BMED Capstone Course Live Peer Evaluation Application

JID 4100

Chunzhen Hu, Katherine Gu, Jiasheng Cao, Xinyu Liang, Yuanhong Zhou

Client: Erick Maxwell

Repository: https://github.com/Sherwin6180/JID-4100-YoungAdventurers

Table of Contents

Terminology

API (Application Programming Interface): A set of protocols and tools that allows different software applications to communicate with each other, enabling data exchange and functionality integration.

AWSRDS: A managed cloud service that simplifies the setup, operation, and scaling of relational databases like MySQL, providing automated backups, updates, and scalability.

Backend: The part of a website or application that is abstracted from users.

Bcrypt: A password-hashing function used to securely encrypt and store user passwords, ensuring protection against unauthorized access.

BMED: An abbreviation for Biomedical Engineering.

ER: Entity-Relationship

FERPA (Family Educational Rights and Privacy Act): A U.S. federal law that protects the privacy of student education records.

Frontend: The part of a website or application that users interact with.

HTTP: Hypertext Transfer Protocol

JSON: JavaScript Object Notation

MySQL: An open source relational database management system that uses SQL (Structured Query Language) to manipulate data.

Node.js: A JavaScript runtime environment that allows developers to build server-side applications capable of handling multiple requests asynchronously.

PII: Personally Identifiable Information

React: JavaScript libraries for building user interfaces.

SSD (System Sequence Diagram): A diagram that shows the sequence of interactions between an actor and system components during a specific scenario, detailing the flow of requests and responses at runtime.

UI (User Interface): A space where humans and the system or machine can interact.

Introduction

Background

Our team is developing a mobile application for the BMED Capstone class, where our overarching goal is to streamline the peer evaluation process. Our application will be able to deliver the functionality of real-time evaluation, where students can evaluate their peers while they present. Additionally, we've heavily prioritized the application's convenience and ease of use, as this has been found to be a major motivating factor for students to provide quality evaluations. The application is built with React Native for the frontend, MySQL for the backend database, and Node.js to support the backend framework. A few key features of our application include a learning management system-oriented layout, where students will have a dashboard showing all the courses that use live-time peer evaluation. Within each course, students will be able to view and complete their assignments – the interface here will focus on mobile usability, where students can make straightforward selections as opposed to filling out long-answer formatted questions. There are some teacher-exclusive features as well, including a view of the roster, assignment creation, and functionality for creating courses and sections.

Document Summary

The System Architecture section offers a high level overview regarding the static and dynamic interactions between our major application components, which include: the React Native front-end, the Node.js back-end, and the MySQL database which is hosted on AWS RDS.

The Data Storage Design section provides an overview of how we store user data within our MySQL relational database. We also discuss some of the security considerations and handling file usage.

The Component Detailed Design section provides a more detailed view of application components, focusing on the static relationship of elements within the application and their dynamic runtime interactions. In particular, the component detailed design is meant to provide much more specifics on what has been discussed and reinforce the ideas presented in regard to the system architecture.

The UI Design section demonstrates all primary screens that the user interacts with and describes the steps to perform certain actions.

System Architecture

Introduction

Goals and Objectives

The primary goal of this peer evaluation mobile application is to create a real-time, group-based system that allows students to assess each other's performance during presentations and track improvements over time. The system is compatible with both iOS and Android, ensuring accessibility to all students, and complies with FERPA regulations to protect the privacy of evaluation data such as ensuring all evaluations will only be viewable by instructors, and students will only be able to view their own feedback.

Key objectives include:

- Enabling students to provide confidential peer evaluations and receive feedback on their performance.
- Allowing instructors to manage student groups and view evaluation results in an easy-to-use interface.
- Providing re-evaluation functionality to allow students to track their improvement over multiple presentations.

Rationale

We have chosen a layered architecture for this system due to its modularity, scalability, and ease of development. A layered design allows for separation of concerns, where different parts of the system (frontend, backend, and database) can be developed and maintained independently. This is especially important as the project may be handed off to other engineers in the future, making it easier to update or replace individual layers without affecting the entire system.

The tech stack we selected includes:

- Frontend: React Native for cross-platform mobile compatibility on iOS and Android devices, ensuring students can use the app on any mobile device.
- Backend: Node.js with Express.js to handle real-time data processing, including evaluations and re-evaluations.
- Database: MySQL to store evaluation data, group assignments, and other relevant information securely.
- Security: Bcrypt for password encryption to protect user credentials and ensure secure authentication.

Security is a critical component of the architecture due to FERPA requirements. The system uses Bcrypt to encrypt passwords and ensures that evaluations remain confidential, accessible only to authorized students and instructors. By maintaining a modular structure, future changes to the

security mechanisms, API integrations, or evaluation logic can be handled at the layer level without disrupting the entire system.

