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	Updated the Software Design Document (SDD) with numerous enhancements and structural changes, with further refinements pending finalization.	Cultura
3.0	9/4/2025	Finalized and updated the Software Design Document (SDD) to reflect the complete and current system architecture, component responsibilities, and integration flow for Capstone 2.	Baguio Cultura Dedumo Vequiso

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.

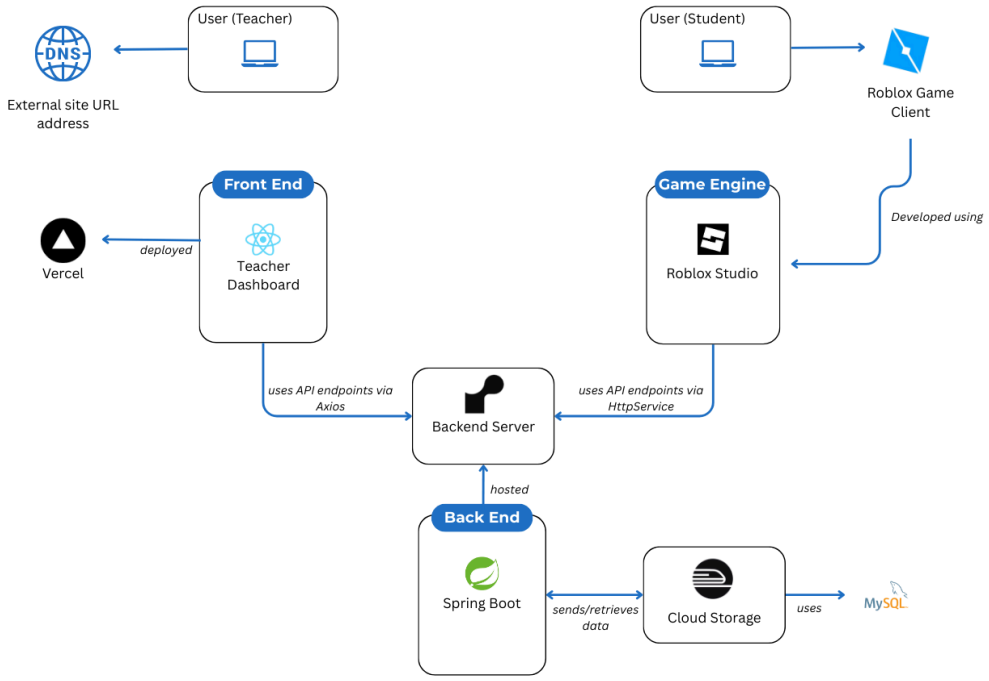
[2] Roblox Corporation. (n.d.). Roblox API Reference. Source: https://create.roblox.com/docs/reference/engine

[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 technologies used in the front-end and back-end.



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 viewpoints 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. **Remote Events**
   - ➢ **Description and Purpose:**
     - Handles communication between the server and clients for a wide range of actions including game start/end, player interactions, object updates, etc.
   - ➢ **Component Type:** RemoteEvent instances 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. **Timer**
   - ➢ **Description and Purpose:**
     Handles countdowns, time limits, or elapsed time tracking for a session or challenge.
   - ➢ **Component Type:**
     Script or LocalScript with custom timer logic (can update UI elements)
8. **Score System**
   - ➢ **Description and Purpose:**
     Tracks and updates player scores based on in-game actions. May display leaderboard or final score.
   - ➢ **Component Type:**
     Script or LocalScript
9. **Leaderboard**
   - ➢ **Description and Purpose:**
     Displays the top scores or stats for current or all-time players. Useful for competition and engagement.
   - ➢ **Component Type:**
     Script or LocalScript

Back-end component(s)

1. Student Entity

- **Description and Purpose:**
  Represents the database table for students/players. Stores fields like id, robloxId, robloxName, realName, and status-related fields (e.g., online, targetWorld, targetLevel). Used to uniquely identify and manage students across game sessions.
- **Component Type:**
  Java class annotated with @Entity (located in the entity package)

2. StudentRequest DTO (Data Transfer Object)

- **Description and Purpose:**
  Used to receive data from the Roblox game client when registering or manually registering a new student. Includes fields like robloxId, robloxName, realName, and classCode.
- **Component Type:**
  Java POJO (Plain Old Java Object), used in HTTP requests (typically in the model or dto package)

3. StudentController

- **Description and Purpose:**
  Handles student-related API endpoints, including automatic and manual registration, fetching active scenarios, student data lookup, and retrieving NPC info for the game. Ensures proper notification of student status changes.
- **Component Type:**
  Java class annotated with @RestController (located in the controller package)

4. StudentService

- **Description and Purpose:**
  Contains core business logic for saving, retrieving, or deleting student records, separated from controller logic for modularity.
- **Component Type:**
  Java class annotated with @Service (located in the service package)

5. StudentRepo

- **Description and Purpose:**
  Provides data access methods for interacting with the Student entity in the database.
- **Component Type:**
  Java interface extending JpaRepository<Student, Long>, annotated with @Repository

6. Timer Entity

- **Description and Purpose:**
  Represents timing details such as start time, end time, and duration for a student's challenge session.
- **Component Type:**
  Java class annotated with @Entity (located in the entity package)

7. TimerRepo

- **Description and Purpose:**
  Manages database operations for Timer entities.
- **Component Type:**
  Java interface extending JpaRepository<Timer, Long> (located in the repository package)

8. TimerService

- **Description and Purpose:**
  Handles logic to create, update, and retrieve timer data for individual students during gameplay.
- **Component Type:**
  Java class annotated with @Service (located in the service package)

9. TimerController

- **Description and Purpose:**
  Receives and processes timing-related API requests from the Roblox client.
- **Component Type:**
  Java class annotated with @RestController (located in the controller package)

10. Score Entity

- **Description and Purpose:**
  Records and stores individual scores achieved by students during scenarios or challenges. Also used to support leaderboard computations.
- **Component Type:**
  Java class annotated with @Entity (located in the entity package)

11. ScoreRepo

- **Description and Purpose:**
  Provides CRUD operations for the Score entity using Spring Data JPA.
- **Component Type:**
  Java interface extending JpaRepository<Score, Long> (located in the repository package)

12. ScoreService

- **Description and Purpose:**
  Contains logic for calculating, storing, and retrieving score data. May also ensure student existence when scores are submitted.
- **Component Type:**
  Java class annotated with @Service (located in the service package)

13. ScoreController

- **Description and Purpose:**
  Accepts HTTP requests to submit or retrieve scores. May also trigger student creation if they don't already exist.
- **Component Type:**
  Java class annotated with @RestController (located in the controller package)

14. Teacher Entity

- **Description and Purpose:**
  Represents authenticated teacher accounts in the system. Stores teacher profile information, email, password, roles, and the worlds they’ve locked. Used to authorize scenario creation and class access.
- **Component Type:**
  Java class annotated with @Entity (located in the entity package)

15. Scenario Entity

- **Description and Purpose:**
  Represents the pool of customizable information items (scenarios) configured by teachers. Includes question content, the correct answer (SAFE or UNSAFE), and class linkage. Can be toggled active/inactive.
- **Component Type:**
  Java class annotated with @Entity (located in the entity package)

16. ScenarioController

- **Description and Purpose:**
  Exposes API endpoints for teachers to create, update, toggle, retrieve, and delete their own scenarios. Also provides counts and active scenario queries.
- **Component Type:**
  Java class annotated with @RestController (located in the controller package)

17. ScenarioService

- **Description and Purpose:**
  Contains business logic for all scenario-related operations: creation, validation, filtering by teacher or status, and authorization checks.
- **Component Type:**
  Java class annotated with @Service (located in the service package)

18. ScenarioDTO

- **Description and Purpose:**
  A Data Transfer Object for returning sanitized scenario data from the backend to clients (e.g., content, active status, timestamps).
- **Component Type:**
  Java POJO (Plain Old Java Object), typically found in the dto package

19. ScenarioRepository

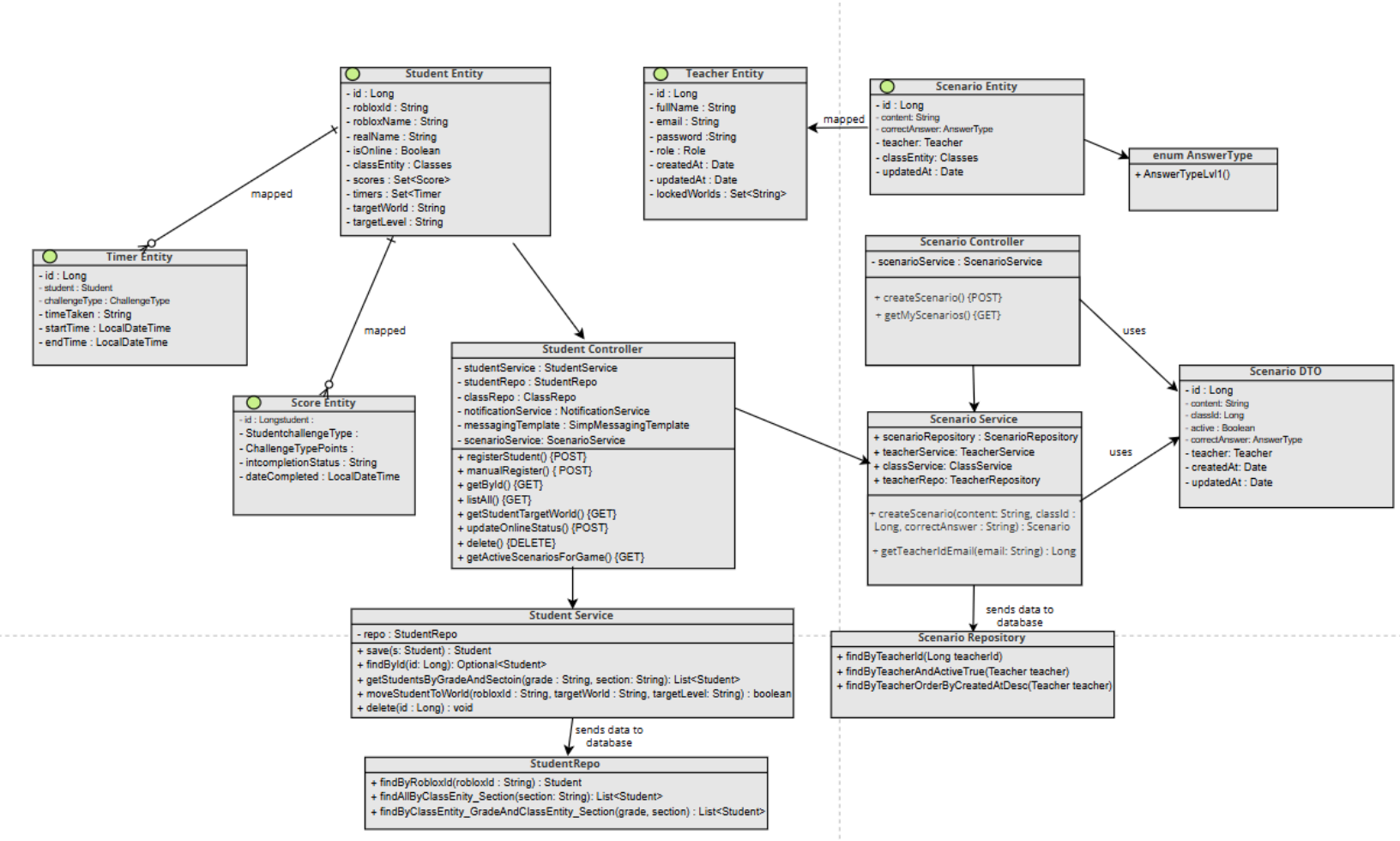
- **Description and Purpose:**
  Provides Spring Data JPA methods to retrieve scenarios by teacher ID, status (active), and more. Includes custom query for all active scenarios.
- **Component Type:**
  Java interface extending JpaRepository<Scenario, Long> (located in the repository package)

20. Enum: AnswerTypeLvl1

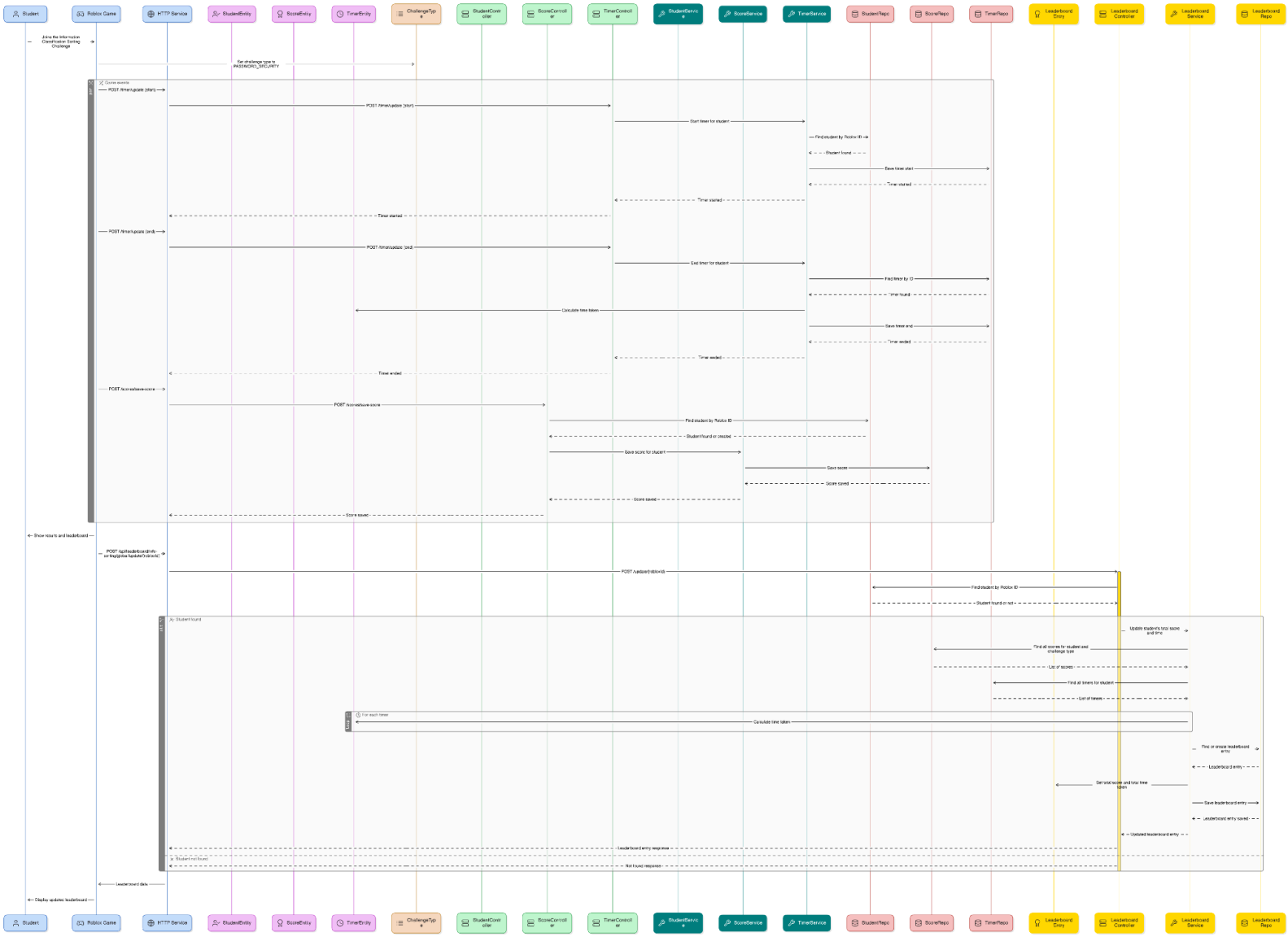
- **Description and Purpose:**
  Enum defining valid answers to scenarios (e.g., SAFE, UNSAFE). Ensures type safety in scenario content and correctness validation.
- **Component Type:**
  Java enum (typically located in the model package)

Object-Oriented Components

- Class Diagram

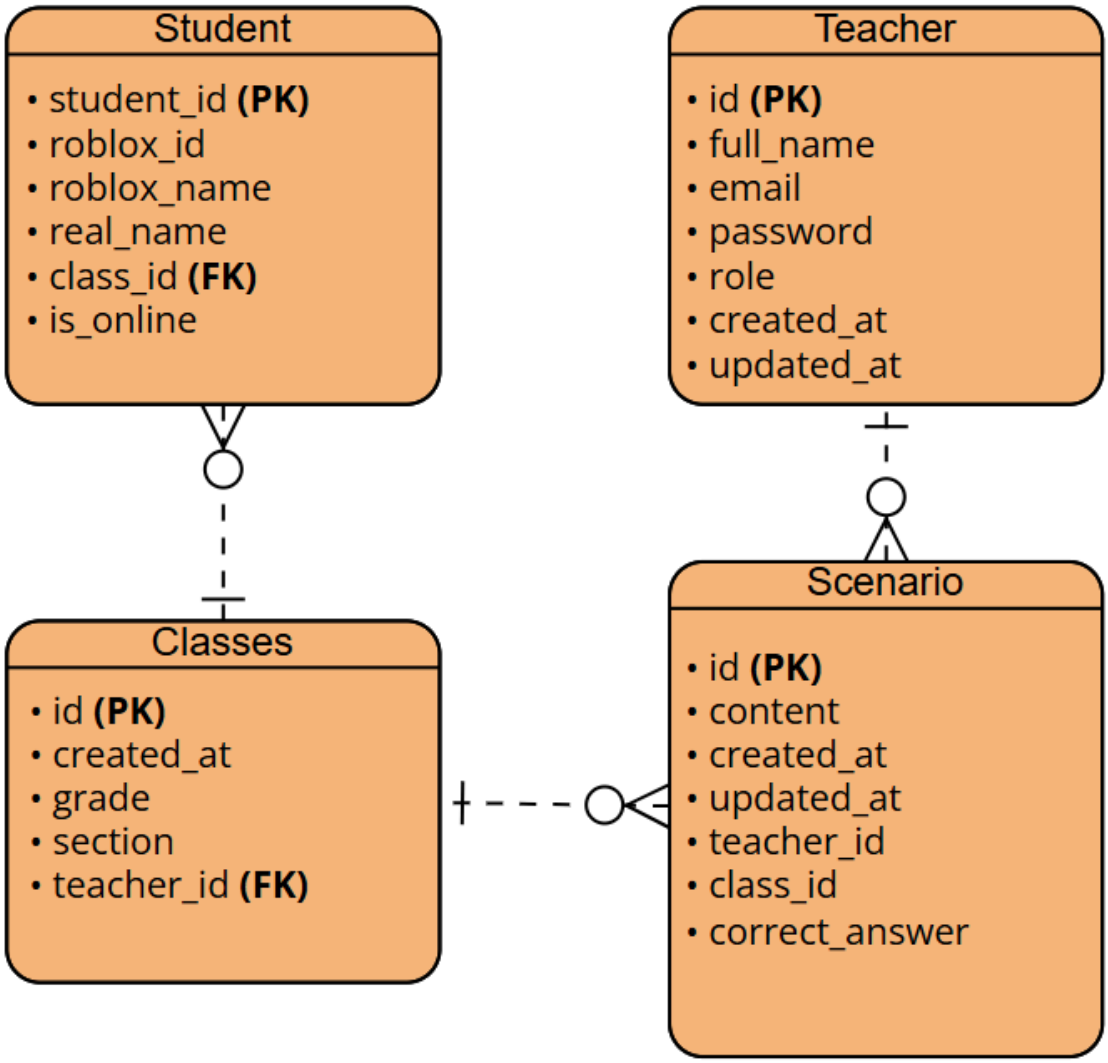


- Sequence Diagram



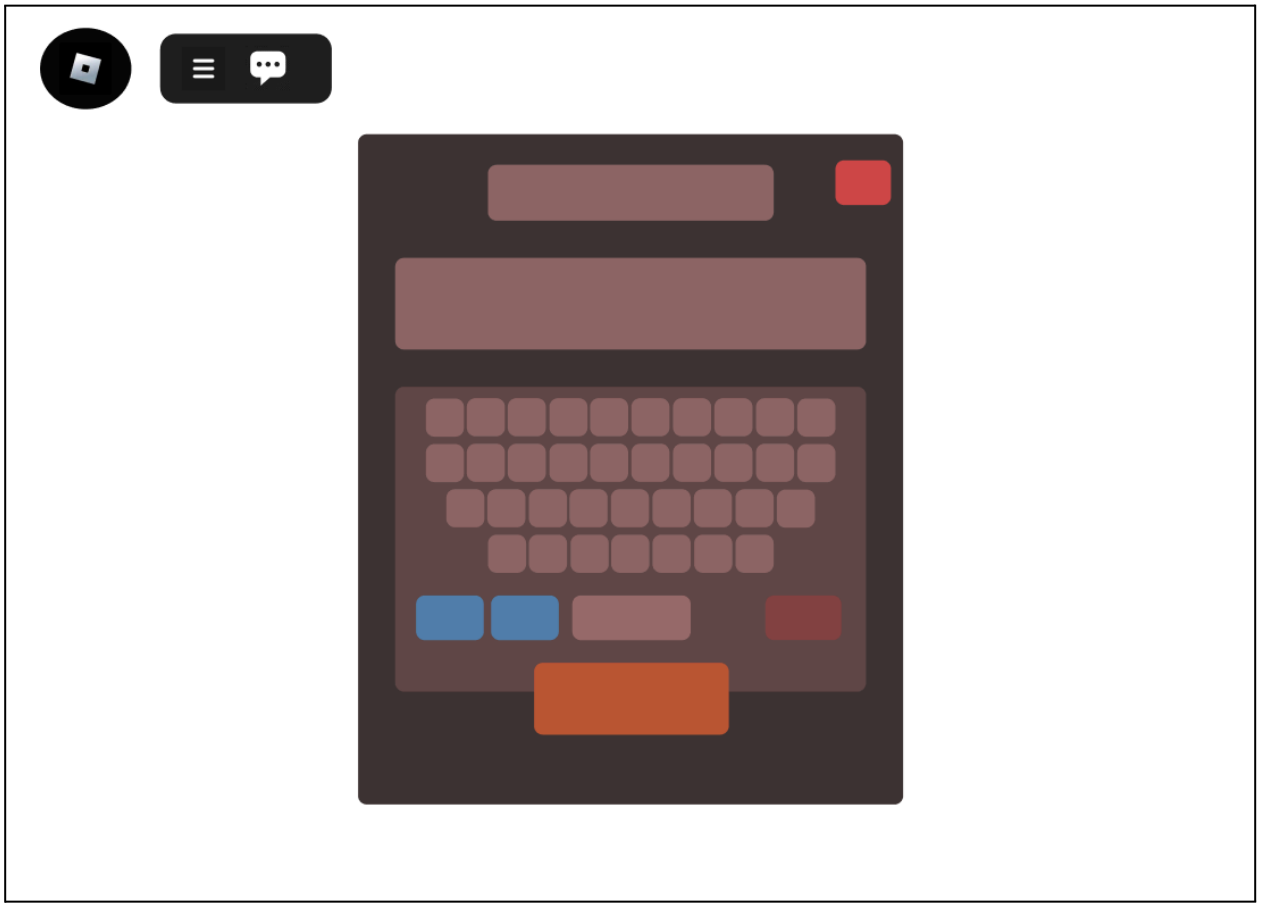
Data Design

- ERD or schema



Transaction 1.2 Password Security Challenge

User Interface Design



Front-end component(s)

1. Workspace Objects

• **Description and purpose:** Static and interactive 3D models placed in the game world. These 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 viewpoints during game start, end, or transitions. Enhances player immersion and guides player focus.

• **Component type:** Roblox Camera objects, manipulated by LocalScripts

3. Sounds / Music

• **Description and purpose:** Background music and sound effects triggered during cutscenes, gameplay, or transitions to enhance the atmosphere and feedback.

• Component type: Roblox Sound instances (placed in SoundService, Workspace, or StarterGui)

4. Cutscenes

• **Description and purpose**: Pre-scripted sequences that control camera, animations, and player input to introduce the story or environment. Typically shown at the start or between gameplay segments.

• Component type: LocalScript using TweenService, Camera manipulation, and UI elements

5. Remote Events

• **Description and purpose**: Enables communication between the server and clients for actions like game state changes, player actions, or updates. Used to trigger remote functions or sync states.

• Component type: RemoteEvent instances in ReplicatedStorage

6. RegisterAPI (HTTP Module)

• **Description and purpose**: Sends the player's Roblox UserId to the backend API to register or update student/player data. Prevents the need for manual registration within the game.

• Component type: ModuleScript or Script using HttpService:PostAsync()

7. Timer

• **Description and purpose**: Tracks countdowns, time limits, or elapsed durations during challenges. Often updates visible UI to reflect time status.

