CEBU INSTITUTE OF TECHNOLOGY UNIVERSITY

COLLEGE OF COMPUTER STUDIES

Software Design Description

for

CyberKids

Signature

Change History

Version	Date	Amendment	Author
1.0	3/21/2025	Initial	Baguio Cultura Dedumo Vequiso
1.1	3/24/2025	Changes of backend structure on each modules	Artezuela Baguio Cultura Dedumo Vequiso
2.0	5/18/2025	Finalized	Cultura

Preface

Table of Contents

1. Introduction

1.1. Purpose

The purpose of this Software Design Document (SDD) is to provide a detailed architectural and technical description of CyberKids, a gamified cybersecurity learning platform developed using Roblox. This document serves as a reference for developers, designers, and stakeholders by outlining the system's design structure, component interactions, and implementation approach. It ensures that all aspects of the system's architecture, modules, and data flow are well-documented to support development, maintenance, and future enhancements.

1.2. Scope

CyberKids aims to enhance cybersecurity awareness among students by integrating interactive learning modules within a gamified environment. The game consists of three levels, each focusing on a different cybersecurity concept:

- Level 1: Online Privacy and Safety – Teaches students how to classify information as personal or private.
- Level 2: Password Security – Helps students build strong passwords using a collection-based mini-game.
- Level 3: Phishing and Scam Awareness – Trains students to identify phishing attempts in a simulated environment.

Key features of CyberKids include:

- Gamified Challenges – Interactive tasks that test students' knowledge in a fun and engaging manner.
- Point-Based Scoring System – Rewards players based on accuracy, speed, and

completion of challenges.

- Leaderboard System – Ranks students to encourage competition and reinforce learning.
- Teacher Dashboard – Provides real-time tracking of student performance and progress reports.

The system is designed to align with the basic security practices in computer subject curriculum for Grade 5 and 6 students under Teacher Rosario, ensuring that in-game activities complement classroom learning.

1.3. Definitions and Acronyms

This section provides definitions of acronyms relevant to the implementation of the CyberKids system.

Acronyms:

Acronyms used in this document shall be interpreted as follows:

- SDD– Software Design Description
- SRS – Software Requirements Specification
- UI– User Interface
- UX – User Experience
- API– Application Programming Interface
- DBMS – Database Management System
- OOP – Object-Oriented Programming

Definitions:

Definitions used in this document shall be interpreted as follows:

Term	Definition
SDD	A document that provides a detailed outline of the software architecture, design decisions, and system components.
SRS	A document that defines the functional and non-functional requirements of the software system.
UI	The visual and interactive elements of the software that allow users to interact with the system.
UX	The overall experience and satisfaction of a user while interacting with the software.
API	A set of rules and protocols that allow different software applications to communicate with each other.
DBMS	A system software used to store, manage, and retrieve data efficiently.
OOP	A programming paradigm that organizes code using objects and classes to promote modularity and reusability.

1.4. References

[1] IEEE Computer Society. (1998). IEEE 830-1998: Recommended Practice for Software Requirements Specifications. IEEE Std 830-1998. Source: IEEE Xplore Digital Library.

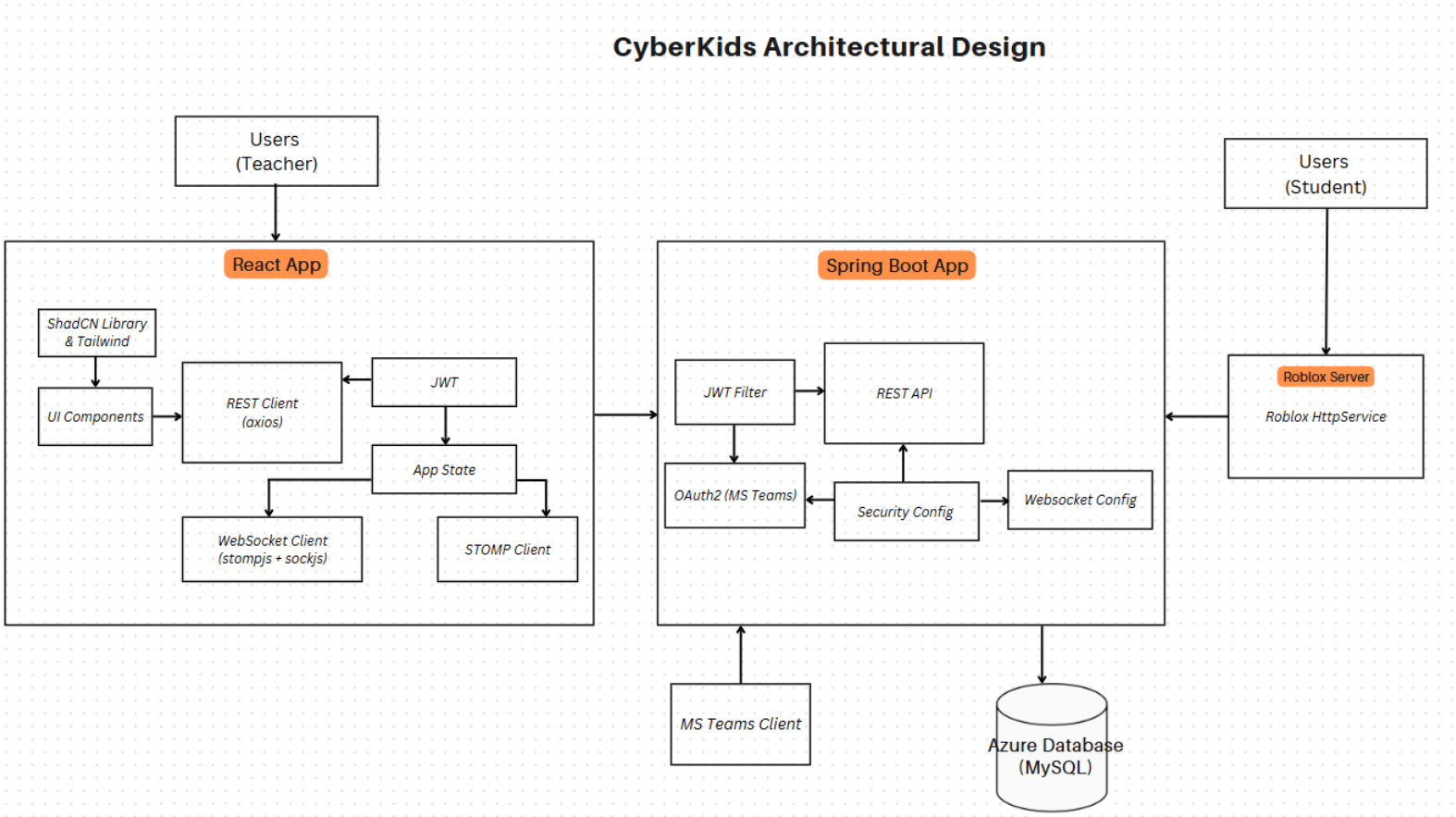
[2] Roblox Corporation. (n.d.). Roblox API Reference. Source: https://create.roblox.com/docs/reference/engine

[3] CyberKids Development Team. (2024). CyberKids Software Requirements Specification (SRS). Internal Document: https://drive.google.com/file/d/1DVdZoIfw4lSzuuthGMLdDelG31R9JWIC/view?usp=drive_link

[4] CyberKids Development Team. (2024). CyberKids Software Engineering Proposal. Internal Document: https://drive.google.com/file/d/1hmgfx1ll3aQNt5iv70arY-bPiUWdCU1-/view?usp=drive_link

2. Architectural Design

Block diagram showing all components of the system including frameworks, libraries, APIs and other technologies used in the front and back ends.

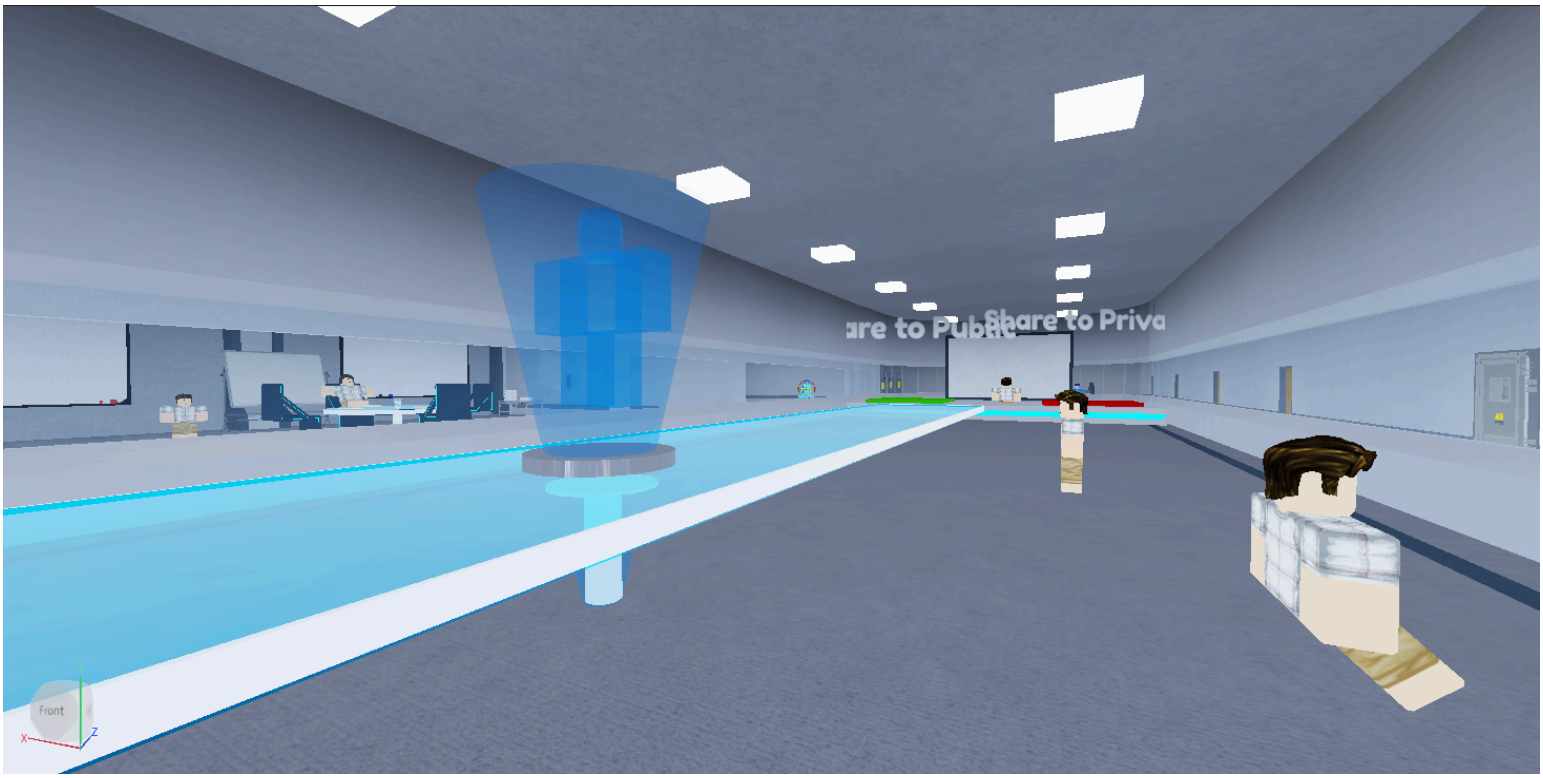


3. Detailed Design

Module 1: Gamified Cybersecurity Game Learning Module

Transaction 1.1 Information Classification Sorting Challenge

User Interface Design



Front-end component(s)

1. **Workspace Objects**
   - ➢ **Description and Purpose:**

     Static and interactive 3D models placed in the game world. These may include environmental assets, game elements, and props used during gameplay or cutscenes
   - ➢ **Component Type:**

     Roblox Workspace (Models, Parts, Meshes, etc.).
2. **Cameras**
   - ➢ **Description and Purpose:**

     Custom camera sequences used to show intro cutscenes or adjust player viewpoint during game start or end. Enhances player immersion and guides focus.
   - ➢ **Component Type:**

     Roblox Camera objects, manipulated by LocalScripts.
3. **Sounds / Music**
   - ➢ **Description and Purpose:**

     Background music and sound effects that play during game start, cutscenes, or transitions. Helps set tone and atmosphere.
   - ➢ **Component Type:**

     Roblox Sound instances in SoundService, Workspace, or StarterGui.
4. **Cutscenes**
   - ➢ **Description and Purpose:**

     Pre-scripted camera and animation sequences triggered at the beginning of the game to introduce the story, environment, or objectives.
   - ➢ **Component Type:**

     LocalScript controlling camera movement, animations, and UI (usually uses TweenService and Camera manipulation).

5. **GameStarted & GameEnded (RemoteEvents)**
   - ➢ **Description and Purpose:**
     - GameStarted: Tells all clients that the game session has officially begun, triggering cutscenes, UI changes, and audio.
     - GameEnded: Signals that the game session is over, allowing the client to display final scores or restart options.
   - ➢ **Component Type:**

     Roblox RemoteEvent objects in ReplicatedStorage.
6. **RegisterAPI (HTTP Module)**
   - ➢ **Description and Purpose:**

     Sends the player's Roblox UserId to your backend database to associate scores and progress with a unique user, avoiding the need for manual account creation.
   - ➢ **Component Type:**

     ModuleScript or Script using HttpService:PostAsync().
7. **HandleGameStart (Client Script)**
   - ➢ **Description and Purpose:**

     Runs on the client when the GameStarted event is fired. Starts the cutscene, enables UI elements, and plays music or animations.
   - ➢ **Component Type:**

     Roblox LocalScript.
8. **HandleGameEnd (Client Script)**
   - ➢ **Description and Purpose:**

     Listens for the GameEnded event. Displays the game over screen, final score, and disables ongoing gameplay elements or input.
   - ➢ **Component Type:**

     Roblox LocalScript.

Back-end component(s)

1. **Student Entity**
   - ➢ **Description and Purpose:**

     Represents the database table for students/players. Stores fields like id, robloxId, and name to uniquely identify and track users across gameplay sessions.
   - ➢ **Component Type:**

     Java Class annotated with @Entity (typically stored in Entity package).
2. **StudentRequest DTO (Data Transfer Object)**
   - ➢ **Description and Purpose:**

     Used to receive data from the client (such as robloxId and optionally name) when registering a new student/player. Separates input data from the full entity.
   - ➢ **Component Type:**

     Java POJO (Plain Old Java Object) used in HTTP requests.
3. **StudentController**
   - ➢ **Description and Purpose:**

     Handles incoming HTTP requests from the Roblox game client for student-related operations (e.g., registering a new player). Routes requests to the appropriate service methods.
   - ➢ **Component Type:**

     Java Class annotated with @RestController.

4. **StudentService**
   - ➢ **Description and Purpose:**

     Contains business logic for creating, retrieving, and deleting student records. Abstracts logic away from the controller to maintain clean code and separation of concerns.
   - ➢ **Component Type:**

     Java Class annotated with @Service.
5. **StudentRepo**
   - ➢ **Description and Purpose:**

     Communicates directly with the database to perform CRUD operations. Uses Spring Data JPA to auto-generate SQL logic for Student entities.
   - ➢ **Component Type:**

     Java Interface extending JpaRepository<Student, Long>, annotated with @Repository.
6. **Timer Entity**
   - ➢ **Description and Purpose:**

     Stores timing information (e.g., start and end times, duration) for each student challenge session.
   - ➢ **Component Type:**

     Java class annotated with `@Entity`, located in the `Entity` package.
7. **TimerRepo**
   - ➢ **Description and Purpose:**

     Provides data access for Timer entities.
   - ➢ **Component Type:**

     Java interface extending `JpaRepository<Timer, Long>`, located in the `Repository` package.

8. **TimerService**
    - ➢ **Description and Purpose:**

      Handles logic for updating and retrieving timer records per challenge or student.
    - ➢ **Component Type:**

      Java class annotated with @Service, located in the Service package.
9. **TimerController**
    - ➢ **Description and Purpose:**

      Receives requests from the Roblox client to log or retrieve challenge timing data.
    - ➢ **Component Type:**

      Java class annotated with @RestController, located in the Controller package.
10. **Score Entity**
    - ➢ **Description and Purpose:**

      Represents points earned by students per challenge, linked to their account and used for leaderboard calculations.
    - ➢ **Component Type:**

      Java class annotated with @Entity, located in the Entity package.
11. **ScoreRepo**
    - ➢ **Description and Purpose:**

      Performs database operations for Score records.
    - ➢ **Component Type:**

      Java interface extending JpaRepository<Score, Long>, located in the Repository package.
12. **ScoreService**
    - ➢ **Description and Purpose:**

      Contains logic for saving and retrieving score data per student.
    - ➢ **Component Type:**

      Java class annotated with @Service, located in the Service package.

13. ScoreController

- **Description and Purpose:**

  Handles HTTP requests to log new scores or retrieve existing ones. Automatically registers new students if not found.

- **Component Type:**

  Java class annotated with @RestController, located in the Controller package.

14. InfoSortingLeaderboardEntry Entity

- **Description and Purpose:**

  Represents a leaderboard entry for the Information Classification Sorting challenge. Stores total score and total time taken.

- **Component Type:**

  Java class annotated with @Entity, located in the LeaderboardInfoSort.Global package.

15. InfoSortingLeaderboard Repo

- **Description and Purpose:**

  Handles database operations for leaderboard entries specific to the Information Classification Sorting challenge.

- **Component Type:**

  Java interface extending JpaRepository<InfoSortingLeaderboardEntry, Long>, located in the LeaderboardInfoSort.Global package.

16. InfoSortingLeaderboard Service

- **Description and Purpose:**

  Calculates and updates leaderboard totals for each student by aggregating relevant scores and times.

- **Component Type:**

  Java class annotated with @Service, located in the LeaderboardInfoSort.Global package.

17. InfoSortingLeaderboard Controller

- **Description and Purpose:**

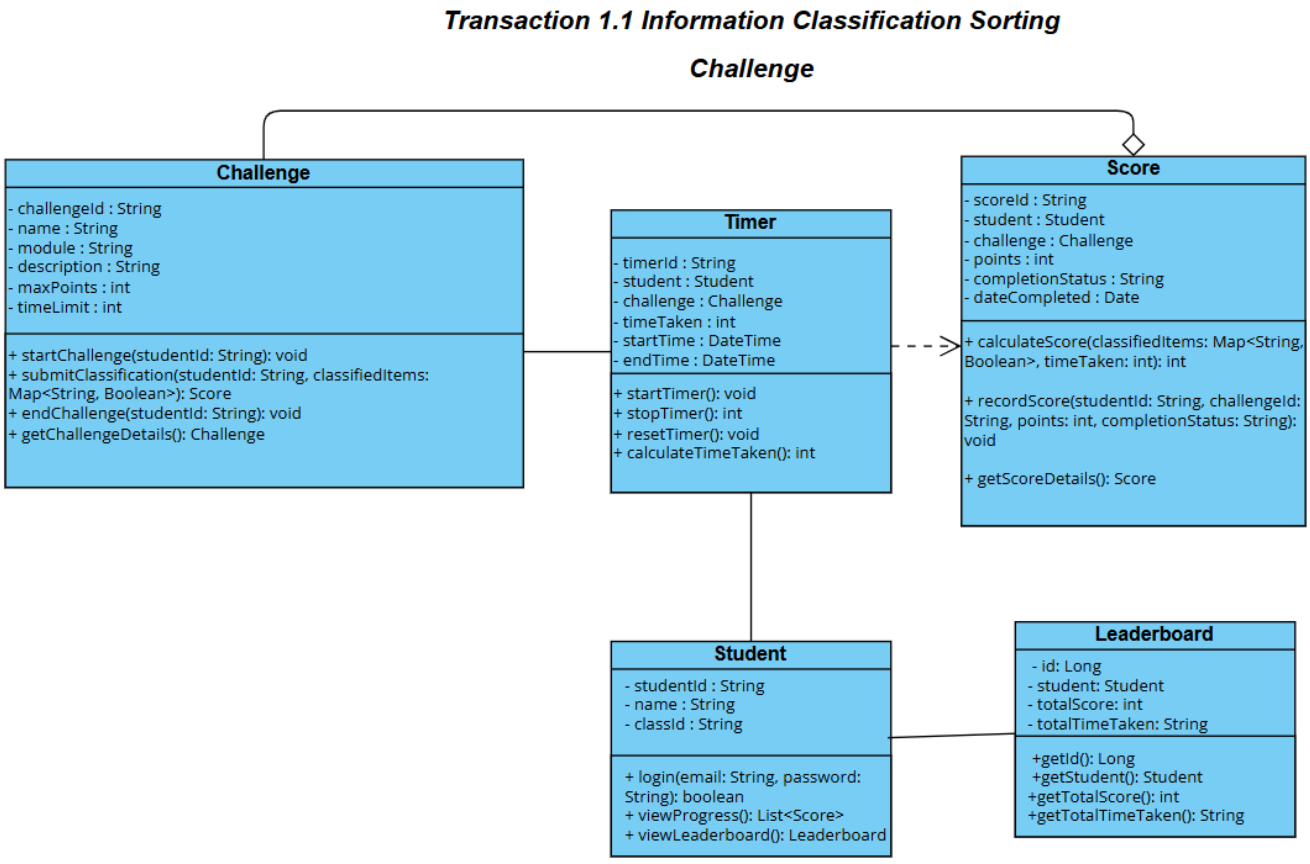
  Receives API requests to trigger leaderboard updates for specific students by robloxId.