The following sections provide both static (in Figure 1) and dynamic (in Figure 2) architectural diagrams to illustrate the system's structure and component interactions during runtime.

Figure 1 shows the static architecture, outlining the main system components: React Native Frontend, Node.js Backend, and MySQL Database on AWS RDS. It highlights the layered structure and the dependencies between these components, showing how the frontend relies on the backend for business logic, which in turn interacts with the database for data storage.

Figure 2 presents the dynamic architecture using a System Sequence Diagram (SSD). It details the runtime interactions during an evaluation submission, from the frontend request to the backend's processing and storing the data in the database. The diagram uses solid arrows for requests and dashed arrows for responses, showing how components communicate in real-time.

Together, these diagrams provide a complete view of the system, covering both its structural layout and runtime behavior.

Static System Architecture

Diagram

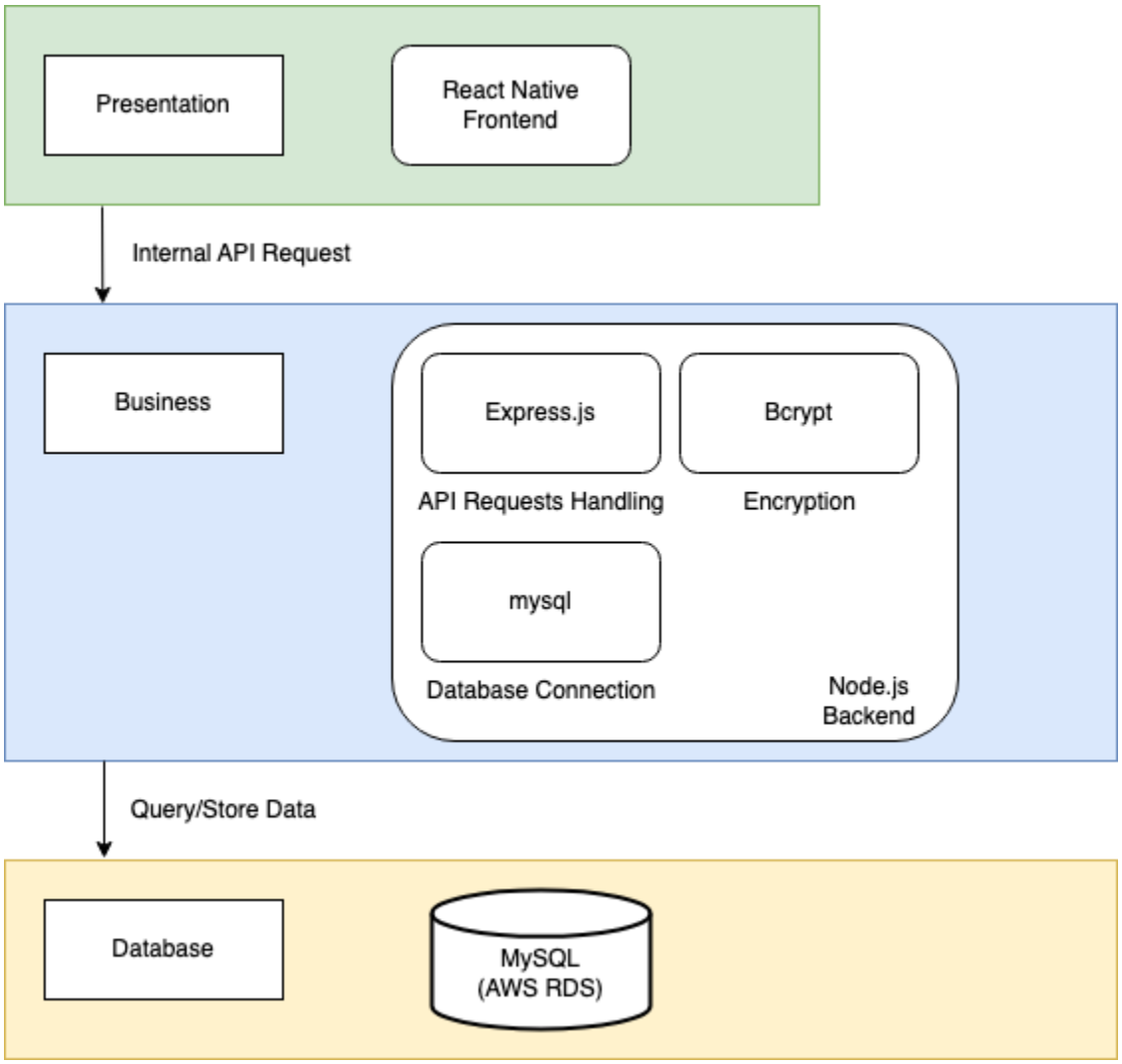

Figure 1 - Static System Diagram

Description

Our peer evaluation application consists of three main parts: the **React Native Frontend**, the **Node.js Backend**, and the **MySQL Database**. These components work together to provide a seamless user experience while ensuring security and data integrity.