• Component type: Script or LocalScript with custom timer logic

8. Score System

• **Description and purpose**: Tracks and updates player scores based on correct actions or scenario completions. May include submission to backend or leaderboard systems.

• Component type: Script or LocalScript

9. Leaderboard

• **Description and purpose**: Displays real-time or historical ranking of players based on their scores. Encourages competition and replayability.

• Component type: Script or LocalScript (can interact with UI elements and back-end via HTTP)

Back-end component(s)

1. Student Entity

- **Description and Purpose:**
  Represents the database table for students/players. Stores fields like id, robloxId, robloxName, realName, and status-related fields (e.g., online, targetWorld, targetLevel). Used to uniquely identify and manage students across game sessions.

- **Component Type:**
  Java class annotated with @Entity (located in the entity package)

2. StudentRequest DTO (Data Transfer Object)

- **Description and Purpose:**
  Used to receive data from the Roblox game client when registering or manually registering a new student. Includes fields like robloxId, robloxName, realName, and classCode.

- **Component Type:**
  Java POJO (Plain Old Java Object), used in HTTP requests (typically in the model or dto package)

3. StudentController

- **Description and Purpose:**
  Handles student-related API endpoints, including automatic and manual registration, fetching active scenarios, student data lookup, and retrieving NPC info for the game. Ensures proper notification of student status changes.

- **Component Type:**
  Java class annotated with @RestController (located in the controller package)

4. StudentService

- **Description and Purpose:**
  Contains core business logic for saving, retrieving, or deleting student records, separated from controller logic for modularity.

- **Component Type:**
  Java class annotated with @Service (located in the service package)

5. StudentRepo

- **Description and Purpose:** Provides data access methods for interacting with the Student entity in the database.

- **Component Type:**
  Java interface extending JpaRepository<Student, Long>, annotated with @Repository

6. Timer Entity

- **Description and Purpose:**
  Represents timing details such as start time, end time, and duration for a student's challenge session.

- **Component Type**:
  Java class annotated with @Entity (located in the entity package)

7. TimerRepo

- **Description and Purpose:**
  Manages database operations for Timer entities.

- **Component Type:**
  Java interface extending JpaRepository<Timer, Long> (located in the repository package)

8. TimerService

- **Description and Purpose:**
  Handles logic to create, update, and retrieve timer data for individual students during gameplay.

- **Component Type:**
  Java class annotated with @Service (located in the service package)

9. TimerController

- **Description and Purpose:**
  Receives and processes timing-related API requests from the Roblox client.

- **Component Type:**
  Java class annotated with @RestController (located in the controller package)

10. Score Entity

- **Description and Purpose:**
  Records and stores individual scores achieved by students during scenarios or challenges. Also used to support leaderboard computations.

- **Component Type:**
  Java class annotated with @Entity (located in the entity package)

11. ScoreRepo

- **Description and Purpose:**
  Provides CRUD operations for the Score entity using Spring Data JPA.

- **Component Type:**
  Java interface extending JpaRepository<Score, Long> (located in the repository package)

12. ScoreService

- **Description and Purpose:**
  Contains logic for calculating, storing, and retrieving score data. May also ensure student existence when scores are submitted.

- **Component Type:**
  Java class annotated with @Service (located in the service package)

13. ScoreController

- **Description and Purpose:**
  Accepts HTTP requests to submit or retrieve scores. May also trigger student creation if they don't already exist.

- **Component Type:**
  Java class annotated with @RestController (located in the controller package)

14. LeaderboardEntry Entity

- **Description and Purpose:**
  Represents an aggregated entry in the leaderboard for the Information Sorting game. Stores total score and time per student.

- **Component Type:**
  Java class annotated with @Entity (located in the LeaderboardInfoSort.Global package)

15. LeaderboardRepo

- **Description and Purpose:**
  Provides CRUD operations for the InfoSortingLeaderboardEntry entity.

- **Component Type:**
  Java interface extending JpaRepository<InfoSortingLeaderboardEntry, Long> (located in the LeaderboardInfoSort.Global package)

16. LeaderboardService

- **Description and Purpose:**
  Contains logic for computing, updating, and retrieving leaderboard data by aggregating student scores and durations.

- **Component Type:**
  Java class annotated with @Service (located in the LeaderboardInfoSort.Global package)

17. Leaderboard Controller

- **Description and Purpose:**
  Exposes endpoints for fetching leaderboard data and triggering updates per student, usually by robloxId.

- **Component Type:**
  Java class annotated with @RestController (located in the LeaderboardInfoSort.Global package)

18. Teacher Entity

- **Description and Purpose:**
  Represents authenticated teacher accounts in the system. Stores teacher profile information, email, password, roles, and the worlds they've locked. Used to authorize scenario creation and class access.

- Component Type:
  Java class annotated with @Entity (located in the entity package)

19. Scenario Entity

- **Description and Purpose:**
  Represents the pool of customizable information items (scenarios) configured by teachers. Includes question content, the correct answer (SAFE or UNSAFE), and class linkage. Can be toggled active/inactive.

- **Component Type:**
  Java class annotated with @Entity (located in the entity package)

20. ScenarioController

- **Description and Purpose**:
  Exposes API endpoints for teachers to create, update, toggle, retrieve, and delete their own scenarios. Also provides counts and active scenario queries.

- **Component Type:**
  Java class annotated with @RestController (located in the controller package)

21. ScenarioService

- **Description and Purpose:**
  Contains business logic for all scenario-related operations: creation, validation, filtering by teacher or status, and authorization checks.

- **Component Type:**
  Java class annotated with @Service (located in the service package)

22. ScenarioDTO

- **Description and Purpose:**
  A Data Transfer Object for returning sanitized scenario data from the backend to clients (e.g., content, active status, timestamps).

- **Component Type:**
  Java POJO (Plain Old Java Object), typically found in the dto package

23. ScenarioRepository

- **Description and Purpose:**
  Provides Spring Data JPA methods to retrieve scenarios by teacher ID, status (active), and more. Includes custom query for all active scenarios.

- **Component Type:**
  Java interface extending JpaRepository<Scenario, Long> (located in the repository package)

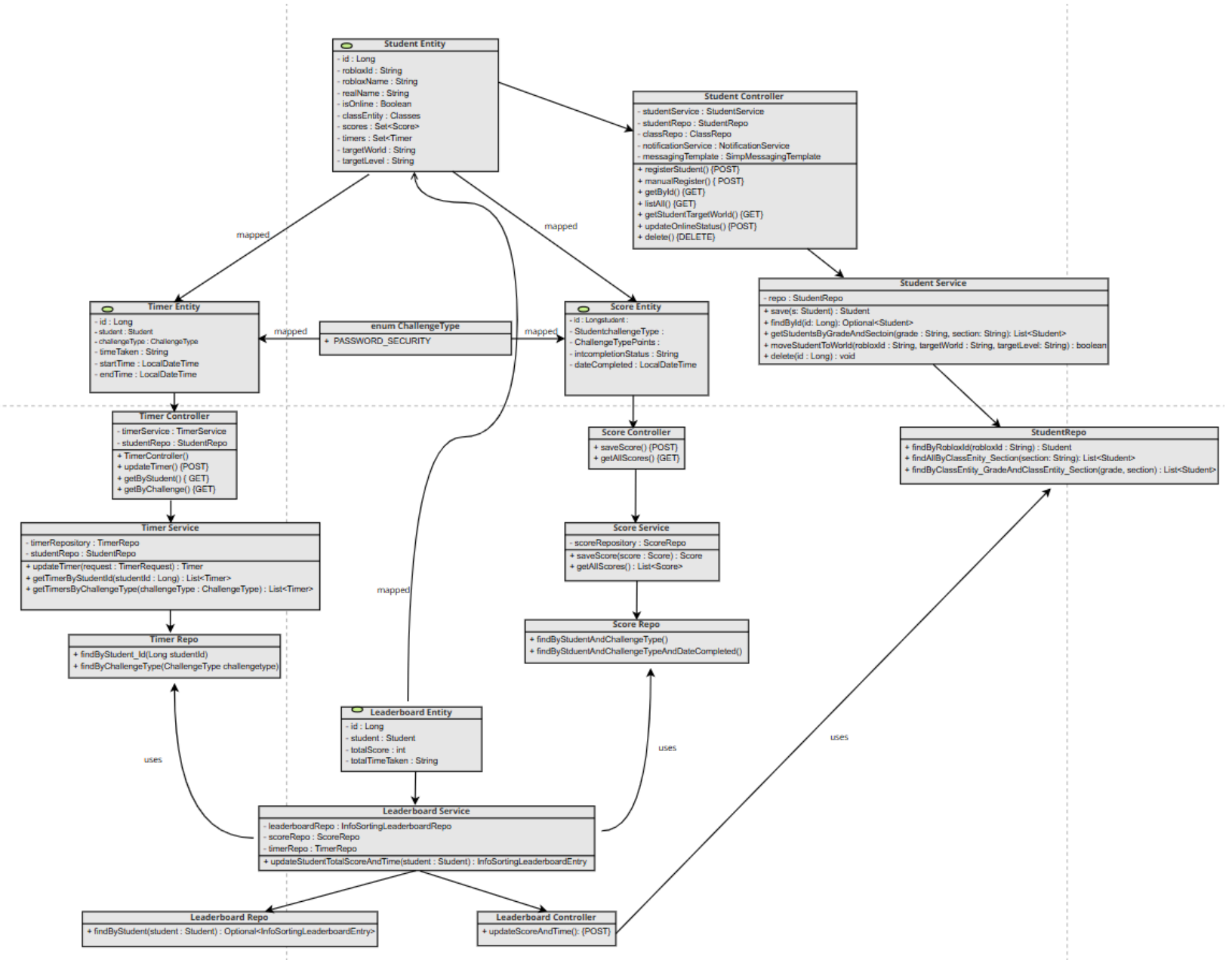
24. Enum: AnswerTypeLvl1

- **Description and Purpose:**
  Enum defining valid answers to scenarios (e.g., SAFE, UNSAFE). Ensures type safety in scenario content and correctness validation.

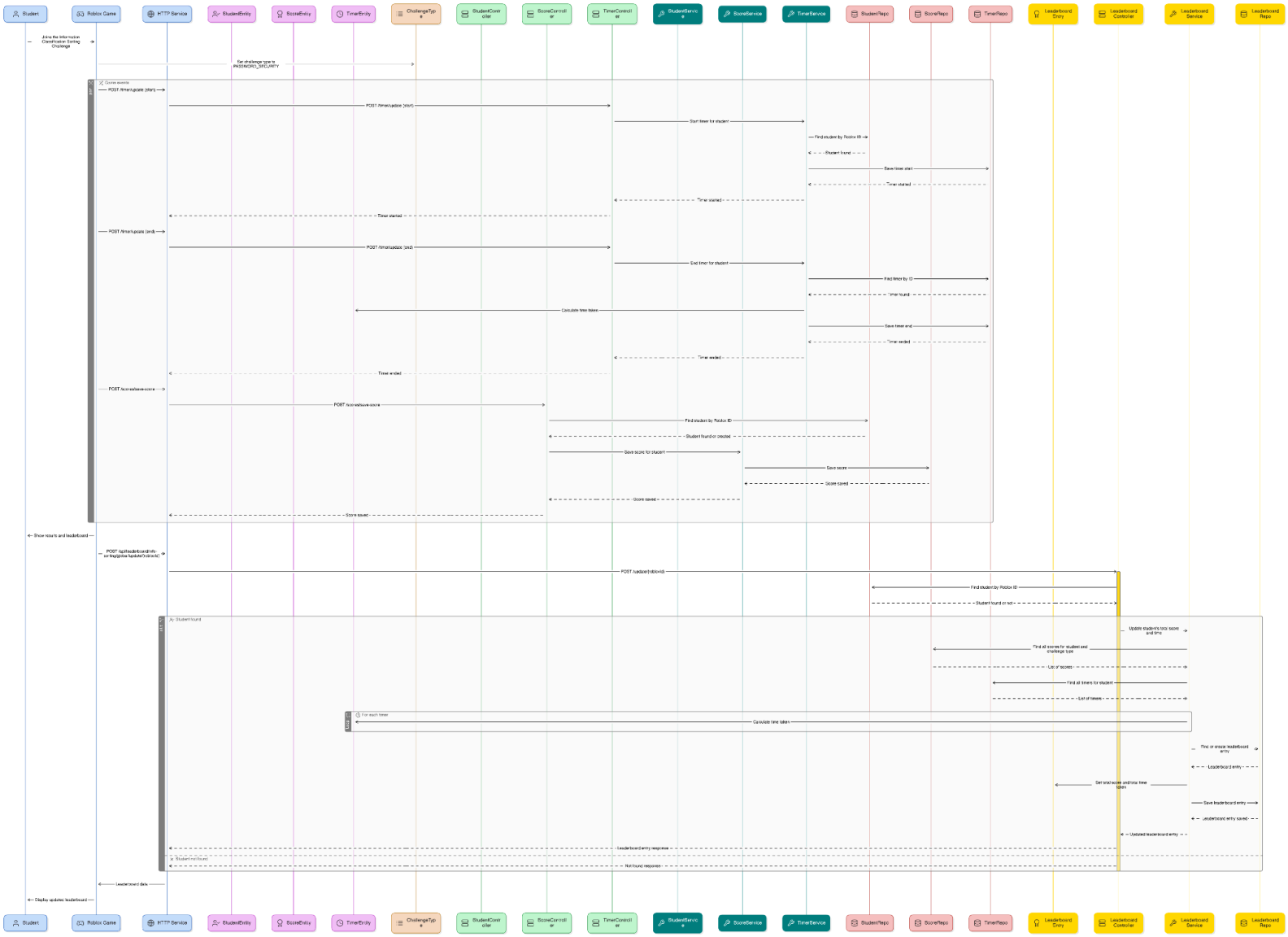
- **Component Type:**
  Java enum (typically located in the model package)

Object-Oriented Components

- Class Diagram

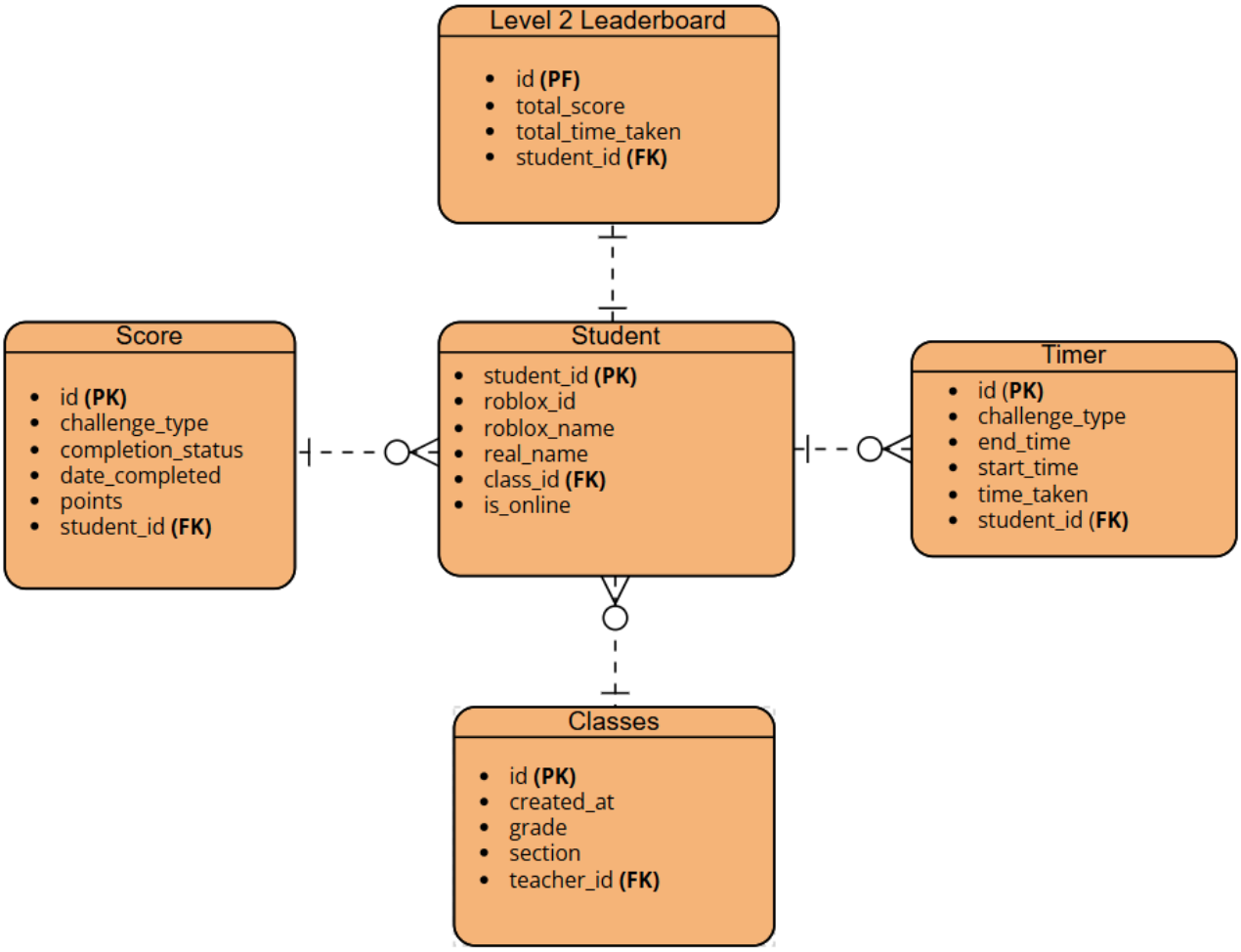


- Sequence Diagram



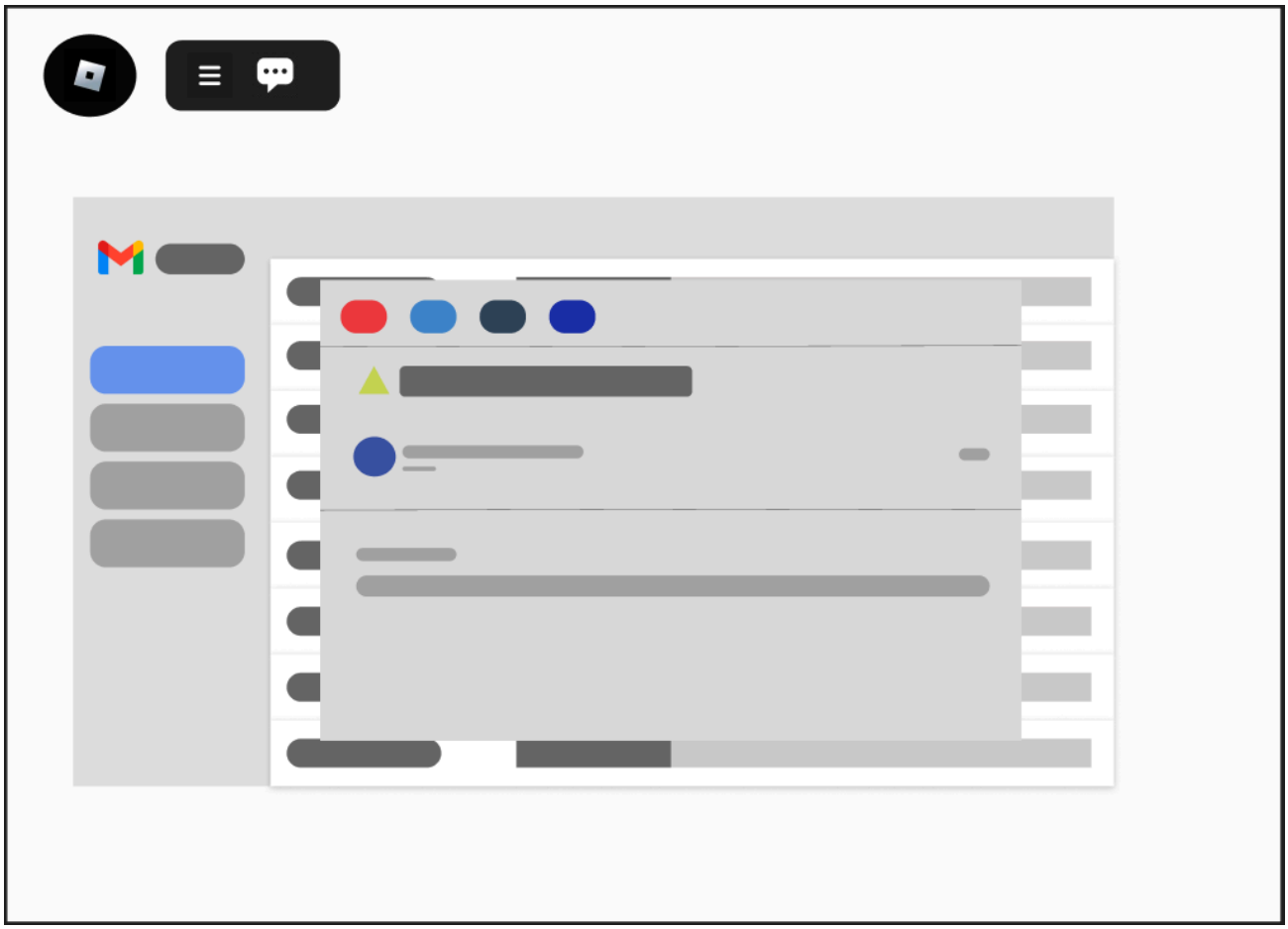
Data Design

- ERD or schema



Transaction 1.3 Phishing Identification Challenge

User Interface Design



Front-end component(s)

1. Workspace Objects

• **Description and purpose:** Static and interactive 3D models placed in the game world. These 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 viewpoints during game start, end, or transitions. Enhances player immersion and guides player focus.
• **Component type:** Roblox Camera objects, manipulated by LocalScripts

3. Sounds / Music

• **Description and purpose:** Background music and sound effects triggered during cutscenes, gameplay, or transitions to enhance the atmosphere and feedback.
• Component type: Roblox Sound instances (placed in SoundService, Workspace, or StarterGui)

4. Cutscenes

• **Description and purpose**r: Pre-scripted sequences that control camera, animations, and player input to introduce the story or environment. Typically shown at the start or between gameplay segments.
• **Component type:** LocalScript using TweenService, Camera manipulation, and UI elements

5. Remote Events

• **Description and purpose**: Enables communication between the server and clients for actions like game state changes, player actions, or updates. Used to trigger remote functions or sync states.
• **Component type:** RemoteEvent instances in ReplicatedStorage

6. RegisterAPI (HTTP Module)

• **Description and purpose**: Sends the player's Roblox UserId to the backend API to register or update student/player data. Prevents the need for manual registration within the game.
• **Component type:** ModuleScript or Script using HttpService:PostAsync()

7. Timer

• **Description and purpose**: Tracks countdowns, time limits, or elapsed durations during challenges. Often updates visible UI to reflect time status.
• **Component type**: Script or LocalScript with custom timer logic

8. Score System

• **Description and purpose**: Tracks and updates player scores based on correct actions or scenario completions. May include submission to backend or leaderboard systems.
• **Component type:** Script or LocalScript

9. Leaderboard

• **Description and purpose**: Displays real-time or historical ranking of players based on their scores. Encourages competition and replayability.

• **Component type:** Script or LocalScript (can interact with UI elements and back-end via HTTP)

Back-end component(s)

1. Student Entity

- **Description and Purpose:**
  Represents the database table for students/players. Stores fields like id, robloxId, robloxName, realName, and status-related fields (e.g., online, targetWorld, targetLevel). Used to uniquely identify and manage students across game sessions.

- **Component Type:**
  Java class annotated with @Entity (located in the entity package)

2. StudentRequest DTO (Data Transfer Object)

- **Description and Purpose:**
  Used to receive data from the Roblox game client when registering or manually registering a new student. Includes fields like robloxId, robloxName, realName, and classCode.

- **Component Type:**
  Java POJO (Plain Old Java Object), used in HTTP requests (typically in the model or dto package)

3. StudentController

- **Description and Purpose:**
  Handles student-related API endpoints, including automatic and manual registration, fetching active scenarios, student data lookup, and retrieving NPC info for the game. Ensures proper notification of student status changes.

- **Component Type:**
  Java class annotated with @RestController (located in the controller package)

4. StudentService

- **Description and Purpose:**
  Contains core business logic for saving, retrieving, or deleting student records, separated from controller logic for modularity.

- **Component Type:**
  Java class annotated with @Service (located in the service package)

5. StudentRepo

- **Description and Purpose**: Provides data access methods for interacting with the Student entity in the database.

- **Component Type:**
  Java interface extending JpaRepository<Student, Long>, annotated with @Repository

6. Timer Entity

- **Description and Purpose:**
  Represents timing details such as start time, end time, and duration for a student's challenge session.

- **Component Type:**
  Java class annotated with @Entity (located in the entity package)

7. TimerRepo

- **Description and Purpose:**
  Manages database operations for Timer entities.