- **Component Type:**

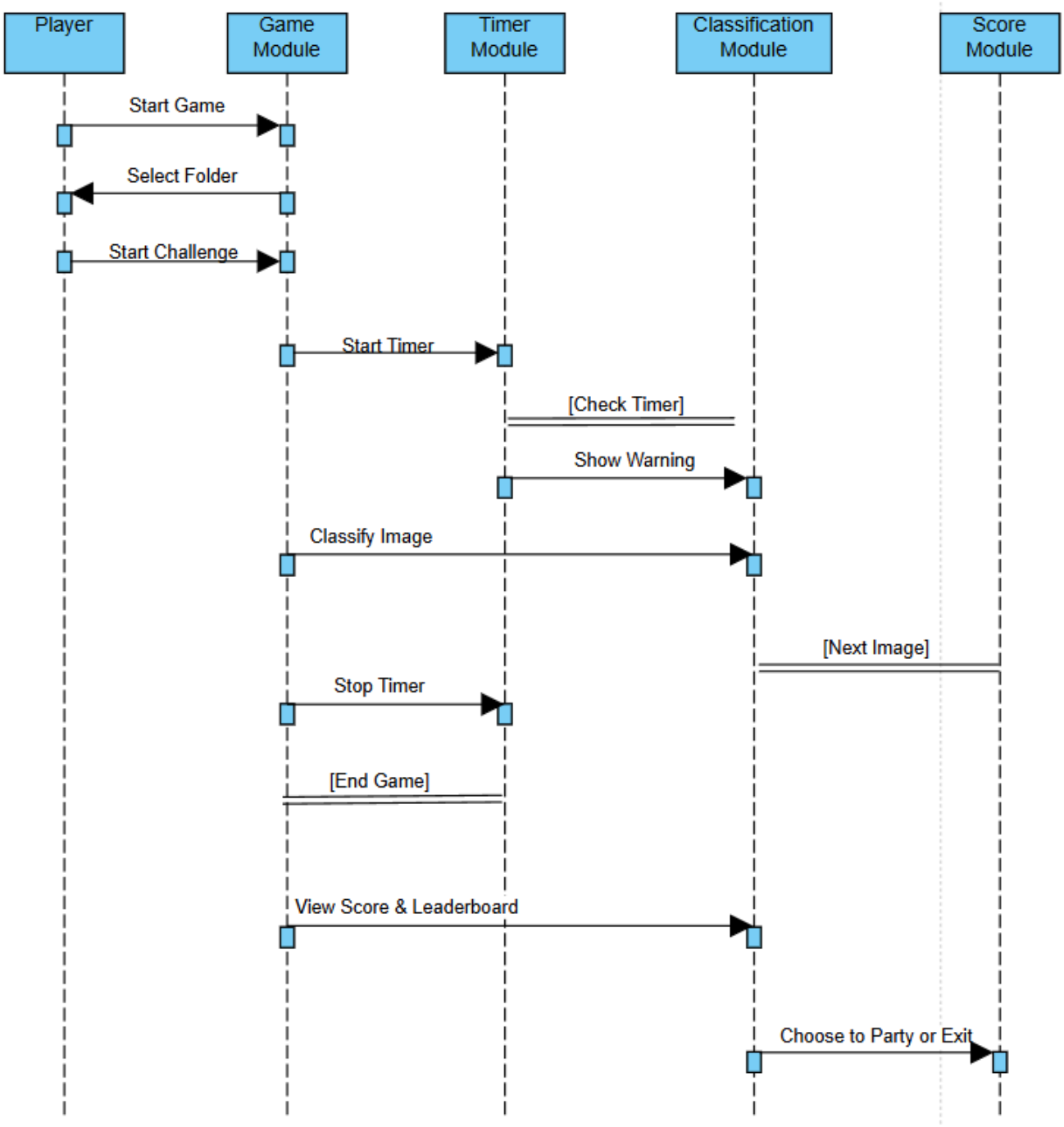
  Java class annotated with @RestController, located in the LeaderboardInfoSort.Global package.

Object-Oriented Components

- Class Diagram

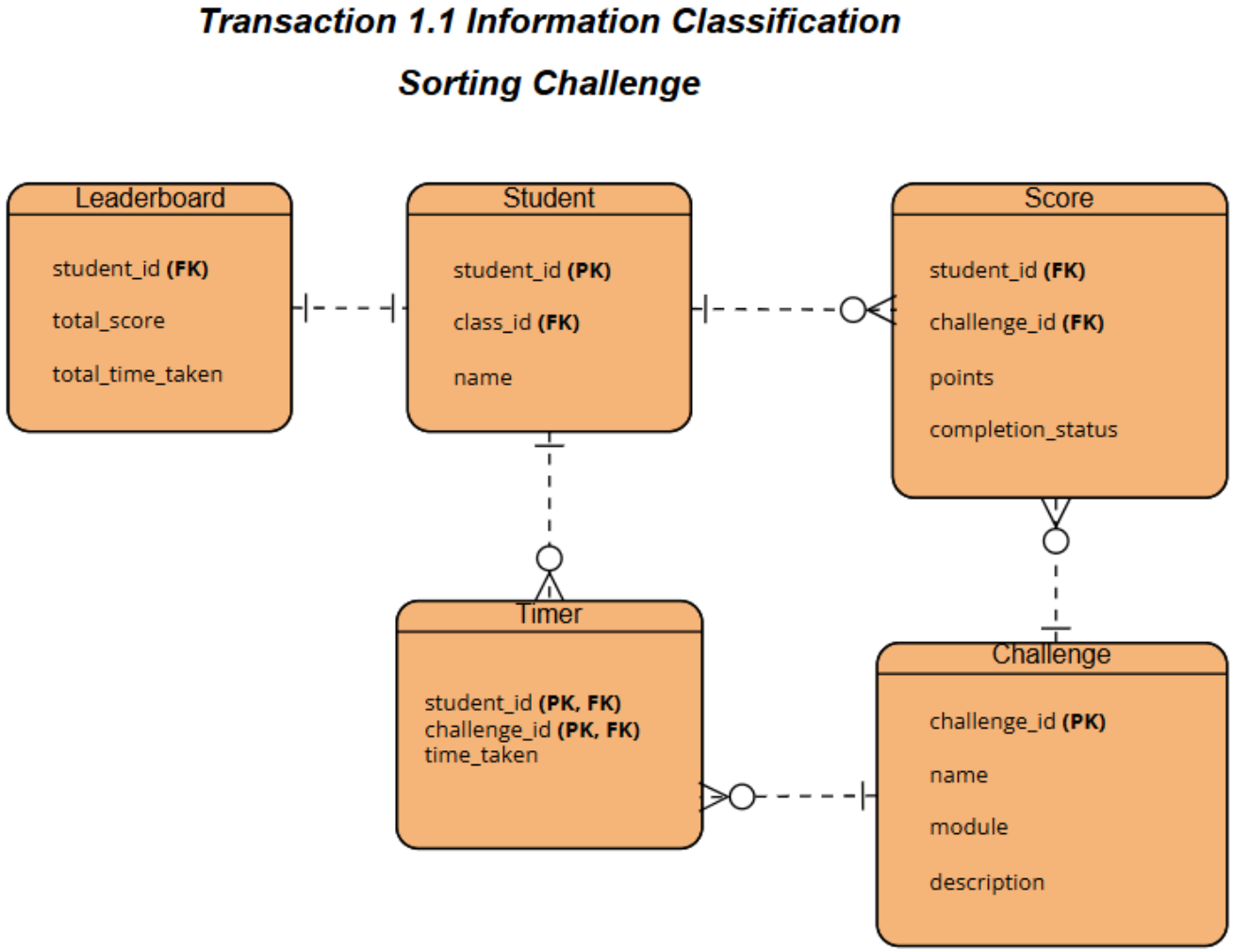


- Sequence Diagram



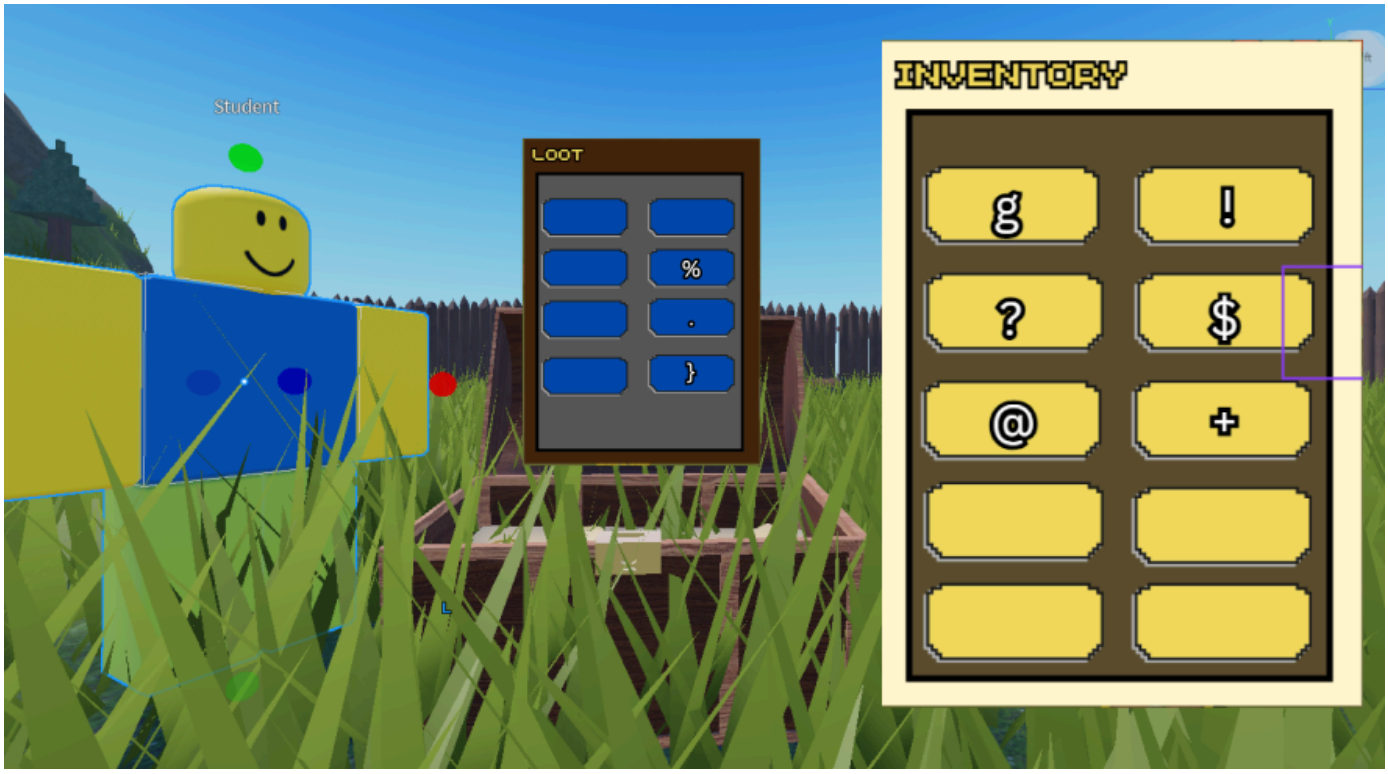
Data Design

- ERD or schema



Transaction 1.2 Password Security Challenge

User Interface Design



Front-end component(s)

1. **ObjectivesGui**
   - ➢ **Description and purpose:** Displays dynamic objectives that guide the player through each stage of the challenge, updating as the player progresses through looting, crafting, and defending. Helps maintain game flow and player clarity.
   - ➢ **Component type:** ScreenGui with TextLabels and LocalScripts in StarterGui.
2. **ChestGui and ChestClient**
   - ➢ **Description and purpose:** ChestGui shows the user interface for looting password fragments. ChestClient handles interactions when a player opens a chest, triggering animations, cutscenes, and updating inventory data.
   - ➢ **Component type:** ScreenGui with ImageButtons and LocalScripts under StarterGui; connected to RemoteEvents.
3. **InventoryGui and InventoryData**
   - ➢ **Description and purpose:** Displays collected password fragments in a visual interface for player review. InventoryData (a ModuleScript or Folder in ReplicatedStorage) stores these items persistently during gameplay.
   - ➢ **Component type:** ScreenGui with scrolling lists and LocalScripts; inventory storage in ReplicatedStorage.
4. **BlacksmithGUI and BlacksmithClient**
   - ➢ **Description and purpose:** Provides a crafting interface where players merge password fragments into one complete password. Includes slots, drag-and-drop features, and validation logic for crafting attempts.
   - ➢ **Component type:** ScreenGui with crafting frames and LocalScripts for merging logic and server communication.
5. **GateGui and GateClient**
   - ➢ **Description and purpose:** UI used for password submission. Players input their crafted password, which is sent to the server for validation. Successful passwords enhance the gate's durability.
   - ➢ **Component type:** ScreenGui with input fields and buttons, controlled by LocalScripts.

6. **GateServer**
   - ➢ **Description and purpose:** Processes submitted passwords to determine gate strength. Assigns appropriate HP and armor values to the gate model based on password complexity.
   - ➢ **Component type:** ServerScript in ServerScriptService.
7. **Zombies and Attack Script**
   - ➢ **Description and purpose:** Enemy NPCs that spawn and attack the player's gate. Their success depends on the strength of the gate, teaching players the consequences of weak passwords.
   - ➢ **Component type:** NPC Models in Workspace with attack logic handled by Script or ServerScript.
8. **ChestSpawner**
   - ➢ **Description and purpose:** Spawns chests at specific locations within the map. Used to control when and where players can access password fragments.
   - ➢ **Component type:** ServerScript in ServerScriptService.
9. **ChestCutscene and ZombieCutscene**
   - ➢ **Description and purpose:** Plays immersive camera sequences when the player opens a chest or encounters zombies. Enhances storytelling and gameplay tension.
   - ➢ **Component type:** LocalScripts using TweenService and Camera manipulation.
10. **Timer GUI**
    - ➢ **Description and purpose:** Displays a countdown that limits the time players have to complete the challenge. Adds urgency and game pacing.
    - ➢ **Component type:** ScreenGui with TextLabel and LocalScript timer logic.
11. **Score System**
    - ➢ **Description and purpose:** Tracks and updates the player's points based on fragment collection, password strength, and gate survival. Displayed at the end of the challenge.
    - ➢ **Component type:** LocalScript calculating score and optionally communicating with the backend.
12. **LeaderboardDisplay**
    - ➢ **Description and purpose:** Presents top scores and fastest times from all players. Promotes competition and encourages replayability.
    - ➢ **Component type:** ScreenGui or BillboardGui fetching data via HttpService or RemoteFunctions.

Back-end component(s)

1. **Student Entity**
   - ➢ **Description and Purpose:**

     Represents the database table for students/players. Stores fields like id, robloxId, and name to uniquely identify and track users across gameplay sessions.
   - ➢ **Component Type:**

     Java Class annotated with @Entity (typically stored in Entity package).
2. **StudentRequest DTO (Data Transfer Object)**
   - ➢ **Description and Purpose:**

     Used to receive data from the client (such as robloxId and optionally name) when registering a new student/player. Separates input data from the full entity.
   - ➢ **Component Type:**

     Java POJO (Plain Old Java Object) used in HTTP requests.
3. **StudentController**
   - ➢ **Description and Purpose:**

     Handles incoming HTTP requests from the Roblox game client for student-related operations (e.g., registering a new player). Routes requests to the appropriate service methods.
   - ➢ **Component Type:**

     Java Class annotated with @RestController.
4. **StudentService**
   - ➢ **Description and Purpose:**

     Contains business logic for creating, retrieving, and deleting student records. Abstracts logic away from the controller to maintain clean code and separation of concerns.
   - ➢ **Component Type:**

     Java Class annotated with @Service.

5. **StudentRepo**
   - ➢ **Description and Purpose:**

     Communicates directly with the database to perform CRUD operations. Uses Spring Data JPA to auto-generate SQL logic for Student entities.
   - ➢ **Component Type:**

     Java Interface extending JpaRepository<Student, Long>, annotated with @Repository.
6. **Timer Entity**
   - ➢ **Description and Purpose:**

     Stores timing information (e.g., start and end times, duration) for each student challenge session.
   - ➢ **Component Type:**

     Java class annotated with `@Entity`, located in the `Entity` package.
7. **TimerRepo**
   - ➢ **Description and Purpose:**

     Provides data access for Timer entities.
   - ➢ **Component Type:**

     Java interface extending `JpaRepository<Timer, Long>`, located in the `Repository` package.
8. **TimerService**
   - ➢ **Description and Purpose:**

     Handles logic for updating and retrieving timer records per challenge or student.
   - ➢ **Component Type:**

     Java class annotated with @Service, located in the Service package.
9. **TimerController**
   - ➢ **Description and Purpose:**

     Receives requests from the Roblox client to log or retrieve challenge timing data.
   - ➢ **Component Type:**

     Java class annotated with @RestController, located in the Controller package.

10. Score Entity

- **Description and Purpose:**

  Represents points earned by students per challenge, linked to their account and used for leaderboard calculations.

- **Component Type:**

  Java class annotated with @Entity, located in the Entity package.

11. ScoreRepo

- **Description and Purpose:**

  Performs database operations for Score records.

- **Component Type:**

  Java interface extending JpaRepository<Score, Long>, located in the Repository package.

12. ScoreService

- **Description and Purpose:**

  Contains logic for saving and retrieving score data per student.

- **Component Type:**

  Java class annotated with @Service, located in the Service package.

13. ScoreController

- **Description and Purpose:**

  Handles HTTP requests to log new scores or retrieve existing ones. Automatically registers new students if not found.

- **Component Type:**

  Java class annotated with @RestController, located in the Controller package.

14. InfoSortingLeaderboardEntry Entity

- **Description and Purpose:**

  Represents a leaderboard entry for the Information Classification Sorting challenge. Stores total score and total time taken.

- **Component Type:**

  Java class annotated with @Entity, located in the LeaderboardInfoSort.Global package.

15. InfoSortingLeaderboard Repo

- **Description and Purpose:**

  Handles database operations for leaderboard entries specific to the Information Classification Sorting challenge.

- **Component Type:**

  Java interface extending JpaRepository<InfoSortingLeaderboardEntry, Long>, located in the LeaderboardInfoSort.Global package.

16. InfoSortingLeaderboard Service

- **Description and Purpose:**

  Calculates and updates leaderboard totals for each student by aggregating relevant scores and times.

- **Component Type:**

  Java class annotated with @Service, located in the LeaderboardInfoSort.Global package.

17. InfoSortingLeaderboard Controller

- **Description and Purpose:**

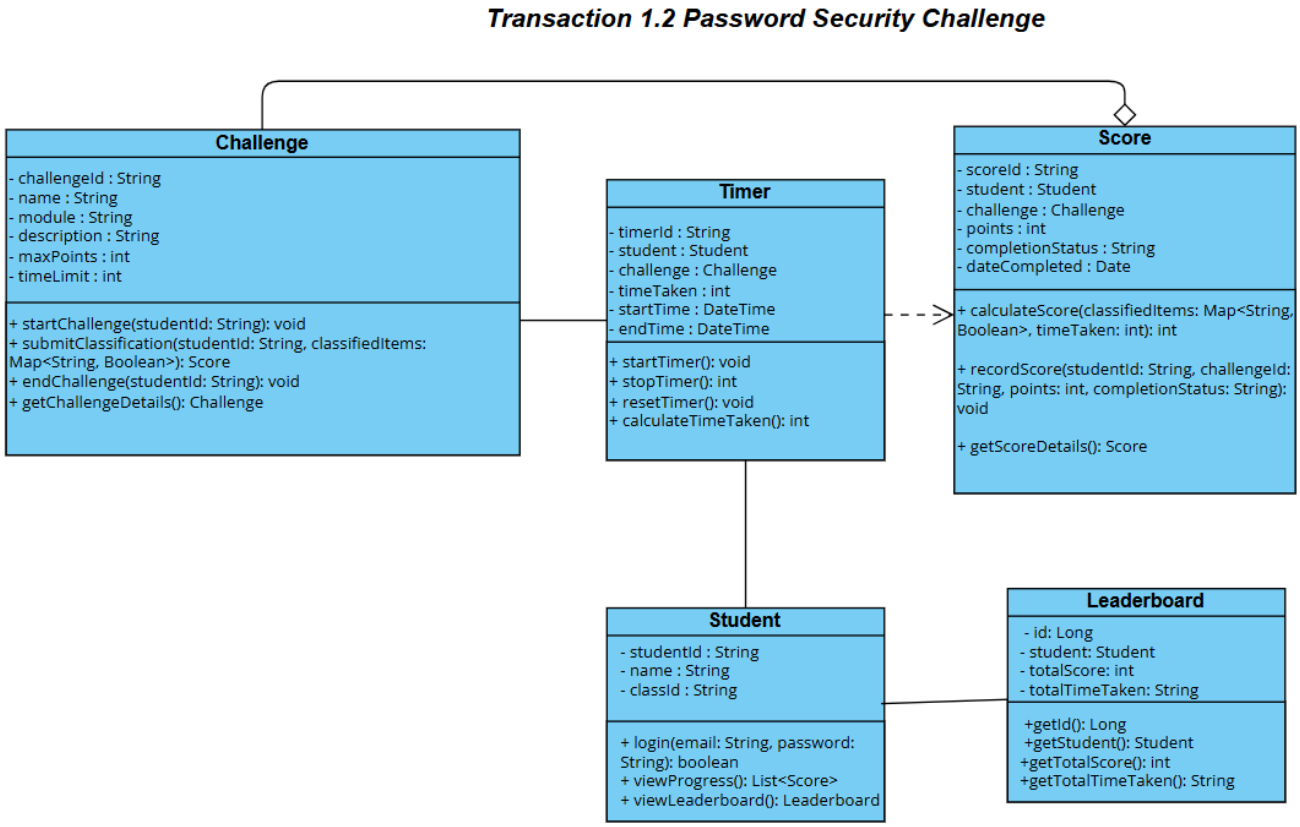
  Receives API requests to trigger leaderboard updates for specific students by robloxId.

