System Design

1. Project Overview

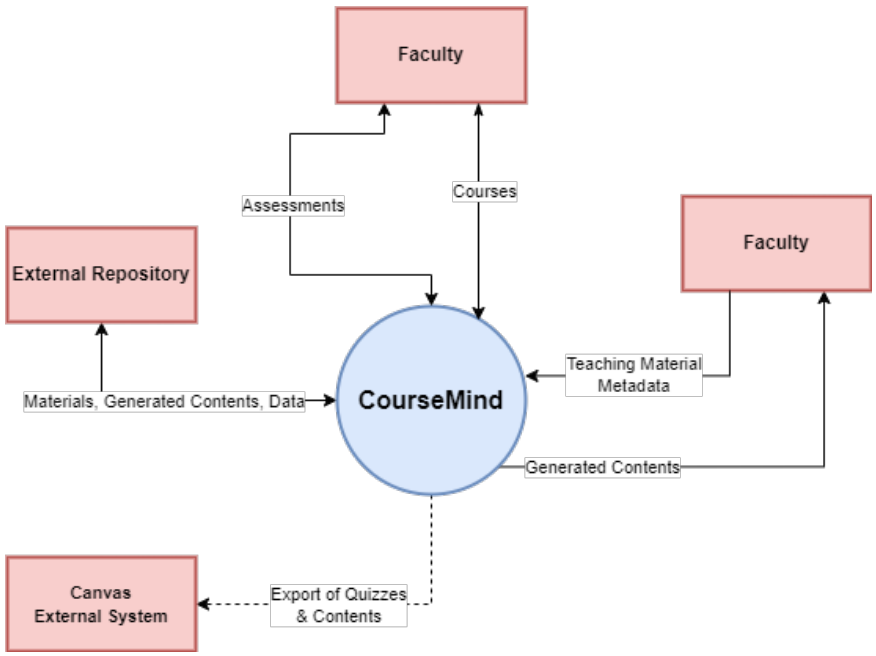
We are developing a web-based application to help professors manage their courses more efficiently by assisting them with course planning, quizzes and assignment creation. The application will provide features to outline the course, auto-generate quizzes from the context and track project submissions and manage deadlines in a centralized interface.

1.1. Introduction

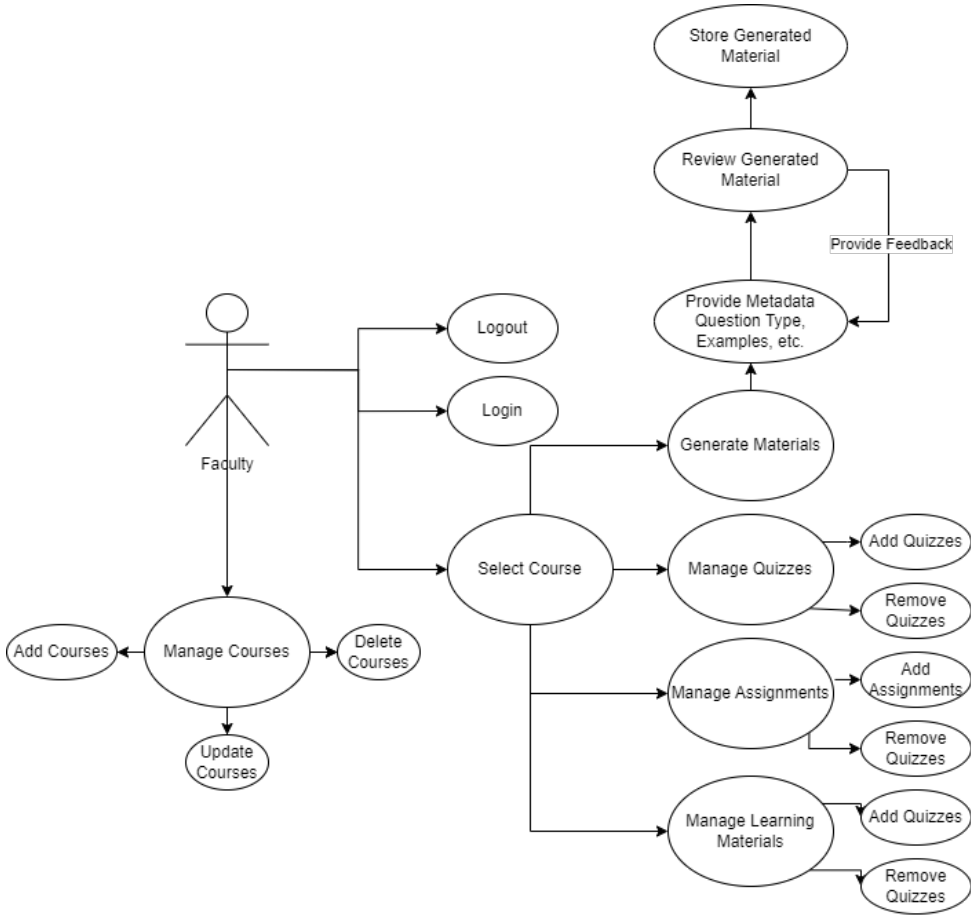
Professors often spend a significant amount of time creating course outlines, assignments, quizzes, and projects. Managing these tasks manually can be inefficient, especially for professors handling multiple courses. The need for a tool that simplifies these tasks has never been greater, as it would allow educators to focus more on teaching and student interaction rather than administrative tasks.