- **Component Type:**
  Java interface extending JpaRepository<Timer, Long> (located in the repository package)

8. TimerService

- **Description and Purpose:**
  Handles logic to create, update, and retrieve timer data for individual students during gameplay.

- **Component Type:**
  Java class annotated with @Service (located in the service package)

9. TimerController

- **Description and Purpose:**
  Receives and processes timing-related API requests from the Roblox client.

- **Component Type**:
  Java class annotated with @RestController (located in the controller package)

10. Score Entity

- **Description and Purpose:**
  Records and stores individual scores achieved by students during scenarios or challenges. Also used to support leaderboard computations.

- **Component Type:**
  Java class annotated with @Entity (located in the entity package)

11. ScoreRepo

- **Description and Purpose:**
  Provides CRUD operations for the Score entity using Spring Data JPA.

- **Component Type:**
  Java interface extending JpaRepository<Score, Long> (located in the repository package)

12. ScoreService

- **Description and Purpose:**
  Contains logic for calculating, storing, and retrieving score data. May also ensure student existence when scores are submitted.

- **Component Type:**
  Java class annotated with @Service (located in the service package)

13. ScoreController

- **Description and Purpose:**
  Accepts HTTP requests to submit or retrieve scores. May also trigger student creation if they don't already exist.

- **Component Type:**
  Java class annotated with @RestController (located in the controller package)

14. LeaderboardEntry Entity

- **Description and Purpose**:
  Represents an aggregated entry in the leaderboard for the Information Sorting game. Stores total score and time per student.

- **Component Type:**
  Java class annotated with @Entity (located in the LeaderboardInfoSort.Global package)

15. LeaderboardRepo

- **Description and Purpose:**
  Provides CRUD operations for the InfoSortingLeaderboardEntry entity.

- **Component Type:**
  Java interface extending JpaRepository<InfoSortingLeaderboardEntry, Long> (located in the LeaderboardInfoSort.Global package)

16. LeaderboardService

- **Description and Purpose:**
  Contains logic for computing, updating, and retrieving leaderboard data by aggregating student scores and durations.

- **Component Type:**
  Java class annotated with @Service (located in the LeaderboardInfoSort.Global package)

17. LeaderboardController

- **Description and Purpose:**
  Exposes endpoints for fetching leaderboard data and triggering updates per student, usually by robloxId.

- **Component Type:**
  Java class annotated with @RestController (located in the LeaderboardInfoSort.Global package)

18. Teacher Entity

- **Description and Purpose:**
  Represents authenticated teacher accounts in the system. Stores teacher profile information, email, password, roles, and the worlds they've locked. Used to authorize scenario creation and class access.

- **Component Type:**
  Java class annotated with @Entity (located in the entity package)

19. Scenario Entity

- **Description and Purpose:**
  Represents the pool of customizable information items (scenarios) configured by teachers. Includes question content, the correct answer (SAFE or UNSAFE), and class linkage. Can be toggled active/inactive.

- **Component Type:**
  Java class annotated with @Entity (located in the entity package)

20. ScenarioController

- **Description and Purpose:**
  Exposes API endpoints for teachers to create, update, toggle, retrieve, and delete their own scenarios. Also provides counts and active scenario queries.

- **Component Type:**
  Java class annotated with @RestController (located in the controller package)

21. ScenarioService

- **Description and Purpose:**
  Contains business logic for all scenario-related operations: creation, validation, filtering by teacher or status, and authorization checks.

- **Component Type:**
  Java class annotated with @Service (located in the service package)

22. ScenarioDTO

- **Description and Purpose:**
  A Data Transfer Object for returning sanitized scenario data from the backend to clients (e.g., content, active status, timestamps).

- **Component Type:**
  Java POJO (Plain Old Java Object), typically found in the dto package

23. ScenarioRepository

- **Description and Purpose:**
  Provides Spring Data JPA methods to retrieve scenarios by teacher ID, status (active), and more. Includes custom query for all active scenarios.

- **Component Type:**
  Java interface extending JpaRepository<Scenario, Long> (located in the repository package)

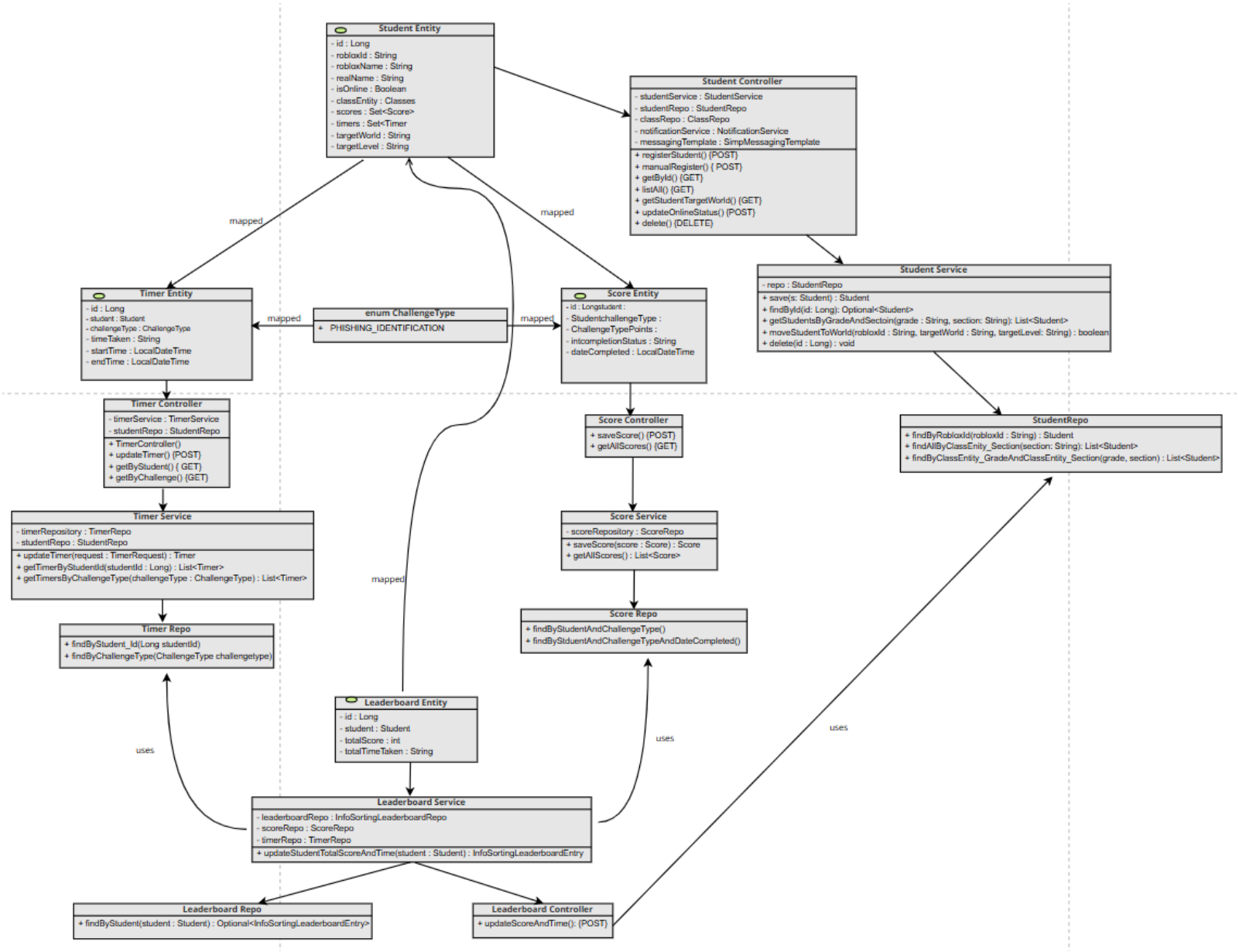
24. Enum: AnswerTypeLvl1

- **Description and Purpose:**
  Enum defining valid answers to scenarios (e.g., SAFE, UNSAFE). Ensures type safety in scenario content and correctness validation.

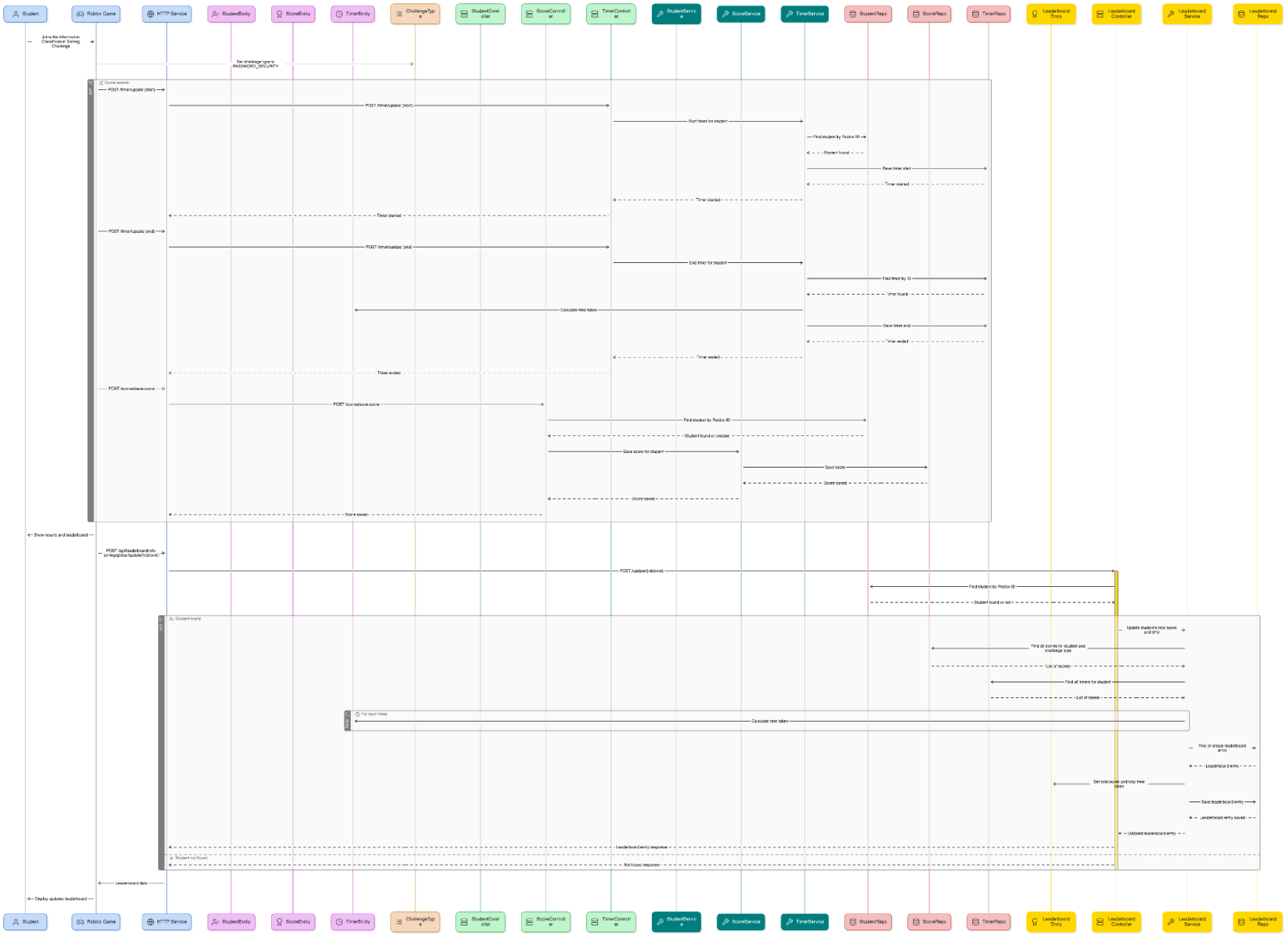
- **Component Type:**
  Java enum (typically located in the model package)

Object-Oriented Components

- Class Diagram

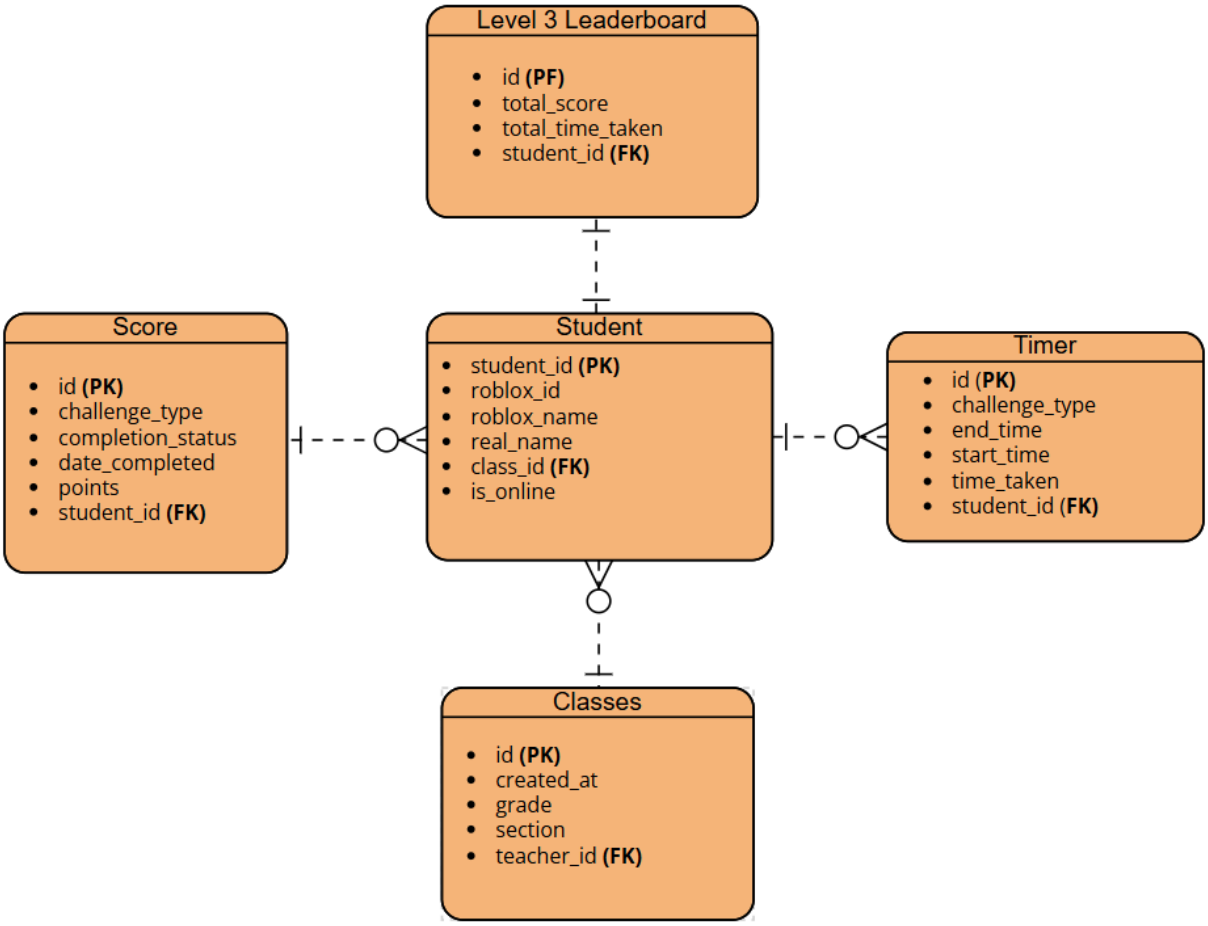


- Sequence Diagram



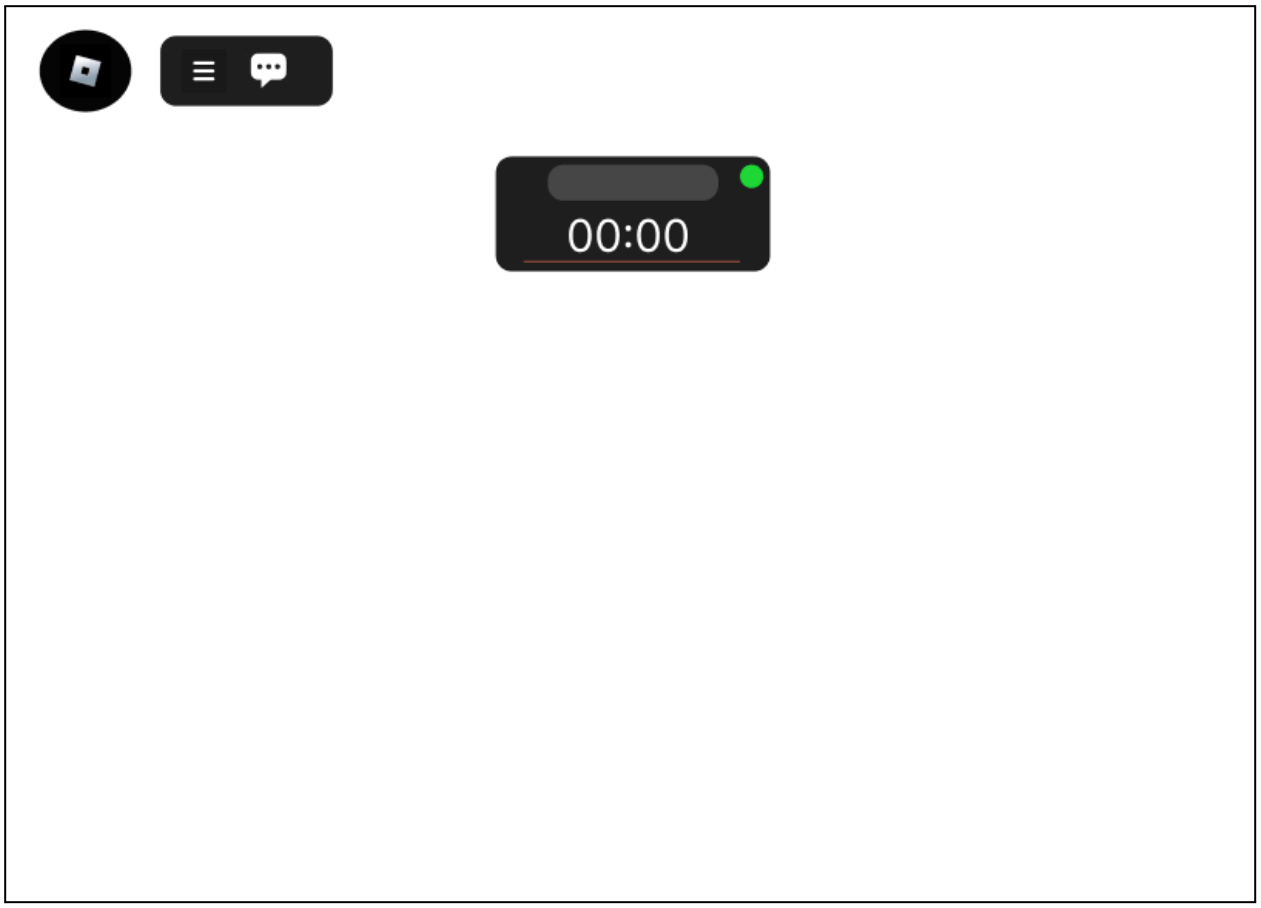
Data Design

- ERD or schema



Transaction 1.4 Timer System

User Interface Design



Front-end component(s)

1. Remote Events

• **Description and purpose**: Enables communication between the server and clients for actions like game state changes, player actions, or updates. Used to trigger remote functions or sync states.
• **Component type:** RemoteEvent instances in ReplicatedStorage

2. Timer

• **Description and purpose**: Tracks countdowns, time limits, or elapsed durations during challenges. Often updates visible UI to reflect time status.
• **Component type:** Script or LocalScript with custom timer logic

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. **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.
4. **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.

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

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

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

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

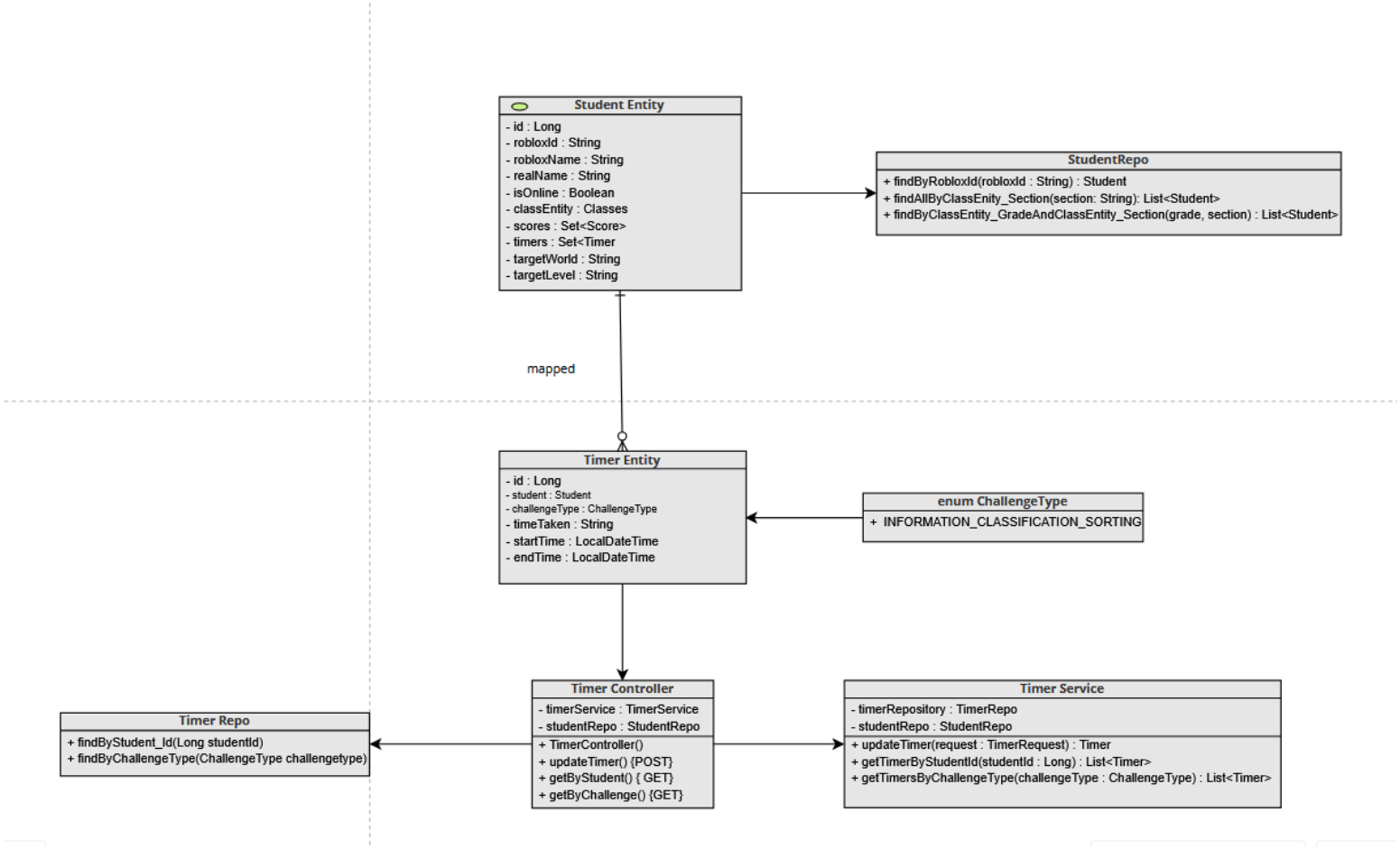
      Java class annotated with @Service, located in the Service package.
7. **TimerController**
    - ➢ **Description and Purpose:**

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

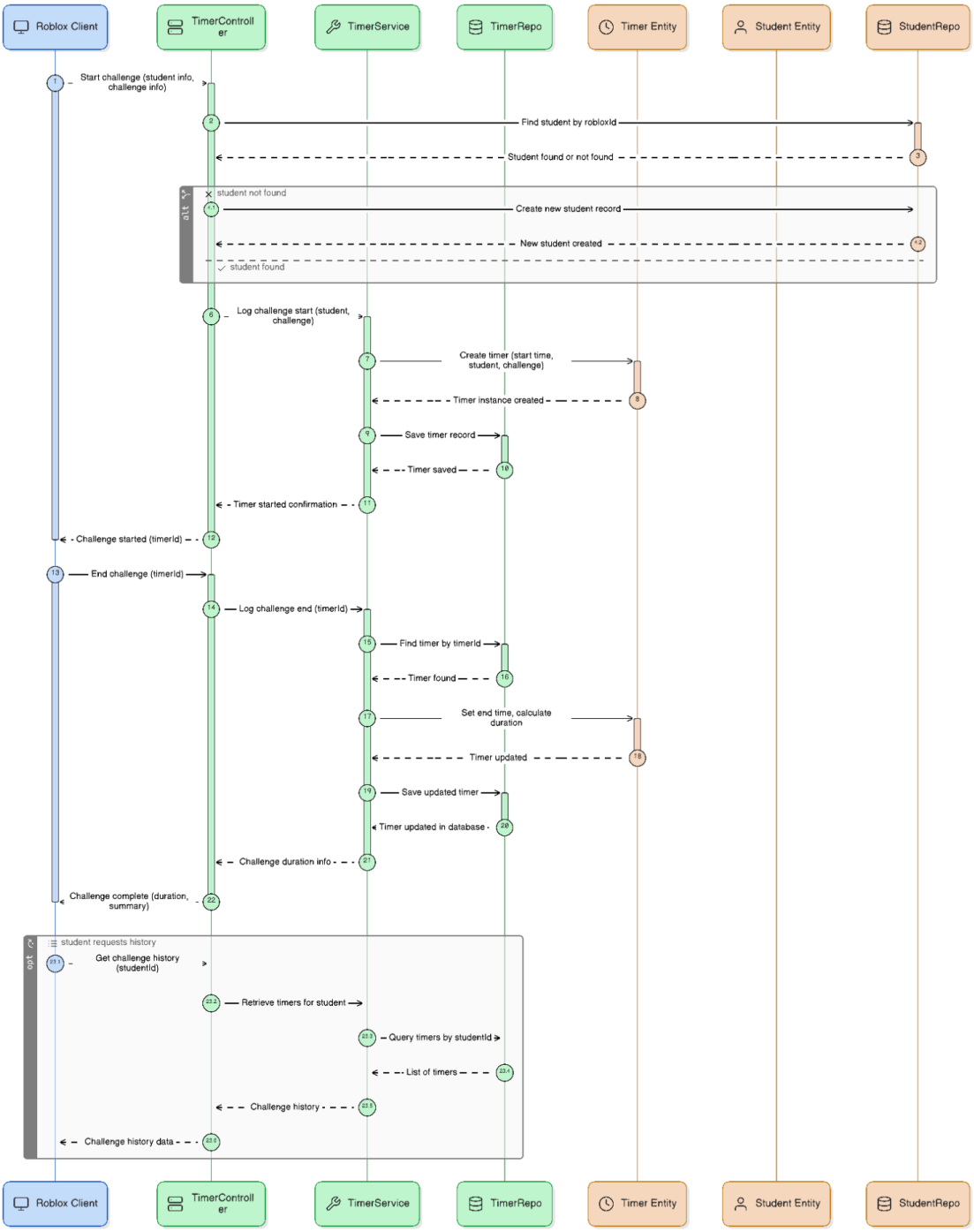
      Java class annotated with @RestController, located in the Controller package.