- **Component Type:**

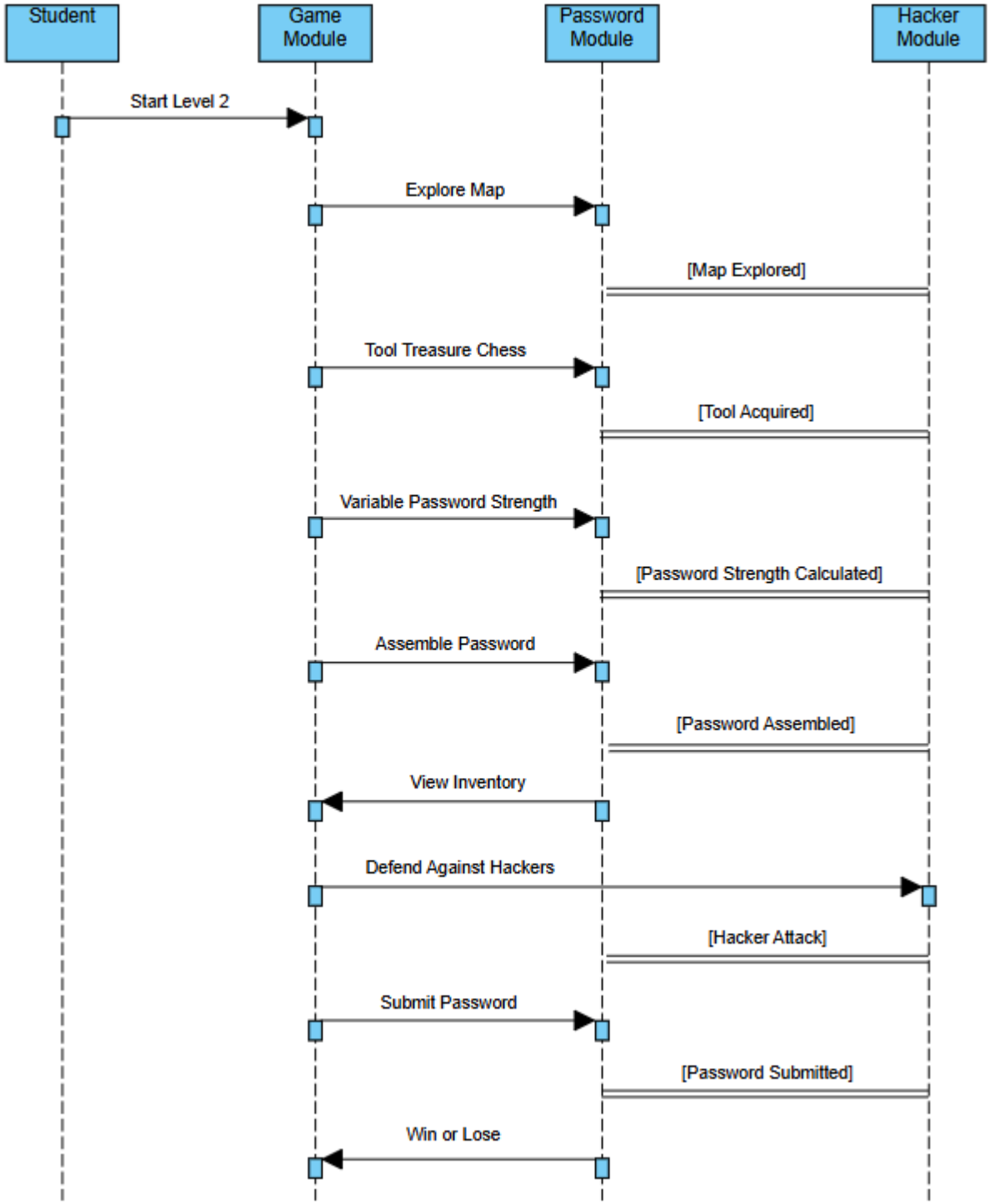
  Java class annotated with @RestController, located in the LeaderboardInfoSort.Global package.

Object-Oriented Components

- Class Diagram

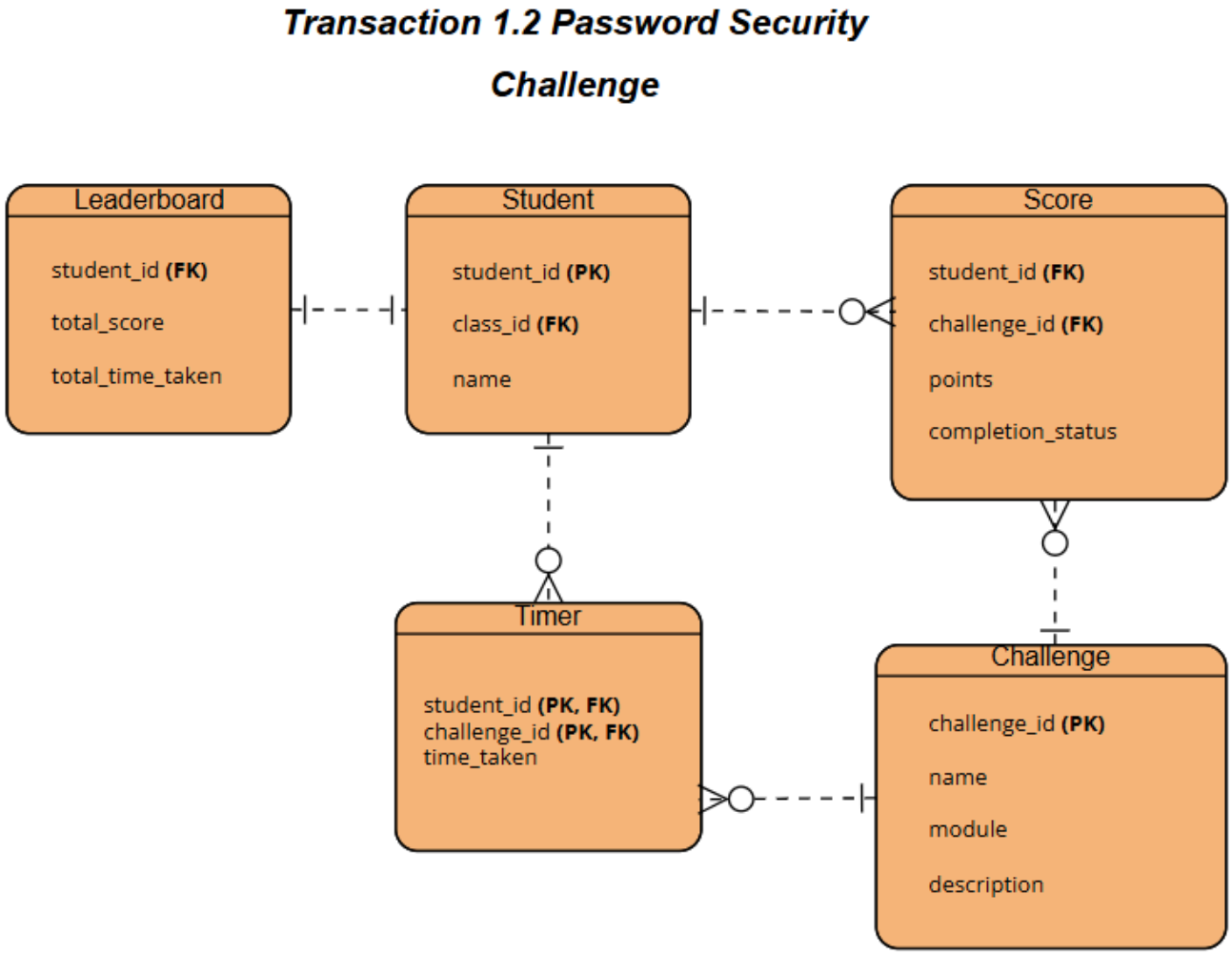


- Sequence Diagram



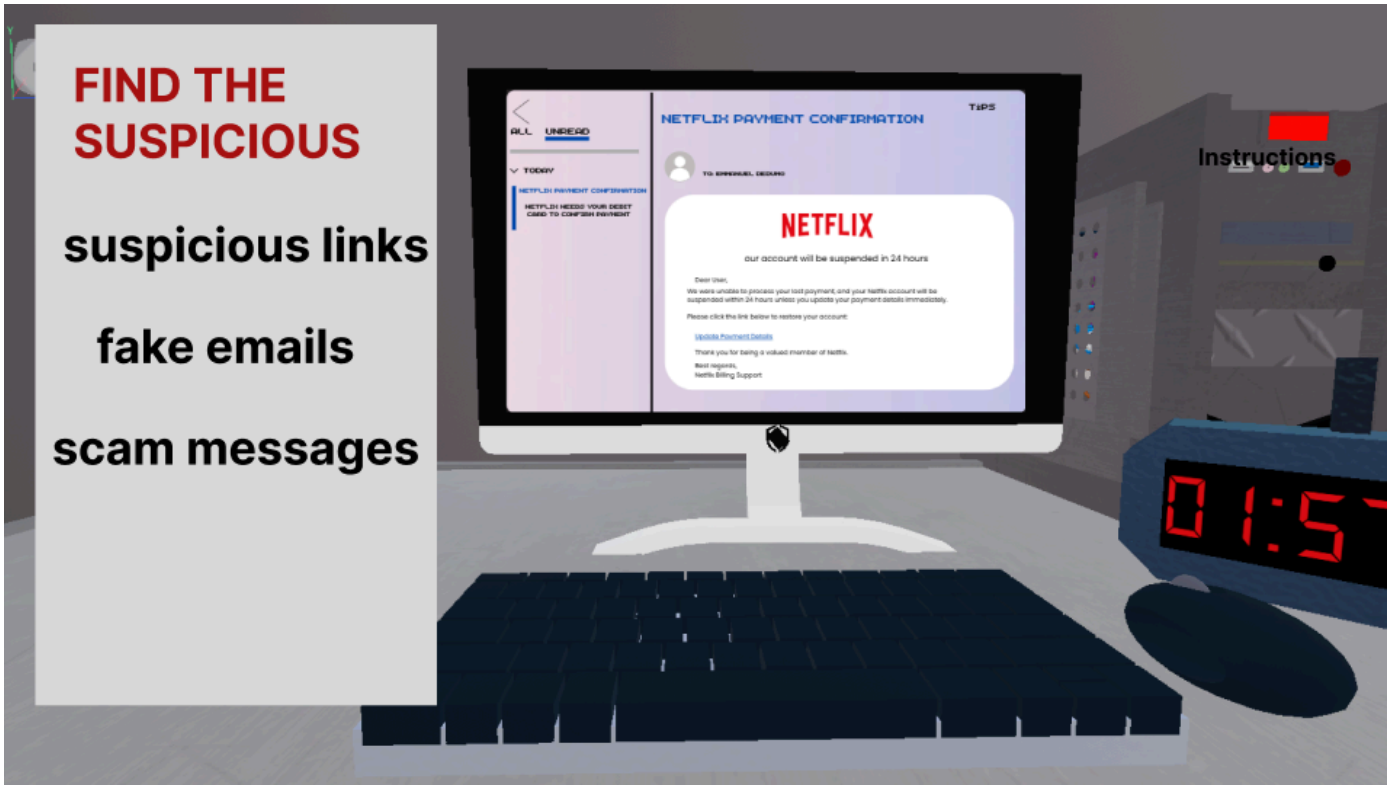
Data Design

- ERD or schema



Transaction 1.3 Phishing Identification Challenge

User Interface Design



Front-end component(s)

1. **Scenario Cutscenes**
    - ➢ **Description and purpose:** Introduces phishing challenges (e.g., suspicious emails or messages) through animated camera sequences or character dialogues.
    - ➢ **Component type:** LocalScripts using TweenService and Camera manipulation.
2. **PhishingScenarioGui**
    - ➢ **Description and purpose:** Displays email or message UIs that simulate realistic phishing attempts. Used for interaction and analysis by the player.
    - ➢ **Component type:** Roblox ScreenGui with TextLabels or Frames in StarterGui.
3. **FeedbackGui**
    - ➢ **Description and purpose:** Provides immediate feedback after each decision (e.g., correct/incorrect choices) to help reinforce learning.
    - ➢ **Component type:** Roblox ScreenGui designed like inbox/browser windows.
4. **ChoiceHandler (Client Script)**
    - ➢ **Description and purpose:** Handles player input for selecting phishing or legitimate options and updates feedback accordingly.
    - ➢ **Component type:** Roblox LocalScript.
5. **GameStarted & GameEnded (RemoteEvents)**
    - ➢ **Description and purpose:** GameStarted: Triggers cutscenes and initializes scenario.
    - ➢ GameEnded: Ends session, shows final score, feedback, and leaderboard entry.
    - ➢ **Component type:** Roblox RemoteEvent in ReplicatedStorage.
6. **TimerGui and TimerClient**
    - ➢ **Description and purpose:** Displays the countdown timer for the player. TimerClient updates time and synchronizes with the backend when time ends.
    - ➢ **Component type:** ScreenGui with Timer LocalScript.
7. **ScoreManager (Client Script)**
    - ➢ **Description and purpose:** Calculates score based on correct phishing identifications and time taken. Updates ScoreGui and sends it to the backend.
    - ➢ **Component type:** LocalScript using HttpService.

8. **LeaderboardGui and LeaderboardClient**
    - ➢ **Description and purpose:** Displays top-performing players and scores pulled from the backend. Updates in real-time or after each session.
    - ➢ **Component type:** ScreenGui in StarterGui and LocalScript.
9. **RegisterAPI & ScoreAPI (HTTP Modules)**
    - ➢ **Description and purpose:** RegisterAPI sends player's Roblox ID to backend for session tracking. ScoreAPI submits score, timer, and challenge type to backend.
    - ➢ **Component type:** ModuleScripts using HttpService:PostAsync().

Back-end component(s)

1. **Student Entity**
   - ➢ **Description and Purpose:**

     Represents the database table for students/players. Stores fields like id, robloxId, and name to uniquely identify and track users across gameplay sessions.
   - ➢ **Component Type:**

     Java Class annotated with @Entity (typically stored in Entity package).
2. **StudentRequest DTO (Data Transfer Object)**
   - ➢ **Description and Purpose:**

     Used to receive data from the client (such as robloxId and optionally name) when registering a new student/player. Separates input data from the full entity.
   - ➢ **Component Type:**

     Java POJO (Plain Old Java Object) used in HTTP requests.
3. **StudentController**
   - ➢ **Description and Purpose:**

     Handles incoming HTTP requests from the Roblox game client for student-related operations (e.g., registering a new player). Routes requests to the appropriate service methods.
   - ➢ **Component Type:**

     Java Class annotated with @RestController.
4. **StudentService**
   - ➢ **Description and Purpose:**

     Contains business logic for creating, retrieving, and deleting student records. Abstracts logic away from the controller to maintain clean code and separation of concerns.
   - ➢ **Component Type:**

     Java Class annotated with @Service.

5. **StudentRepo**
   - ➢ **Description and Purpose:**

     Communicates directly with the database to perform CRUD operations. Uses Spring Data JPA to auto-generate SQL logic for Student entities.
   - ➢ **Component Type:**

     Java Interface extending JpaRepository<Student, Long>, annotated with @Repository.
6. **Timer Entity**
   - ➢ **Description and Purpose:**

     Stores timing information (e.g., start and end times, duration) for each student challenge session.
   - ➢ **Component Type:**

     Java class annotated with `@Entity`, located in the `Entity` package.
7. **TimerRepo**
   - ➢ **Description and Purpose:**

     Provides data access for Timer entities.
   - ➢ **Component Type:**

     Java interface extending `JpaRepository<Timer, Long>`, located in the `Repository` package.
8. **TimerService**
   - ➢ **Description and Purpose:**

     Handles logic for updating and retrieving timer records per challenge or student.
   - ➢ **Component Type:**

     Java class annotated with @Service, located in the Service package.
9. **TimerController**
   - ➢ **Description and Purpose:**

     Receives requests from the Roblox client to log or retrieve challenge timing data.
   - ➢ **Component Type:**

     Java class annotated with @RestController, located in the Controller package.

10. Score Entity

- **Description and Purpose:**

  Represents points earned by students per challenge, linked to their account and used for leaderboard calculations.

- **Component Type:**

  Java class annotated with @Entity, located in the Entity package.

11. ScoreRepo

- **Description and Purpose:**

  Performs database operations for Score records.

- **Component Type:**

  Java interface extending JpaRepository<Score, Long>, located in the Repository package.

12. ScoreService

- **Description and Purpose:**

  Contains logic for saving and retrieving score data per student.

- **Component Type:**

  Java class annotated with @Service, located in the Service package.

13. ScoreController

- **Description and Purpose:**

  Handles HTTP requests to log new scores or retrieve existing ones. Automatically registers new students if not found.

- **Component Type:**

  Java class annotated with @RestController, located in the Controller package.

14. InfoSortingLeaderboardEntry Entity

- **Description and Purpose:**

  Represents a leaderboard entry for the Information Classification Sorting challenge. Stores total score and total time taken.

- **Component Type:**

  Java class annotated with @Entity, located in the LeaderboardInfoSort.Global package.

15. InfoSortingLeaderboard Repo

- **Description and Purpose:**

  Handles database operations for leaderboard entries specific to the Information Classification Sorting challenge.

- **Component Type:**

  Java interface extending JpaRepository<InfoSortingLeaderboardEntry, Long>, located in the LeaderboardInfoSort.Global package.

16. InfoSortingLeaderboard Service

- **Description and Purpose:**

  Calculates and updates leaderboard totals for each student by aggregating relevant scores and times.

- **Component Type:**

  Java class annotated with @Service, located in the LeaderboardInfoSort.Global package.

17. InfoSortingLeaderboard Controller

- **Description and Purpose:**

  Receives API requests to trigger leaderboard updates for specific students by robloxId.

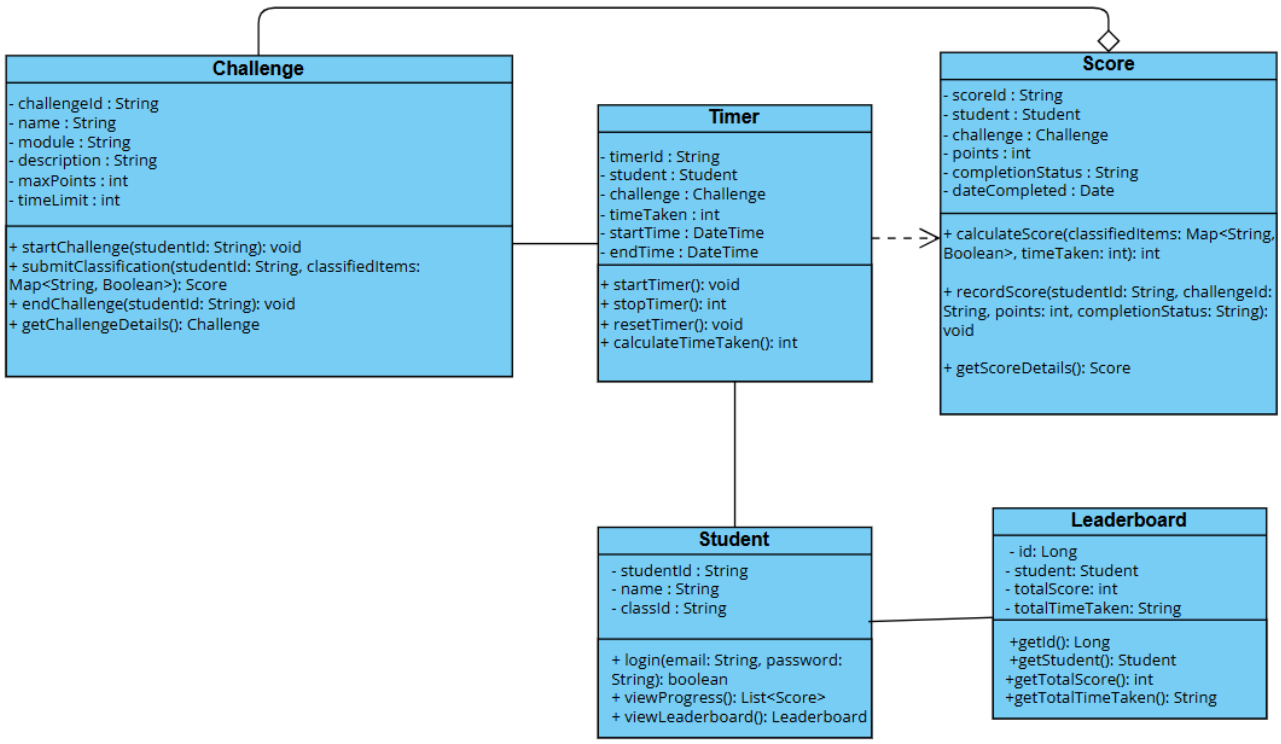
- **Component Type:**

  Java class annotated with @RestController, located in the LeaderboardInfoSort.Global package.