The **React Native Frontend** forms the **presentation layer** of our system. This layer is built using React Native, a popular JavaScript framework, which allows us to create a cross-platform mobile application that works on both Android and iOS devices. This frontend handles all user interactions, such as submitting evaluations and viewing results. We chose React Native because it allows for rapid development and ensures compatibility across multiple mobile platforms.

The **Node.js Backend** serves as the **business layer** of the application. It uses Express.js to handle incoming API requests from the frontend and Bcrypt for encryption to securely manage user credentials. This backend processes the evaluations and connects to the database to store or retrieve data. We opted for Node.js due to its asynchronous nature, which makes it ideal for handling multiple simultaneous user requests efficiently.

The **MySQL Database**, hosted on **AWS RDS**, represents the **database layer** of our architecture. This is where all user data, evaluations, and group information are stored. Using AWS RDS allows us to scale our database as needed without worrying about maintenance. MySQL was chosen for its reliability and structured query capabilities, which suit our application's need for structured, relational data.

By employing a layered architecture, we ensure that each component operates independently, allowing for better modularity and future scalability. For instance, if our database technology needs to change in the future, it can be swapped out without affecting the frontend or backend logic. This clear separation of concerns was the main reason behind choosing this architecture.

Dynamic System Architecture

Diagram

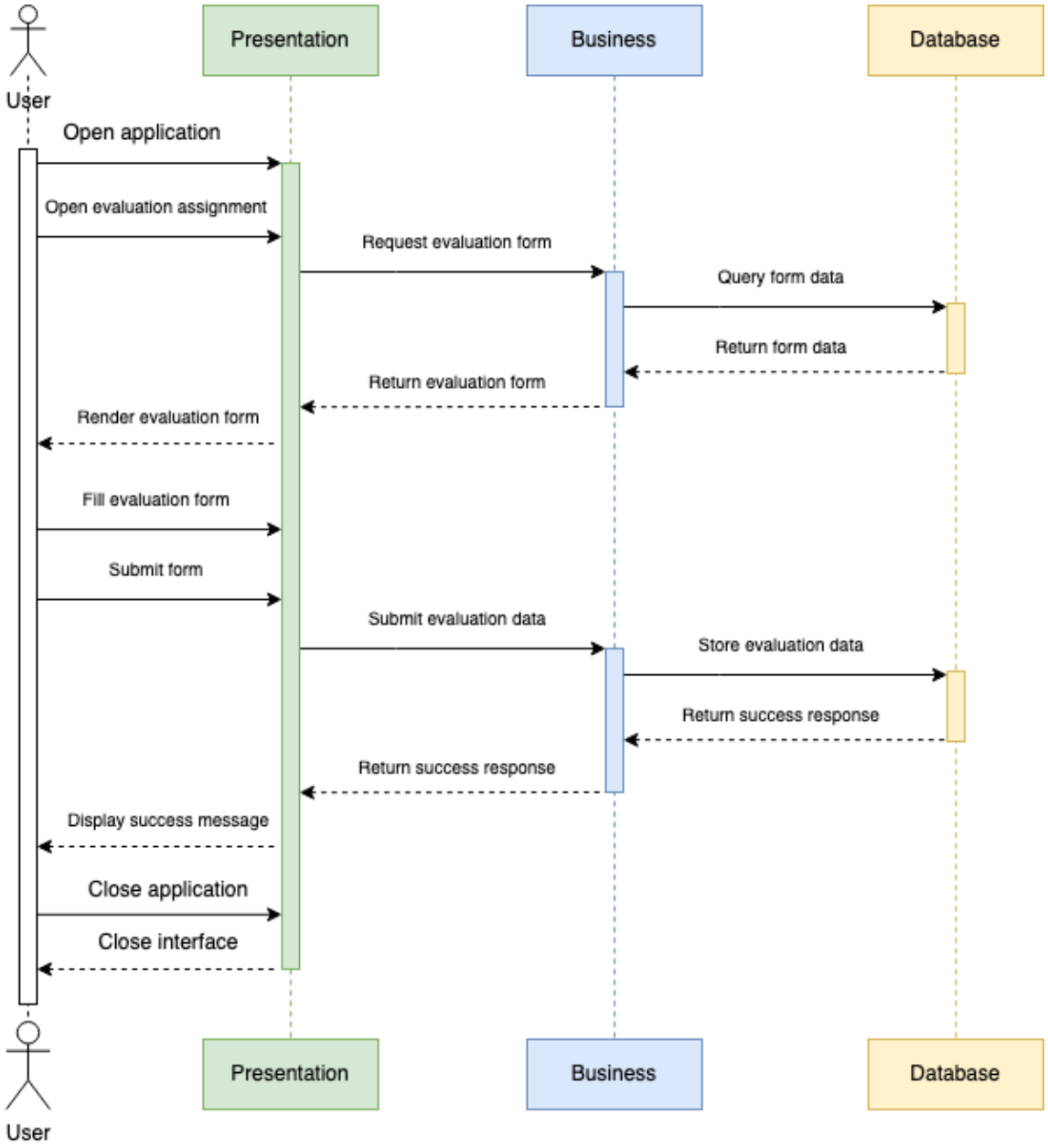

Figure 2 - Dynamic System Diagram to demonstrate evaluation form submission feature

Description

This dynamic architecture diagram illustrates the process of a student submitting an evaluation, including the fetching of the evaluation form from the database. We chose this feature because it involves all critical components of the system: the frontend, backend, and database. The scenario demonstrates the system's runtime behavior, focusing on how components interact when performing key operations such as fetching, submitting, and storing evaluation data.

The process begins when the user opens the application, and the **React Native Frontend** sends a request to the backend to fetch the evaluation form. This is crucial to ensure the user has the most up-to-date form data before submitting their evaluation. The **Node.js Backend** handles this request by querying the **MySQL Database** (hosted on AWS RDS) to retrieve the necessary form data. Once the form data is returned, the backend sends it back to the frontend, where it is rendered for the user to fill out.