This project aims to build an application that assists professors in planning their courses, generating questions, and organizing small projects in a more streamlined manner. The tool will simplify the creation of quizzes, assignments, given the course outlines and materials, saving professors time while ensuring consistency and quality in their content delivery.

1.2. Context Diagram



1.3. Use case Diagram



1.4. Use cases and User Stories

USE CASE NAME:	**Generating Examples from teaching materials**	**USE CASE TYPE**
USE CASE ID:	**1**	**Business Requirements:** □
PRIORITY:	**1**	**System Analysis:** □
SOURCE:	**Teacher**	**System Design:** ?
PRIMARY BUSINESS ACTOR	**Teacher who wants to teach some topic to students**	

PRIMARY SYSTEM ACTOR	**CourseMind System (working on generating examples)**	
OTHER PARTICIPATING ACTORS:	• **Generative AI Model (part of CourseMind)**	
OTHER INTERESTED STAKEHOLDERS:		
DESCRIPTION:	**Focuses on generating examples to teach students from given teaching materials**	
PRE-CONDITION:	**Teacher has uploaded some teaching materials for the selected course from which Teacher wants CourseMind to create examples.**	
TRIGGER:	**Teacher wants to teach something and seeks to generate examples**	
TYPICAL COURSE OF EVENTS:	**Actor Action**	**System Response**
	Step 1: Teacher navigates to selected course	**Step 4: Gives examples for faculty to filter through**
	Step 2: Selects uploaded material from which teacher wants to create examples	
	Step 3: Gives some details about what type of examples teacher wants	
	Step 5: Selects and exports these examples in JSON	
ALTERNATE COURSES:	**If Teacher doesn't like the examples generated by CourseMind, teacher can provide feedback to CourseMind which allows teacher to get different examples**	

	If user CourseMind doesn't get any good examples it would return with error message
CONCLUSION:	**Teacher will have examples for particular topic ready to help students with studying**
POST-CONDITION:	**Teachers will have examples generated and stored into a database as well as JSON of all examples exported.**
BUSINESS RULES	• **Teacher should be logged in** • **Should have uploaded some reference material**
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS	• **Only authorized users can access** • **It creates examples on 1 topic at a time**
ASSUMPTIONS:	• **Teachers will provide high quality teaching material to create examples from**
OPEN ISSUES:	

		USE CASE TYPE
USE CASE ID:	**8**	**Business Requirements:** □
PRIORITY:	**1**	**System Analysis:** □
SOURCE:	**Teacher**	**System Design:** ?
PRIMARY BUSINESS ACTOR	**Teacher who is responsible for managing course quizzes**	
PRIMARY SYSTEM ACTOR	**CourseMind System (helps teachers to manage course quizzes)**	
OTHER PARTICIPATING ACTORS:	• **Generative AI Model (part of CourseMind)**	
OTHER INTERESTED STAKEHOLDERS:		

DESCRIPTION:	**This feature focuses on how teachers manage the course quizzes for a particular course including the ability to add, remove and review assignments.**	
PRE-CONDITION:	**Teacher should have successfully logged into the system and has selected a course for which they want to manage quizzes.**	
TRIGGER:	**Teacher wants to manage assignments for students**	
TYPICAL COURSE OF EVENTS:	**Actor Action**	**System Response**
	Step 1: Teacher navigates to selected course's dashboard	**ii) Remove Quizzes:** **Teacher selects an existing quiz to remove.** **The system asks for confirmation before removing the assignment. Teacher confirms, and the quiz is removed.**
	Step 2: Selects "Manage Assignments" option and the system displays all the existing assignments for selected course	**iii) Review Quiz:** **Teacher selects an existing quiz to review.** **The system displays the details of the selected quiz. Teachers can update the details or leave them unchanged.**
	Step 3: Teacher can choose to do	
	i) Add Assignments: **Teacher selects Add Assignment** **The system prompts for assignment details (title, due date, description, etc)**	

ALTERNATE FLOW:	**If Teacher attempts to add a quiz without providing all required information (e.g., missing title or due date), the system will prompt them to complete the missing fields before submission.**
	If Teacher tries to remove a quiz that has active student submissions, the CourseMind system will prompt them with a warning that removal will also delete the submissions, requiring confirmation.
CONCLUSION:	**Teacher will be able to add, delete and update quizzes to keep the course materials up to date and relevant**
POST-CONDITION:	**If the quiz is added or removed successfully, the course quiz list is updated accordingly.** **If there is an issue with adding or removing quiz (e.g., network error), an error message is displayed to the faculty member.**
BUSINESS RULES	• **Teacher should be logged in** • **Should have quizzes to be managed**
IMPLEMENTATION CONSTRAINTS AND SPECIFICATIONS	• **Only authorized users can access**
ASSUMPTIONS:	• **Teachers will perform high quality management to manage quizzes**
OPEN ISSUES:	

1.5. Project Scope

- Develop a user-friendly interface for professors to create and manage their courses.
- Build a question bank for quizzes and assignments with the ability of auto-generation.
- Enable professors to add, update, delete new course assignments and quizzes.
- Allow course content and assignments to be exported to share with others.