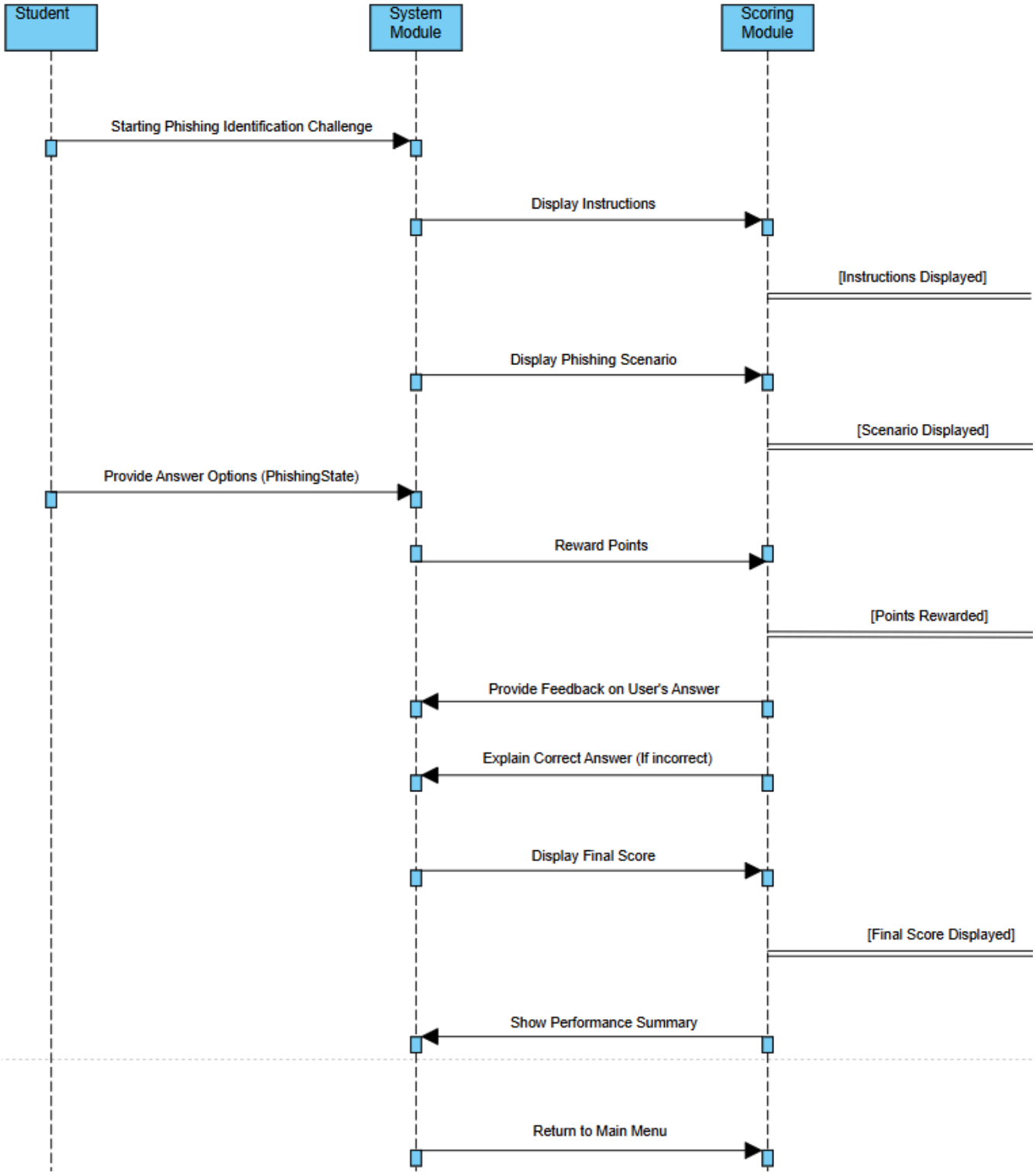
Object-Oriented Components

- Class Diagram

Transaction 1.3 Phishing Identification Challenge



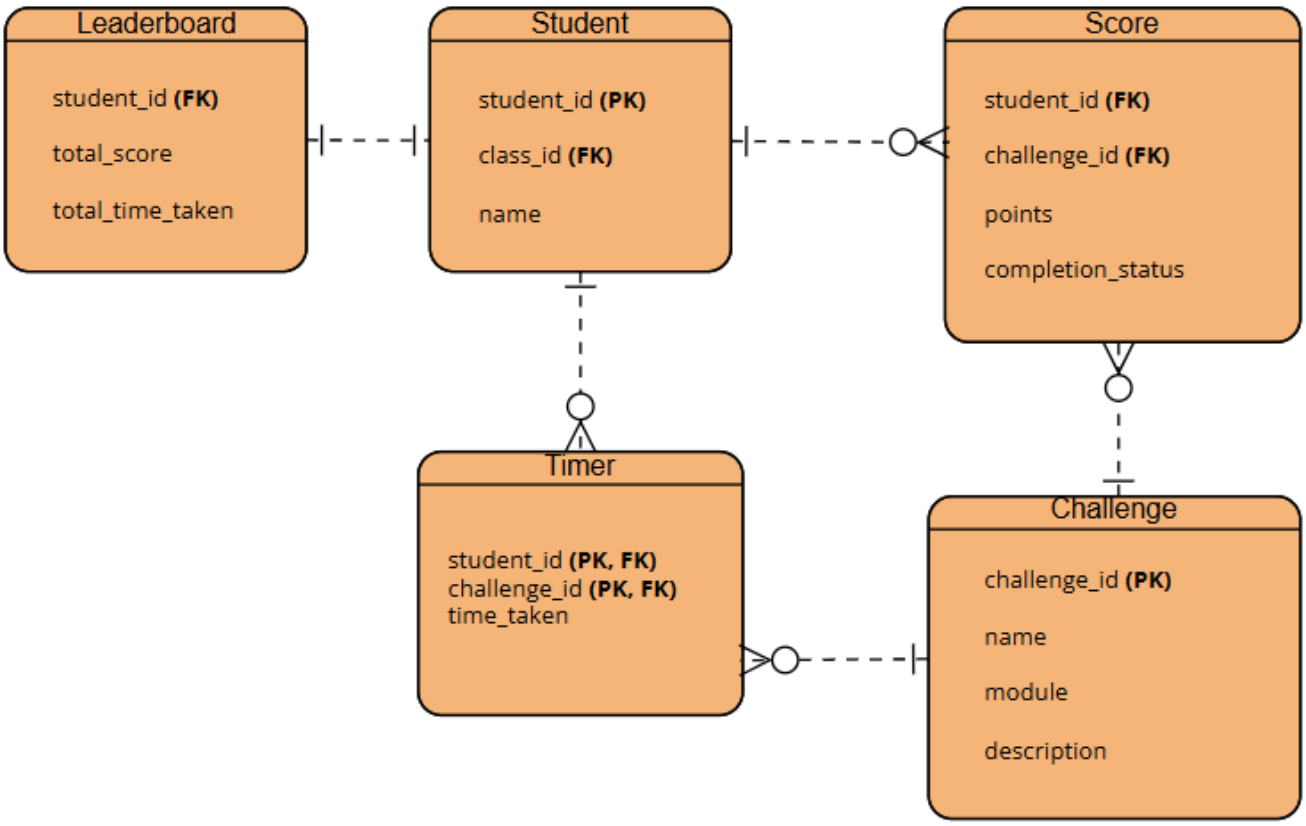
- Sequence Diagram



Data Design

- ERD or schema

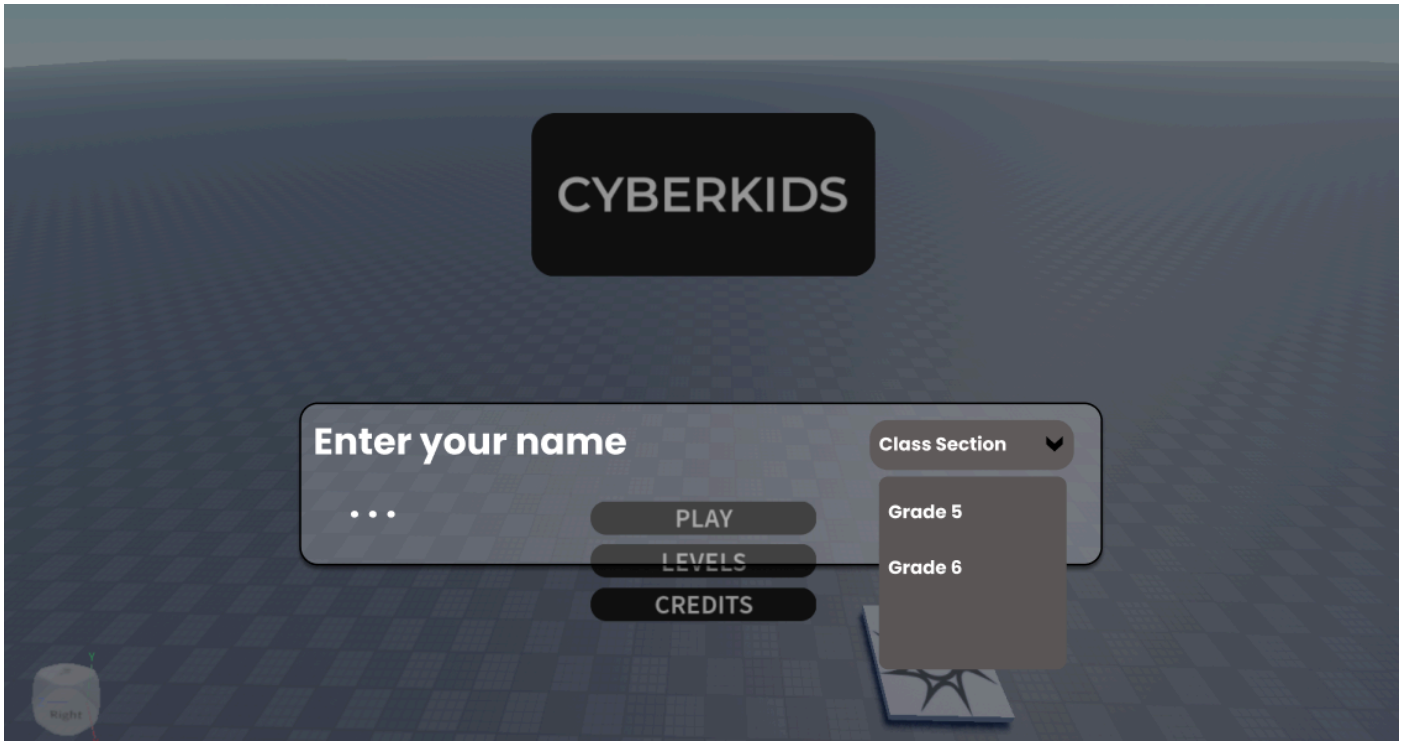
Transaction 1.3 Phishing Identification Challenge



Module 2: Student Real-Name Integration

Transaction 2.1 Real-Name Input and Submission

User Interface Design



Front-end component(s)

1. **RealNameInputGui**
   - ➢ **Description and purpose:** A graphical user interface shown to the player at the start of the game. Allows students to input their real name for registration. Typically includes textboxes for real name and Roblox name, and a submit button.
   - ➢ **Component type:** Roblox ScreenGui containing TextBox and TextButton instances.
2. **RealNameClient (LocalScript)**
   - ➢ **Description and purpose:** Handles user interaction on the GUI. Validates inputs and sends the data (realName, robloxName, robloxId) to the backend using HttpService:PostAsync(). Displays feedback to the player upon success or failure.
   - ➢ **Component type:** Roblox LocalScript placed inside the GUI or StarterPlayerScripts.
3. **HttpService Configuration**
   - ➢ **Description and purpose:** Enables Roblox to communicate with the external backend. Used to send the registration request from the client to the Spring Boot API (/api/student/register).
   - ➢ **Component type:** Enabled HttpService with proper permissions in Roblox Studio (API Services must be on).
4. **Player.UserId (RobloxId Source)**
   - ➢ **Description and purpose:** Used as the unique Roblox identifier in the registration payload. Retrieved directly using Players.LocalPlayer.UserId.
   - ➢ **Component type:** Built-in property of the Player object in Roblox.

Back-end component(s)

1. **Student Entity**
   - ➢ **Description and Purpose:**

     Represents the database table for students/players. Stores fields like id, robloxId, and name to uniquely identify and track users across gameplay sessions.
   - ➢ **Component Type:**

     Java Class annotated with @Entity (typically stored in Entity package).
2. **StudentRequest DTO (Data Transfer Object)**
   - ➢ **Description and Purpose:**

     Used to receive data from the client (such as robloxId and optionally name) when registering a new student/player. Separates input data from the full entity.
   - ➢ **Component Type:**

     Java POJO (Plain Old Java Object) used in HTTP requests.
3. **StudentController**
   - ➢ **Description and Purpose:**

     Handles incoming HTTP requests from the Roblox game client for student-related operations (e.g., registering a new player). Routes requests to the appropriate service methods.
   - ➢ **Component Type:**

     Java Class annotated with @RestController.
4. **StudentService**
   - ➢ **Description and Purpose:**

     Contains business logic for creating, retrieving, and deleting student records. Abstracts logic away from the controller to maintain clean code and separation of concerns.
   - ➢ **Component Type:**

     Java Class annotated with @Service.

5. **StudentRepo**
   - ➢ **Description and Purpose:**

     Communicates directly with the database to perform CRUD operations. Uses Spring Data JPA to auto-generate SQL logic for Student entities.
   - ➢ **Component Type:**

     Java Interface extending JpaRepository<Student, Long>, annotated with @Repository.
6. **Validation & Error Handling**
   - ➢ **Description and Purpose:**

     Ensures clean and valid submissions. Prevents duplicate Roblox IDs and handles empty or invalid input, returning clear status codes like 404 or 409.
   - ➢ **Component Type:**

     Bean validation annotations and exception handling logic.
7. **CORS / Security Configuration**
   - ➢ **Description and Purpose:**

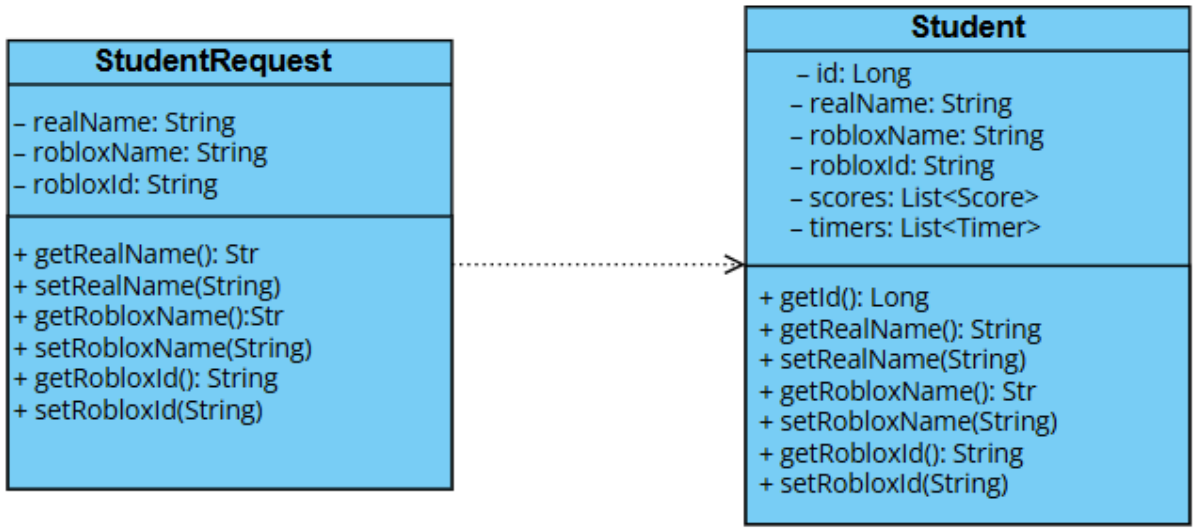
     Enables secure communication between Roblox frontend and backend by allowing specific origins (e.g., Roblox domain) and restricting sensitive endpoints.
   - ➢ **Component Type:**

     CORS configuration and Spring Security classes.

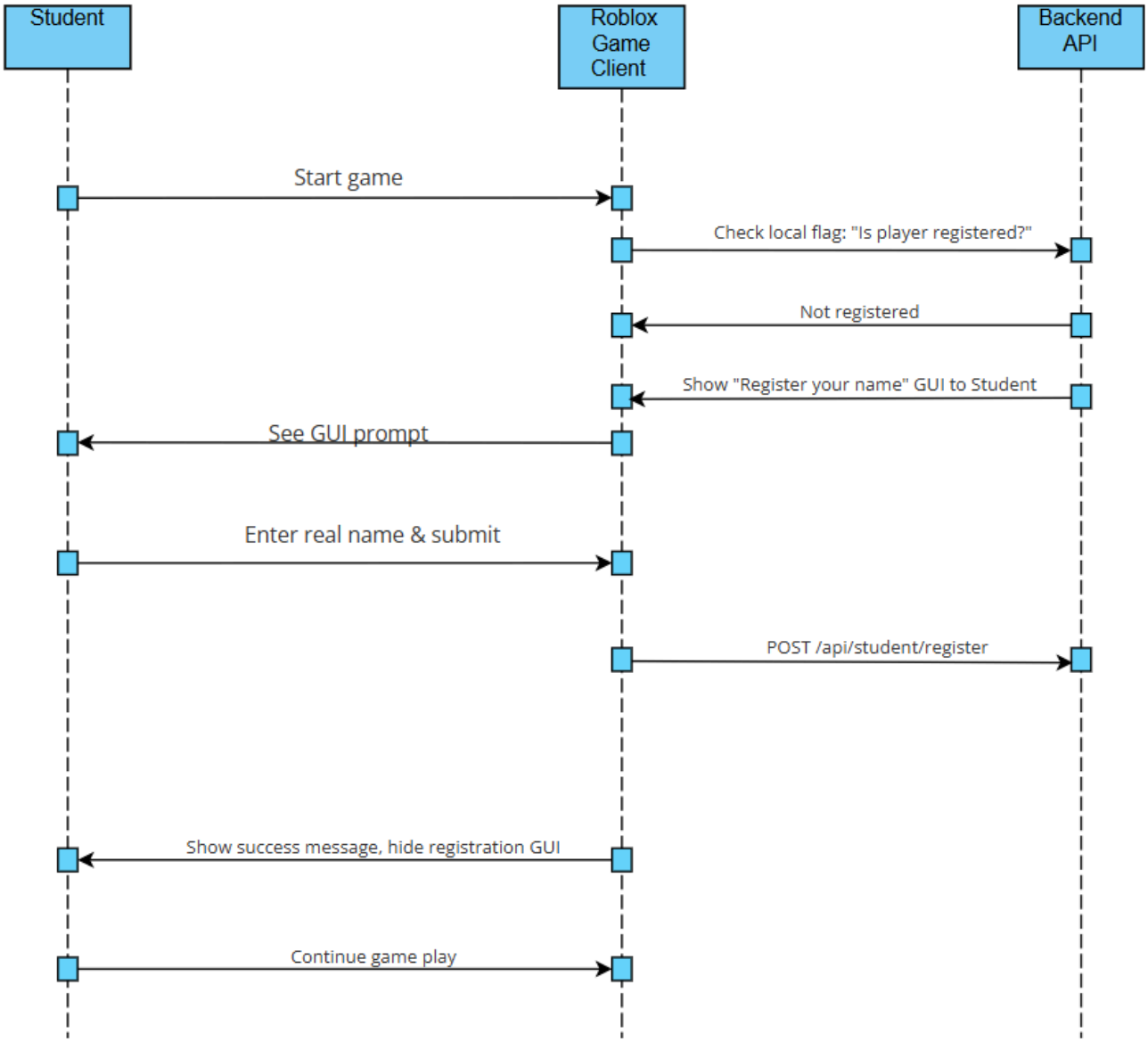
Object-Oriented Components

- Class Diagram

Transaction 2.1 Real-Name Input and Submission

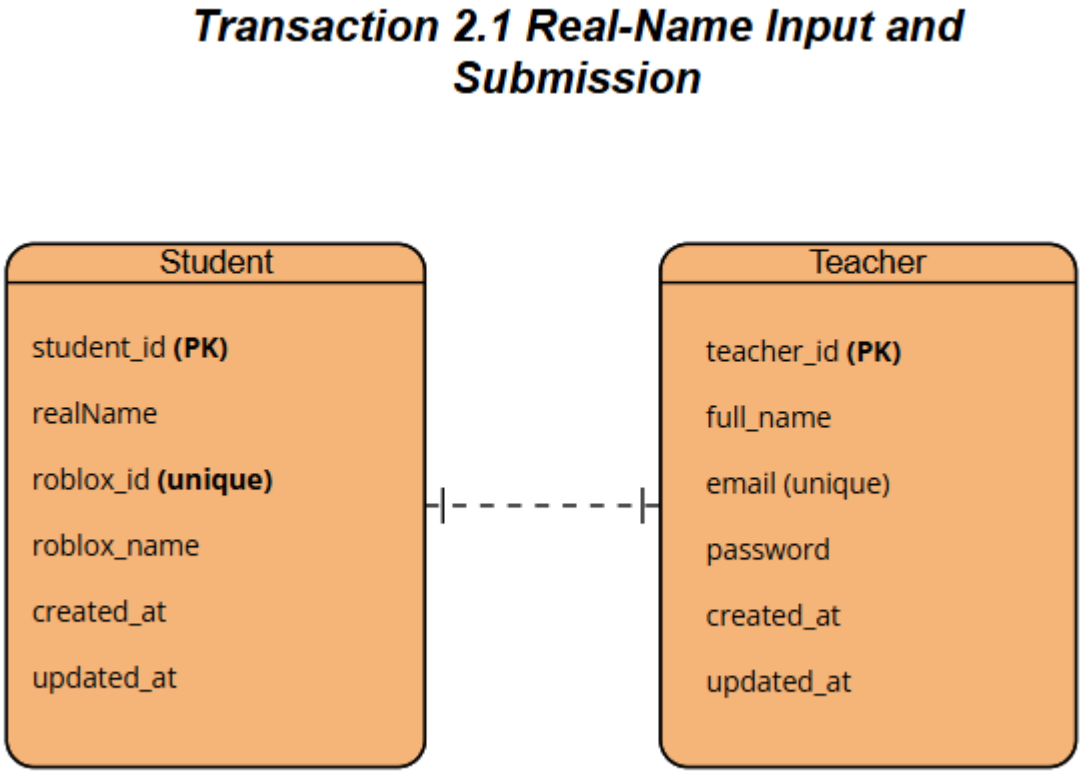


- Sequence Diagram



Data Design

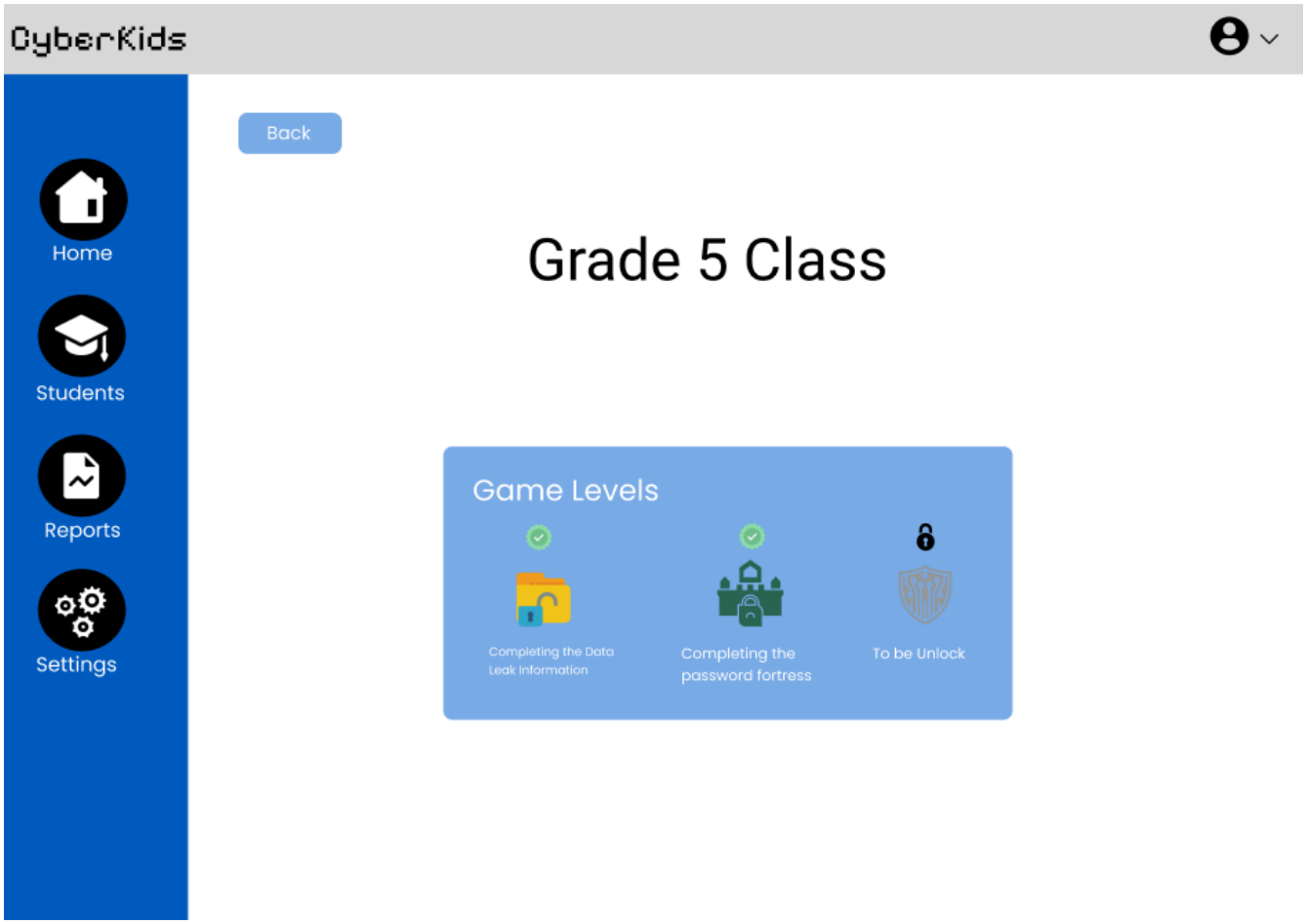
- ERD or schema



Module 3: Teachers Dashboard

Transaction 3.1 Lock / Unlock Game Levels

User Interface Design



Front-end component(s)

1. **Dashboard.jsx**
   - ➢ **Description and Purpose:**

     Main page for the teacher's dashboard. It displays toggles for Level 1, 2, and 3 by reusing the LevelToggle component. The teacherId is extracted from the route params to load and send settings specific to that teacher.
   - ➢ **Component Type:**

     React Page Component (typically placed in src/pages/Dashboard.jsx).
2. **api.js (HTTP Client Setup)**
   - ➢ **Description and Purpose:**

     Configures the Axios HTTP client with a base API URL and a request interceptor to include the JWT token in all API calls. Ensures authenticated requests for protected endpoints like level lock/unlock.
   - ➢ **Component Type:**

     JavaScript module (typically placed under src/api.js or src/utils/api.js).
3. **LevelToggle.jsx**
   - ➢ **Description and Purpose:**

     A reusable toggle switch component for displaying and updating the open/locked status of a specific level (1, 2, or 3). It fetches the current setting from the backend on mount and allows toggling the state with a POST API call.
   - ➢ **Component Type:**

     React Functional Component (typically placed in src/components/LevelToggle.jsx).