Object-Oriented Components

- Class Diagram

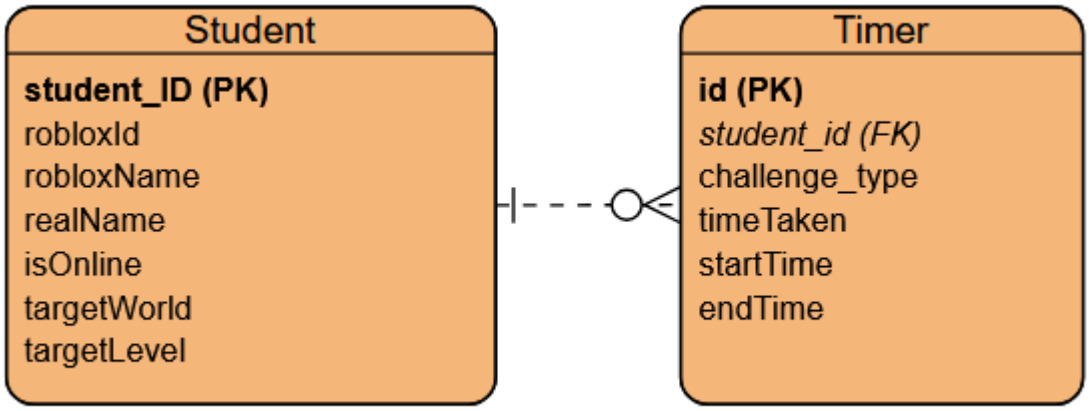


- Sequence Diagram



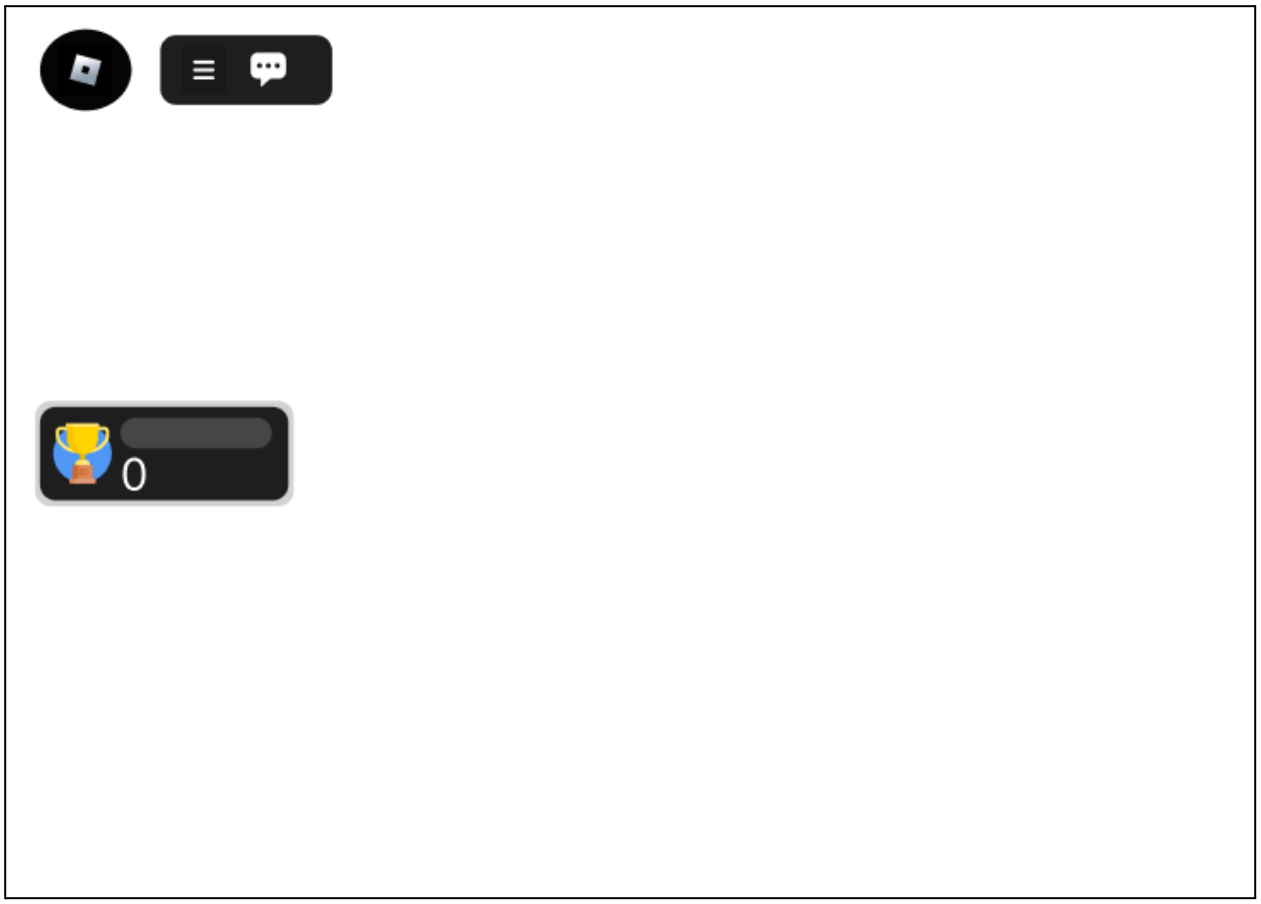
Data Design

- ERD or schema



Transaction 1.5 Scoring System

User Interface Design



Front-end component(s)

1. Remote Events

• **Description and purpose**: Enables communication between the server and clients for actions like game state changes, player actions, or updates. Used to trigger remote functions or sync states.
• **Component type:** RemoteEvent instances in ReplicatedStorage

2. Score System

• **Description and purpose**: Tracks and updates player scores based on correct actions or scenario completions. May include submission to backend or leaderboard systems.
• **Component type:** Script or 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. **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.
4. **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.

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

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

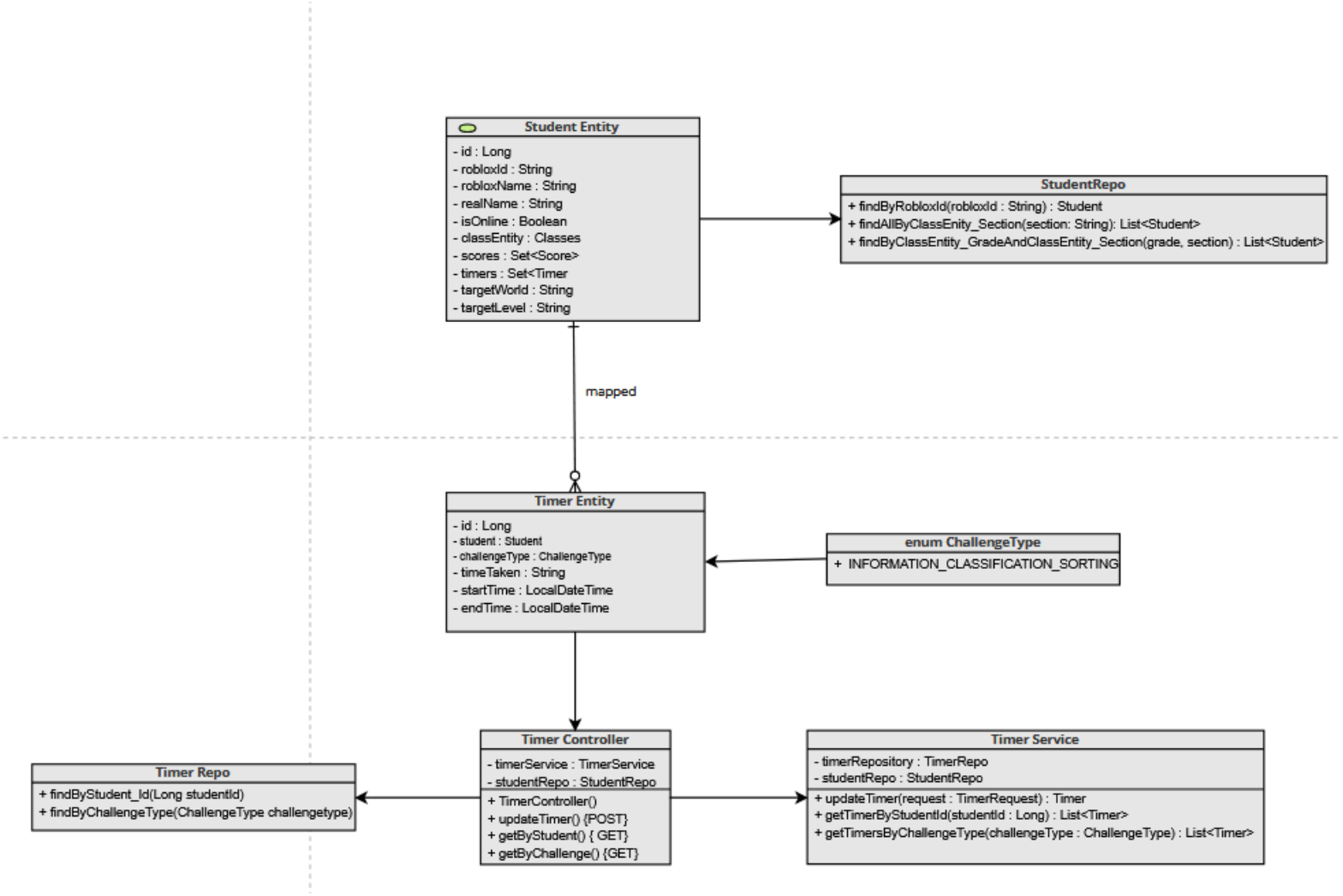
      Java interface extending JpaRepository<Score, Long>, located in the Repository package.
6. **ScoreService**
    - ➢ **Description and Purpose:**

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

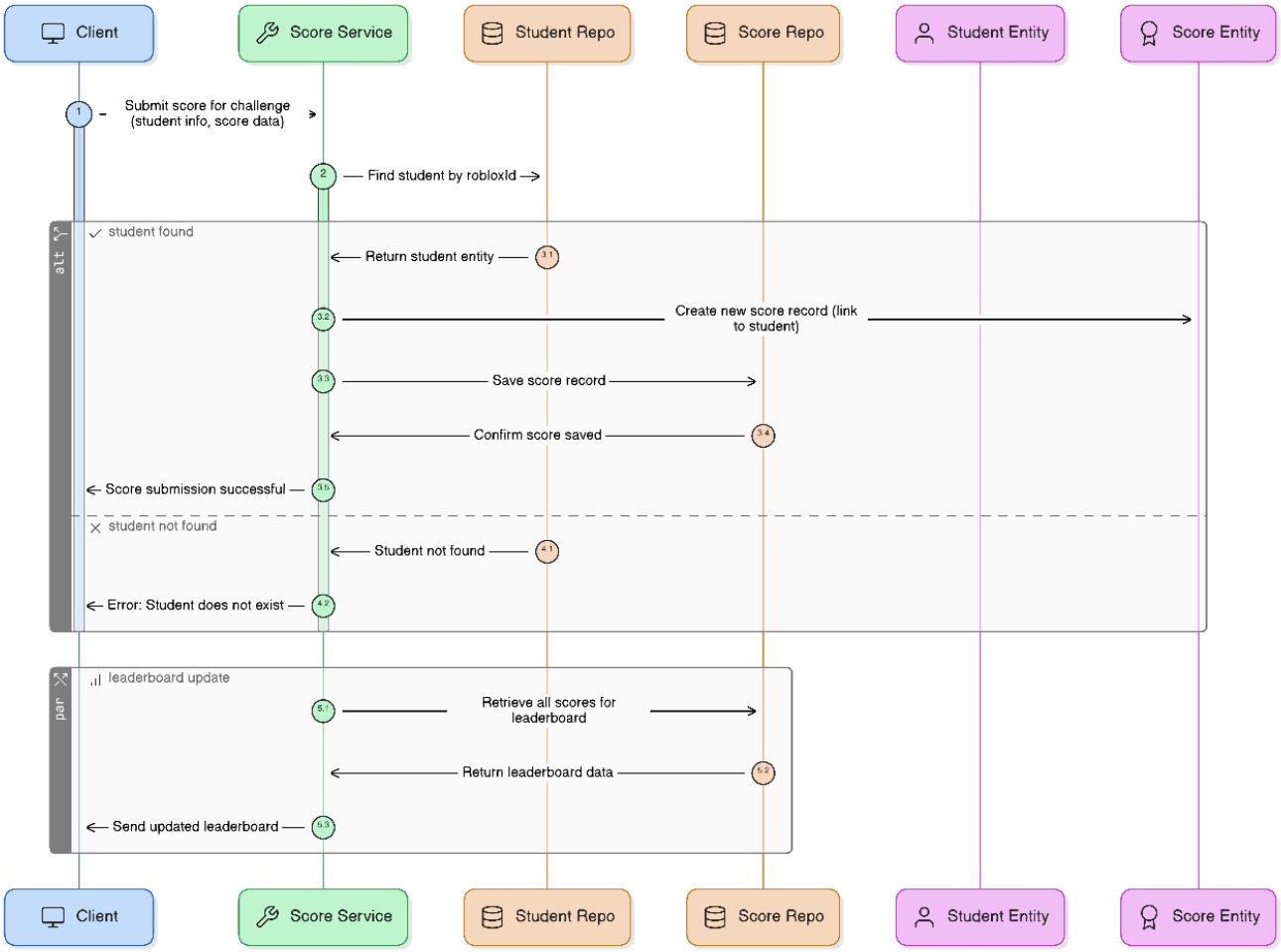
      Java class annotated with @Service, located in the Service package.

Object-Oriented Components

- Class Diagram

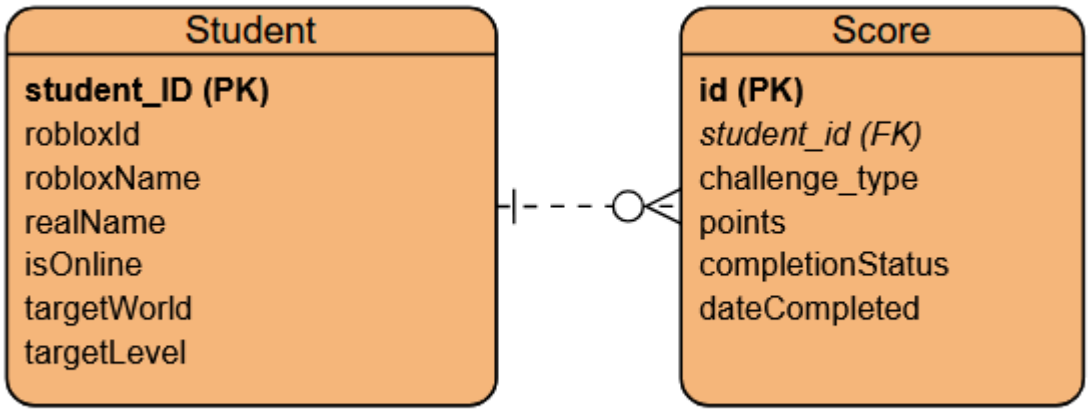


- Sequence Diagram



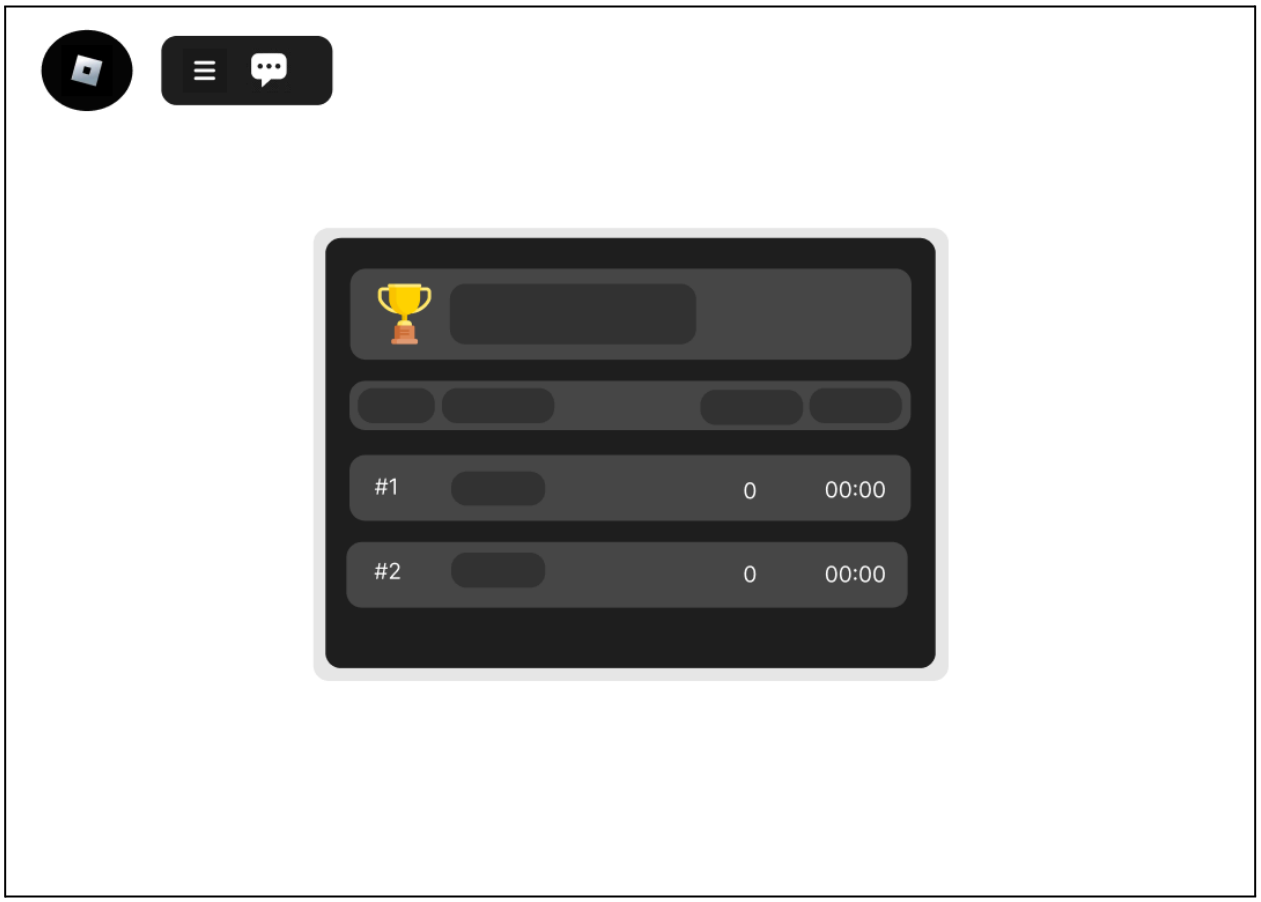
Data Design

- ERD or schema



Transaction 1.6 Leaderboard System

User Interface Design



Front-end component(s)

1. Remote Events

• **Description and purpose**: Enables communication between the server and clients for actions like game state changes, player actions, or updates. Used to trigger remote functions or sync states.
• **Component type:** RemoteEvent instances in ReplicatedStorage

2. Leaderboard

• **Description and purpose**: Displays real-time or historical ranking of players based on their scores.

Encourages competition and replayability.

• Component type: Script or LocalScript (can interact with UI elements and back-end via HTTP)

Back-end component(s)

1. Student Entity

- **Description and Purpose:**
  Represents the database table for students/players. Stores fields like id, robloxId, robloxName, realName, and status-related fields (e.g., online, targetWorld, targetLevel). Used to uniquely identify and manage students across game sessions.

- **Component Type:**
  Java class annotated with @Entity (located in the entity package)

2. StudentRequest DTO (Data Transfer Object)

- **Description and Purpose:**
  Used to receive data from the Roblox game client when registering or manually registering a new student. Includes fields like robloxId, robloxName, realName, and classCode.

- **Component Type:**
  Java POJO (Plain Old Java Object), used in HTTP requests (typically in the model or dto package)

3. StudentController

- **Description and Purpose:**
  Handles student-related API endpoints, including automatic and manual registration, fetching active scenarios, student data lookup, and retrieving NPC info for the game. Ensures proper notification of student status changes.

- **Component Type:**
  Java class annotated with @RestController (located in the controller package)

4. StudentService

- **Description and Purpose:**
  Contains core business logic for saving, retrieving, or deleting student records, separated from controller logic for modularity.

- **Component Type:**
  Java class annotated with @Service (located in the service package)

5. StudentRepo

- **Description and Purpose**: Provides data access methods for interacting with the Student entity in the database.

- **Component Type:**
  Java interface extending JpaRepository<Student, Long>, annotated with @Repository

6. LeaderboardEntry Entity

- **Description and Purpose:**
  Represents an aggregated entry in the leaderboard for the Information Sorting game. Stores total score and time per student.

- **Component Type:**
  Java class annotated with @Entity (located in the LeaderboardInfoSort.Global package)

7. LeaderboardRepo

- **Description and Purpose:**
  Provides CRUD operations for the InfoSortingLeaderboardEntry entity.

- **Component Type:**
  Java interface extending JpaRepository<InfoSortingLeaderboardEntry, Long> (located in the LeaderboardInfoSort.Global package)

8. LeaderboardService

- **Description and Purpose:**
  Contains logic for computing, updating, and retrieving leaderboard data by aggregating student scores and durations.