1.6. Tasks

- Task 1 - Authentication - UI
- Task 2 - Dashboard with list of courses - UI

- Task 3 - Inner page for list of assignments and quizzes for each course - UI
- Task 4 - Create a new course - UI
- Task 5 - Add new materials for course - UI
- Task 6 - Create new quizzes - UI
- Task 7 - Create new assignment - UI
- Task 8 - Create database schema - backend (structuring)
- Task 9 - Authentication - backend API
- Task 10 - List of courses - backend API
- Task 11 - CRUD for courses - backend API
- Task 12 - CRUD for materials for course - backend API
- Task 13 - CRUD for quizzes - backend API
- Task 14 - CRUD for assignment - backend API
- Task 15 - Get a model to create questions from given context - AI Model (POC)
- Task 16 - Convert document to data chunks - AI Model
- Task 17 - Use data chunks to find key information in chunks - AI Model
- Task 18 - Use key information and chunks to create answers - AI Model
- Task 19 - Get a model to generate assignments or short projects problems for course - AI Model
- Task 20 - Integration of Authentication UI & API
- Task 21 - Integration of Courses CRUD UI & API
- Task 22 - Integration of Quizzes CRUD UI & API
- Task 23 - Integration of Assignments CRUD UI & API
- Task 24 - Integration of AI Model to feed new materials
- Task 25 - Integration of AI Model to generate new things

2. Project Architecture

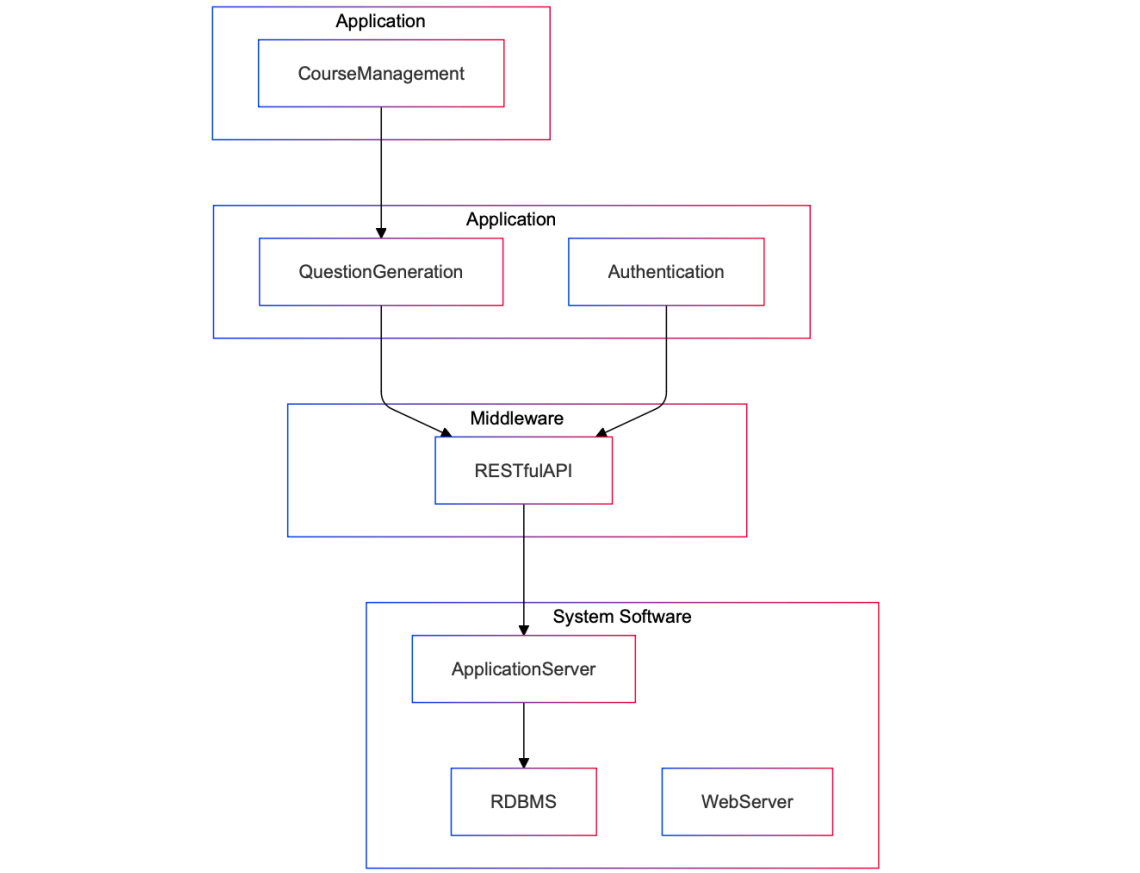
CourseMind architecture balances simplicity with flexibility. Clearly defined layers, secure data storage, and responsive control flow, the system performs its job of supporting professors in course management, assignments, and quizzes without technical complexities. The application will be highly accessible, reliable, and scalable in such a setting; it will be an easy and useful tool for educators.