Back-end component(s)

1. **TeacherSettings Entity**
   - ➢ **Description and Purpose:**

     Represents the database table for storing level lock/unlock settings per teacher. Each row holds flags for whether Levels 1, 2, and 3 are open (unlocked) or closed (locked). The primary key is teacherId, which corresponds to the Teacher entity's ID.
   - ➢ **Component Type:**

     Java Class annotated with @Entity (typically stored in the entity or model package).
2. **LevelLockRequest DTO**
   - ➢ **Description and Purpose:**

     Acts as a Data Transfer Object (DTO) for incoming level lock/unlock requests. It holds the level number (1, 2, or 3) and a boolean indicating whether the level should be open (true) or locked (false).
   - ➢ **Component Type:**

     Plain Old Java Object (POJO), typically placed in a dto or payload package. Used with @RequestBody in controller methods.
3. **TeacherSettingsRepository**
   - ➢ **Description and Purpose:**

     Provides access to the teacher_settings table for creating, reading, and updating lock status. Inherits basic CRUD operations from JpaRepository.
   - ➢ **Component Type:**

     Java Interface extending JpaRepository<TeacherSettings, Long>, typically stored in the repository package.
4. **TeacherSettingsController**
   - ➢ **Description and Purpose:**

     Exposes REST API endpoints that allow teachers to fetch and update their level lock settings. Ensures the corresponding settings row exists for the given teacherId.
   - ➢ **Component Type:**

     Java Class annotated with @RestController, typically located in the controller package.
5. **JWT Authorization / Security Filter**

- **Description and Purpose:**

  Ensures that only authenticated teachers can access or modify their own level settings. Verifies that the {tid} path variable matches the teacher ID in the JWT token.

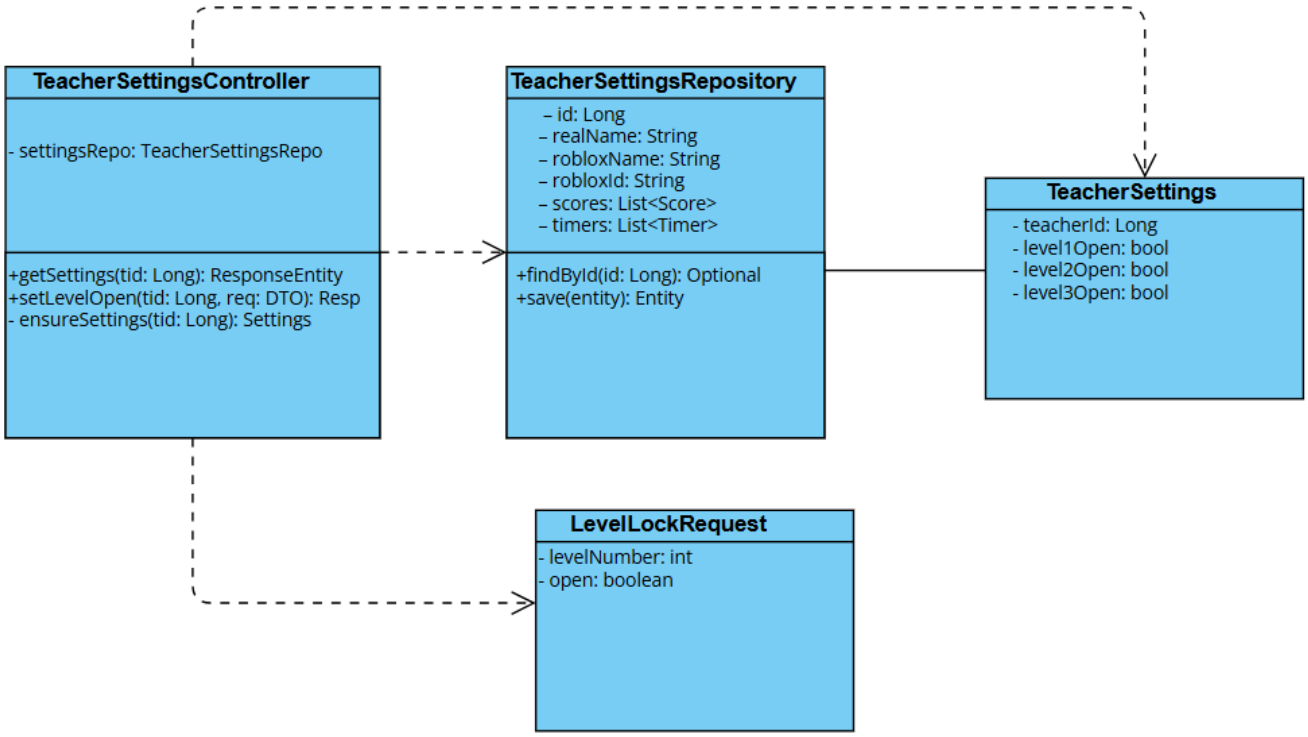
- **Component Type:**

  Security logic typically enforced via Spring Security configuration (e.g. SecurityConfig, filters, method-level security with @PreAuthorize).

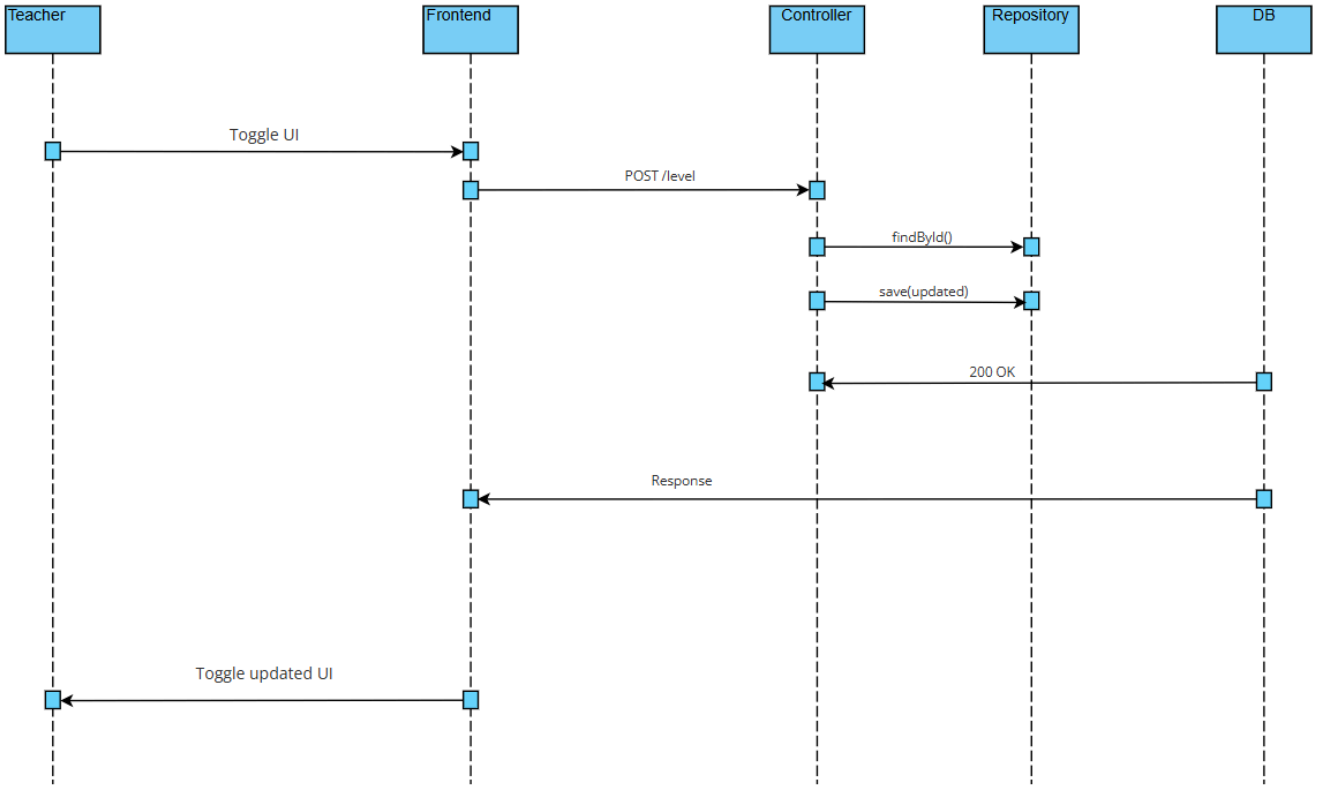
Object-Oriented Components

- Class Diagram

Transaction 3.1 Lock / Unlock Game Levels

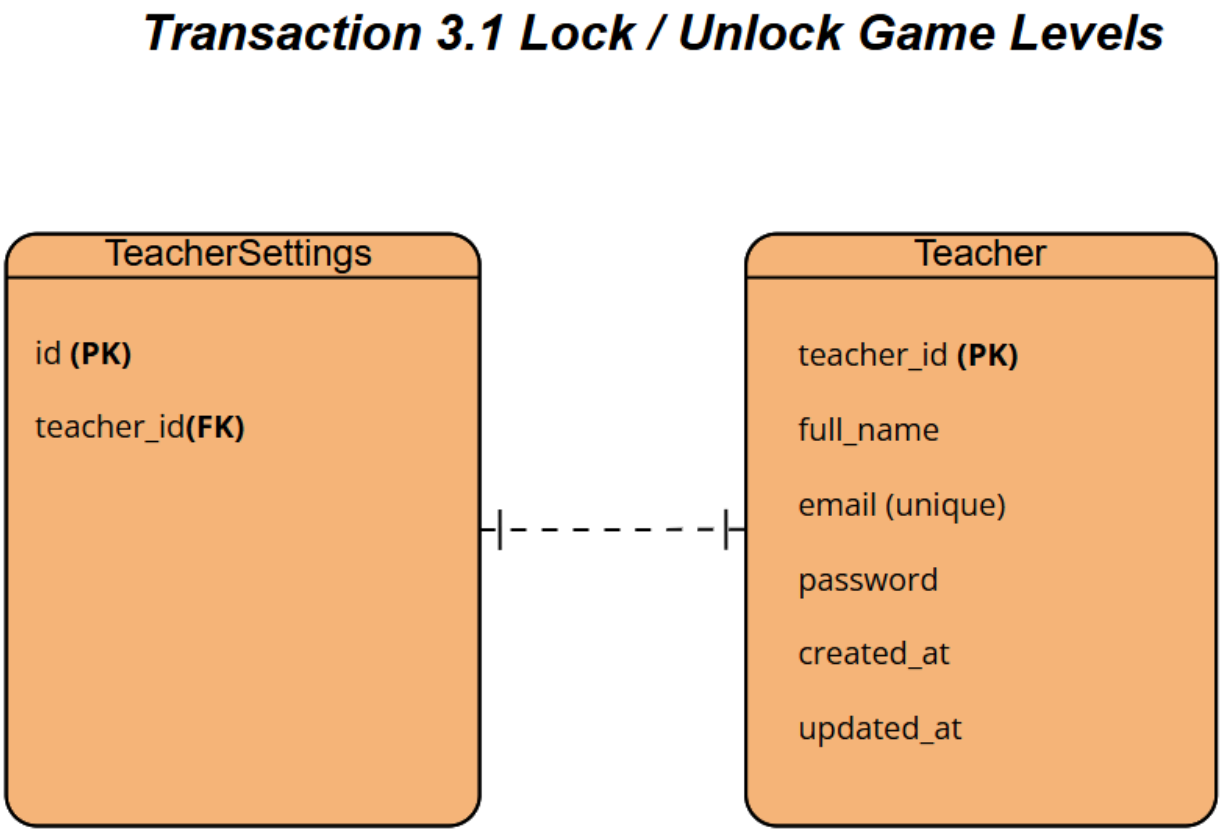


- Sequence Diagram



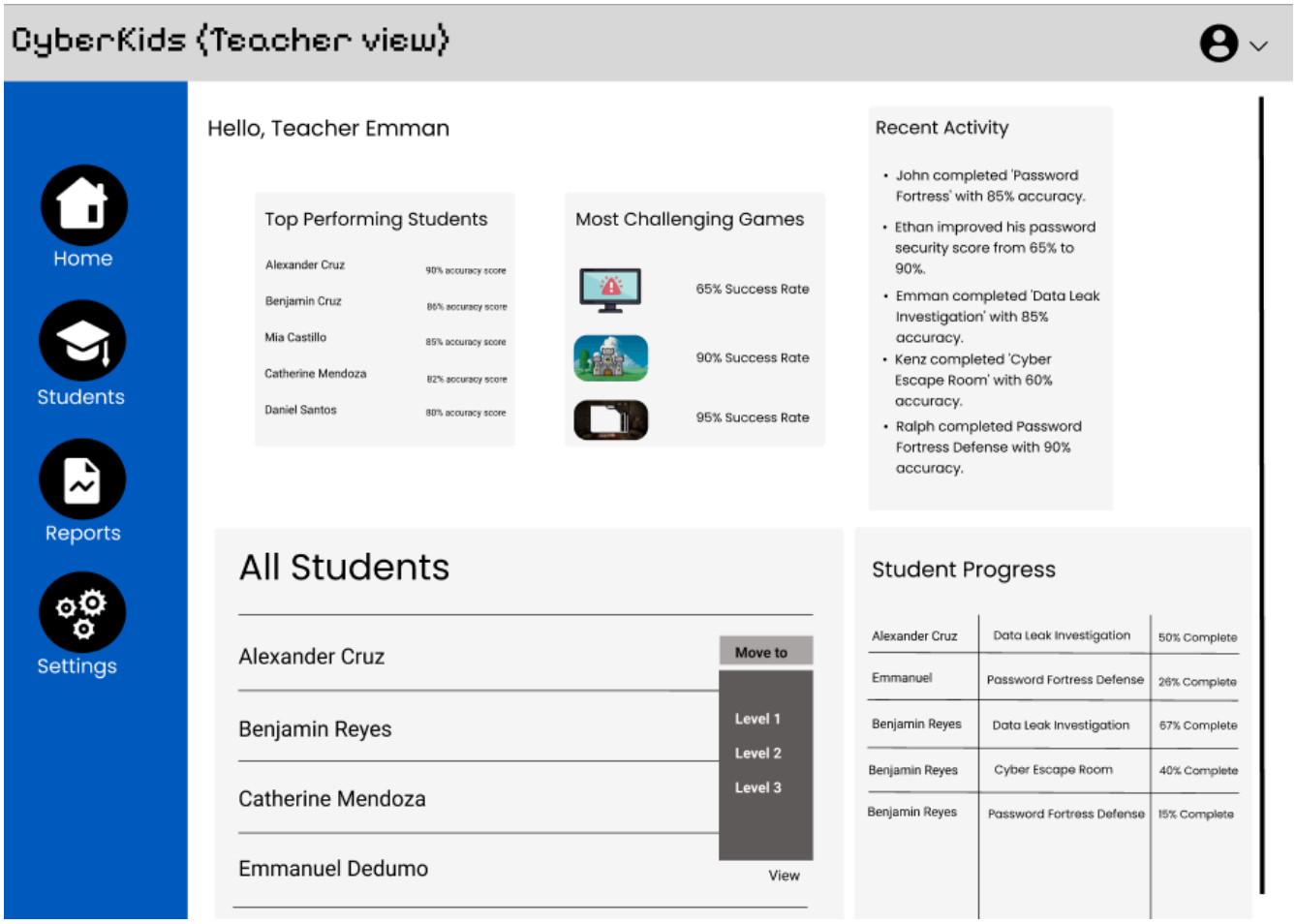
Data Design

- ERD or schema



Transaction 3.2 Reassign Players to Levels

User Interface Design



Front-end component(s)

1. **PlayerReassign.jsx**
   - ➢ **Description and Purpose:**
     This is the main React page for reassigning students to different levels. It fetches all students and renders a table of StudentRow components. Each row shows the student's name, Roblox ID, current level, and level reassignment buttons.
   - ➢ **Component Type:**
     React Page Component (typically placed in src/pages/PlayerReassign.jsx)
2. **StudentRow.jsx**
   - ➢ **Description and Purpose:**
     A reusable row component in the reassign table. It loads a student's current level via fetchGameState, and provides buttons to let the teacher reassign the student to levels 1–3 using reassignPlayer.
   - ➢ **Component Type:**
     React Component (typically placed in src/components/StudentRow.jsx.

Back-end component(s)

1. **StudentGameState**
   - ➢ **Description and Purpose:**
     Represents each student’s current level and the level they are targeted to be reassigned to. Used to support dynamic level teleportation in Roblox based on teacher input.
   - ➢ **Component Type:**
     Java class annotated with @Entity (typically stored in entity package).
2. **StudentGameState**
   - ➢ **Description and Purpose:**
     Data Transfer Object (DTO) used by the teacher dashboard to specify a student and the new level to assign them.
   - ➢ **Component Type:**
     Plain Java Object (typically stored in dto package).

3. **StudentGameStateRepository**

- ➢ **Description and Purpose:**

  Interface that provides database access for StudentGameState records using Spring Data JPA.

- ➢ **Component Type:**

  Java interface extending JpaRepository (typically stored in repository package).

4. **StudentGameStateController**

- ➢ **Description and Purpose:**

  REST controller that handles requests to reassign players and allow Roblox to fetch player-level state.

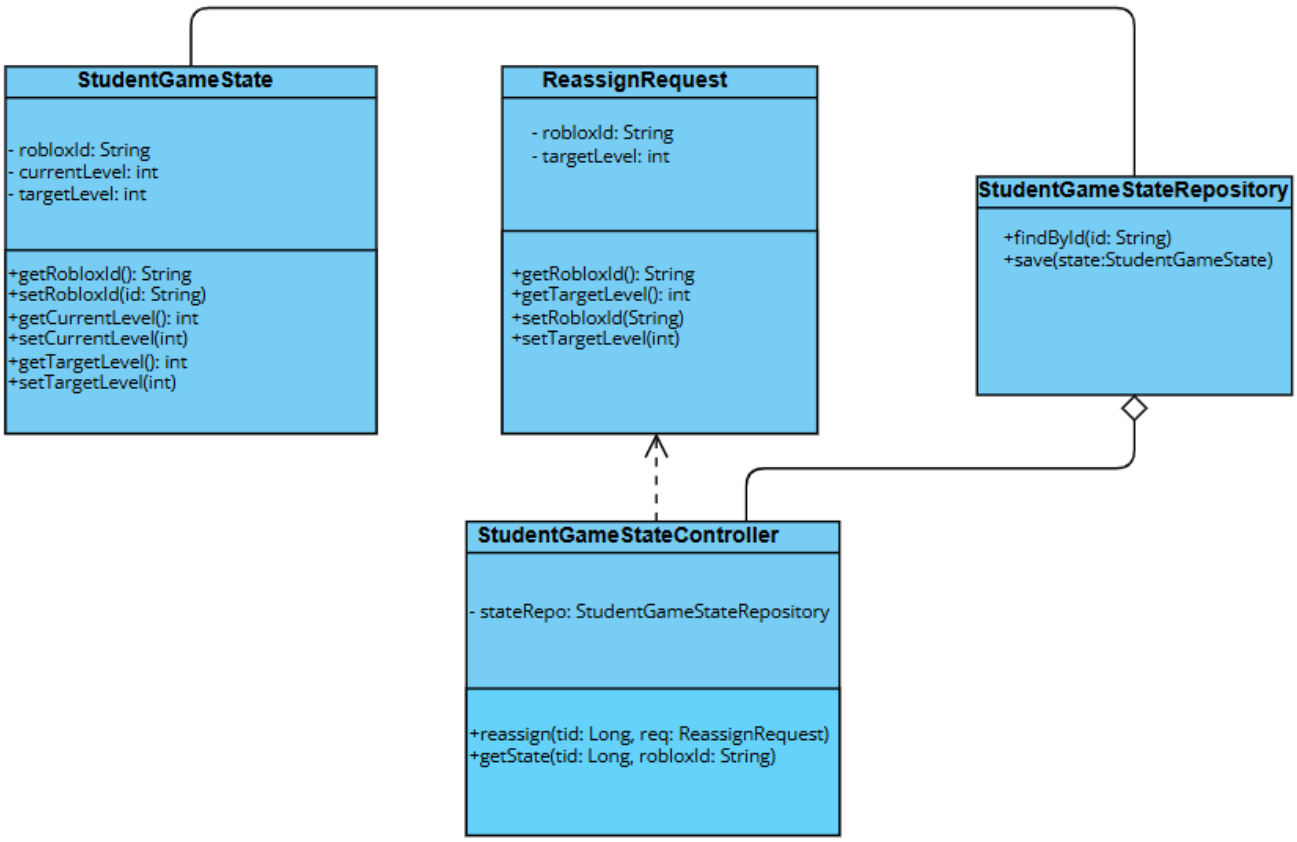
- ➢ **Component Type:**

  Java class annotated with @RestController (typically stored in controller package).