- **Component Type:**
  Java class annotated with @Service (located in the LeaderboardInfoSort.Global package)

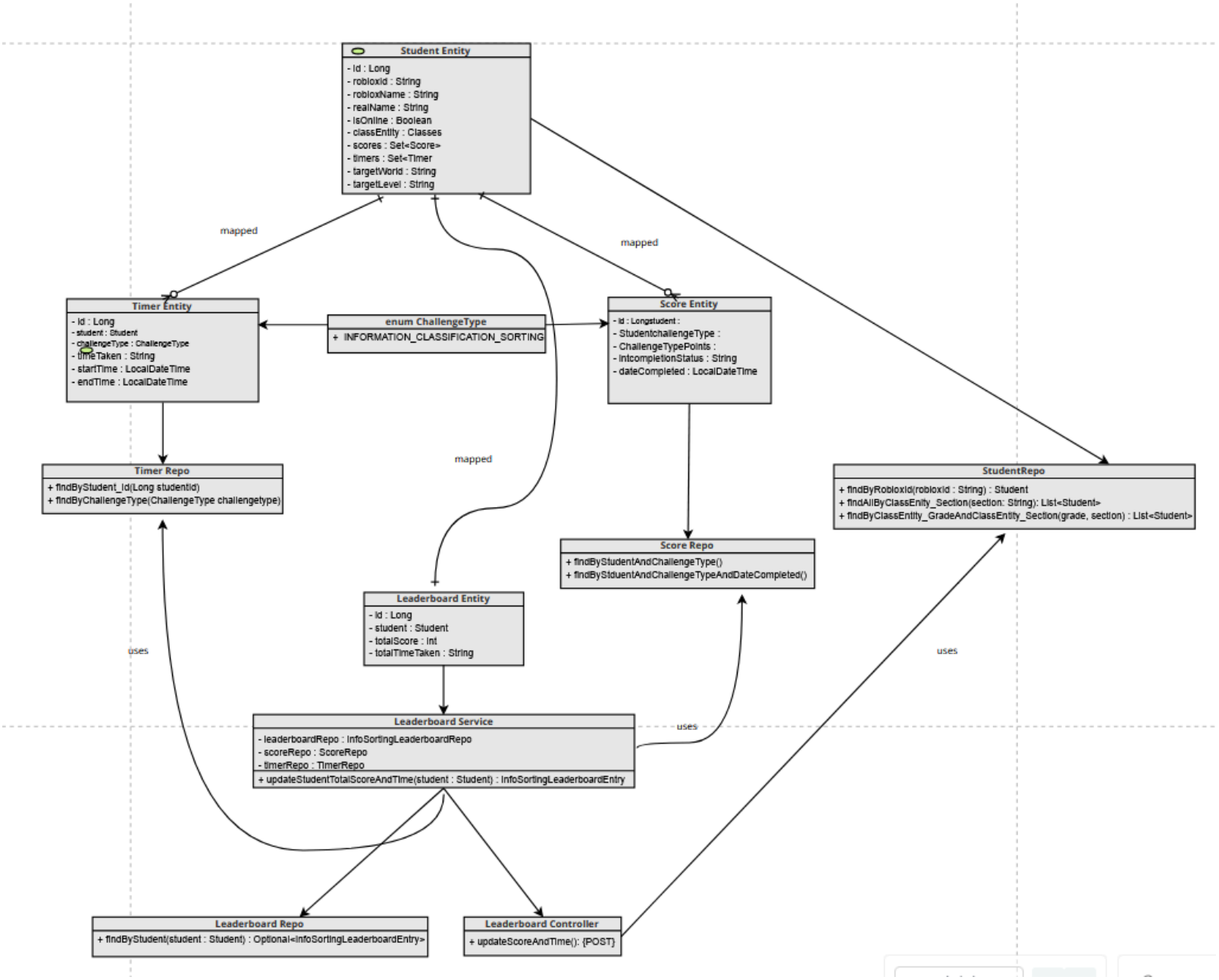
9. Leaderboard Controller

- **Description and Purpose:**
  Exposes endpoints for fetching leaderboard data and triggering updates per student, usually by robloxId.

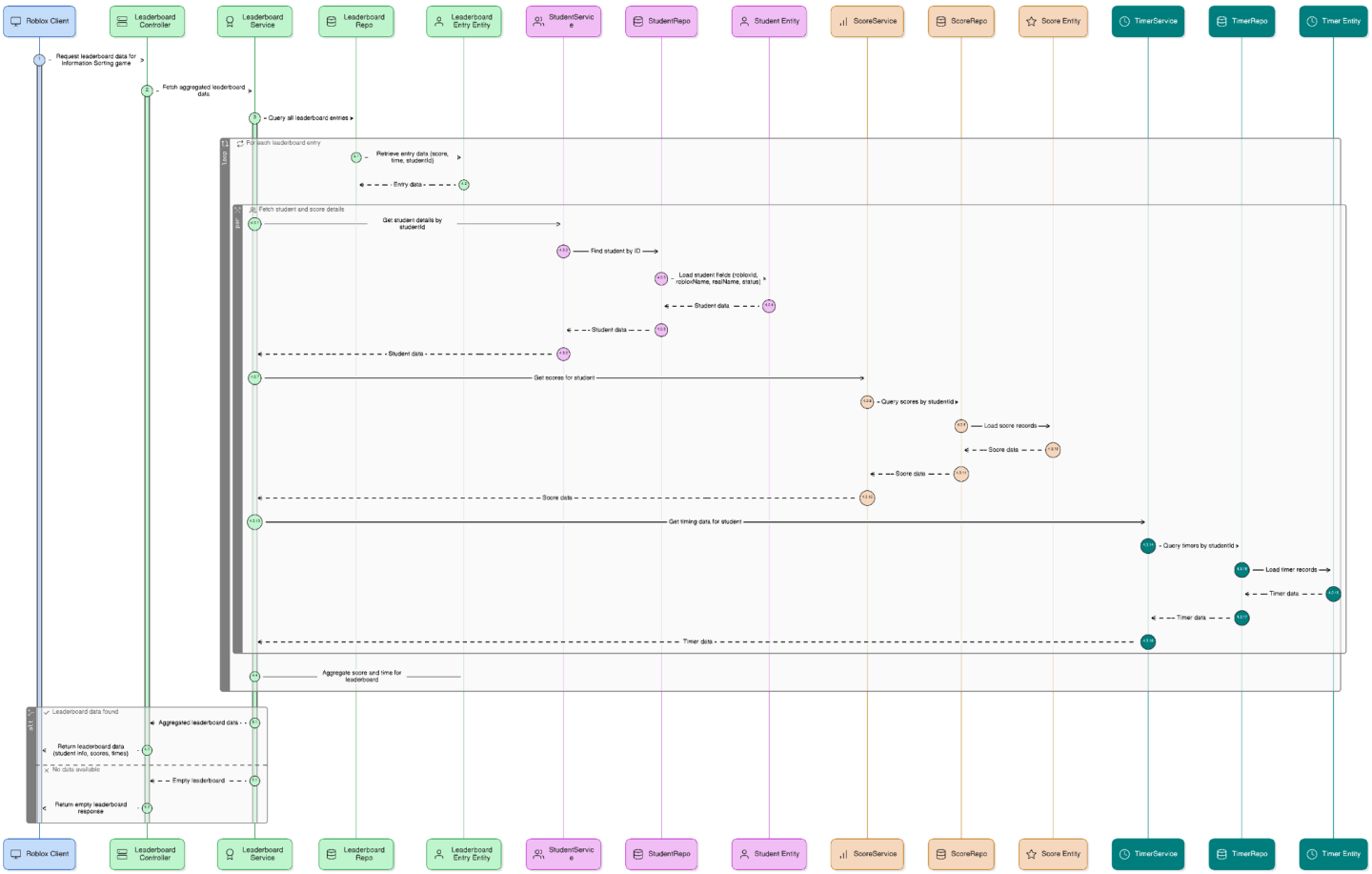
- **Component Type:**
  Java class annotated with @RestController (located in the LeaderboardInfoSort.Global package)

Object-Oriented Components

- Class Diagram

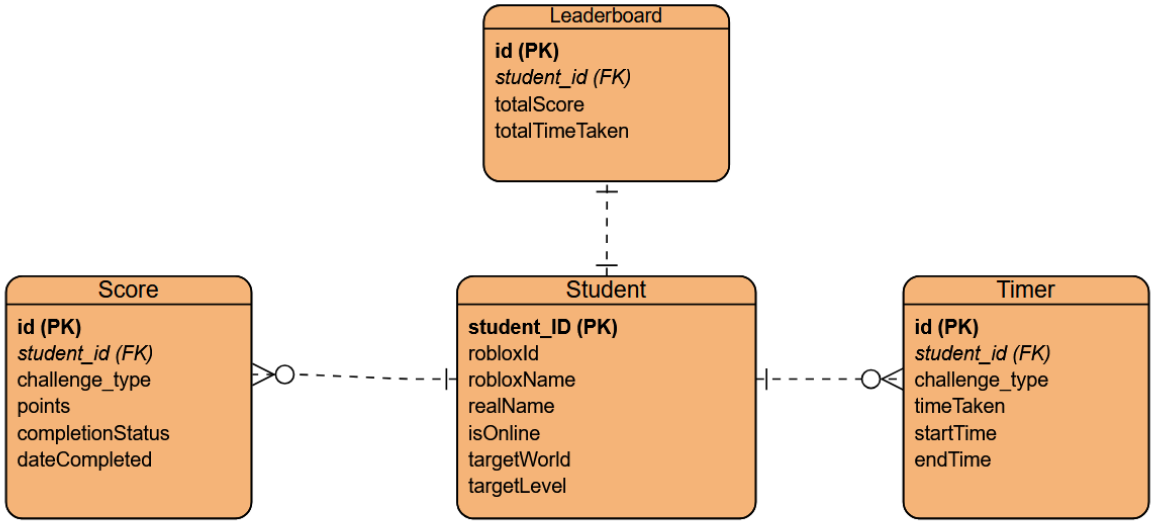


- Sequence Diagram



Data Design

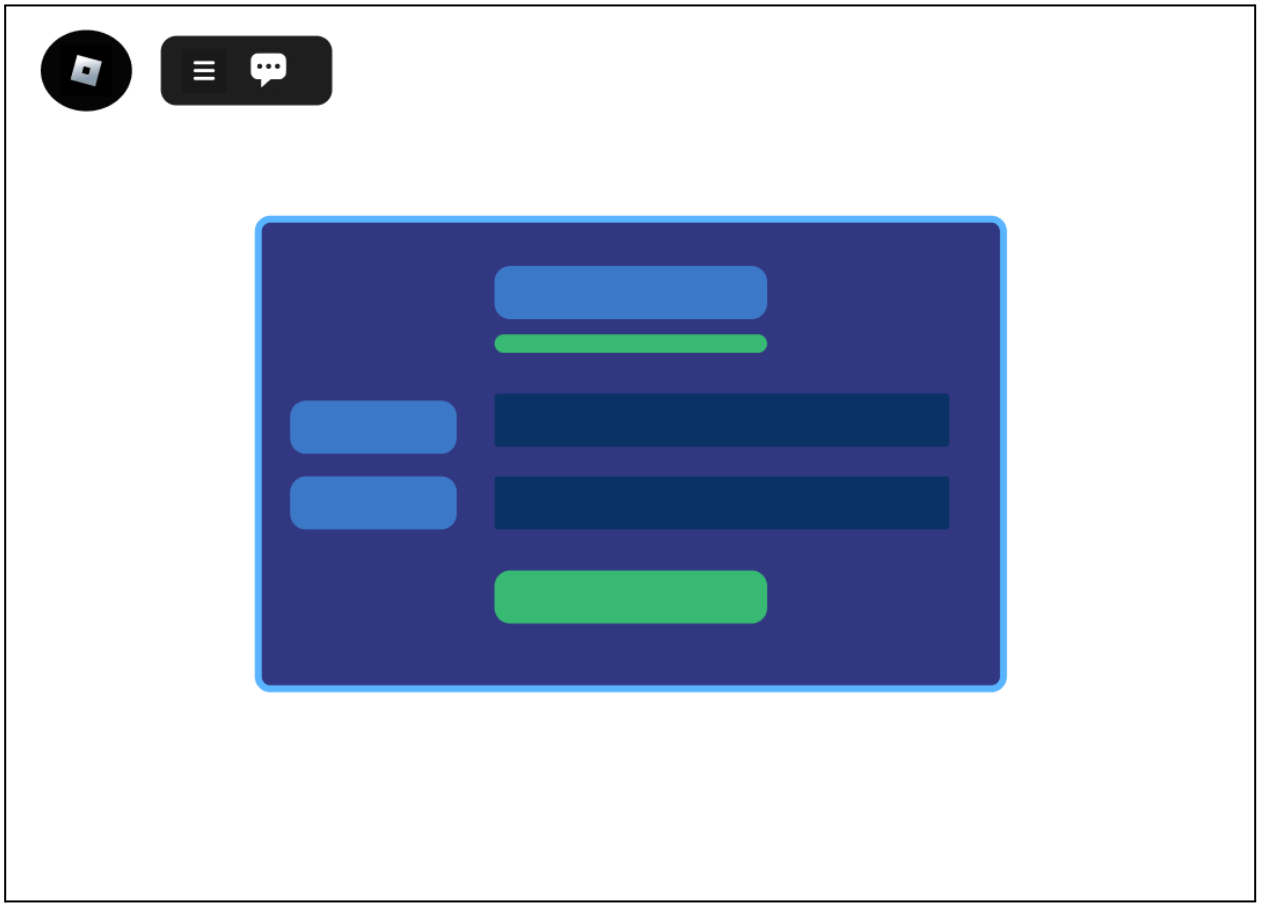
- ERD or schema



Module 2: Student Real-Name Integration

Transaction 2.1 Real-Name Input and Submission

User Interface Design



Front-end component(s)

1. GUI LocalScript

• Description and purpose:
Handles player UI interactions including buttons, text labels, visibility toggles, and responding to RemoteEvents. Controls scenario prompts, answers, transitions, and instructional feedback.

• Component type:
LocalScript placed inside StarterGui or ScreenGui objects in the Roblox UI hierarchy

2. Cameras

• **Description and purpose:** Custom camera sequences used to show intro cutscenes or adjust player viewpoints during game start, end, or transitions. Enhances player immersion and guides player focus.
• **Component type:** Roblox Camera objects, manipulated by LocalScripts

3. Remote Events

• **Description and purpose**: Enables communication between the server and clients for actions like game state changes, player actions, or updates. Used to trigger remote functions or sync states.
• **Component type:** RemoteEvent instances in ReplicatedStorage

4. RegisterAPI (HTTP Module)

• **Description and purpose**: Sends the player's Roblox UserId to the backend API to register or update student/player data. Prevents the need for manual registration within the game.
• **Component type:** ModuleScript or Script using HttpService:PostAsync()

Back-end component(s)

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

     Represents a class group in the system, containing a grade, section, assigned teacher, and list of students. It helps in organizing and linking students to their respective teacher.
   - ➢ **Component Type:**

     Data model/entity for mapping to the classes table in the database.
2. **Teacher Entity**
   - ➢ **Description and Purpose:**

     Represents a teacher user in the system with authentication credentials, assigned role, and tracked locked worlds. It facilitates managing teacher-specific features like class assignment and system access.
   - ➢ **Component Type:**

     Data model/entity for mapping to the teachers table in the database.
3. **Class Repository**
   - ➢ **Description and Purpose:**

     Provides data access methods for Classes entities, including finding classes by teacher ID and by grade/section combination. It simplifies interaction with the database using Spring Data JPA.
   - ➢ **Component Type:**

     Repository interface for performing CRUD and custom queries on the classes table.
4. **Teacher Repository**
   - ➢ **Description and Purpose:**

     Provides database operations for the Teacher entity, including a method to find a teacher by email for authentication or profile lookup. It leverages Spring Data JPA for simplified query handling.
   - ➢ **Component Type:**

     Repository interface for managing teachers table interactions.

5. Student Controller

- **Description and Purpose:**

  Acts as the main REST controller for managing student-related operations, including registration, manual profile completion, student lookup, online status updates, and deleting students. It also manages WebSocket notifications for real-time status updates.

- **Component Type:**

  REST Controller

6. **Class Controller**

- **Description and Purpose:**

  Handles REST endpoints for managing class data, such as creating, updating, deleting, and retrieving classes and enrolled students. It uses teacher authentication to fetch classes specific to the logged-in teacher.

- **Component Type:**

  REST Controller

7. **Class Service**

- **Description and Purpose:**

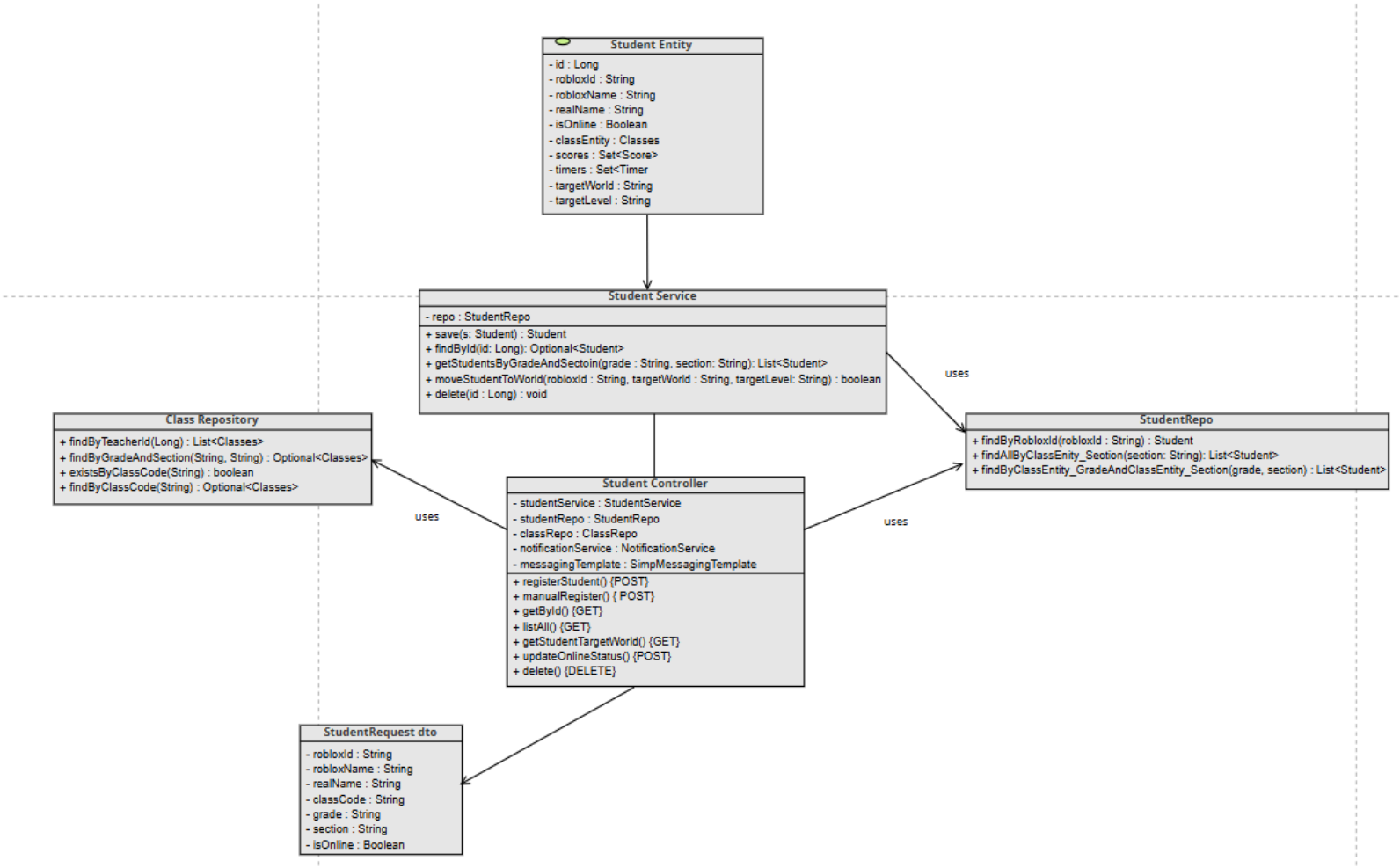
  Provides business logic for managing class-related operations, including creating, updating, retrieving, and deleting classes. It also associates classes with teachers using their email addresses for authentication purposes.

- **Component Type:**

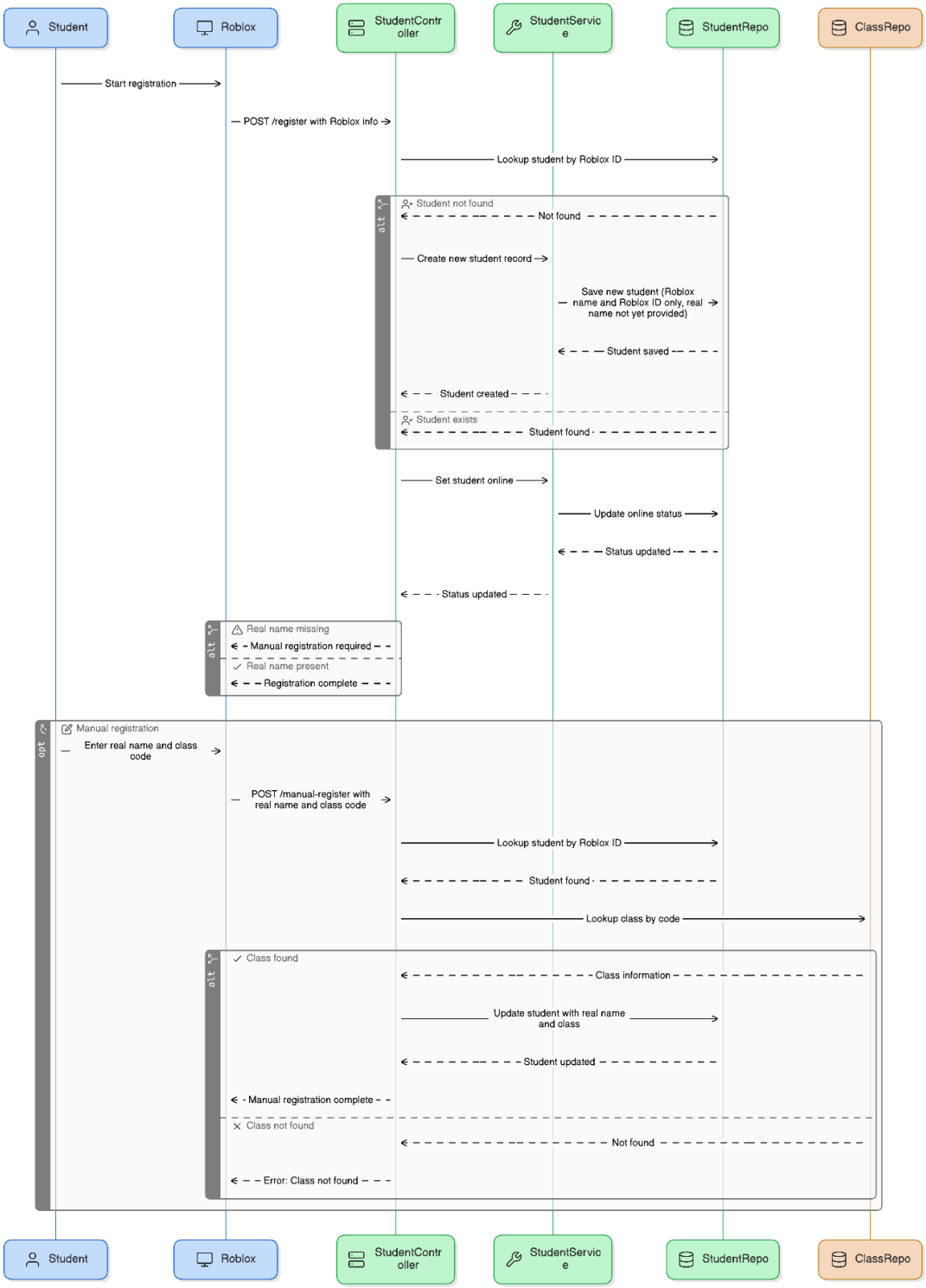
  Service Layer

Object-Oriented Components

- Class Diagram

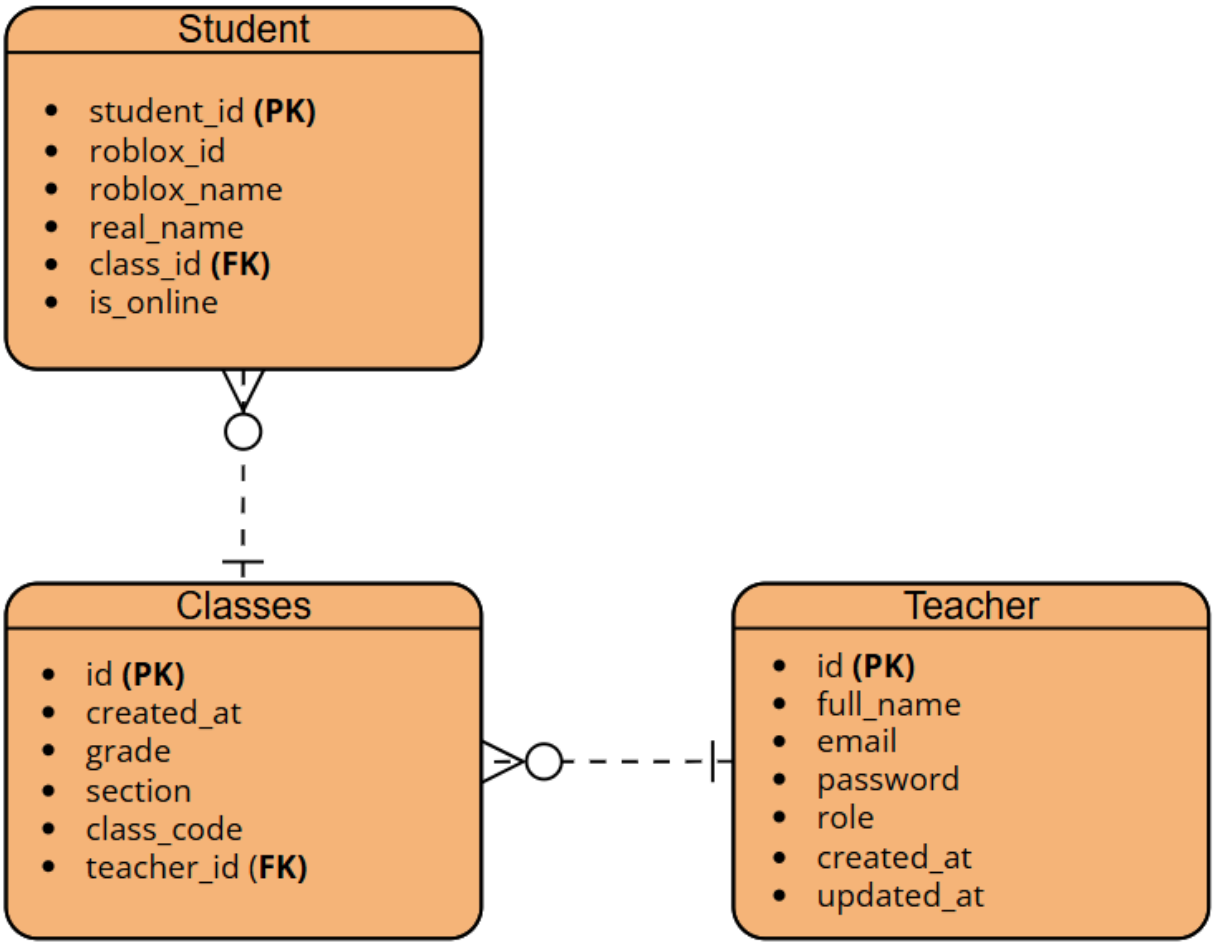


- Sequence Diagram



Data Design

- ERD or schema



Module 3: Teachers Dashboard

Transaction 3.1 Lock / Unlock Game Levels

User Interface Design

Front-end component(s)

1. Overview Component

➢ **Description and purpose**: Main dashboard for Transaction 3.1, allowing teachers to lock or unlock access to three Roblox game levels: Info Classification, Password Security, and Phishing ID. Each world is shown with its status and a toggle control. Locked worlds appear gray with a red badge; unlocked worlds show colored cards with green badges.