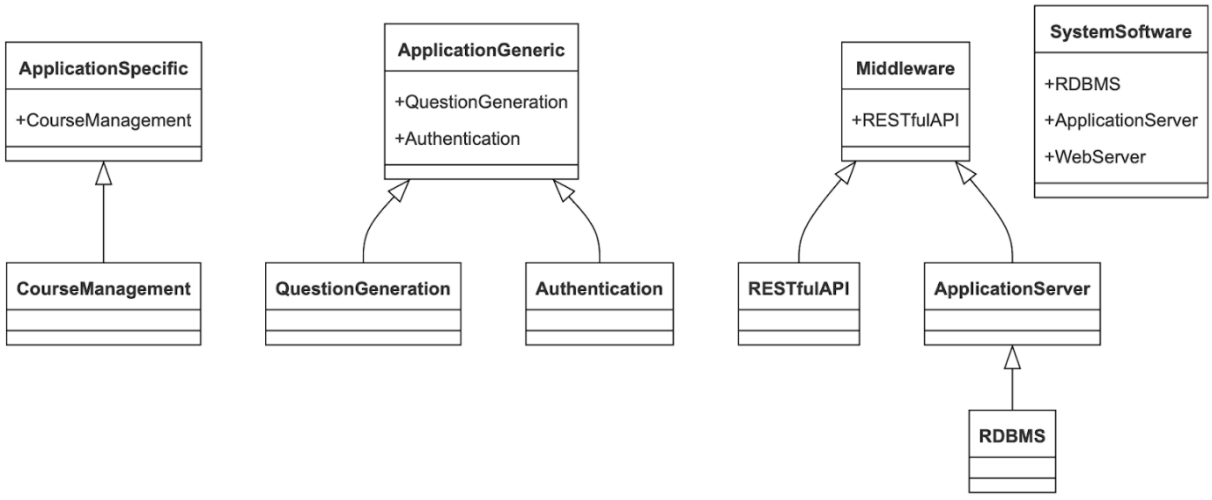
2.1. Subsystem Architecture

This section explains the system’s layers and why they’re organized this way, focusing on flexibility and ease of maintenance.

The architecture followed is the multi-tier architecture; it has divided the system into layers. This separation between presentation layer, business layer, and data storage makes the system very scalable and maintainable. Yes, we did think over the monolithic design, but we saw that in such a design, scalability may be a problem, and the system probably can't handle many users at once.

For the architectural overview, we would typically include a subsystem package diagram





Application-Specific: Handles course management, quiz generation, and assignment modules, providing core functionalities for professors.

Application-Generic: Comprises reusable components like user interfaces and authentication services along with managing common features like question generation and authentication

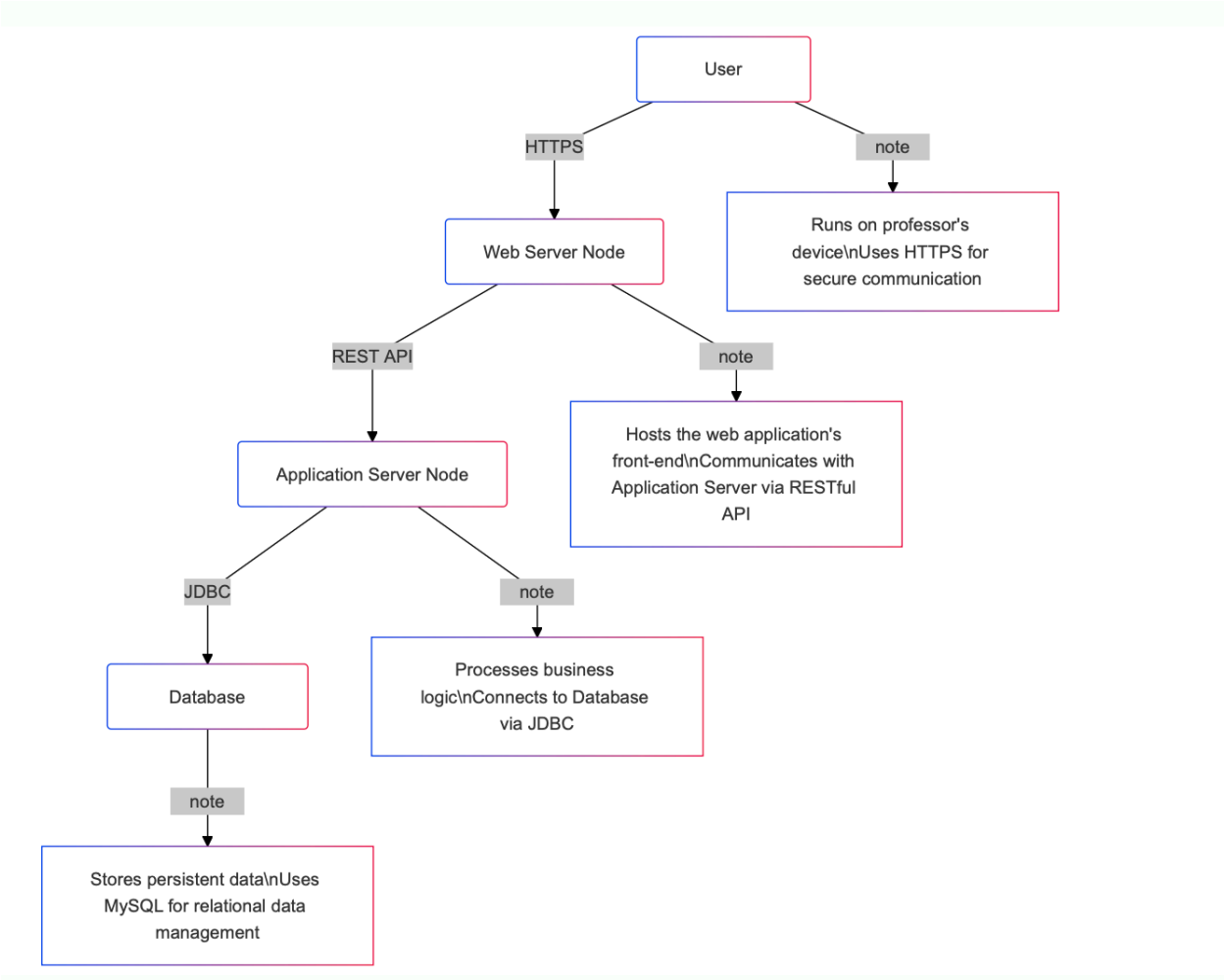
Middleware: The REST API acts as the communication layer between the frontend and backend