After the user completes the form and submits their evaluation, the frontend sends the evaluation data to the backend. The **Node.js Backend** processes this data, including validation and encryption, before storing it in the database. The **MySQL Database** stores the evaluation data, and upon successful completion, returns a success response to the backend, which in turn sends a success message back to the frontend, informing the user that their submission was successful.

The **System Sequence Diagram (SSD)** was selected for this dynamic architecture because it effectively captures the step-by-step interaction between the system components during this specific runtime scenario. By using solid arrows for requests and dashed arrows for responses, we provide a clear visualization of the dependencies and communication between the components. Additionally, activation bars help illustrate when each component is actively processing a request or response.

This scenario was chosen because it encapsulates the core functionality of the system and demonstrates the interactions between key architectural components. It also showcases how data flows through the system, from the user interaction on the frontend to database operations on the backend. This SSD complements the static architecture by maintaining conceptual consistency and providing a detailed view of the system's behavior during runtime.

Component Detailed Design

Introduction

The following section showcases the structural and dynamic components of the peer evaluation system developed for Georgia Tech's BMED department. The design focuses on the system's core functionalities, including teacher management, course creation, and student evaluation, while maintaining conceptual integrity with the system architecture diagrams. To achieve this, a class diagram (Figure 3) is used to depict the static relationships and dependencies among system components, reflecting the layered architecture described earlier. The diagram aligns with the modular design principles, ensuring consistency in how system elements like users, courses, and evaluations are structured.

In addition, a sequence system diagram (SSD) (Figure 4) provides a detailed view of the dynamic interactions involved in key processes, such as publishing a course, and reinforces the system's runtime behavior as illustrated in the architecture diagrams. By maintaining this consistency between the conceptual architecture and the detailed system design, we ensure modularity, scalability, and clarity in both the static structure and the interactions between components at runtime. Together, these diagrams give a comprehensive understanding of how the system operates while adhering to the goals of layered modularity and security outlined in the system architecture.