➢ **Component type**: React Page Component (src/components/Overview.jsx)

2. toggleWorldLock Function

➢ **Description and purpose**: Core function for Transaction 3.1 that sends API requests to lock/unlock game levels. It updates the worldStatus state, saves to localStorage, and includes loading and error handling.

➢ **Component type**: API Integration Function within Overview

3. Game Worlds Grid UI

➢ **Description and purpose**: Displays the three game levels in a responsive card layout. Each card shows the game title, icon, description, current status (Locked/Unlocked), and a toggle button. Visuals dynamically change based on lock status.

➢ **Component type**: Inline JSX Component within Overview's render method

4. Confirmation Modal

➢ **Description and purpose**: Appears when teachers lock/unlock a world to confirm the action and prevent accidental changes. Includes details about the selected world and impact of the change.

➢ **Component type**: Inline JSX Component within Overview's render method

5. World Status State Management

➢ **Description and purpose**: Manages the lock/unlock states of game worlds using useState and localStorage. Ensures status is preserved across sessions and page reloads.

➢ **Component type**: State Management Logic within Overview

6. showConfirmModal Function

➢ **Description and purpose**: Utility function that configures the confirmation modal by preparing the

selected world's details and setting up the modal's messaging and actions.

➢ **Component type**: Utility Function within Overview

Back-end component(s)

1. **Teacher Entity**
   - ➢ **Description and Purpose:**
     Represents a teacher user in the system with authentication credentials, assigned role, and tracked locked worlds. It facilitates managing teacher-specific features like class assignment and system access.
   - ➢ **Component Type:**
     Java Class annotated with @Entity (typically stored in the entity or model package).
2. **Teacher Repository**
   - ➢ **Description and Purpose:**
     Provides database operations for the Teacher entity, including a method to find a teacher by email for authentication or profile lookup. It leverages Spring Data JPA for simplified query handling.
   - ➢ **Component Type:**
     Repository interface for managing teachers table interactions.
3. **Teacher Controller**
   - ➢ **Description and Purpose:**
     REST controller that handles HTTP requests related to teacher functionalities, including teacher registration, profile retrieval, managing world lock/unlock states, moving students between virtual worlds, fetching student status history, and deleting notifications.
   - ➢ **Component Type:**
     Controller Layer

4. Teacher Service

- **Description and Purpose:**

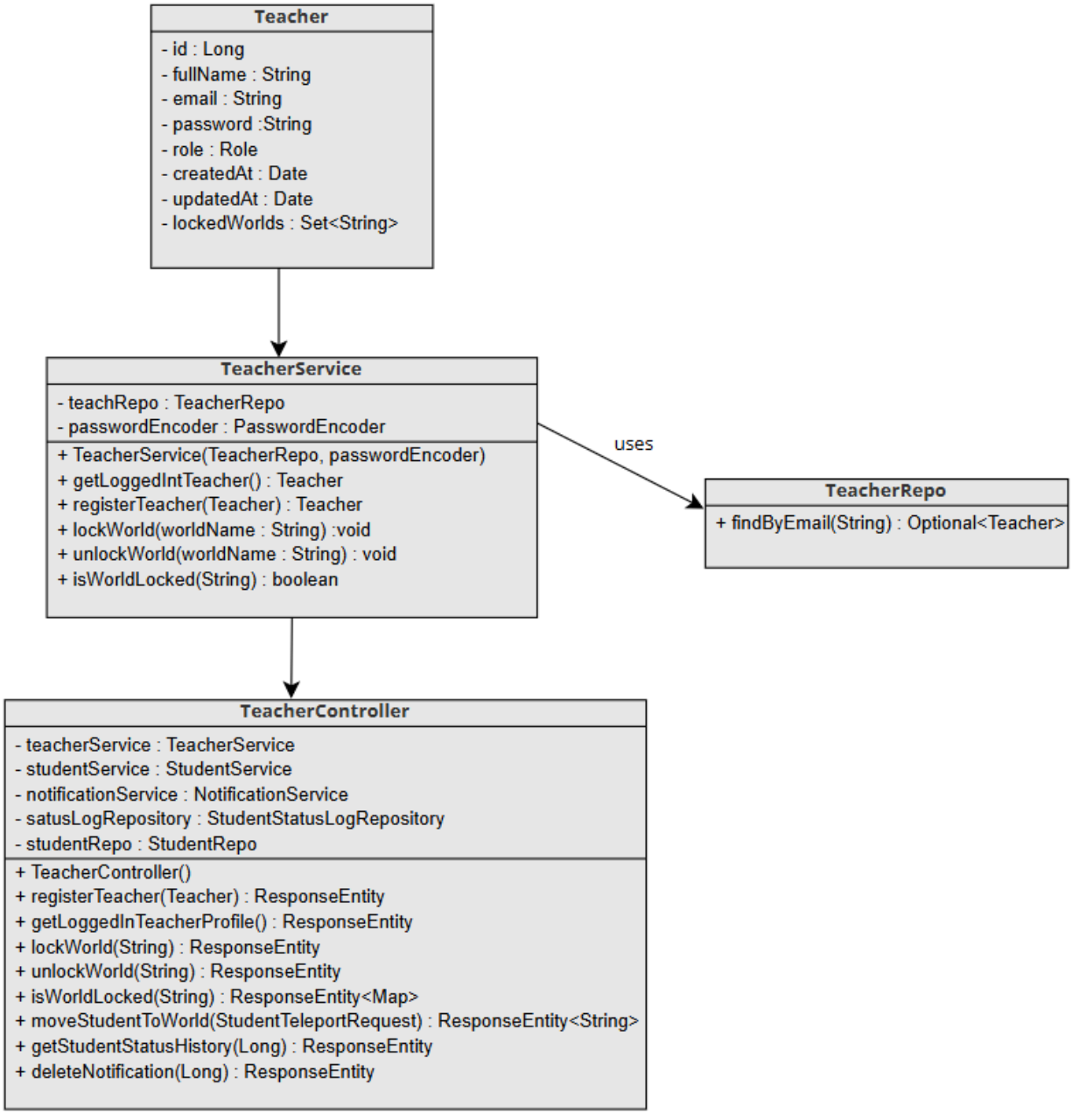
  Service layer managing business logic related to Teacher entities. It handles registration, retrieving the logged-in teacher, and managing locked virtual worlds associated with teachers.

- **Component Type:**

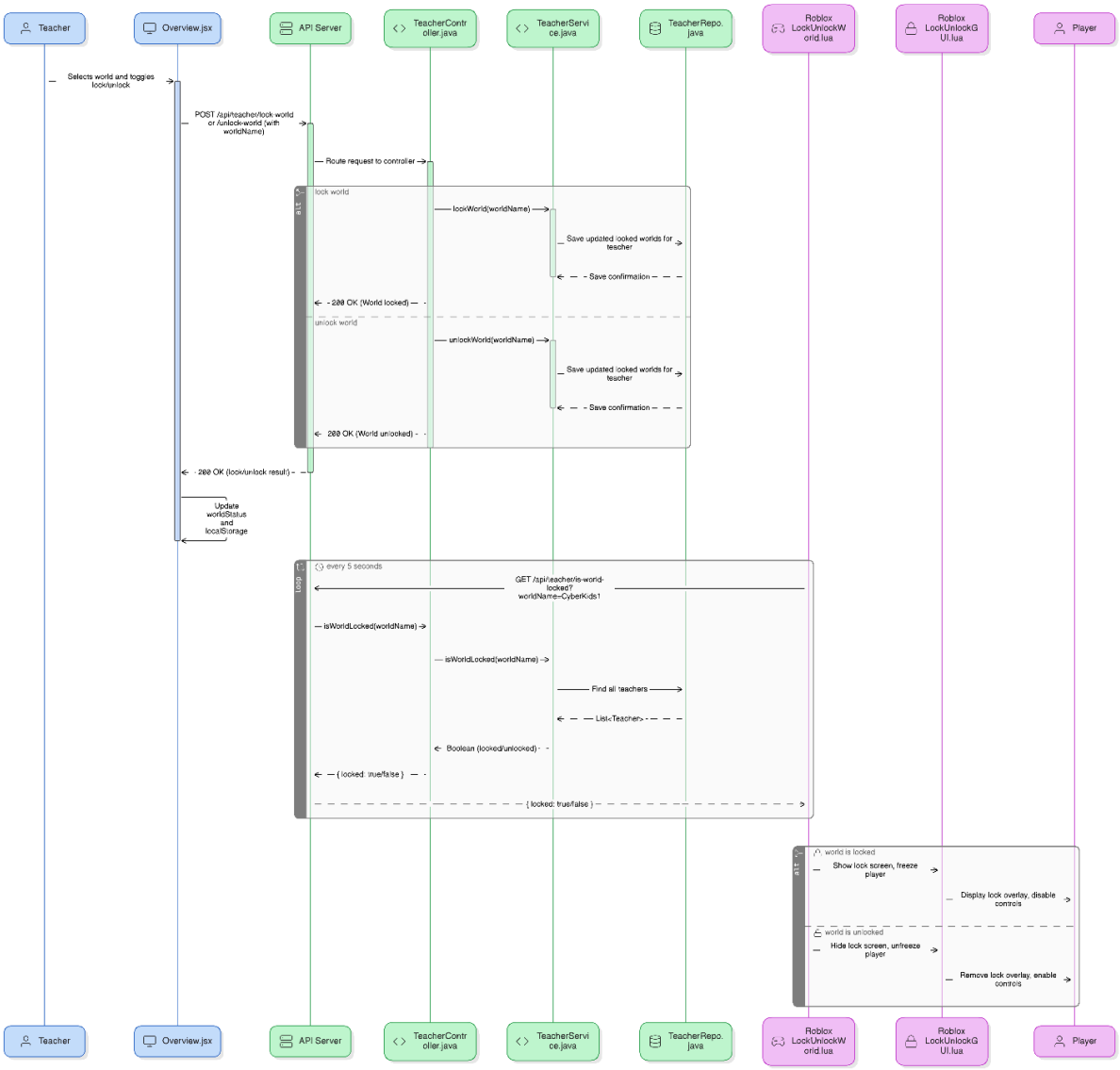
  Service Layer

Object-Oriented Components

- Class Diagram

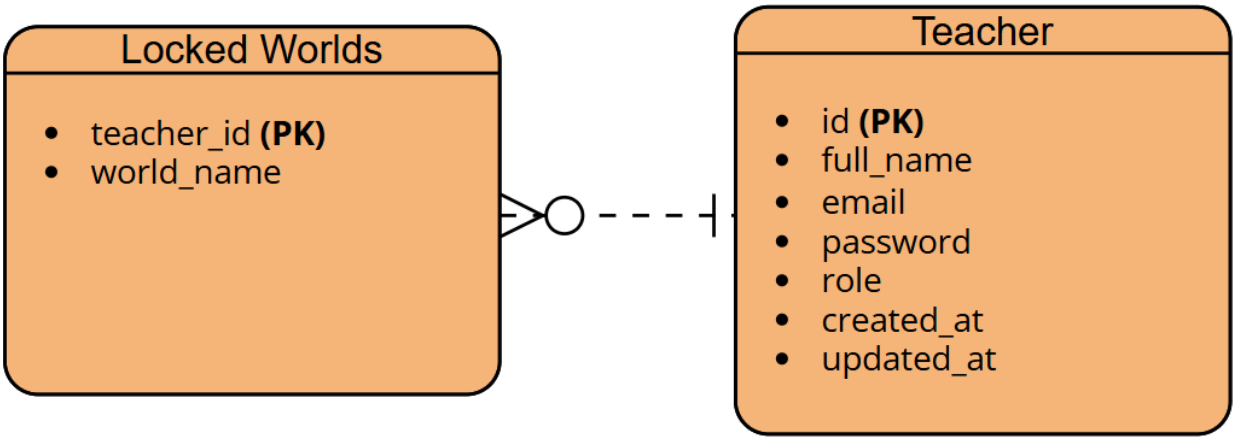


- Sequence Diagram



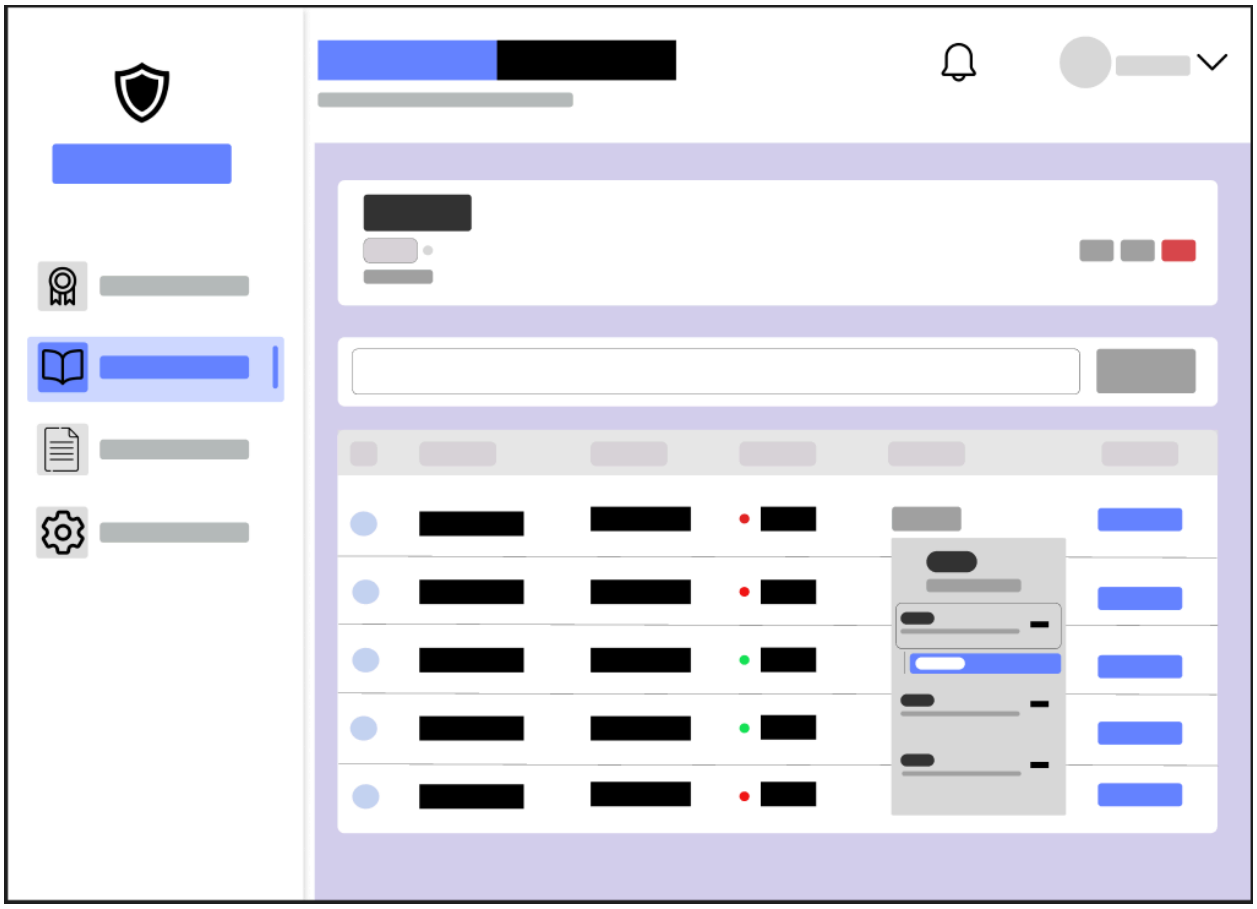
Data Design

- ERD or schema



Transaction 3.2 Reassign Players to Levels

User Interface Design



Front-end component(s)

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

     This is the main React component that handles the complete class management system, including the player reassignment functionality for Transaction 3.2. Within the students view, it renders a comprehensive table where each student row contains a "View Progress" dropdown that expands to show level assignment options. Teachers can reassign students to Main Menu, Level 1, Level 2, or Level 3 by selecting a level and clicking "Assign Player". The component manages real-time student status updates via WebSocket and provides visual feedback during the reassignment process.
   - ➢ **Component Type:**

     React Page Component (typically placed in src/components/Class.jsx)

2. **moveStudentToWorld Function**
   - ➢ **Description and Purpose:**

     A core utility function within ClassComponent that handles the actual player reassignment logic for Transaction 3.2. It sends a POST request to the teacher API endpoint (`${TEACHER_API_URL}/move-student`) with the student's Roblox ID, target world name (mapped from level), and target level number.
   - ➢ **Component Type:**

     Utility Function within Class.jsx

3. **Level Selection Dropdown UI**
   - ➢ **Description and Purpose:**

     An expandable dropdown interface nested within each student row in the students table. This component renders available game levels (Main Menu, Level 1-3) with their corresponding challenge types and completion status indicators. Each level option includes an "Assign Player" button that triggers the moveStudentToWorld function.
   - ➢ **Component Type:**

     Inline JSX Component within Class.jsx render method

Back-end component(s)

1. **Teacher Entity**
   - ➢ **Description and Purpose:**
     Represents a teacher user in the system with authentication credentials, assigned role, and tracked locked worlds. It facilitates managing teacher-specific features like class assignment and system access.
   - ➢ **Component Type:**
     Java Class annotated with @Entity (typically stored in the entity or model package).
2. **Teacher Controller**
   - ➢ **Description and Purpose:**
     REST controller that handles HTTP requests related to teacher functionalities, including teacher registration, profile retrieval, managing world lock/unlock states, moving students between virtual worlds, fetching student status history, and deleting notifications.
   - ➢ **Component Type:**
     Controller Layer
3. **StudentService**
   - ➢ **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. **StudentRepo**
   - ➢ **Description and Purpose:**
     A Spring Data JPA repository interface for accessing and managing Student entities. It defines custom query methods for common access patterns involving student data, especially related to their class assignments and identifiers.
   - ➢ **Component Type:**
     Repository Layer

5. Student Entity

- **Description and Purpose:**

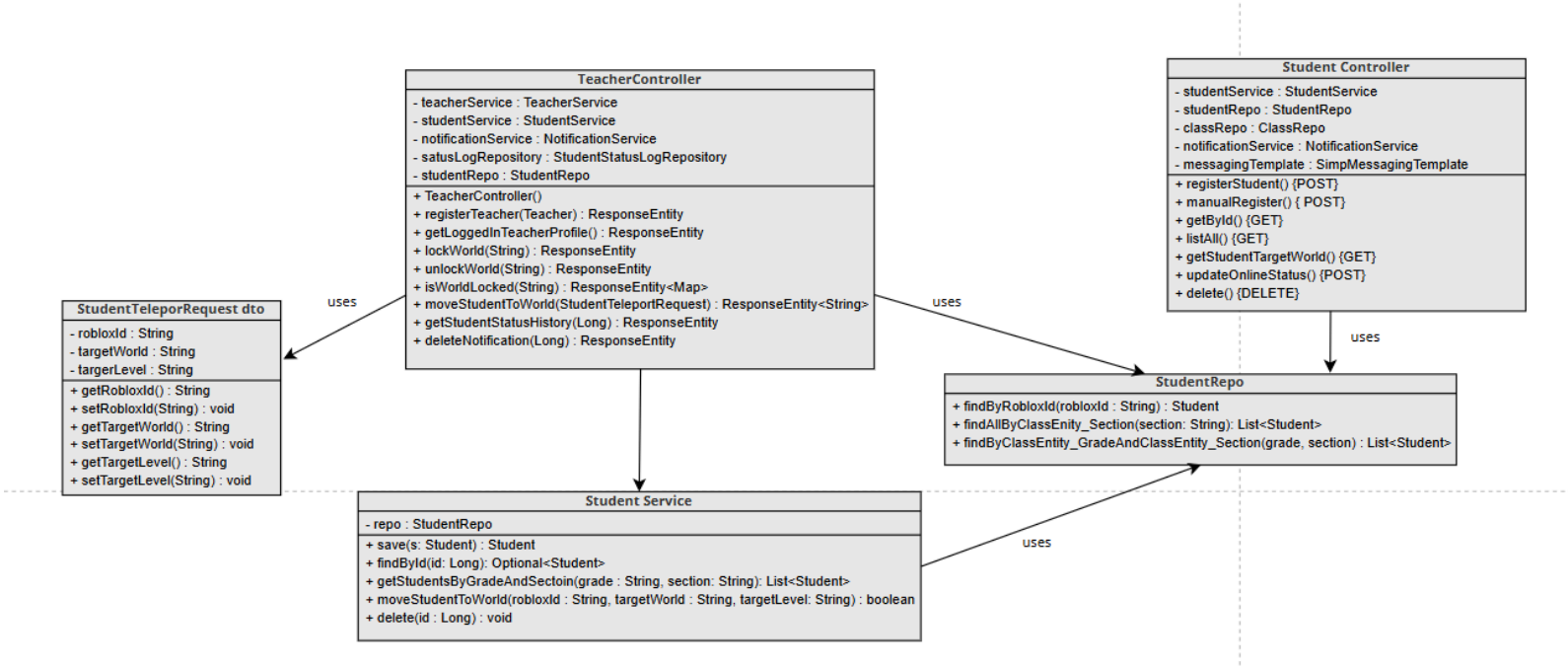
  Represents a student in the CyberKids platform, including their Roblox credentials, real name, class assignment, scores, timers, and gameplay target (world and level). Acts as a core domain object for tracking student progress and status.