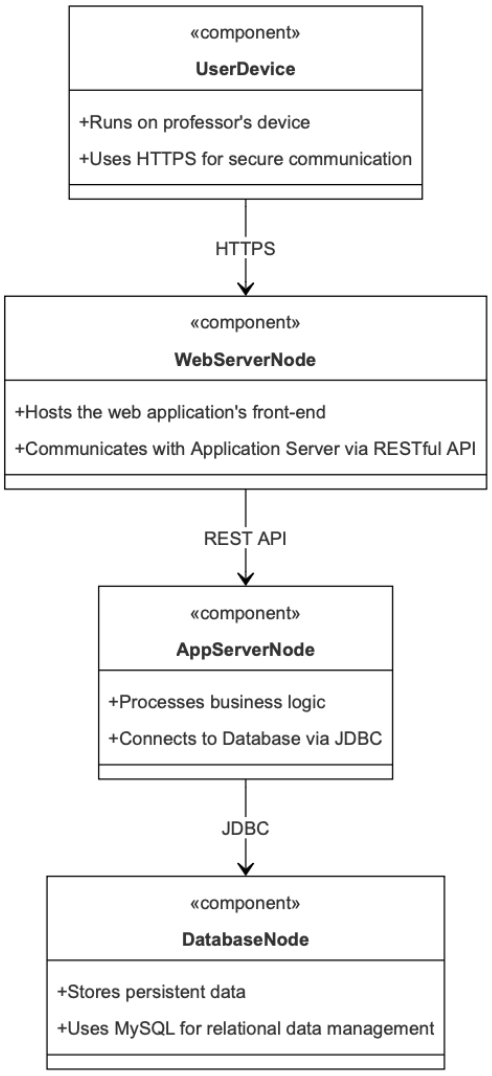
System Software: This is the foundation, including the operating system and the web server where the backend operates.

Reason for this Architecture: This layered approach keeps things organized and makes future updates easier. By separating layers, we can improve one layer (like the user interface) without affecting other layers.

2.2. Deployment Architecture

The deployment diagram maps out where each part of the "CourseMind" application will run and how they communicate. Basically, it shows how different parts of the system are physically distributed and communicate with each other. In this case, we have separated the front-end, back-end, and database layers to enhance scalability, maintainability, and security.





Detailed Explanation:

- **User Devices**: This represents user devices (laptops, desktops, tablets, or smartphones) used by professors where the application front end will be accessed through web browsers and facilitating professors' interaction with the system over a secure HTTPS connection and accessibility without requiring installation.
- **Backend Server**: This server hosts the core functionality that is UI components, handling authentication, course management, and quiz/ assignment generation.
- **webserver Node**: Contains the main logic for course management and content generation. The server can handle multiple requests simultaneously, enabling professors to create and manage course content concurrently.
- **Central Database**: Stores course content, quizzes, assignments, and user credentials. Communication between the backend and database occurs via JDBC or SQL protocols for reliable data transactions