Static Diagram

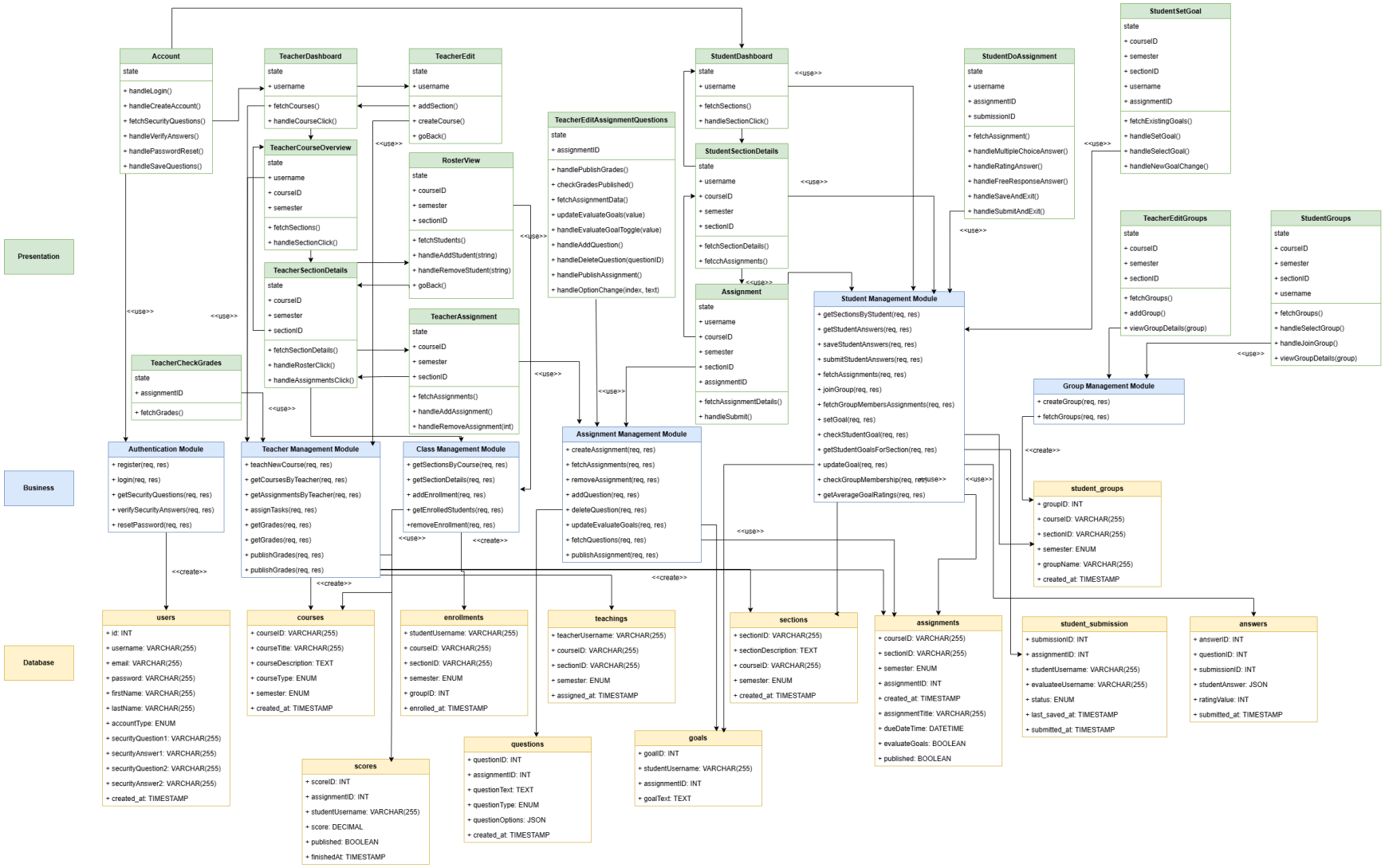

Figure 3 - Class Diagram

Description

The class diagram above provides a high-level overview of the system's architecture, with a focus on key components like account management, teacher dashboard, course management, and student evaluation modules. The design includes both frontend and backend components, such as **Account**, **TeacherDashboard**, and **TeacherManagementModule**, as well as database entities like **courses**, **sections**, and **teachings**. Each class is annotated with its state, attributes, and core methods, showing how they contribute to overall system functionality. For example, the **TeacherManagementModule** handles key operations like **teachNewCourse()** and **getCoursesByTeacher()**. Additionally, it interacts with the Database where tables like **courses** and **teachings** store relevant data, represented through associations between classes and their respective modules.