- **Component Type:**

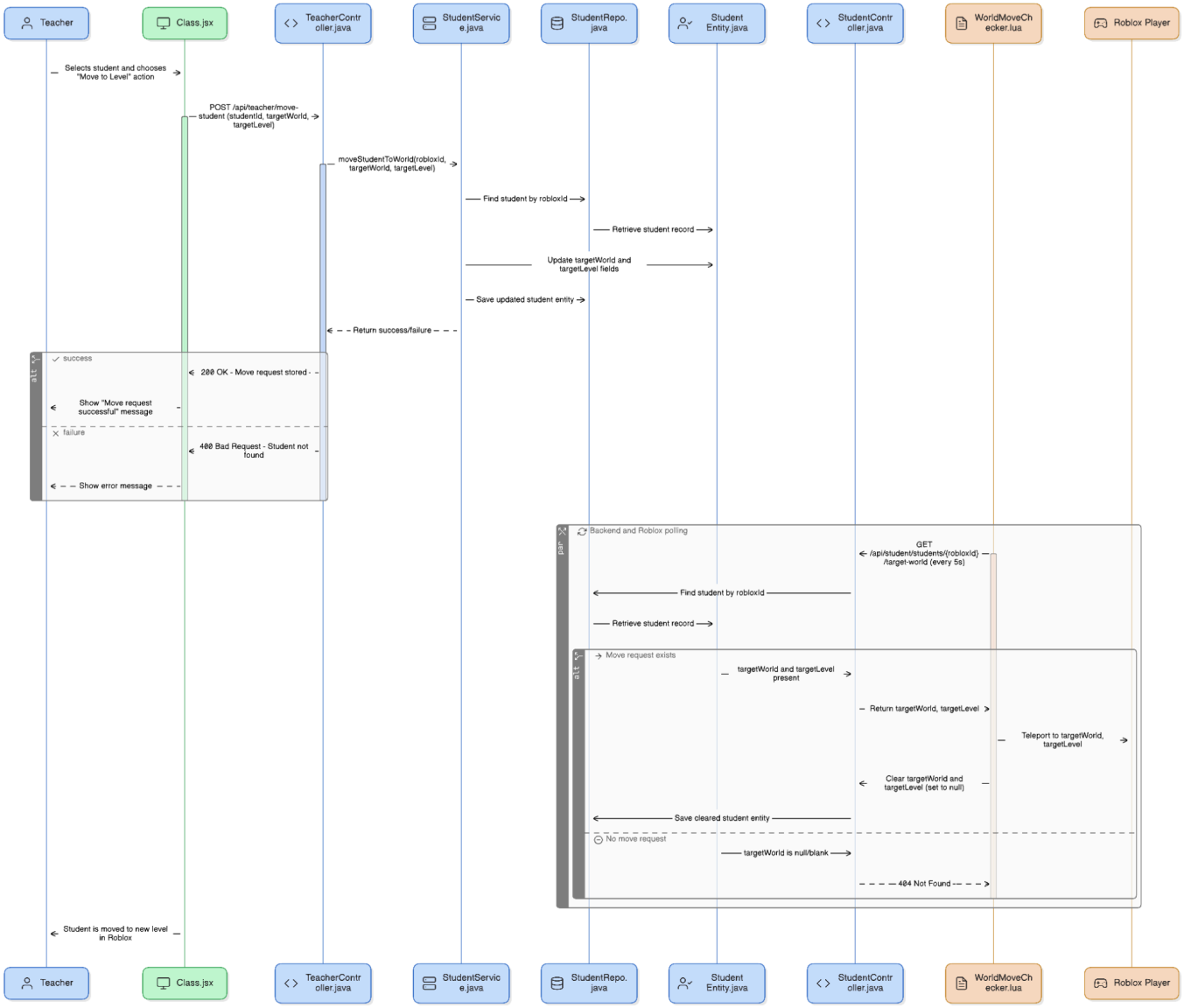
  Entity (Domain Model)

Object-Oriented Components

- Class Diagram

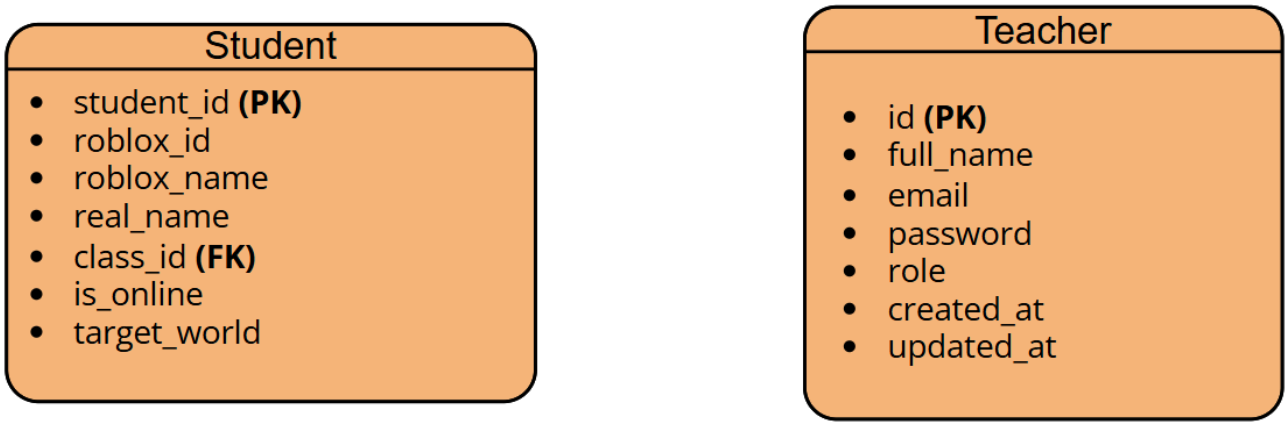


- Sequence Diagram



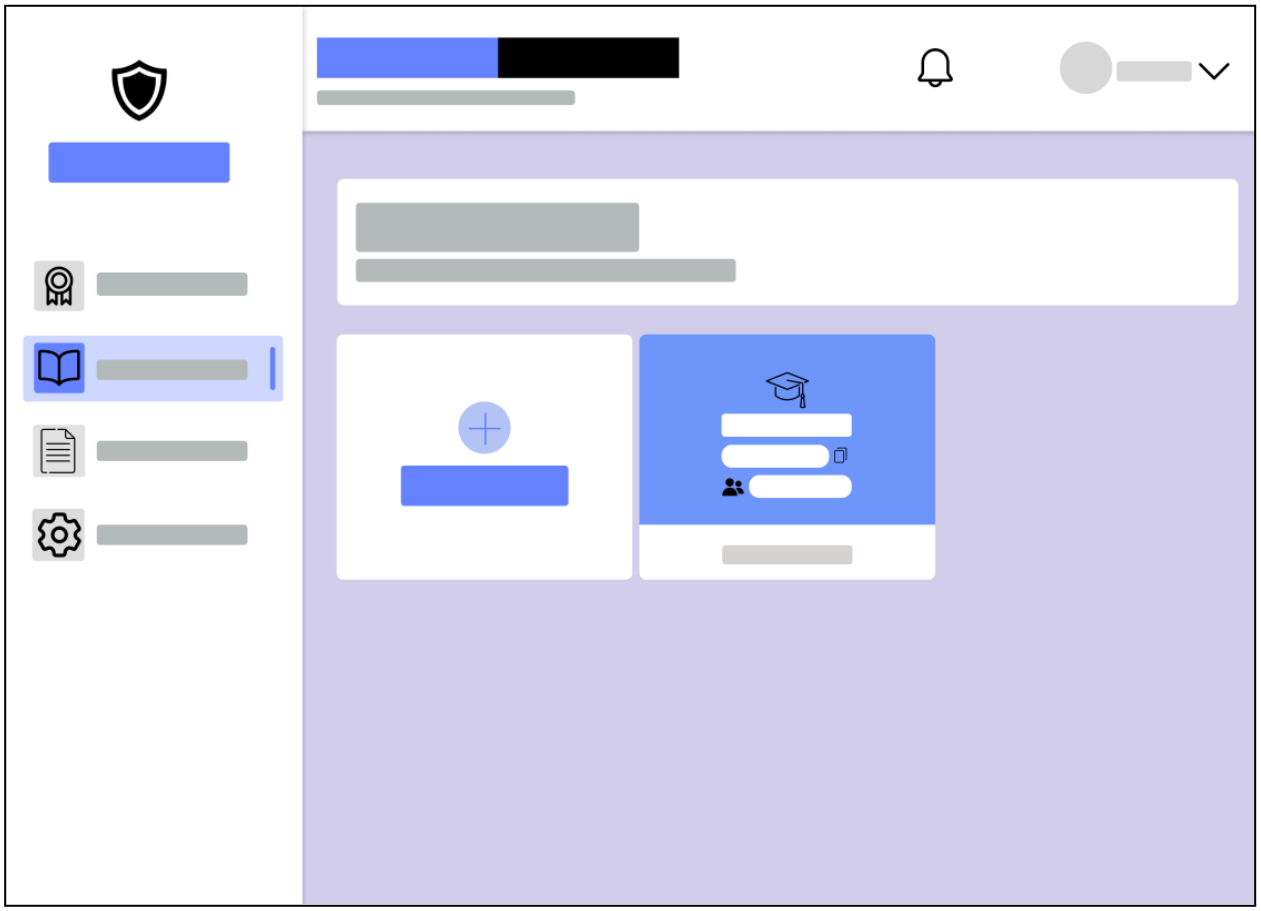
Data Design

- ERD or schema



Transaction 3.3 Class Creation and Enrollment

User Interface Design



Front-end component(s)

1. Clas.jsx

➢ **Description and purpose**:

This is the main React component that handles the complete class management system, including the class creation functionality for Transaction 3.3. In the classes view, it renders a grid of class cards with a dedicated "Create New Class" card that allows teachers to input grade and section information. The component manages the class creation form state and displays existing classes with their auto-generated class codes for student enrollment.

➢ **Component type**:

React Page Component (typically placed in src/components/ClassComponent.jsx)

2. Create New Class Card UI

➢ **Description and purpose**:

An interactive card component within the classes grid that provides the interface for Transaction 3.3. When clicked, it expands to show input fields for grade (e.g., "Grade 5") and section (e.g., "A"). The component toggles between a button state showing "Create New Class" with a plus icon and a form state with grade/section inputs and Create/Cancel buttons. It uses isCreatingClass state to manage the UI transition.

➢ **Component type**:

Inline JSX Component within ClassComponent’s render method

3. handleCreateClass Function

➢ **Description and purpose**:

The core function that processes class creation for Transaction 3.3. It validates that both grade and section fields are filled, sends a POST request to the classes API endpoint with the grade and section data, and refreshes the classes list upon successful creation. The system automatically generates a class code on the backend, which is then displayed in the updated class list for student enrollment purposes.

➢ **Component type**:

Utility Function within ClassComponent

4. Class Display Cards

➢ **Description and purpose**:

Visual components that display created classes with their auto-generated class codes for Transaction 3.3 enrollment functionality. Each card shows the grade and section (e.g., "Grade 5 - A"), the auto-generated class code with a copy button for easy sharing with students, and the number of enrolled students. The class codes enable students to join the class through the enrollment process.

➢ **Component type**:

Inline JSX Component within ClassComponent's render method

5. Class Code Copy Functionality

➢ **Description and purpose**:

A utility feature that supports the enrollment aspect of Transaction 3.3 by allowing teachers to easily copy and share class codes with students. It includes a copy button next to each class code that uses the browser's clipboard API, provides visual feedback when copied (checkmark icon), and includes a fallback method for older browsers. This facilitates the student enrollment process by making class codes easily shareable.

➢ **Component type**:

Utility Function (copyClassCode) within ClassComponent

Back-end component(s)

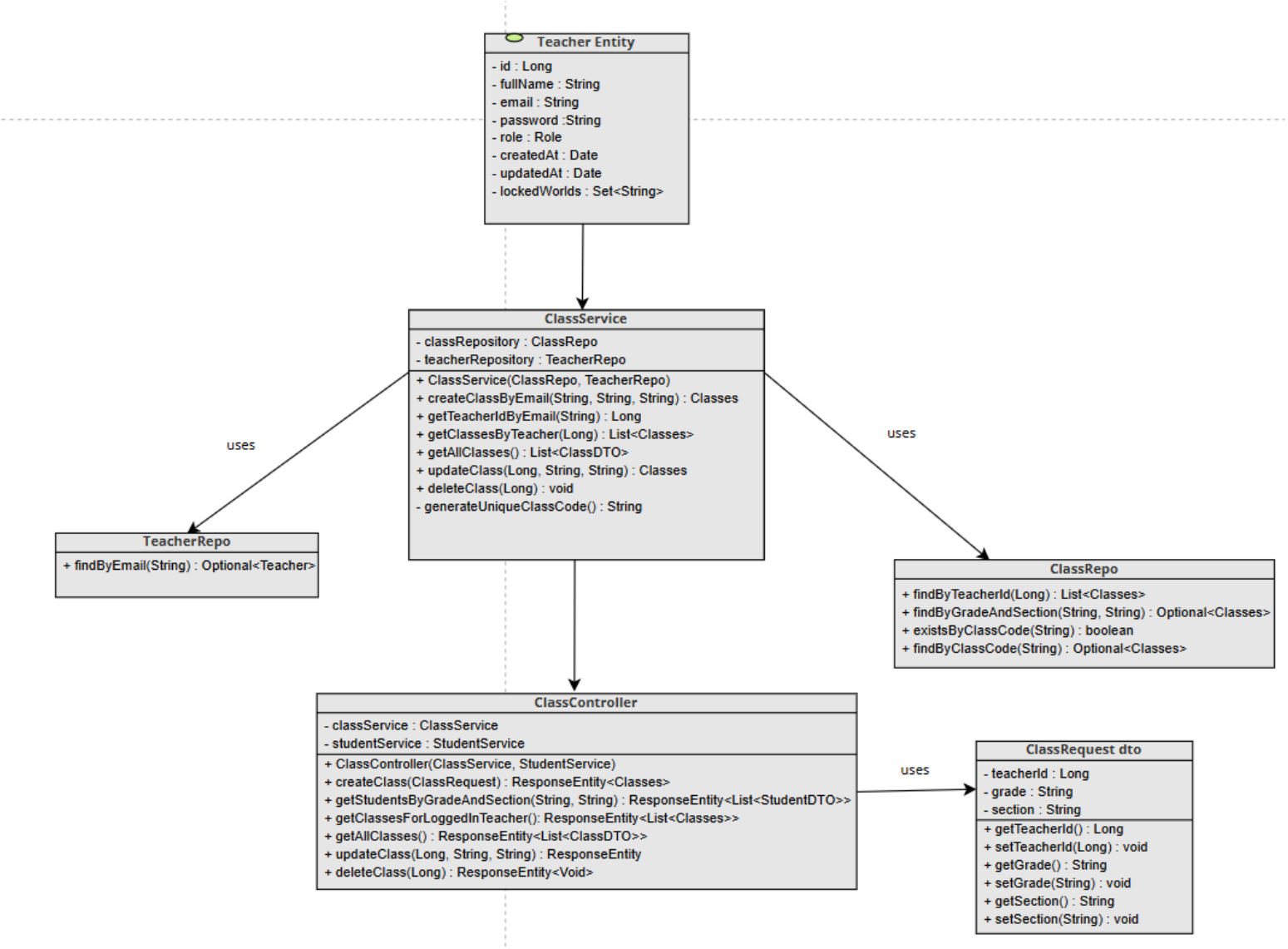
1. **Student Entity**
   - ➢ **Description and Purpose:**
     Represents a student in the CyberKids platform, including their Roblox credentials, real name, class assignment, scores, timers, and gameplay target (world and level). Acts as a core domain object for tracking student progress and status.
   - ➢ **Component Type:**
     Entity (Domain Model)
2. **Teacher Entity**
   - ➢ **Description and Purpose:**
     Represents a teacher user in the system with authentication credentials, assigned role, and tracked locked worlds. It facilitates managing teacher-specific features like class assignment and system access.
   - ➢ **Component Type:**
     Java Class annotated with @Entity (typically stored in the entity or model package).
3. **Teacher Repo**
   - ➢ **Description and purpose:** Provides access to Teacher entity data in the database, extending Spring Data JPA to leverage built-in CRUD operations. Includes a custom method to retrieve a teacher using their email address. Essential for operations that rely on authenticated user data, especially during class creation.
   - ➢ **Component type:** Repository (Data Access Layer)
4. **Class Service**
   - ➢ **Description and purpose:** Handles business logic related to the Classes entity. Supports creating a class linked to a teacher by email, updating and deleting class information, retrieving classes per teacher, and listing all classes in DTO format.
   - ➢ **Component type:** Service (Business Logic Layer)
5. **Class Controller**
   - ➢ **Description and purpose:** Exposes RESTful endpoints for managing classes. Facilitates class creation based on the authenticated teacher's email, retrieval of classes and students per grade/section, and CRUD operations on classes.
   - ➢ **Component type:** Controller (Presentation Layer)

6. Student Service

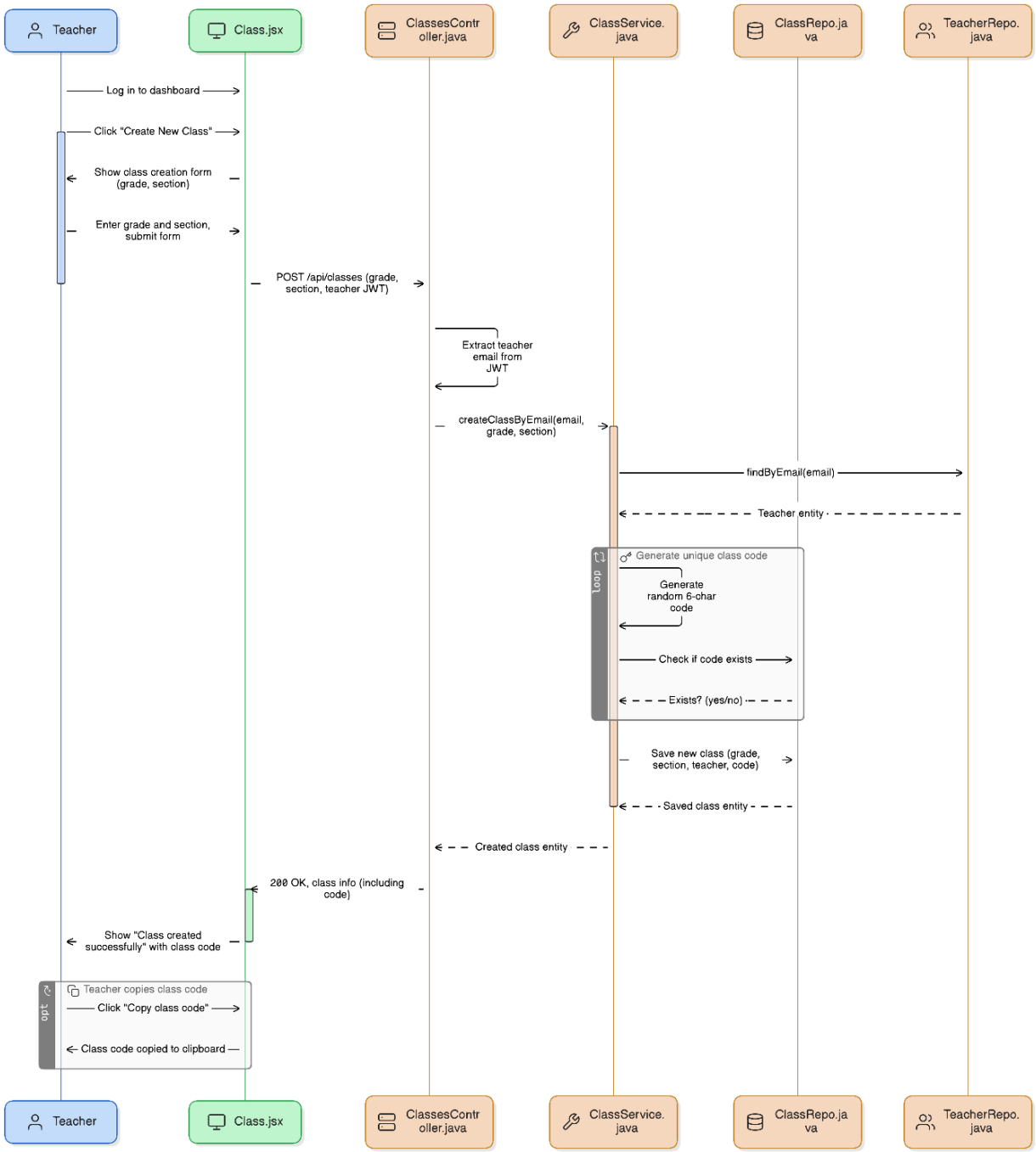
- **Description and purpose:** Encapsulates logic for managing Student entities. Supports operations such as saving new students, retrieving students by grade/section, moving students to new in-game locations, and deleting student records.
- **Component type:** Service (Business Logic Layer)

Object-Oriented Components

- Class Diagram

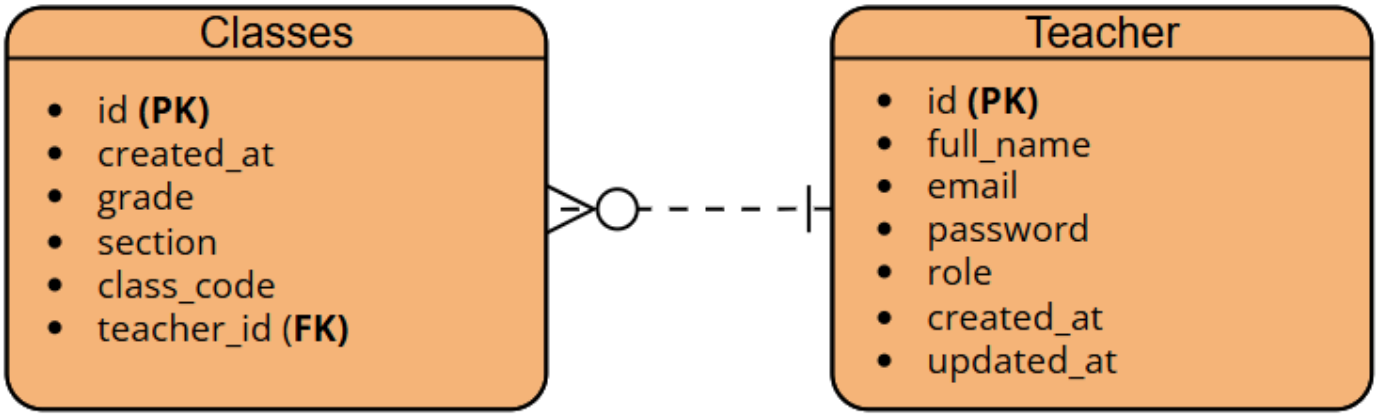


- Sequence Diagram



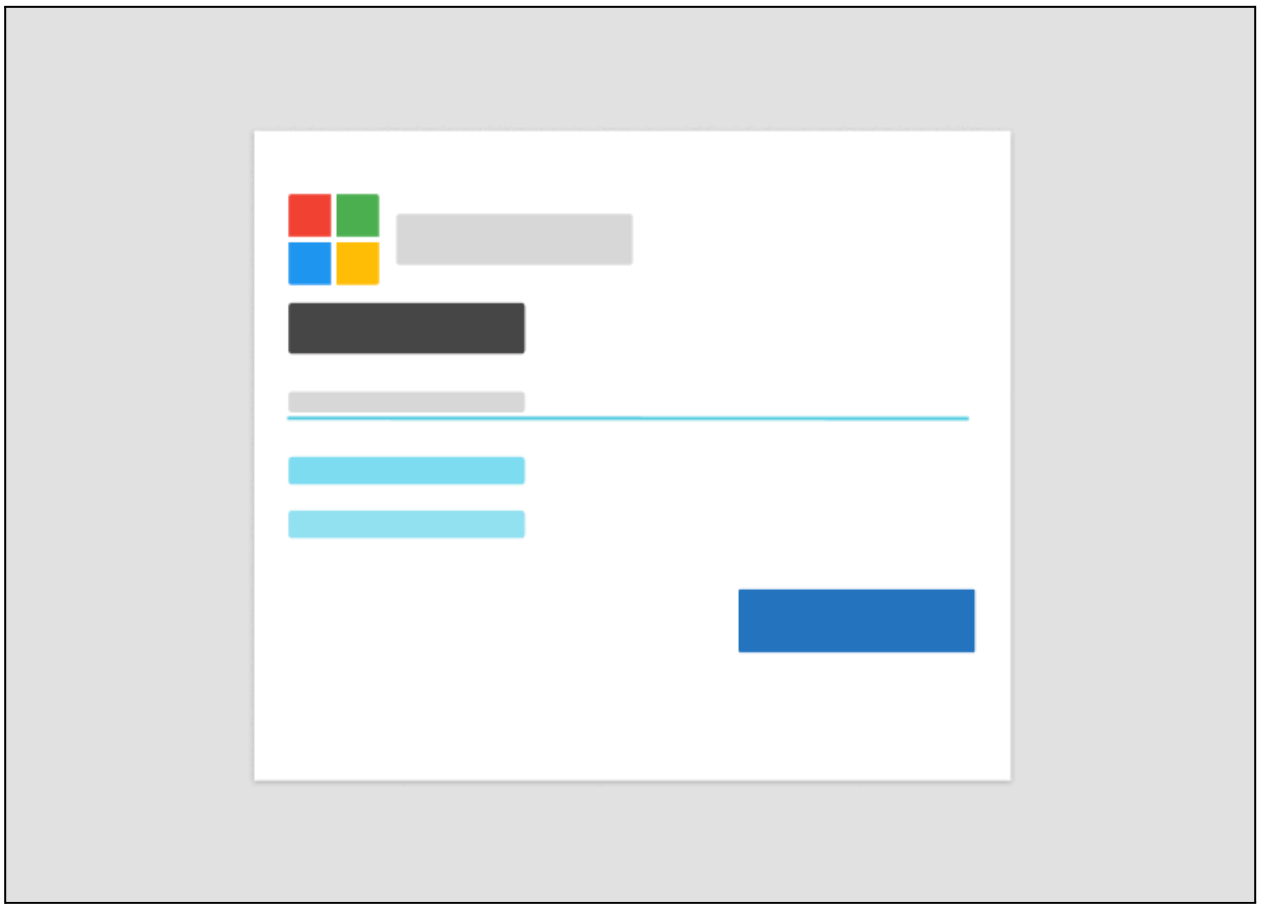
Data Design

- ERD or schema



Transaction 3.4 Microsoft Teams Authentication

User Interface Design



Front-end component(s)

1. **Microsoft Teams Login Button**
   - ➢ **Description and Purpose:**

     A button component integrated within the signup form that initiates the Microsoft Teams OAuth2 authentication flow. When clicked, it redirects users to the Microsoft OAuth2 authorization endpoint through the backend API.
   - ➢ **Component Type:**

     React functional component button element with click handler, located within the Signup component.
2. **handleMicrosoftLogin Function**
   - ➢ **Description and Purpose:**

     JavaScript function that handles the Microsoft Teams login initiation by redirecting the browser to the backend OAuth2 authorization endpoint. Constructs the proper URL using the API_URL environment variable.
   - ➢ **Component Type:**

     JavaScript function within React component, handles OAuth flow initiation.
3. **OAuthCallback Component**
   - ➢ **Description and Purpose:**

     React component that processes the OAuth2 redirect callback from Microsoft Teams authentication. Extracts authentication parameters (token, userId, role, email, name) from URL query parameters, stores them in localStorage, and redirects to the dashboard upon successful authentication.
   - ➢ **Component Type:**

     React functional component with useEffect hooks for URL parameter processing and navigation.