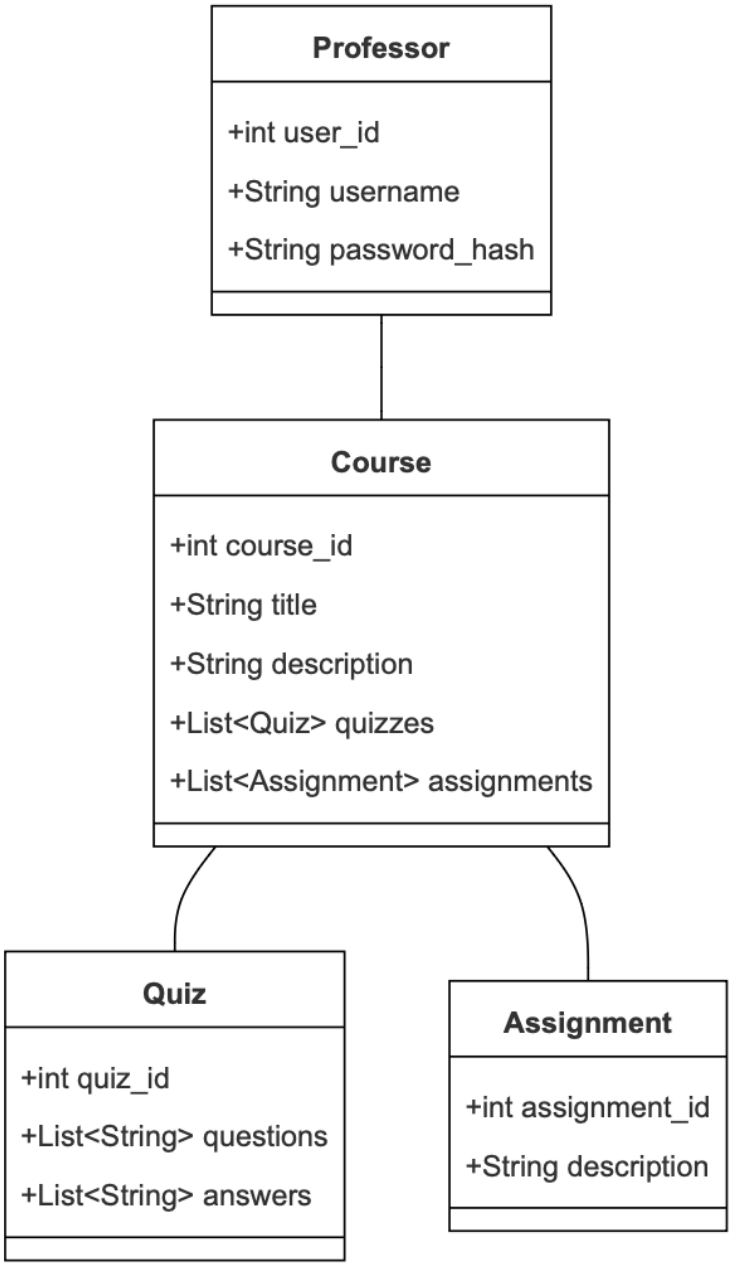
Communication: The frontend (user's device) and backend server communicate via HTTP/HTTPS for security. The backend uses SQL or similar database protocols to interact with the database.

2.3. Persistent Data Storage

Our system uses MongoDB, enabling the storage of the data in a persistent database. The scalability and flexibility of MongoDB to store complex data in non-relational ways make it an apt choice for CourseMind.

The major data models for CourseMind are the following:

- **User**: Stores information about users, including user ID, name, email address, and role, such as a student or instructor.
- **Course**: Lists courses, course ID, title, description, and instructor ID.
- **Module**: Represents course modules, including module ID, course ID, title, and description.
- **Lecture**: Stores information about lectures, including lecture ID, module ID, title, content, and media links.
- **Assignment**: Contains information for assignments, such as assignment ID, module ID, title, description, and due date.
- **Quiz**: Holds quizzes associated with modules, including quiz ID, module ID, questions, and answer keys.



Each model is represented as a MongoDB collection, with documents according to each entry for the respective entity. This structure enables efficient querying and processing of data to support the course management functionality in CourseMind.

2.4. Global Control Flow

Our Project has an event-driven control model with elements of procedural flow where necessary. It waits for user inputs, otherwise known as events, to activate various functions associated with course management, assessments, and AI-based content generation.

Control Flow Model

Event-Based Flow: Primarily, CourseMind operates in an event-driven mode. Until someone takes an action within the system (typically, a course instructor logging in and creating a course or quiz), it remains idle. Every action (such as button clicks to add a module, submit an assignment, etc.) creates an event that executes some processes in the Controller. This allows the user to engage with the application at their own speed and in any particular order, depending on what they need.