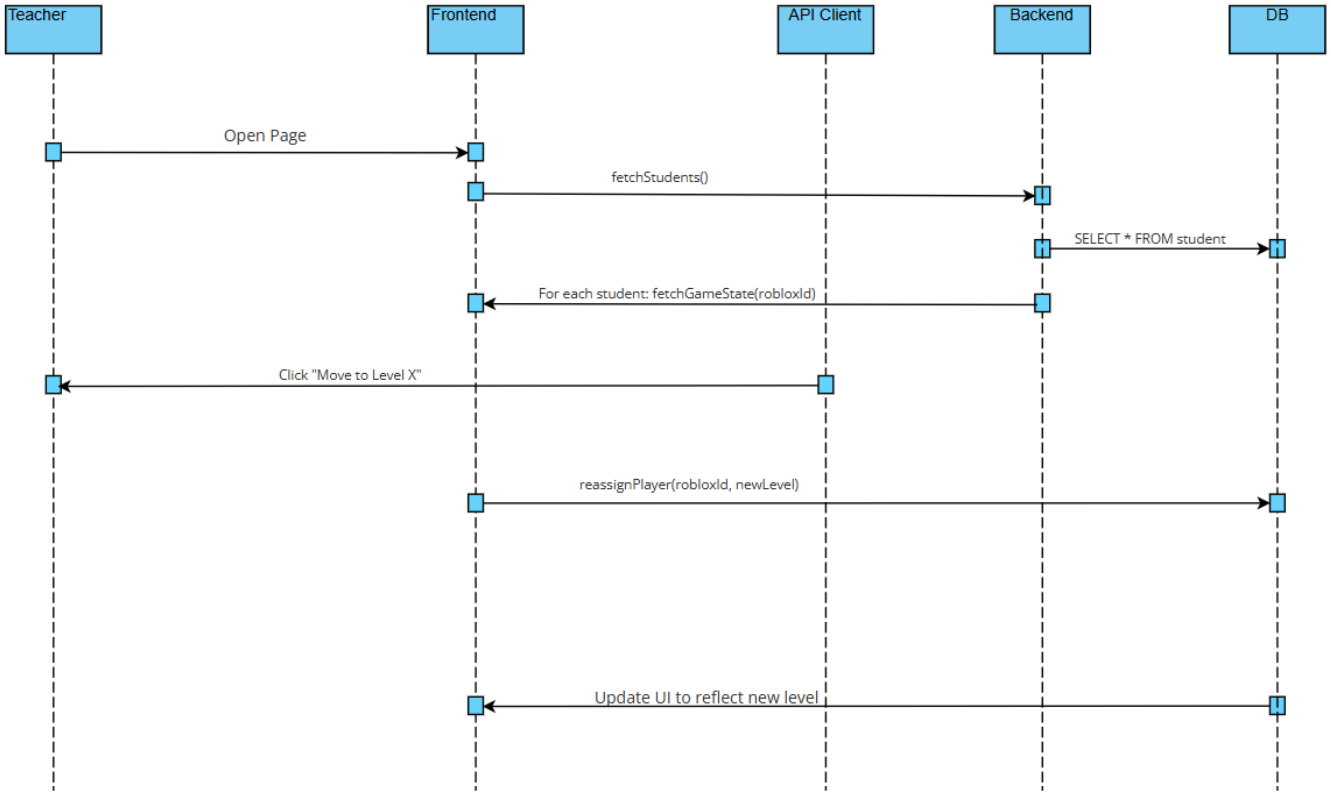
Object-Oriented Components

- Class Diagram

Transaction 3.2 Reassign Players to Levels

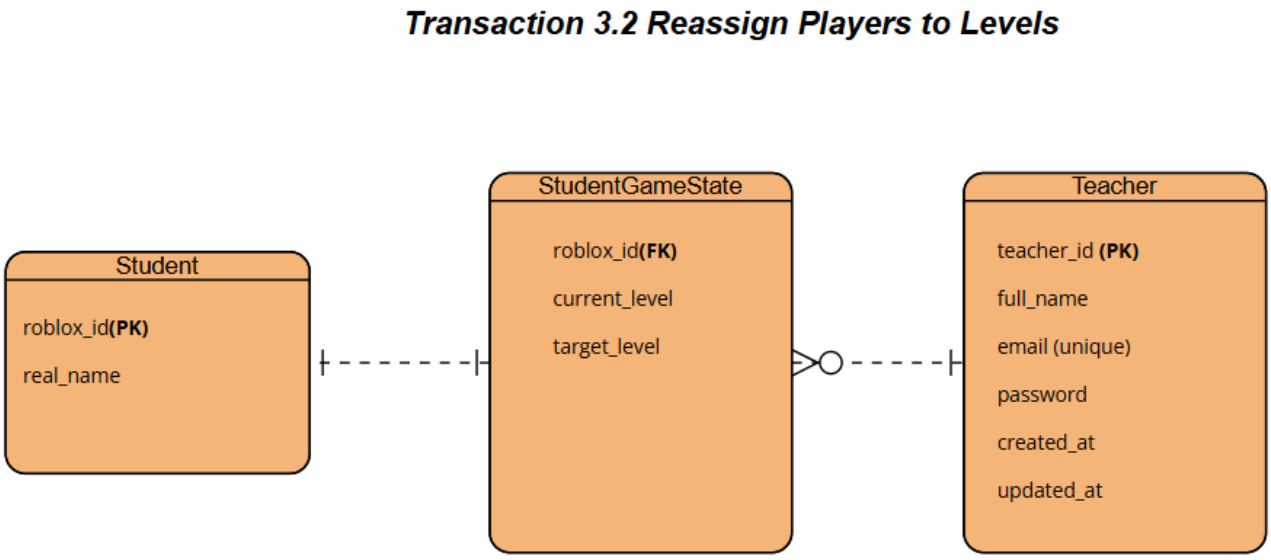


- Sequence Diagram



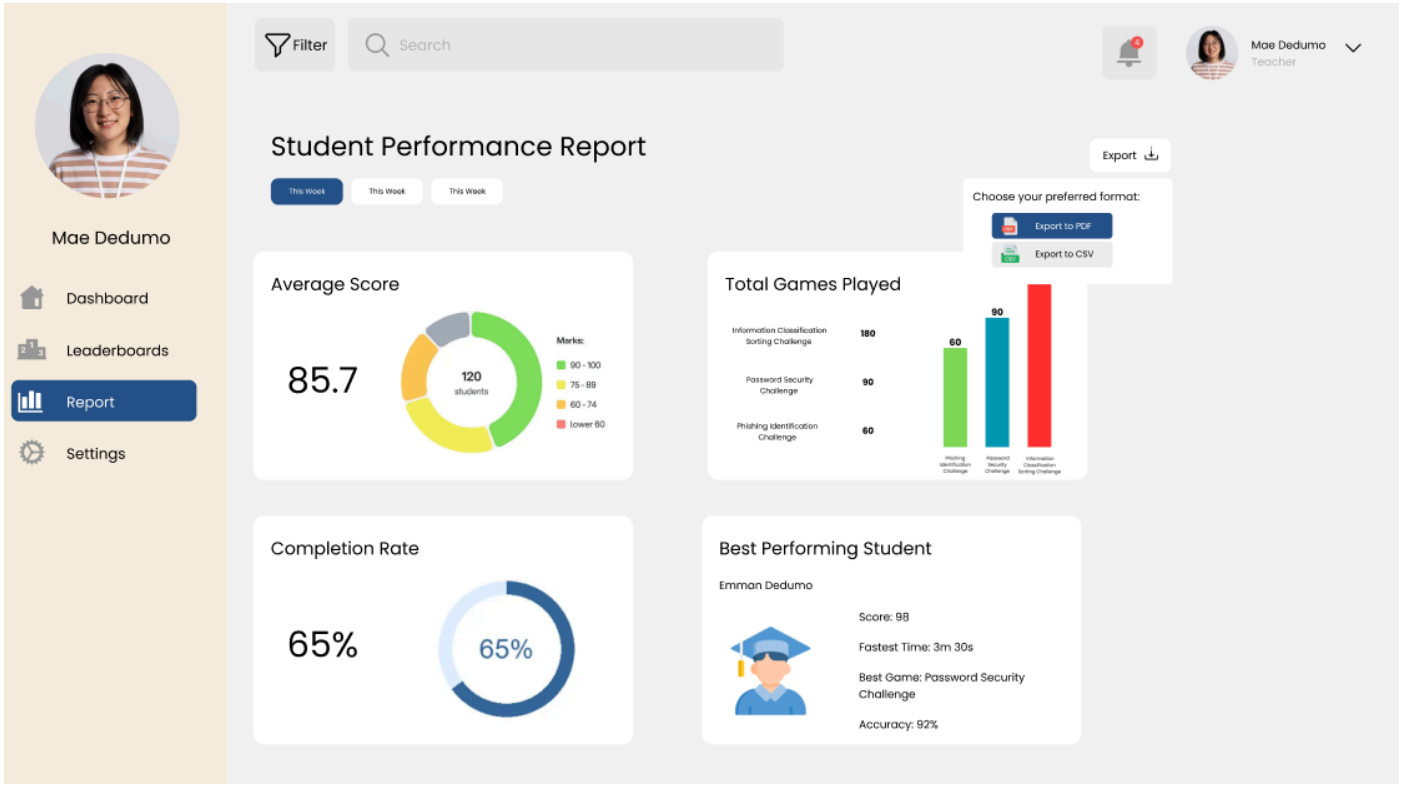
Data Design

- ERD or schema



Transaction 3.3 Generate Reports (CSV/PDF)

User Interface Design



Front-end component(s)

1. **Export Options Panel**
   - ➢ **Description and purpose:** Allows teachers to choose report formats (PDF, CSV)
   - ➢ **Component type:** React.js Component (Material UI Dialog, Buttons)
2. **Student Report Viewer**
   - ➢ **Description and purpose:** Displays student performance data before exporting.
   - ➢ **Component type:** React.js Component (Material UI Table, Typography)
3. **Download Button**
   - ➢ **Description and purpose:** Triggers report generation and downloads the file.
   - ➢ **Component type:** React.js Component (Button, Icon)

Back-end component(s)

1. **ReportController**
   - ➢ **Description and purpose:** Handles API requests for generating and exporting student reports in different formats (PDF, CSV).
   - ➢ **Component type:** Spring Boot REST Controller *(@RestController)*
2. **ReportService**
   - ➢ **Description and purpose:** Business logic for compiling student performance data, formatting it, and generating reports.
   - ➢ **Component type**: Spring Boot Service *(@Service)*
3. **ReportRepository**
   - ➢ **Description and purpose:** Fetches necessary student performance data from the database for report generation.
   - ➢ **Component type:** Spring Data JPA Repository *(@Repository)*
4. **StudentPerformance Model**
   - ➢ **Description and purpose:** Represents the structure of student performance data, including game scores, completion rates, and achievements.
   - ➢ **Component type:** Java Entity *(@Entity)* with JPA annotations
5. **PDFGenerator Utility**
   - ➢ **Description and purpose:** Handles the conversion of student data into a structured

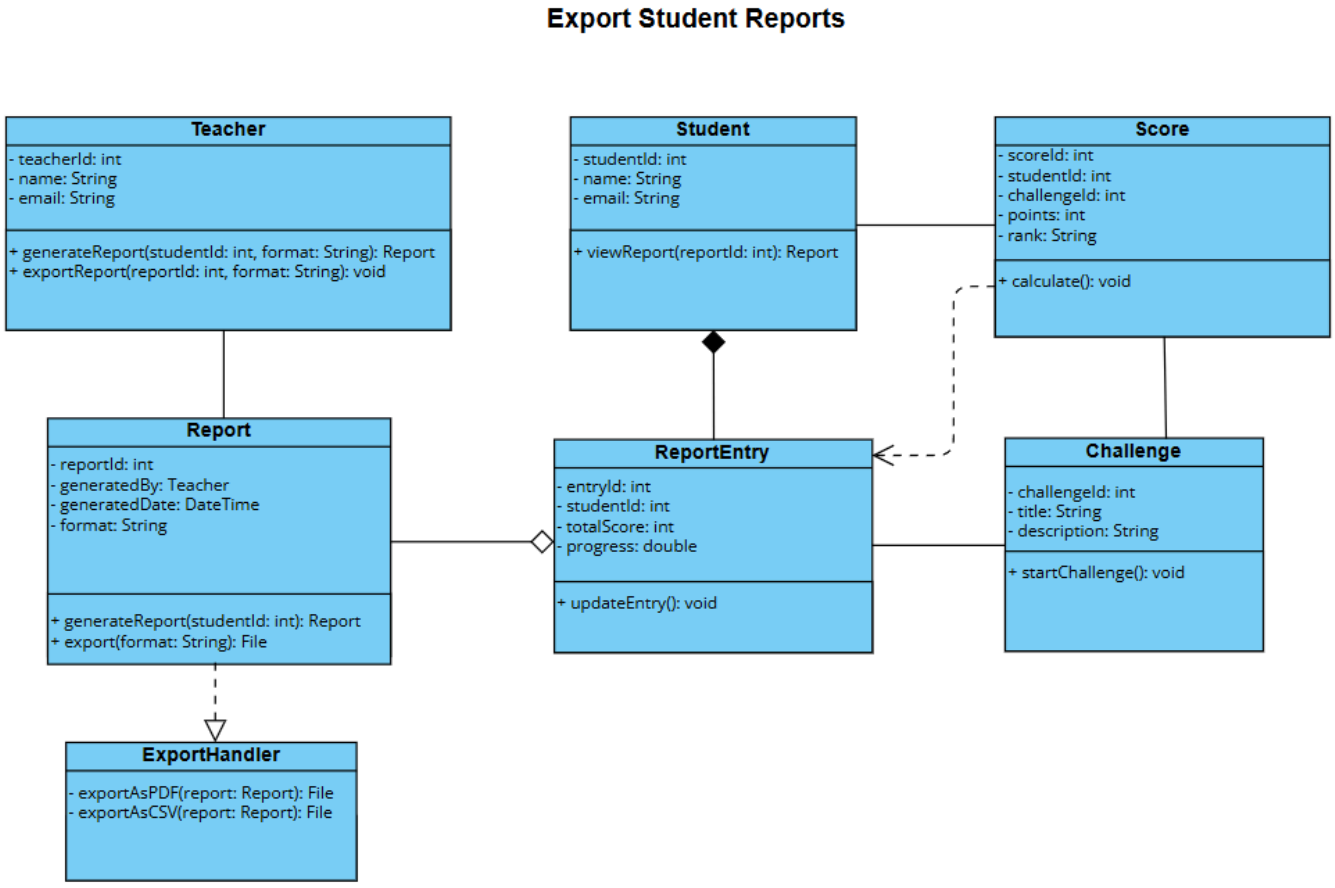
PDF format.

- ➢ **Component type:** Java Library (e.g., Apache PDFBox, iText)

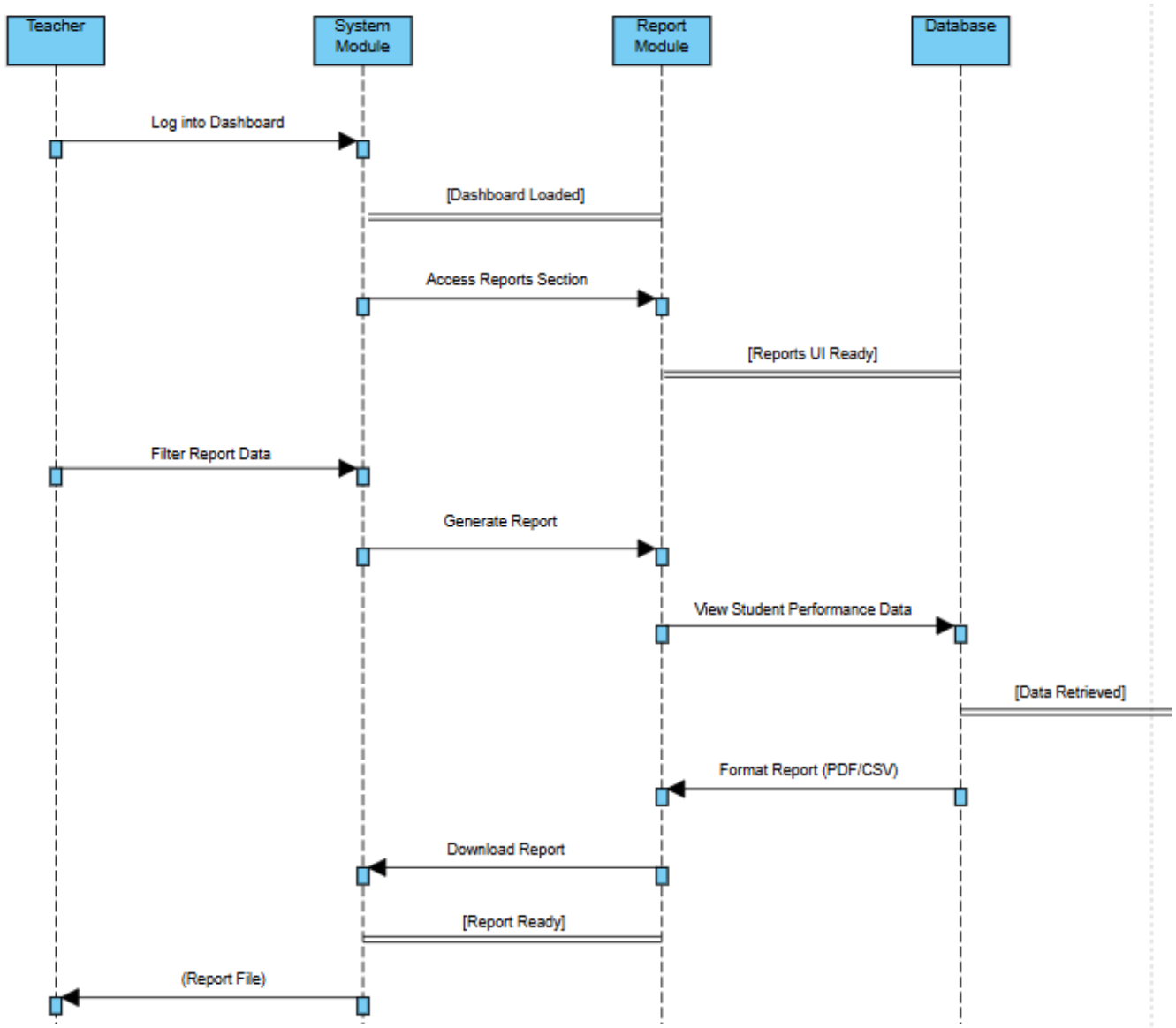
6. **CSVGenerator Utility**
   - ➢ **Description and purpose:** Formats student data into a downloadable CSV file.
   - ➢ **Component type:** Java Library (e.g., OpenCSV)