Dynamic

Diagram

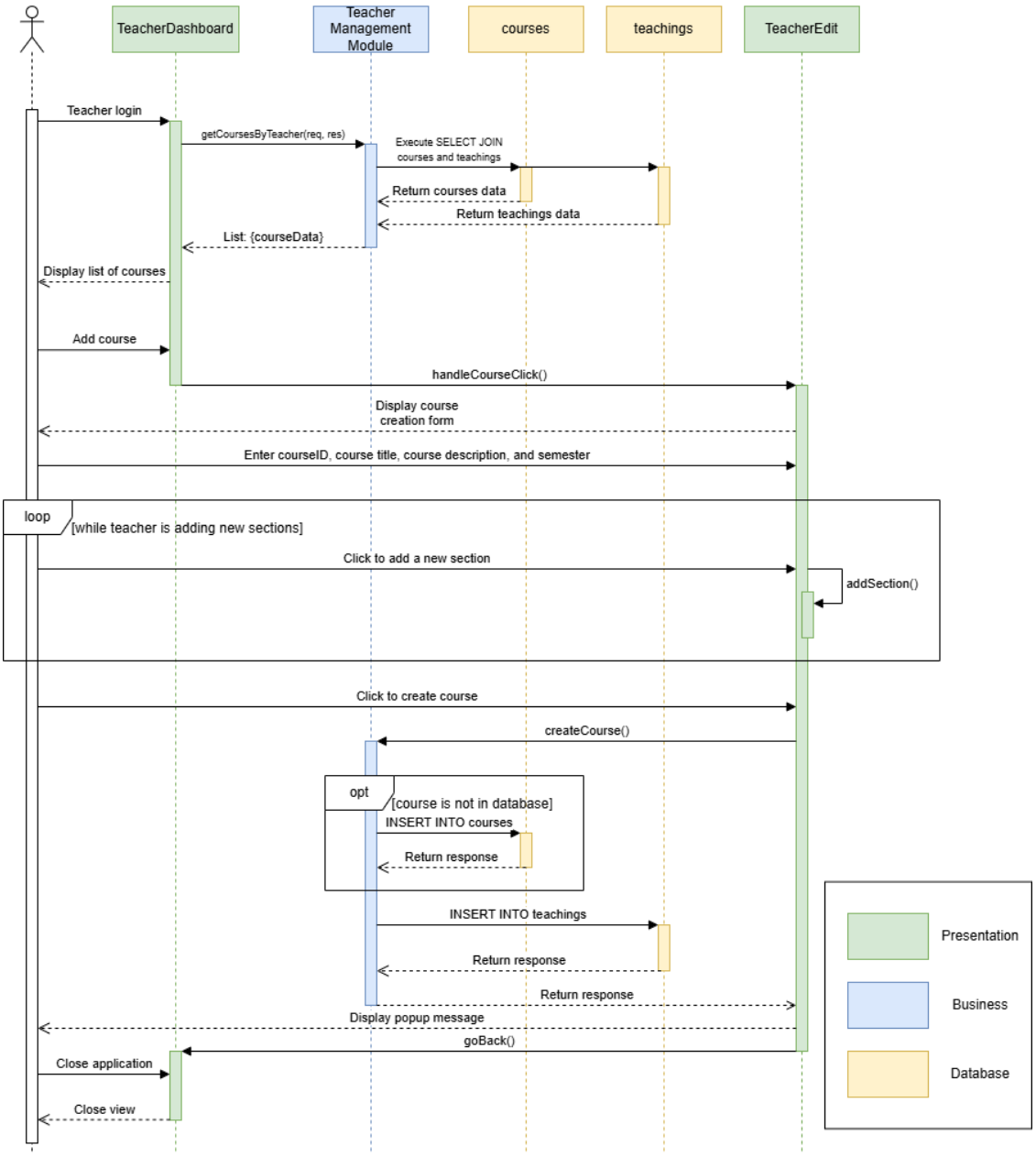

Figure 4 - SSD for creating courses and sections

Description

The sequence diagram illustrates the runtime behavior when a teacher publishes a course. The interaction begins with the teacher logging into the system and selecting an option to add a course, triggering the **createCourse()** method in the **TeacherManagementModule**. This action results in a **SELECT JOIN** query being executed across the **courses** and **teachings** tables, fetching the relevant data. After displaying the course creation form, the teacher inputs course details, and the system proceeds to handle adding new sections through a loop. The course is then stored in the database, with conditional logic checking if the course already exists, followed by inserting the course and section data into the **teachings** table. Upon completion, a success message is displayed to the teacher, and the sequence concludes by closing the application interface.

Data Design

Introduction

In Figure 5, we present an Entity-Relationship (ER) diagram that reflects the structure of our mobile peer evaluation application. This diagram details entities such as courses, users, and assignments, along with their attributes and relationships. The data is stored using MySQL, a relational database, which ensures efficient storage and retrieval of information. Following the diagram, we discuss how data is exchanged between the frontend and backend, file usage, and the security measures we've implemented to protect the data.