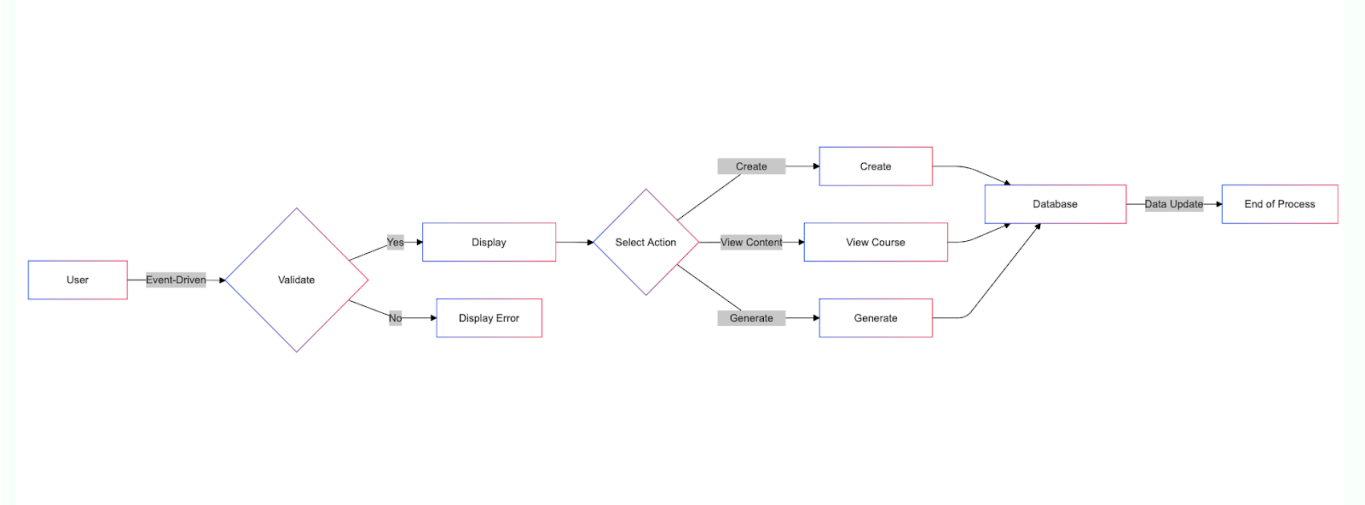
Procedural Flow: Certain processes like authentication or sequential loading of course modules follow a linear procedural flow. These actions have an order (login → load dashboard → show courses) but only if it is the result of a corresponding user event.

Time Dependency: The CourseMind system does not work under firm real-time constraints but includes periodic tasks. For example, it may check at regular intervals to save recent user progress or generate AI-driven content recommendations. These periodic checks allow the system to save user actions periodically without enforcing strict real-time requirements.

Concurrency: Currently, multi-threading isn't used for user-facing tasks because MongoDB's single-threaded nature meets the application's CRUD operation needs. However, concurrency is managed in specific cases:

- **Asynchronous Operations**: CourseMind uses asynchronous operations for data retrieval and saving to prevent blocking, especially with large datasets like multiple courses or assessments. This approach keeps the UI responsive, even for long-running tasks.
- **AI Processing**: For more sophisticated tasks like AI-driven quiz generation, asynchronous processing is used to run these operations in the background. This allows users to keep working within the application while AI functions generate content without interruptions.

Rationale for Control Flow Model: This hybrid of event-driven and procedural control flow was chosen to ensure a responsive, interactive experience for users, especially instructors managing courses at their own pace. By handling data operations and AI processing asynchronously, the system provides prompt feedback, even for data retrieval or complex computations. Since CourseMind's educational environment doesn't demand strict real-time constraints, an asynchronous, event-driven model is ideal.



3. System Design Details

3.1. Static View

Each class (User, Course, Quiz, Assignment) has specific tasks:

User Class: Manages user credentials and sessions. The login() and logout() methods handle user authentication, ensuring secure access to the application.

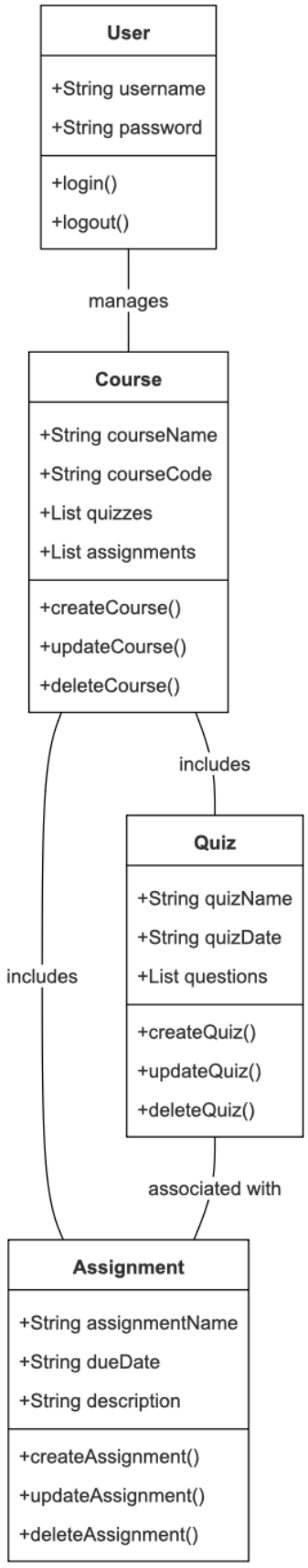
Course Class: Represents a course, with methods for creation, updates, and deletion. It allows professors to add new courses or modify existing ones.

