	**Issue Repository**	**Issue Entity**	**Announcement Controller**	**Announcement Service**	**Announcement Repository**	**Announcement Entity**
Feedback	X	X	X	X								
View/ Provide Feedback	X	**X**	X	X								
Issues					X	X	X	X				
View/ Report Issues					X	X	X	X				
Announcement									X	X	X	X
Post/ View									X	X	X	X

Architecture/Detailed Design	**LessonController**	**LessonService**	**LessonRepository**	**LessonPage Entity**	**LessonContentEntity**	**PracticeQuestions**
Lesson	X	X	X	X		
Practice Questions	X	**X**	X	X	X	X
Media Handler	X	**X**	X	X	X	

Manage Lesson	X	**X**	X	X	X	X

Individual Student Contribution

Student name	Requirements analysis	User models & UI prototype	SW architecture	SW detailed design	Implementation	Project management
Muskan Sagar	Developed abuse case model, Wrote abuse case description, Developed abuse case diagram, Developed security scenarios, Filled bi-directional traceability matrix between abuse cases and security scenarios, Reviewed SRS	Created UI Flow diagram and crafted Storyboard diagrams for three distinct scenarios, including the Primary persona, Special persona, and Attacker persona	Created Preliminary architecture decisions document. Defined brief description of system and top priority requirements.Developed trade-off analysis table. Identified risks associated with selected technology and developed respective mitigations. Identified major architectural elements and their relation. Designed behavior view of the architecture.	Defined structural and behavioral diagram for Course and Course management components.		Conducted team meetings, Assigned tasks, Resolved team queries, Tracked project progress to complete deliverables on time

Sachin Velmurugan	Deduced functional requirements, Developed the use cases and features, Developed quality attributes, Created SRS, Designed the context diagram	Designed UI sketches, Created prototype for primary persona, Identified and created the primary persona. Reviewed UI document.	Applied architecture styles and tactics for our product and drew refined architecture. Developed ER diagram for information view.	Applied MVC design pattern, and observer design pattern. Created class diagram and sequence diagram for authentication component.		
Barath Bhaskaran	Deduced the non functional requirements, Developed scenarios for quality attributes,designed utility tree	Designed UI sketches.Identified and Created Attacker persona. Created wireframe for attacker persona.Reviewed UI document.	Designed Logical view and preliminary architecture. Developed component and interface description for the components.	Created structural and behavioral diagram for Lesson and Announcements components.		
Swathi Selvakumaran	Deduced functional requirements, Designed the use case diagram, Worked on security scenarios, helped with bi-directional traceability matrix between abuse cases and security scenarios	Identified and created a special persona. Designed UI sketches, Created UI prototype for special persona, Reviewed UI document.	Applied architecture styles and tactics for our product and drew event driven architecture, Developed the deployment view for the application. Elaborated Component and Interface description in logical	Created structural and behavioral diagram for feedback and issues component, Developed Architecture to detailed design		

			view, Developed allocation view	traceability table		
Keerthan Mahesh	Developed functional requirements, non-functional requirements use cases, use case diagram, functional features template, trace features use case matrix, reviewed, validated and formatted SRS document	Developed UI Flow diagram. Implemented Storyboard diagrams for three scenarios - Primary persona, Special persona, Attacker persona	Implemented Quality Attributes and top priority quality scenarios. Constructed Constraints for the web application. Designed and implemented the Utility Tree diagram. Consolidated the required sections into the SAD document.	Implemented the ProfileController, User service, User Repository design patterns. Also assisted in implementing the Lesson component design pattern.		Allocated responsibilities to team members, brainstormed ideas, monitored task completions