Diagram

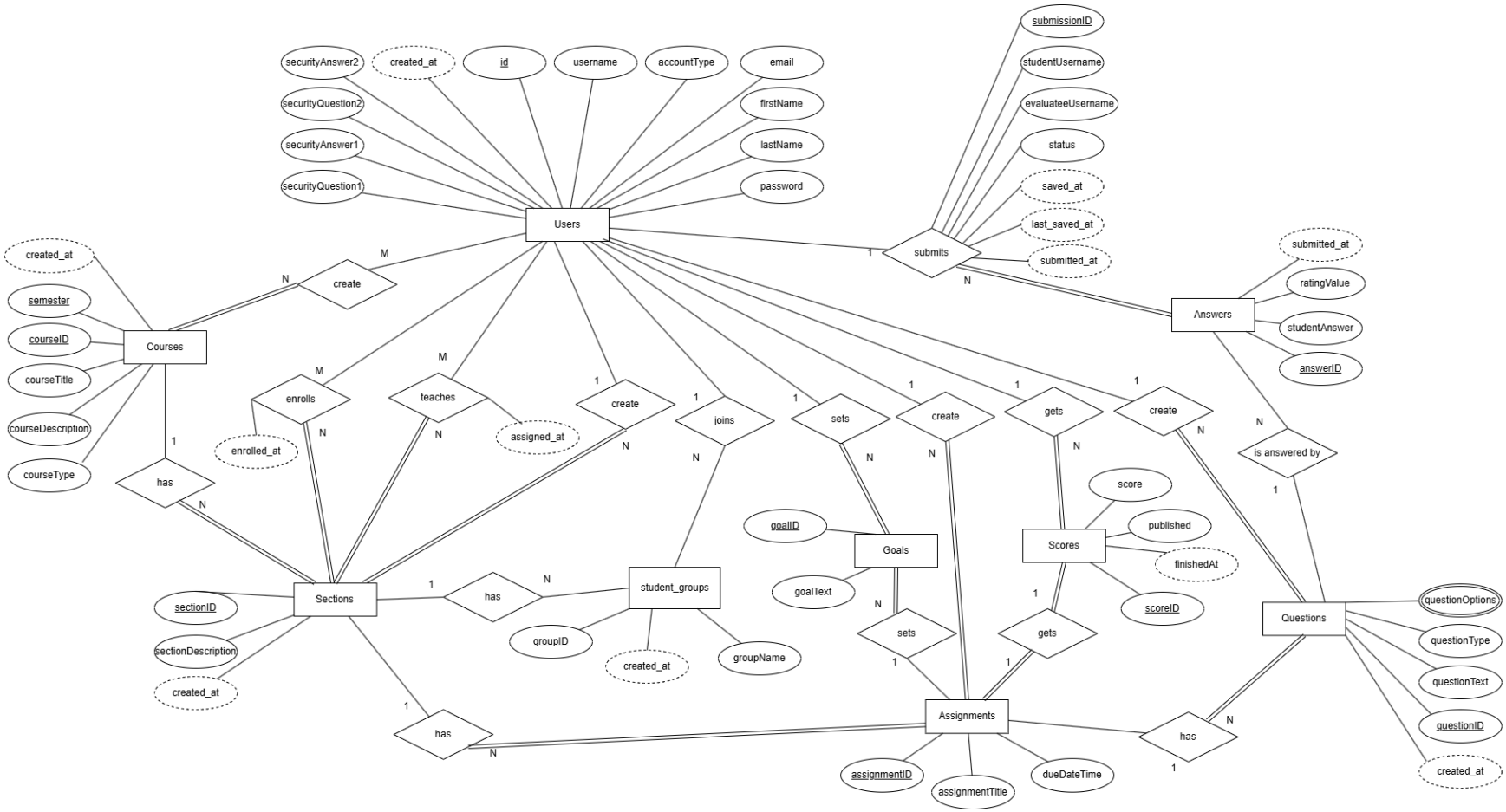

Figure 5 – Entity Relationship Diagram

File Use

Our system is purely based on text and does not rely on external files for input or output, as all data is directly handled through the server and stored in the database. Therefore, no file formats are utilized.

Data Exchange

Our system uses JSON (JavaScript Object Notation) format for data transfer between the frontend (React Native) and backend (Node.js with Express). For instance, student submissions and answers are transferred in JSON format. The data is transferred over HTTPS (HyperText Transfer Protocol) for security, ensuring the integrity and confidentiality of the data.

Data Security

User Login & Data Encryption: Data is encrypted at rest using the database's native encryption capabilities. Passwords are not stored in plaintext but instead stored in hashed form using Bcrypt.
Secure Channels: All data exchange between the client and server happens over HTTPS, ensuring the data transmitted between the front-end and the server is encrypted and cannot be intercepted.
Sensitive Information: The system only stores email as the PII (Personally Identifiable Information) which is protected by MySQL's native encryption capabilities.

User Interface Design

Introduction

This user interface (UI) design aims to create an intuitive and visually unified experience for the course management application, primarily for student and instructor users. The main goals of the UI are to provide easy navigation, effective assignment management, and efficient peer assessment functionality. The design prioritizes simplicity and usability so that users can focus on their tasks without unnecessary distractions: this will be critical for students especially as their peers are presenting themselves in a live environment, and teachers as they will be able to focus on constructing a straightforward assignment. Below is a description of the main UI components and their intended functionality.