Object-Oriented Components

- Class Diagram

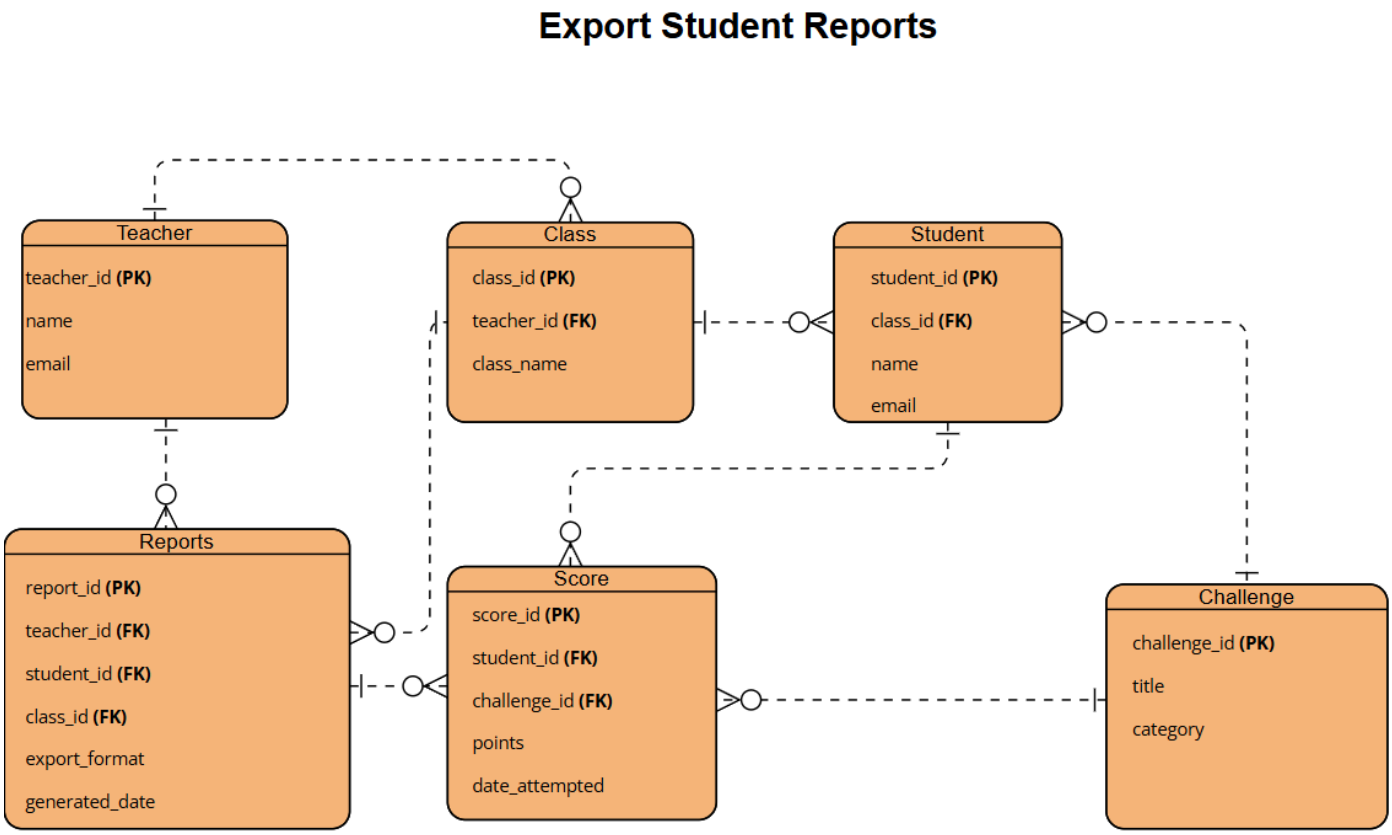


- Sequence Diagram



Data Design

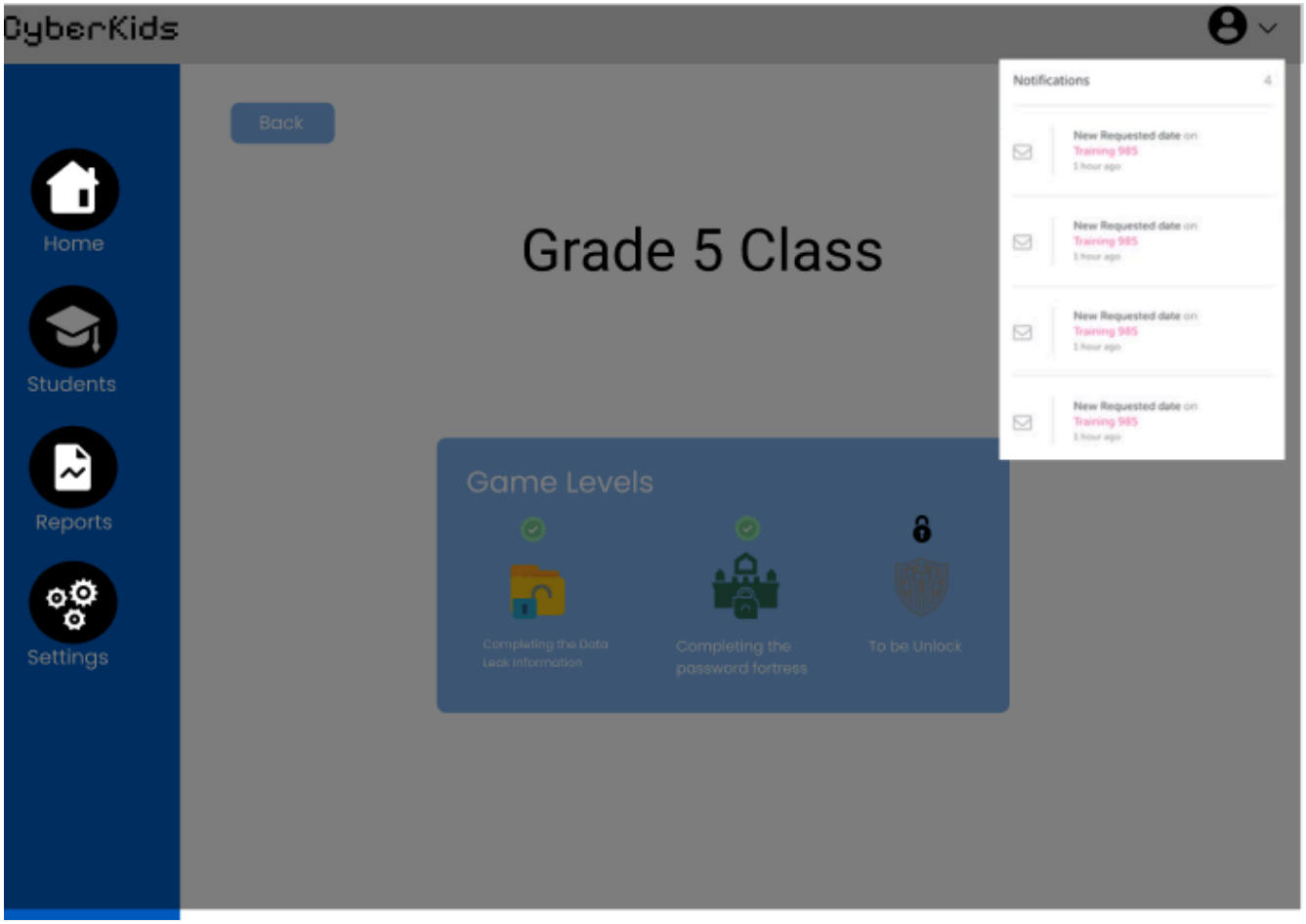
- ERD or schema



Module 4: Notification System

Transaction 4.1 Retrieve Notifications

User Interface Design



Front-end component(s)

1. **Notifications.jsx**
   - ➢ **Description and purpose:** This component is responsible for displaying real-time notifications to teachers. It retrieves the notification history via an API call when mounted and opens a WebSocket connection to receive live notifications pushed from the backend.
   - ➢ **Component type:** React UI Component typically placed in the src/components directory.
2. **websocket.js**
   - ➢ **Description and purpose:** This module manages the WebSocket connection using STOMP over SockJS. It handles establishing the connection, subscribing to the teacher's notification topic, receiving messages, and disconnecting cleanly.
   - ➢ **Component type:** JavaScript utility module typically placed in the src directory or a utilities folder like src/utils.
3. **Dashboard.jsx**
   - ➢ **Description and purpose:** The main teacher dashboard page imports and renders the Notifications component, passing the teacher's ID to it. It also renders other UI components such as level toggles and player reassignment.
   - ➢ **Component type:** React Page Component typically placed in the src/pages directory.

Back-end component(s)

1. **WebSocketConfig.java**
   - ➢ **Description and Purpose:**

     Configures WebSocket messaging using STOMP protocol. Defines the endpoint /ws-notifications for clients to connect, and sets the message broker with /app as prefix for sending messages and /topic for subscribing.
   - ➢ **Component Type:**

     Java Configuration class (@Configuration) – typically placed under src/main/java/.../config/WebSocketConfig.java.
2. **NotificationEntity**
   - ➢ **Description and Purpose:**

     Entity representing a notification with fields for recipient (teacherId), message, and timestamp. Used for persisting notification history in the database.
   - ➢ **Component Type:**

     JPA Entity – typically placed under src/main/java/.../entity/Notification.java.
3. **NotificationRepository**
   - ➢ **Description and Purpose:**

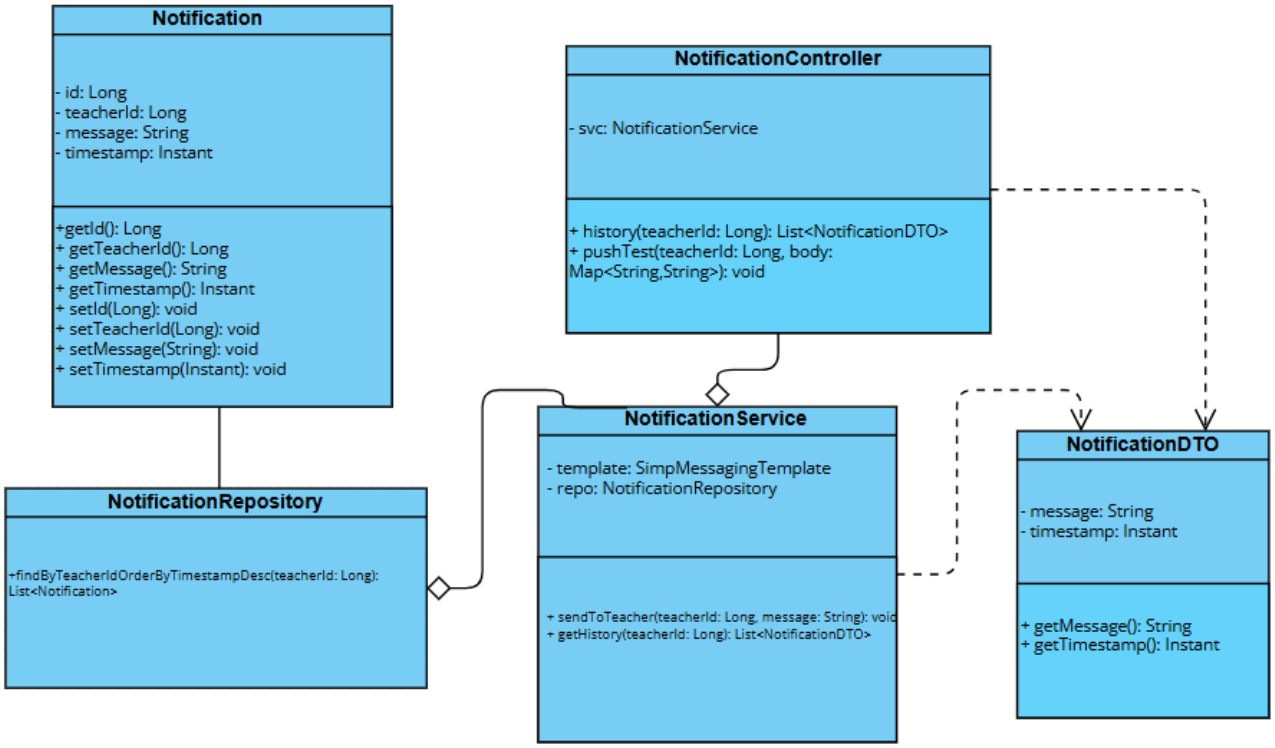
     Provides access methods to retrieve notifications, particularly findByTeacherIdOrderByTimestampDesc(Long) for fetching a teacher's notification history sorted by time.
   - ➢ **Component Type:** Spring Data JPA Repository Interface – typically placed under src/main/java/.../repository/NotificationRepository.java.

4. **NotificationDTO**
   - ➢ **Description and Purpose:**
     Data Transfer Object (DTO) sent to the frontend over WebSocket or REST. Contains only necessary fields: message and timestamp.
   - ➢ **Component Type:** DTO Class – typically placed under src/main/java/.../dto/NotificationDTO.java.
5. **NotificationService**
   - ➢ **Description and Purpose:**
     Broadcasts them to subscribed WebSocket clients via SimpMessagingTemplate.
   - ➢ **Component Type:** Service Class – typically placed under src/main/java/.../service/NotificationService.java.
6. **NotificationController**
   - ➢ **Description and Purpose:** Retrieves past notifications and sends a test notification manually (optional)
   - ➢ **Component Type:** REST Controller – typically placed under src/main/java/.../controller/NotificationController.java.

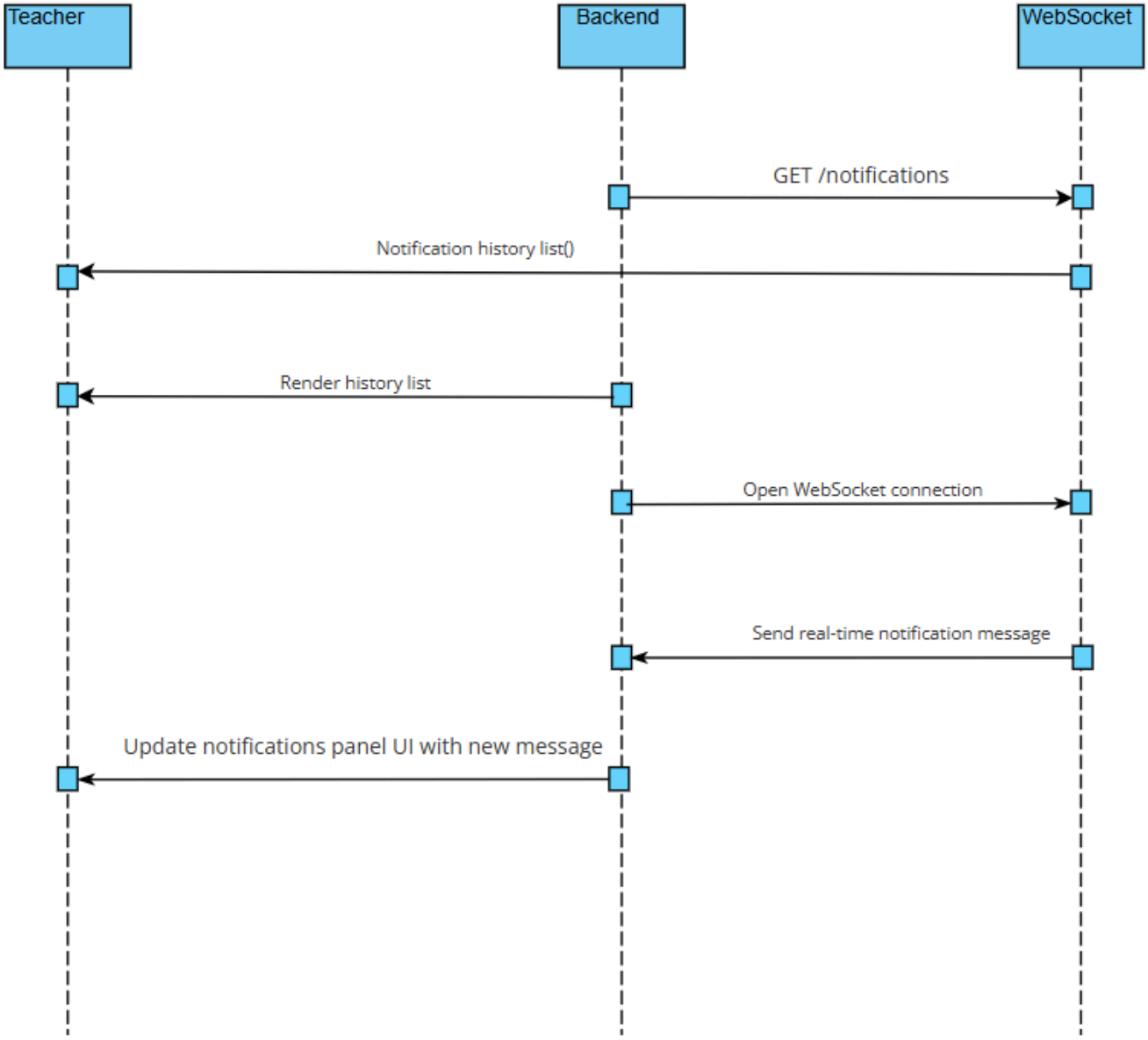
Object-Oriented Components

- Class Diagram

Transaction 4.1 Retrieve Notifications

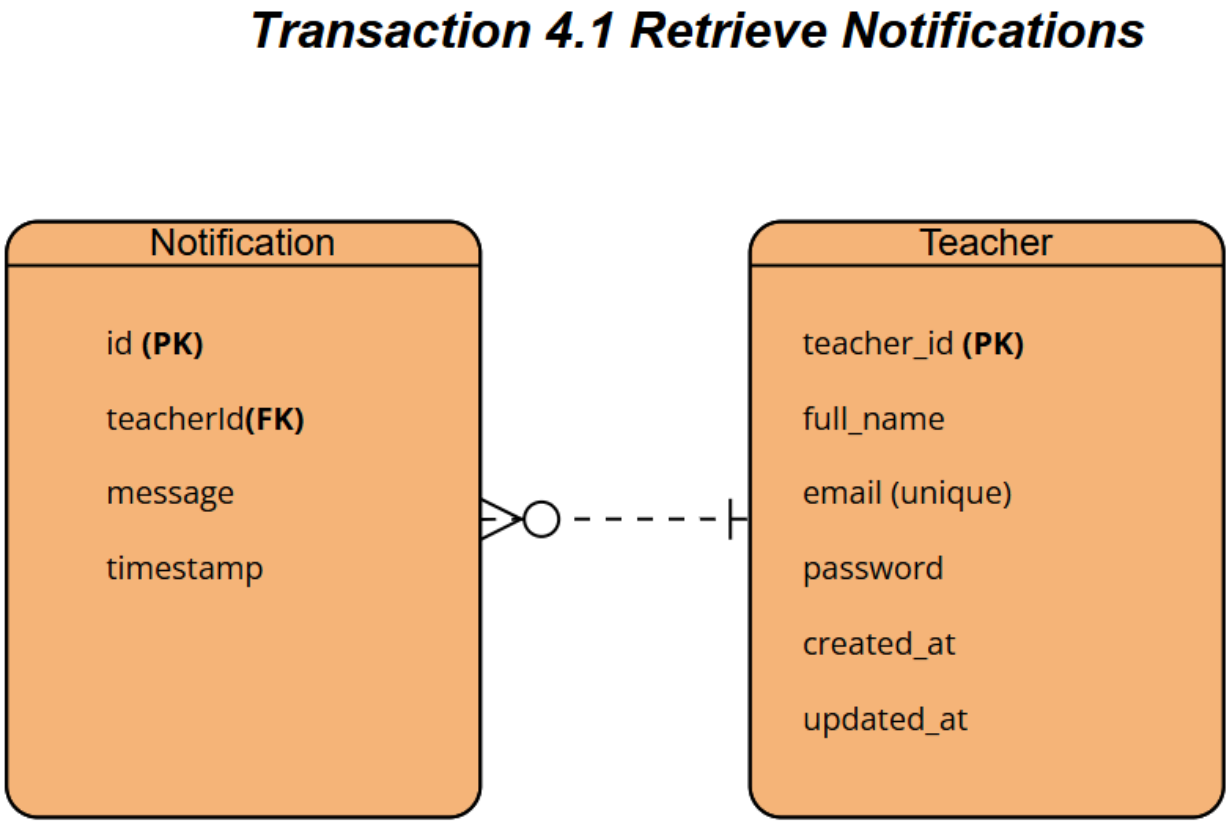


- Sequence Diagram



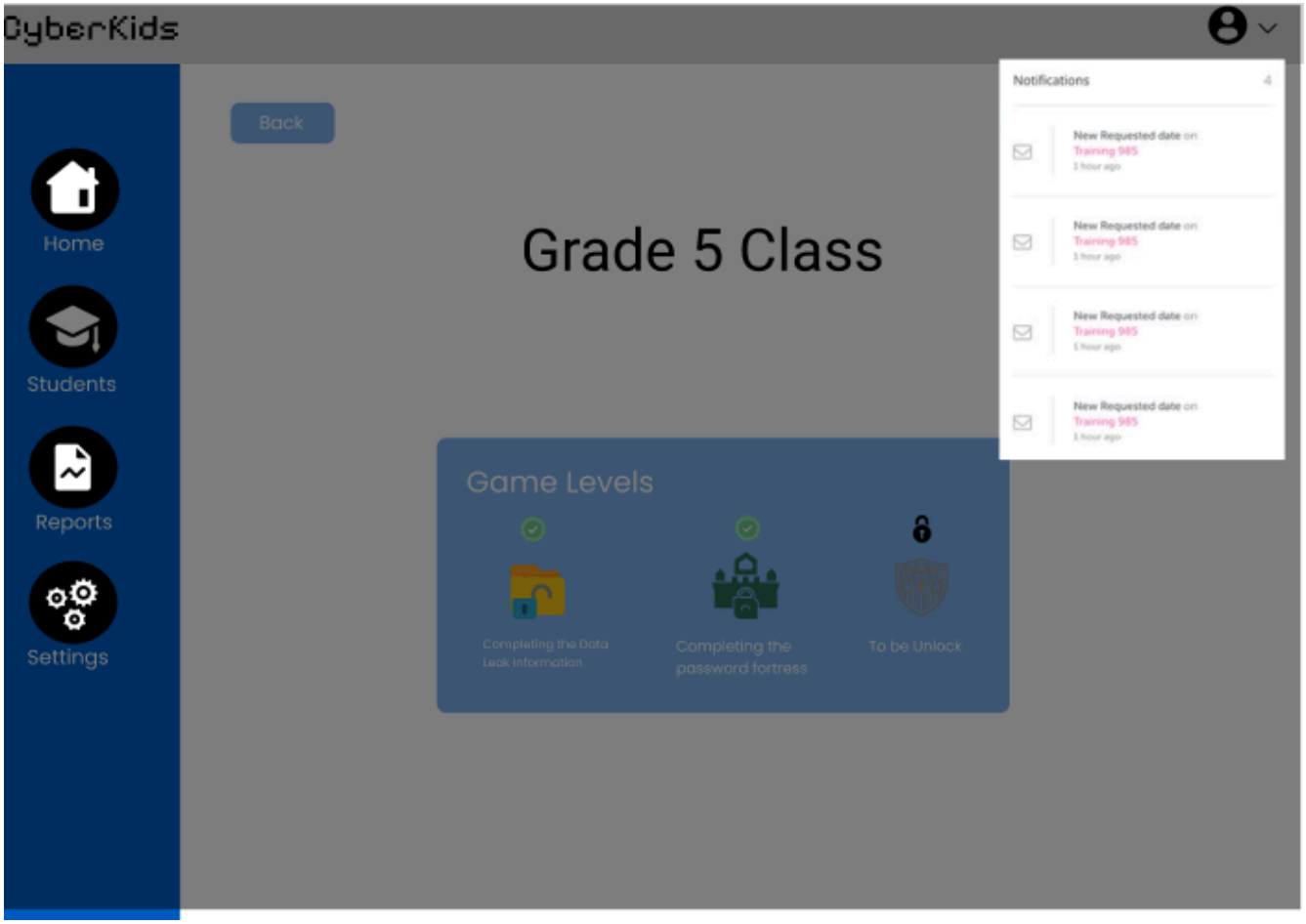
Data Design

- ERD or schema



Transaction 4.2 Mark Notification as Read/Unread

User Interface Design



Front-end component(s)

1. **Notifications.jsx**
   - ➢ **Description and Purpose:**

     Displays the list of teacher notifications, including both read and unread. Connects to the WebSocket for real-time updates and allows teachers to toggle notifications between read and unread using a button. Updates the UI accordingly based on the read status of each item.
   - ➢ **Component Type:**

     React UI Component – typically placed under src/components/Notifications.jsx.
2. **Dashboard.jsx**
   - ➢ **Description and Purpose:**

     Main page for the teacher's dashboard. Embeds the Notifications component to display and manage the teacher's notification feed in real-time.
   - ➢ **Component Type:**

     React UI Component – typically placed under src/components/Dashboard.jsx or src/pages/Dashboard.jsx.
3. **ws-notifications.js**
   - ➢ **Description and Purpose:**

     Handles WebSocket connection logic. Manages subscribing to teacher-specific notification topics and delivering real-time notification data to the Notifications component.
   - ➢ **Component Type:**

     WebSocket Utility Module – typically placed under src/ws-notifications.js.

Back-end component(s)

1. **Notification.java**
   - ➢ **Description and Purpose:**

     Represents the notification entity stored in the database. Extended to include a read field which defaults to false, allowing notifications to be marked as read or unread.
   - ➢ **Component Type:**

     JPA Entity – typically placed under src/main/java/.../model/Notification.java.
2. **NotificationRepository**
   - ➢ **Description and Purpose:**

     Data access layer for interacting with the Notification table. Includes custom methods to retrieve all notifications, as well as only unread ones for a specific teacher.
   - ➢ **Component Type:**

     Spring Data JPA Repository – typically placed under src/main/java/.../repository/NotificationRepository.java.
3. **MarkReadRequest.java**
   - ➢ **Description and Purpose:**

     A simple DTO used in the PATCH request to specify whether a notification should be marked as read or unread.
   - ➢ **Component Type:**

     POJO (Plain Old Java Object) / DTO – typically placed under src/main/java/.../dto/MarkReadRequest.java.
4. **NotificationDTO.java**
   - ➢ **Description and Purpose:**

     A simple DTO used in the PATCH request to specify whether a notification should be marked as read or unread.
   - ➢ **Component Type:**

     Data Transfer Object – typically placed under src/main/java/.../dto/NotificationDTO.java.