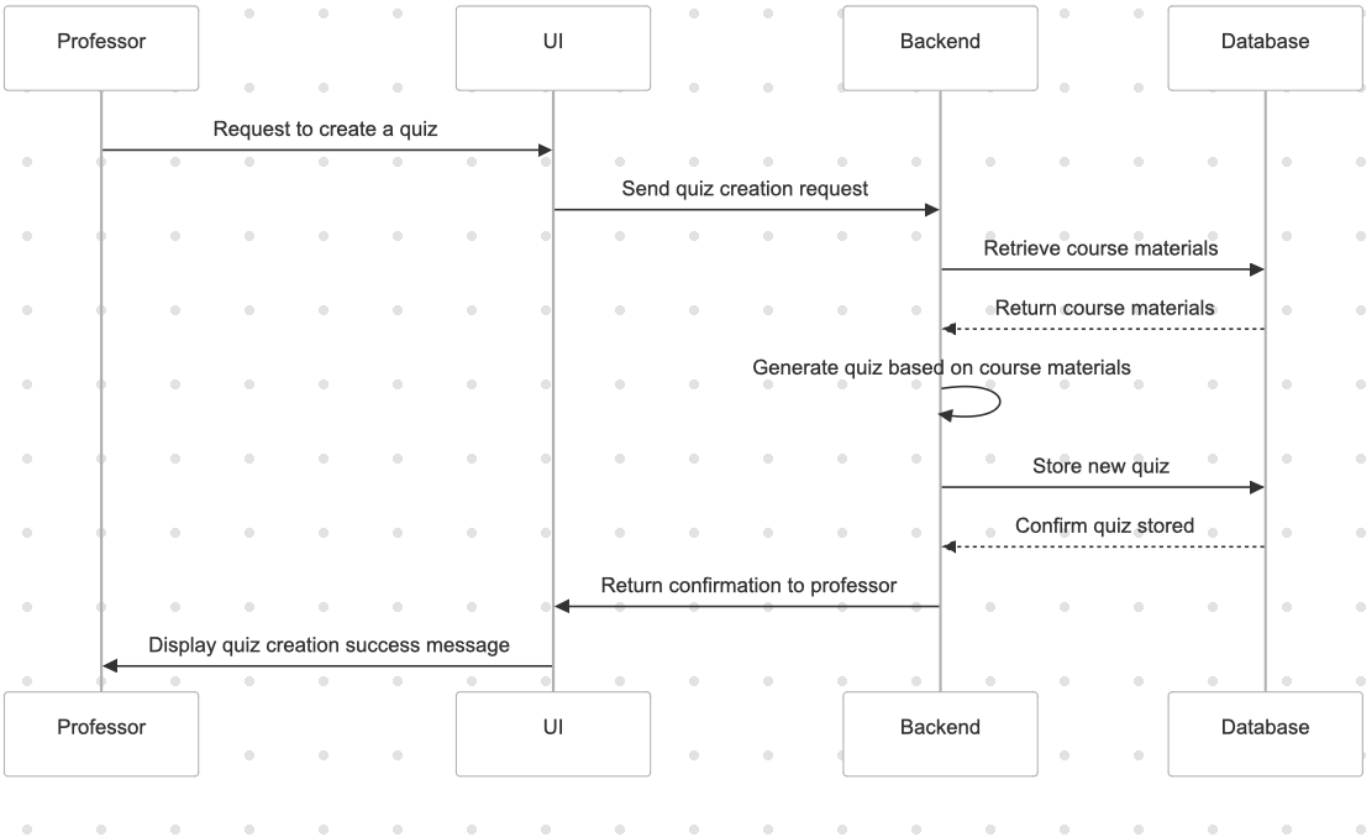
Quiz Class: Contains details of the quizzes, including the questions. The generateQuiz() method automates quiz creation, leveraging course material to populate questions.

Assignment Class: Represents assignments within a course. It has methods for creation and submission, supporting the management of assignments within the course context.

3.2. Dynamic View

It shows the interactions for the quiz generation process, mapping out the sequential flow and responsibilities of each component.


User
+String username
+String password
+login()
+logout()
manages
Course
+String courseName
+String courseCode
+List quizzes
+List assignments
+createCourse()
+updateCourse()
+deleteCourse()
includes
includes
Quiz
+String quizName
+String quizDate
+List questions
+createQuiz()
+updateQuiz()
+deleteQuiz()
associated with
Assignment
+String assignmentName
+String dueDate
+String description
+createAssignment()
+updateAssignment()
+deleteAssignment()



Professor Initiates Quiz Creation: The professor uses the front-end interface to request quiz creation for a specific course.

Front End Sends Request to Backend: The front end sends an HTTP request to the backend, containing relevant course details.

Backend Fetches Course Data: The backend queries the database to retrieve course content that will serve as the basis for quiz questions.

Quiz Generation and Storage: Using the retrieved course content, the backend generates quiz questions, then stores the new quiz in the database.

Confirmation: Finally, the backend sends a success message to the front end, confirming that the quiz was created. This feedback is displayed to the professor

Purpose of this Design: The class structure keeps functions separate and organized. The sequence flow shows how each component contributes to tasks, making the process efficient and easy to follow.