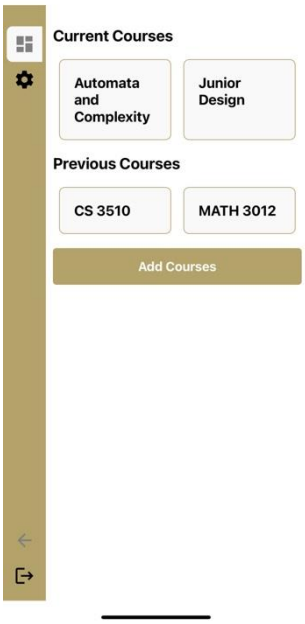

Figure 6 – Teacher's Dashboard

Figure 6 is the course management page for teacher users. The ‘Current courses’ and ‘Completed courses’ sections display the courses the teacher is teaching and has completed. Each course is displayed as a box, and the teacher can click to go to the section choose page and then course details page. The ‘Add course’ button at the bottom of the page allows the teacher to add new courses to expand the course list. At the left of the screen there are four buttons, from top to bottom: ‘Quick Return to Dashboard’, ‘Settings’, ‘Back’, and ‘Logout’. They appear in the same position on the left side of the screen on many pages to help users quickly jump to the page they want to go to.

Figure 7 – Teacher's Course Details

Figure 7 is the course details page for teacher users. Users can quickly view the course ID, section ID and semester information to get a better overview of the course. At the bottom of the page, there are three function buttons: ‘Student List’, ‘Groups’ and ‘Assignments’. Clicking on ‘Student List’ allows the user to view the list of students in the course; the ‘Groups’ function helps the user manage the student groups in the course; and the ‘Assignments’ button allows the teacher to view and edit assignments, so that they can manage the course more effectively.

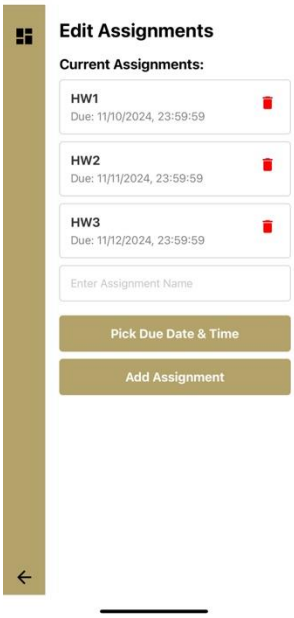

Figure 8 – Teacher’s Assignment Overall

Figure 8 is the Edit Assignments page for teacher users, allowing the user to view and manage the current assignment list. Each assignment entry displays the assignment name and due date, and the red trash can icon on the right can be used to delete the assignment. The input box at the bottom allows the user to enter a new assignment name, click the ‘Select due date and time’ button to set the due date, and then click ‘Add assignment’ to add the new assignment to the list.

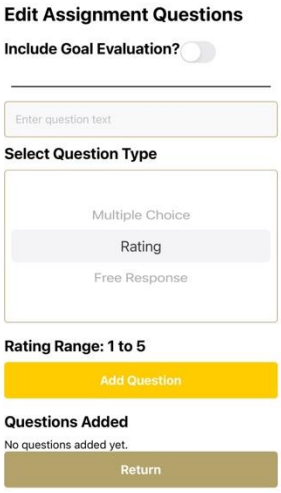

Figure 9 – Creating Assignment Details

Figure 9 is the Edit Assignment Question page for teacher users. Allowing the user to add questions to the assignment. The ‘Include goal evaluation’ switch at the top allows the user to choose whether to include a goal evaluation question. If the user selects ‘include goal evaluation’, this assignment will automatically generate goal evaluation questions for all groups and group members when it is assigned to students. The user can enter the question content in the text box and select the question type (multiple choice, graded or free response) and other settings. Click the ‘Add question’ button to add the question to the assignment. The added question will be displayed in the ‘Questions Added’ section below, and the user can click the Return button to return to the previous screen.

Figure 10 – Student's Assignment

Figure 10 is the course detail page for student users. On this page, users can view the list of assignments, including the assignment name, deadline and completion status. At the bottom of the page, there are two buttons: ‘My semester goals’ allows user to view and set user’s semester goals, and ‘My groups’ is used to view information about the groups the user is in.

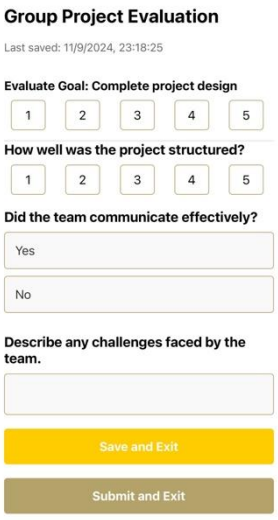

Figure 11 – Student's Assignment Detail

Figure 11 is the assignment page for student users. Users can evaluate the other student in detail. If this assignment has been set to include a goal evaluation question, an evaluation question

related to this student's goal will be shown on the top. The questions set by the teacher user are displayed below the evaluation goal question. The student user can answer these questions on this page. The 'Save and exit' button at the bottom of the page allows students to save their current progress and exit, while the 'Submit and exit' button is used to submit the evaluation results and exit the page.