5. **NotificationService.java**
   - ➢ **Description and Purpose:**

     Encapsulates business logic for managing notifications. Provides a markRead method to safely validate and update a notification's read status.
   - ➢ **Component Type:**

     Spring Service – typically placed under src/main/java/.../service/NotificationService.java.
6. **NotificationController.java**
   - ➢ **Description and Purpose:**

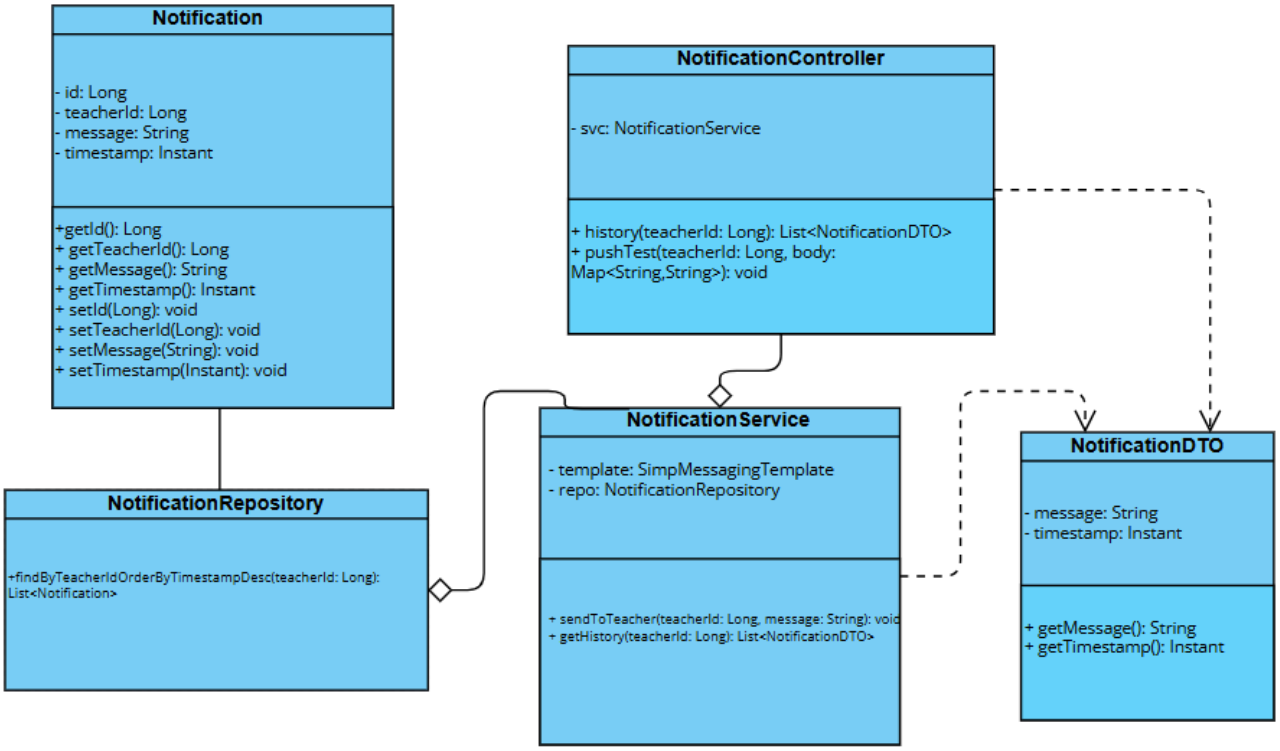
     REST controller exposing endpoints for retrieving all or only unread notifications, and marking specific notifications as read/unread. Routes include GET for history and PATCH for updates.
   - ➢ **Component Type:**

     Spring REST Controller – typically placed under src/main/java/.../controller/NotificationController.java.

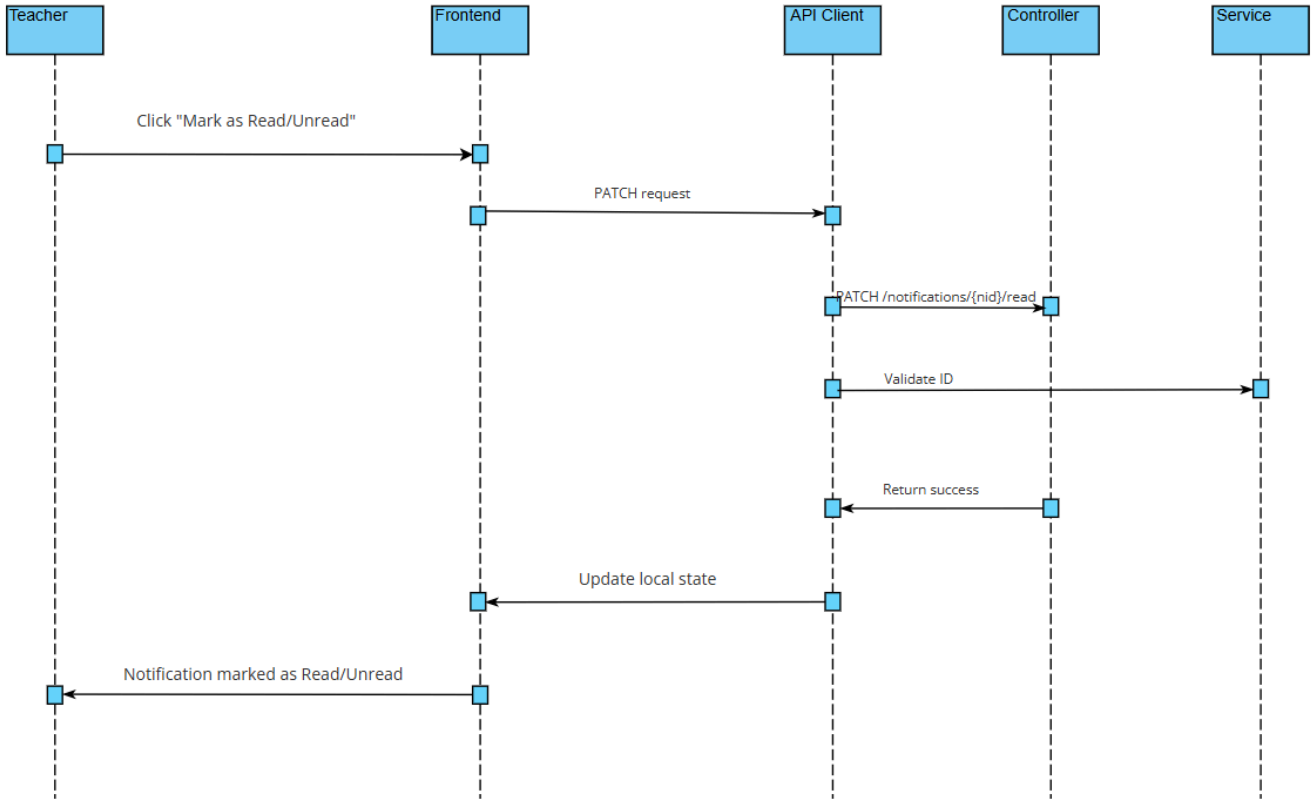
Object-Oriented Components

- Class Diagram

Transaction 4.2 Mark Notification as Read/Unread



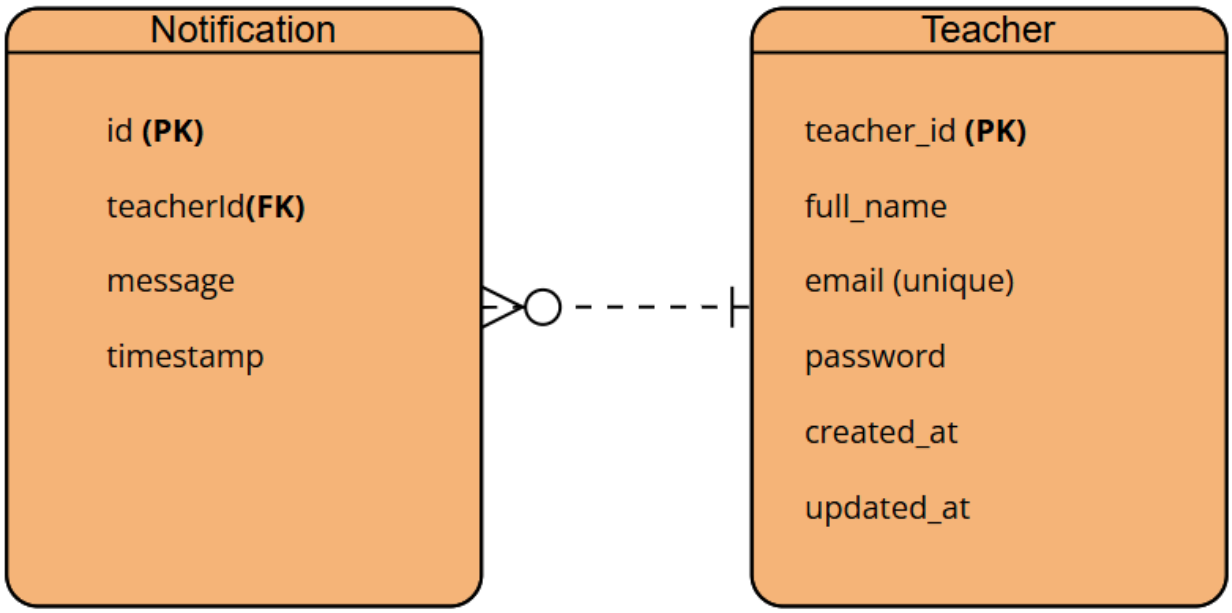
- Sequence Diagram



Data Design

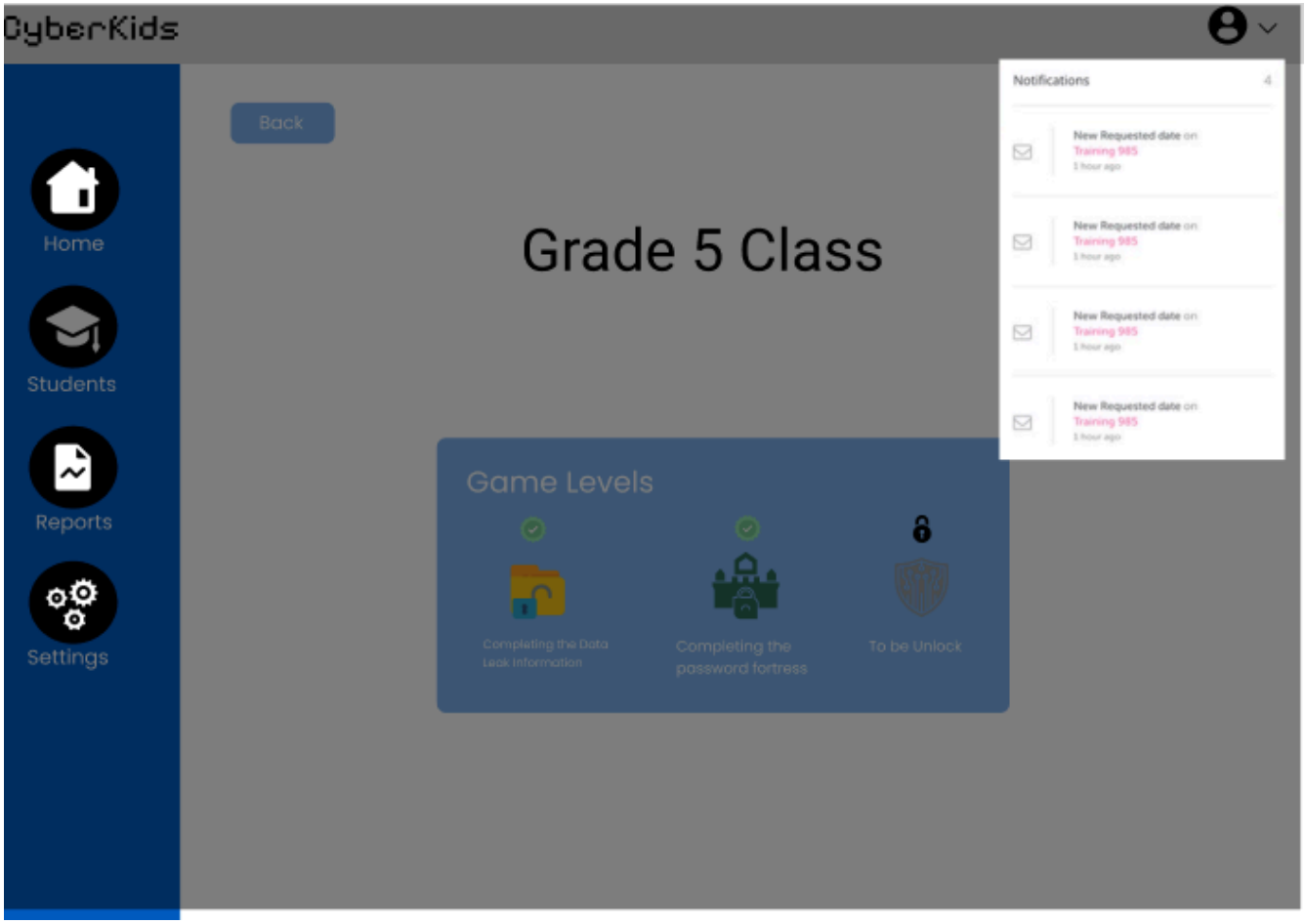
- ERD or schema

Transaction 4.2 Mark Notification as Read/Unread



Transaction 4.3 Delete Notification

User Interface Design



Front-end component(s)

1. **Notifications.jsx**
   - ➢ **Description and Purpose:**

     Displays the list of teacher notifications, including both read and unread. Connects to the WebSocket for real-time updates and allows teachers to toggle notifications between read and unread using a button. Updates the UI accordingly based on the read status of each item.
   - ➢ **Component Type:**

     React UI Component – typically placed under src/components/Notifications.jsx.
2. **Dashboard.jsx**
   - ➢ **Description and Purpose:**

     Main page for the teacher's dashboard. Embeds the Notifications component to display and manage the teacher's notification feed in real-time.
   - ➢ **Component Type:**

     React UI Component – typically placed under src/components/Dashboard.jsx or src/pages/Dashboard.jsx.
3. **ws-notifications.js**
   - ➢ **Description and Purpose:**

     Handles WebSocket connection logic. Manages subscribing to teacher-specific notification topics and delivering real-time notification data to the Notifications component.
   - ➢ **Component Type:**

     WebSocket Utility Module – typically placed under src/ws-notifications.js.

Back-end component(s)

1. **Notification.java**
   - ➢ **Description and Purpose:**

     Represents the notification entity stored in the database. Extended to include a read field which defaults to false, allowing notifications to be marked as read or unread.
   - ➢ **Component Type:**

     JPA Entity – typically placed under src/main/java/.../model/Notification.java.
2. **NotificationRepository**
   - ➢ **Description and Purpose:**

     Data access layer for interacting with the Notification table. Includes custom methods to retrieve all notifications, as well as only unread ones for a specific teacher.
   - ➢ **Component Type:**

     Spring Data JPA Repository – typically placed under src/main/java/.../repository/NotificationRepository.java.
3. **MarkReadRequest.java**
   - ➢ **Description and Purpose:**

     A simple DTO used in the PATCH request to specify whether a notification should be marked as read or unread.
   - ➢ **Component Type:**

     POJO (Plain Old Java Object) / DTO – typically placed under src/main/java/.../dto/MarkReadRequest.java.
4. **NotificationDTO.java**
   - ➢ **Description and Purpose:**

     A simple DTO used in the PATCH request to specify whether a notification should be marked as read or unread.
   - ➢ **Component Type:**

     Data Transfer Object – typically placed under src/main/java/.../dto/NotificationDTO.java.

5. **NotificationService.java**
   - ➢ **Description and Purpose:**

     Encapsulates business logic for managing notifications. Provides a markRead method to safely validate and update a notification's read status.
   - ➢ **Component Type:**

     Spring Service – typically placed under src/main/java/.../service/NotificationService.java.
6. **NotificationController.java**
   - ➢ **Description and Purpose:**

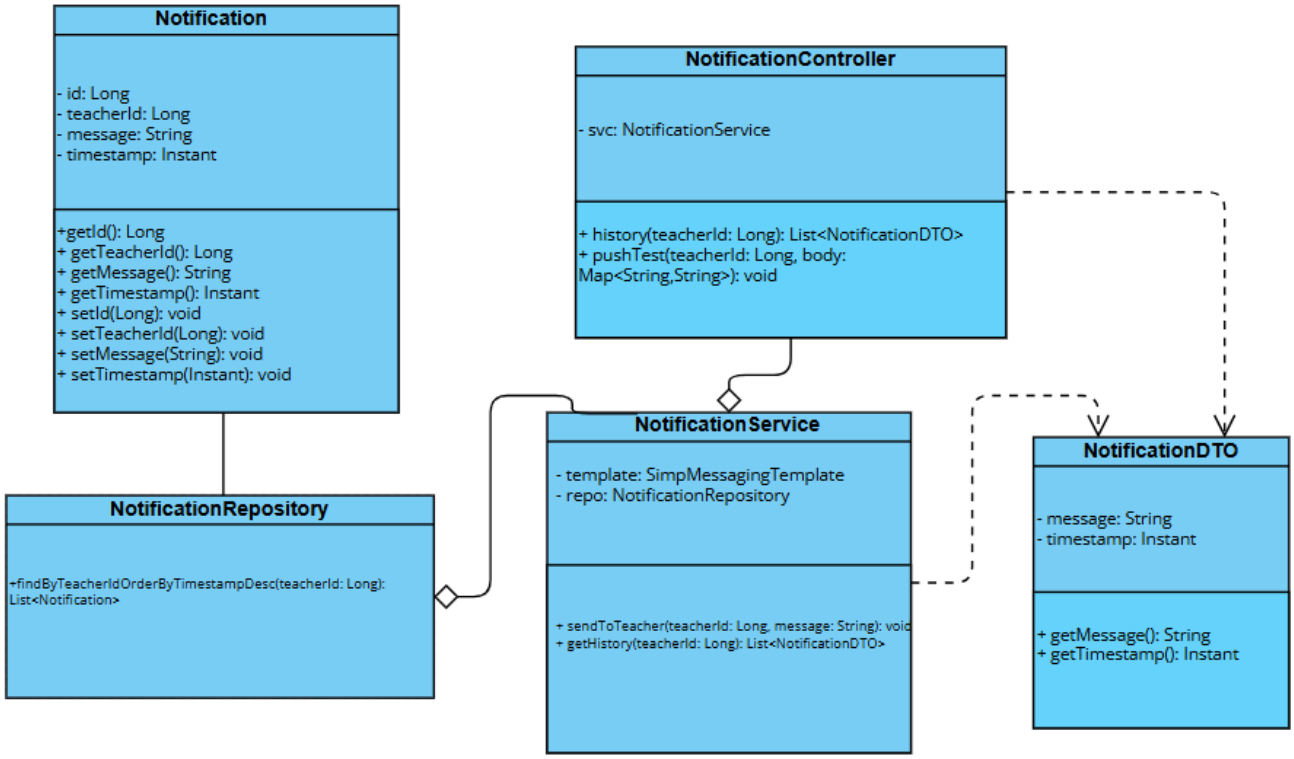
     REST controller exposing endpoints for retrieving all or only unread notifications, and marking specific notifications as read/unread. Routes include GET for history and PATCH for updates.
   - ➢ **Component Type:**

     Spring REST Controller – typically placed under src/main/java/.../controller/NotificationController.java.

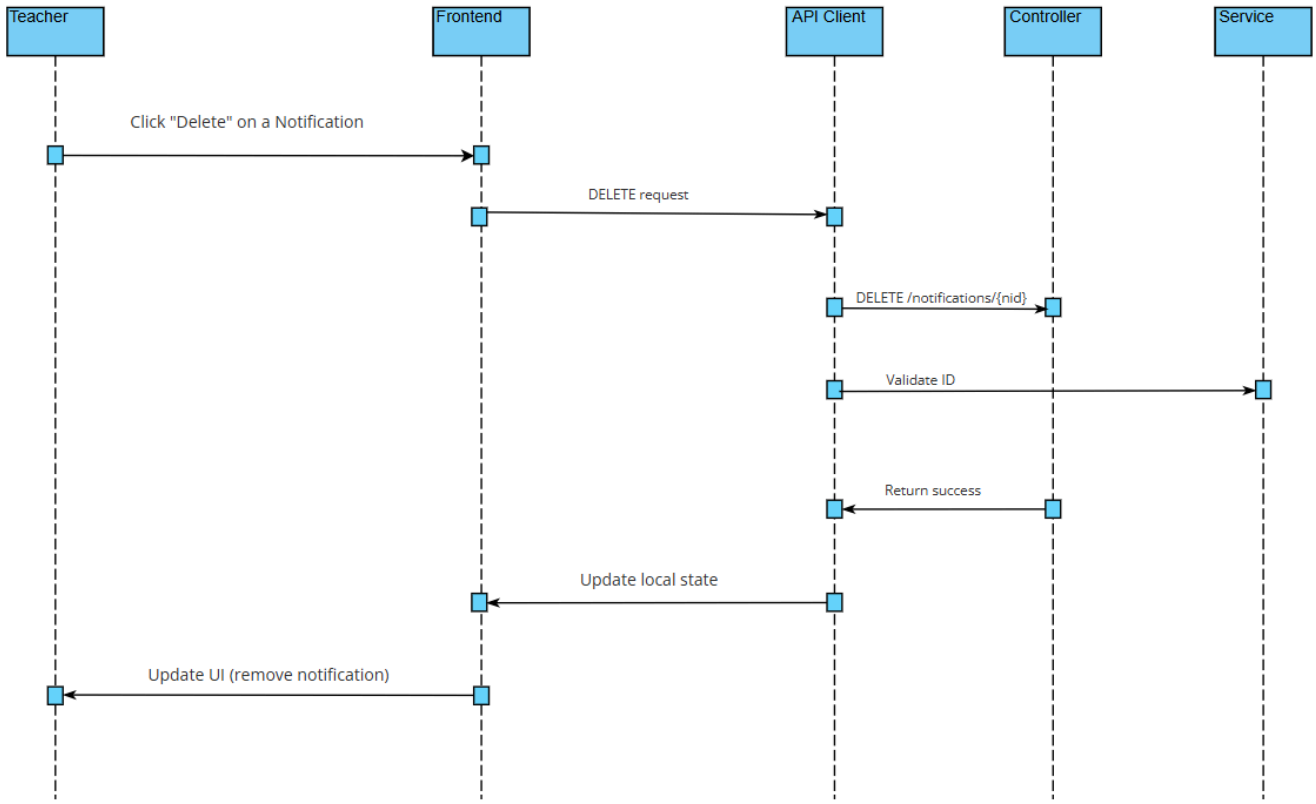
Object-Oriented Components

- Class Diagram

Transaction 4.3 Delete Notification



- Sequence Diagram



Data Design

- ERD or schema