4. **processOAuthRedirect Function**
   - ➢ **Description and Purpose:**

     Core processing function within OAuthCallback that handles parameter extraction, token storage using jwtUtils, user data creation, and error handling for the OAuth authentication flow. Manages the complete post-authentication workflow.
   - ➢ **Component Type:**

     Async JavaScript function within React component, handles OAuth callback processing.
5. **jwtUtils Integration**
   - ➢ **Description and Purpose:**
     - Utility integration for JWT token management that stores authentication tokens received from the Microsoft Teams OAuth flow. Provides consistent token storage mechanism for both traditional and OAuth authentication methods.
   - ➢ **Component Type:** Imported utility module with token management functions, used across authentication components.
6. **OAuth Error Handling UI**
   - ➢ **Description and Purpose:**

     User interface elements that display authentication errors, loading states, and success messages during the Microsoft Teams OAuth process. Provides visual feedback and user guidance throughout the authentication flow.
   - ➢ **Component Type:**

     React JSX elements with conditional rendering based on authentication state, includes error messages and loading spinners.
7. **URL Parameter Parser**
   - ➢ **Description and Purpose:**

     JavaScript functionality that extracts OAuth callback parameters from URL query string including token, userId, role, email, and name. Handles URL decoding and parameter validation for the authentication process.
   - ➢ **Component Type:**

     JavaScript URLSearchParams API usage within React component for OAuth callback processing.

8. Navigation Integration

- **Description and Purpose:**

  React Router navigation logic that handles redirection to dashboard after successful OAuth authentication or back to login page on authentication failure. Manages proper route transitions in the single-page application.

- **Component Type:**

  React Router useNavigate hook integration with fallback window.location handling.

Back-end component(s)

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

     Represents the database table for teachers/educators in the system. Stores fields like id, fullName, email, password, role, and lockedWorlds to manage teacher accounts, authentication, and world access control functionality.
   - ➢ **Component Type:**

     Java Class annotated with @Entity, located in the Entity package.
2. **TeacherRepo**
   - ➢ **Description and Purpose:**

     Communicates directly with the database to perform CRUD operations for Teacher entities. Uses Spring Data JPA to auto-generate SQL logic and provides methods like findByEmail() for OAuth authentication flow.
   - ➢ **Component Type:**

     Java Interface extending JpaRepository<Teacher, Long>, located in the Repository package.
3. **OAuthController**
   - ➢ **Description and Purpose:**

     Handles incoming HTTP requests from Microsoft Teams OAuth2 authentication flow. Processes OAuth2User data, creates or finds teacher accounts, generates JWT tokens, and redirects users to the frontend with authentication credentials.
   - ➢ **Component Type:**

     Java Class annotated with @RestController, located in the MsTeams package.
4. **JwtUtil**
   - ➢ **Description and Purpose:**

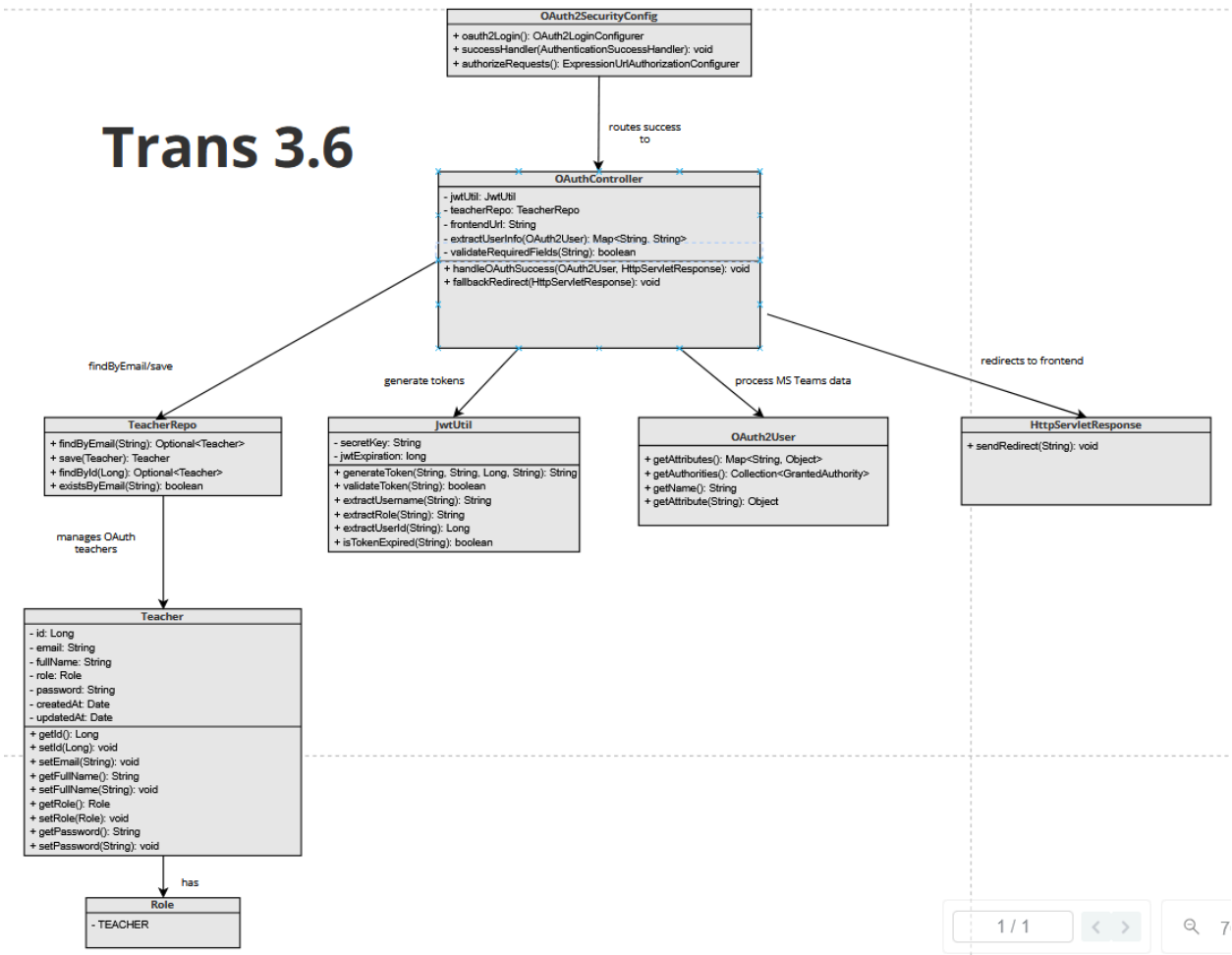
     Utility class responsible for JWT token generation, validation, and extraction of user information. Creates secure tokens containing teacher email, role, and ID for session management across the application.
   - ➢ **Component Type:**

     Java Class annotated with @Component, located in the JWT package.

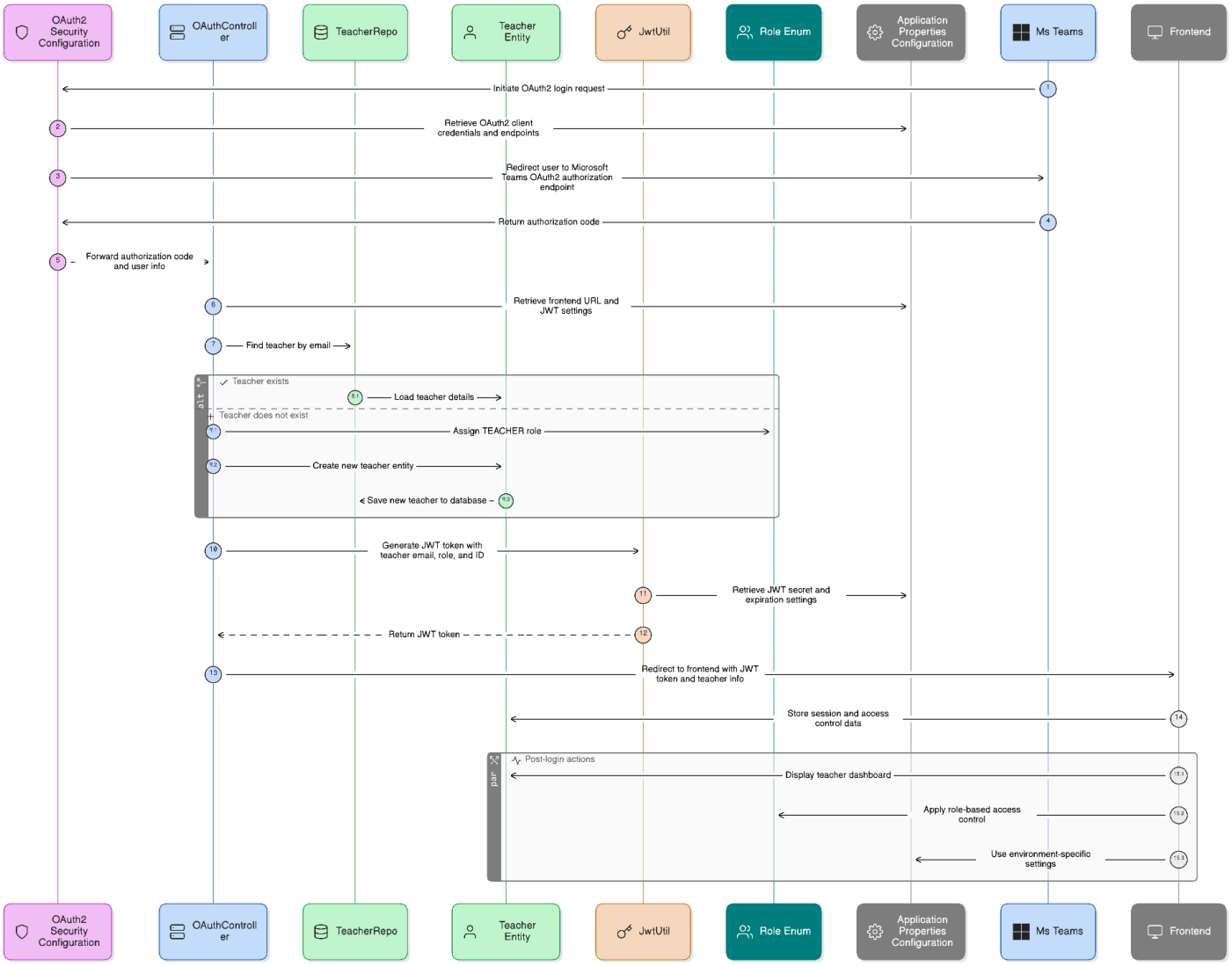
5. **Role Enum**
   - ➢ **Description and Purpose:**
     Defines user roles within the system (TEACHER, ADMIN, STUDENT) for role-based access control. Used to assign appropriate permissions and access levels to authenticated users.
   - ➢ **Component Type:**
     Java Enum, located in the Model package.
6. **OAuth2 Security Configuration**
   - ➢ **Description and Purpose:**
     Configures Spring Security OAuth2 settings for Microsoft Teams integration. Defines authentication endpoints, success handlers, and security policies for the OAuth2 authentication flow.
   - ➢ **Component Type:**
     Java Class annotated with @Configuration and @EnableWebSecurity, located in the Security package.
7. **Application Properties Configuration**
   - ➢ **Description and Purpose:**
     Stores configuration values such as frontend URL, OAuth2 client credentials, and JWT settings. Provides externalized configuration for different environments (development, production).
   - ➢ **Component Type:**
     Properties file (application.yml or application.properties) with corresponding @Value annotations in Java classes.

Object-Oriented Components

- Class Diagram

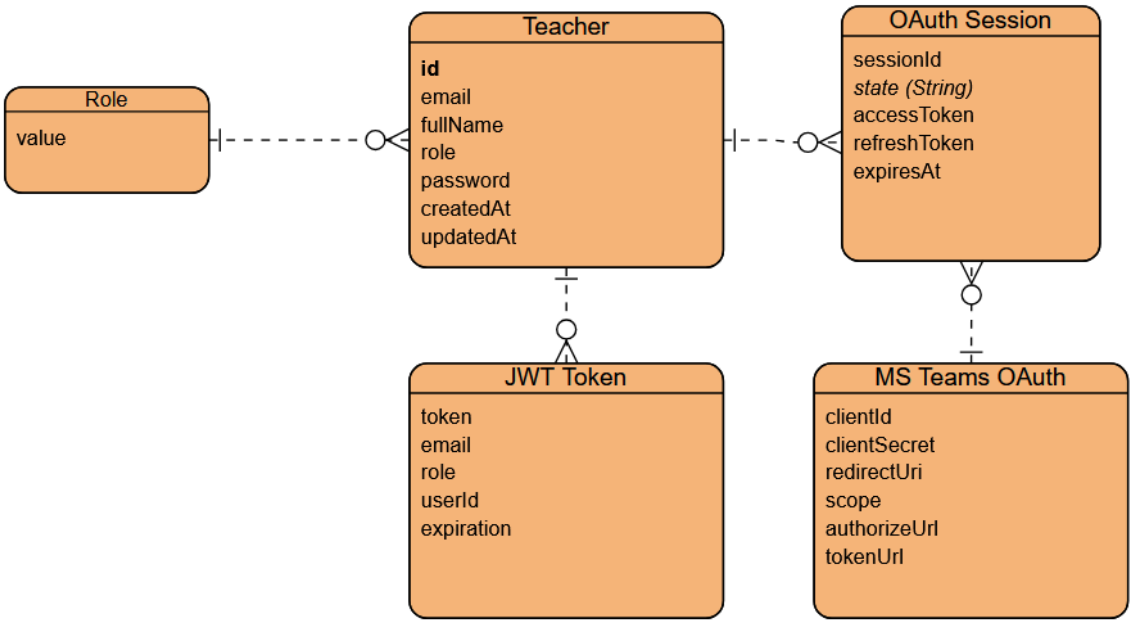


- Sequence Diagram



Data Design

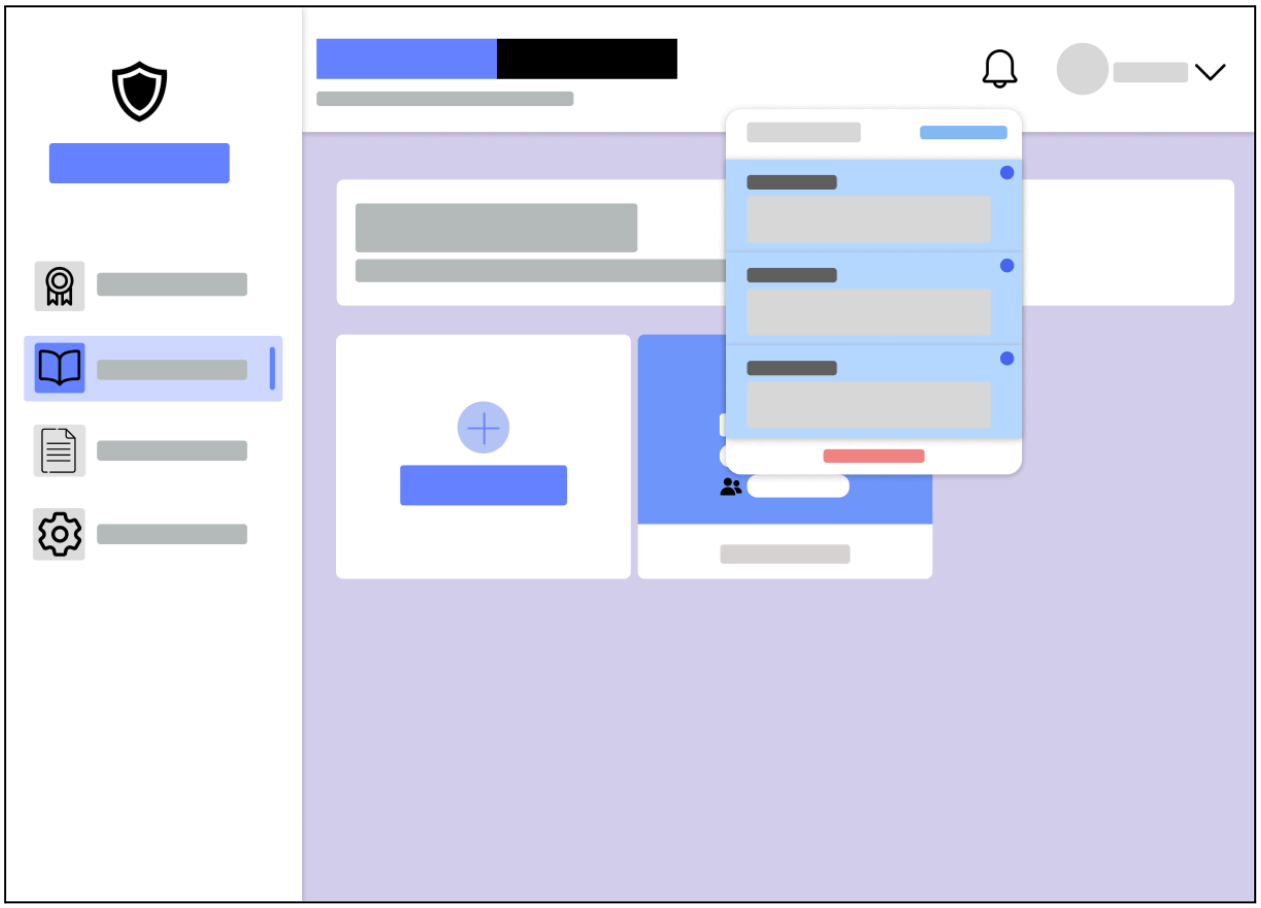
- ERD or schema



Module 4: Real-Time Communication System

Transaction 4.1 Delete Retrieved Notifications

User Interface Design



Front-end component(s)

1. TeacherDashboard.jsx

➢ **Description and purpose**: This is the main dashboard component that handles the complete teacher interface, including the notification management system for Transaction 4.1. It provides a notification bell icon in the header that displays unread notification count, a dropdown menu showing recent notifications, and a full notification modal. Each notification includes a "Delete" button that permanently removes the notification from the system by calling the deleteNotification function.

➢ **Component type**: React Page Component (typically placed in src/components/TeacherDashboard.jsx)

2. deleteNotification Function

➢ **Description and purpose**: The core function that handles Transaction 4.1 by permanently deleting notifications from the teacher's dashboard. It sends a DELETE request to the API endpoint (${API_URL}/api/teacher/notification/${id}) with authentication headers, removes the notification from the local state, and updates the unread count if the deleted notification was unread. The function includes error handling with user feedback for failed deletion attempts.

➢ **Component type**: Utility Function within TeacherDashboard

3. Notification Dropdown UI

➢ **Description and purpose**: A dropdown component that appears when clicking the notification bell icon, displaying recent notifications with options to mark as read or delete. Each notification item shows the title, message, timestamp, and includes a delete option for Transaction 4.1. The dropdown provides quick access to notification management without opening the full modal.

➢ **Component type**: Inline JSX Component within TeacherDashboard’s render method

4. Notification Modal

➢ **Description and purpose**: A full-screen modal that displays all notifications with enhanced management capabilities for Transaction 4.1. It shows notifications in a scrollable list format, each with individual "Mark as read" and "Delete" buttons. The modal provides bulk actions like "Mark all as read" and gives teachers a comprehensive view of all their notifications with the ability to permanently delete any notification.

➢ **Component type**: Inline JSX Component within TeacherDashboard's render method

5. Notification State Management

➢ **Description and purpose**: A state management system that tracks notifications array, unread count, and handles real-time updates via WebSocket. For Transaction 4.1, it manages the removal of deleted notifications from the local state, ensuring the UI immediately reflects the deletion without requiring a page refresh. It also handles updating the unread count when deleted notifications were previously unread.

➢ **Component type**: State Management Logic within TeacherDashboard

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. **Teacher Entity**
   - ➢ **Description and Purpose:**

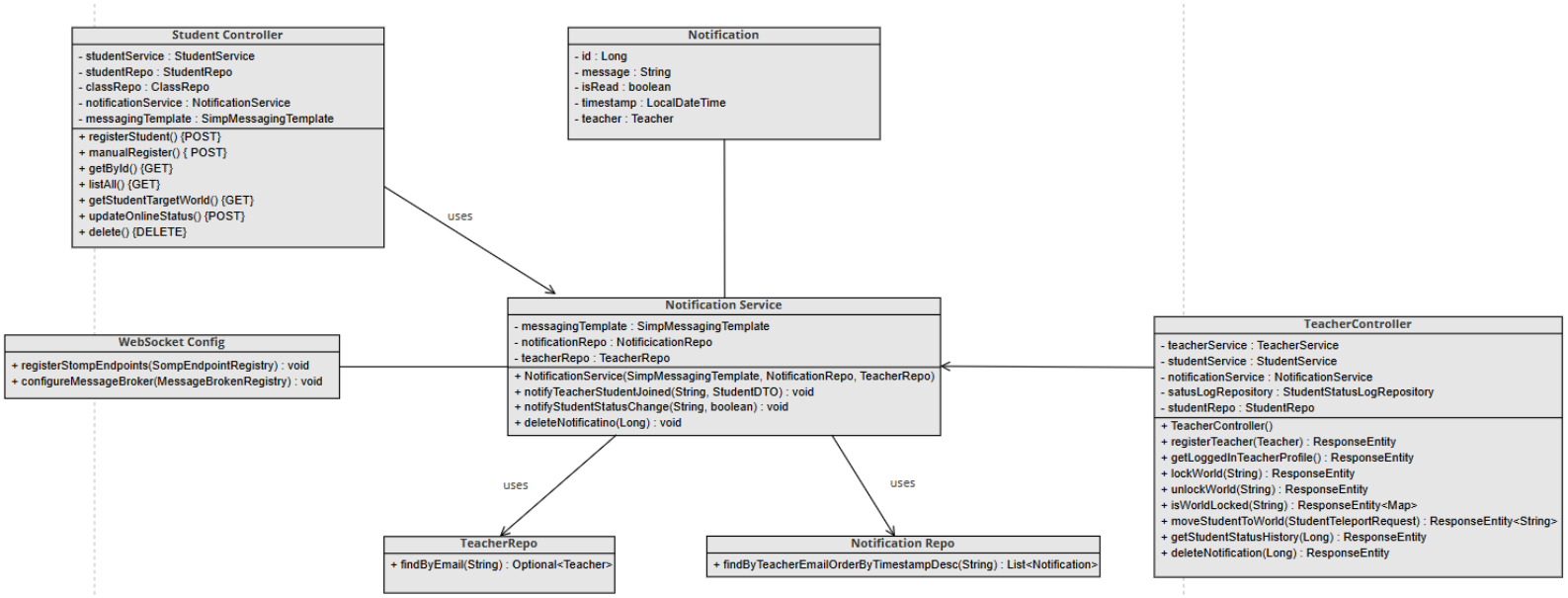
     Represents a teacher user in the system with authentication credentials, assigned role, and tracked locked worlds. It facilitates managing teacher-specific features like class assignment and system access.
   - ➢ **Component Type:**

     Java Class annotated with @Entity (typically stored in the entity or model package).

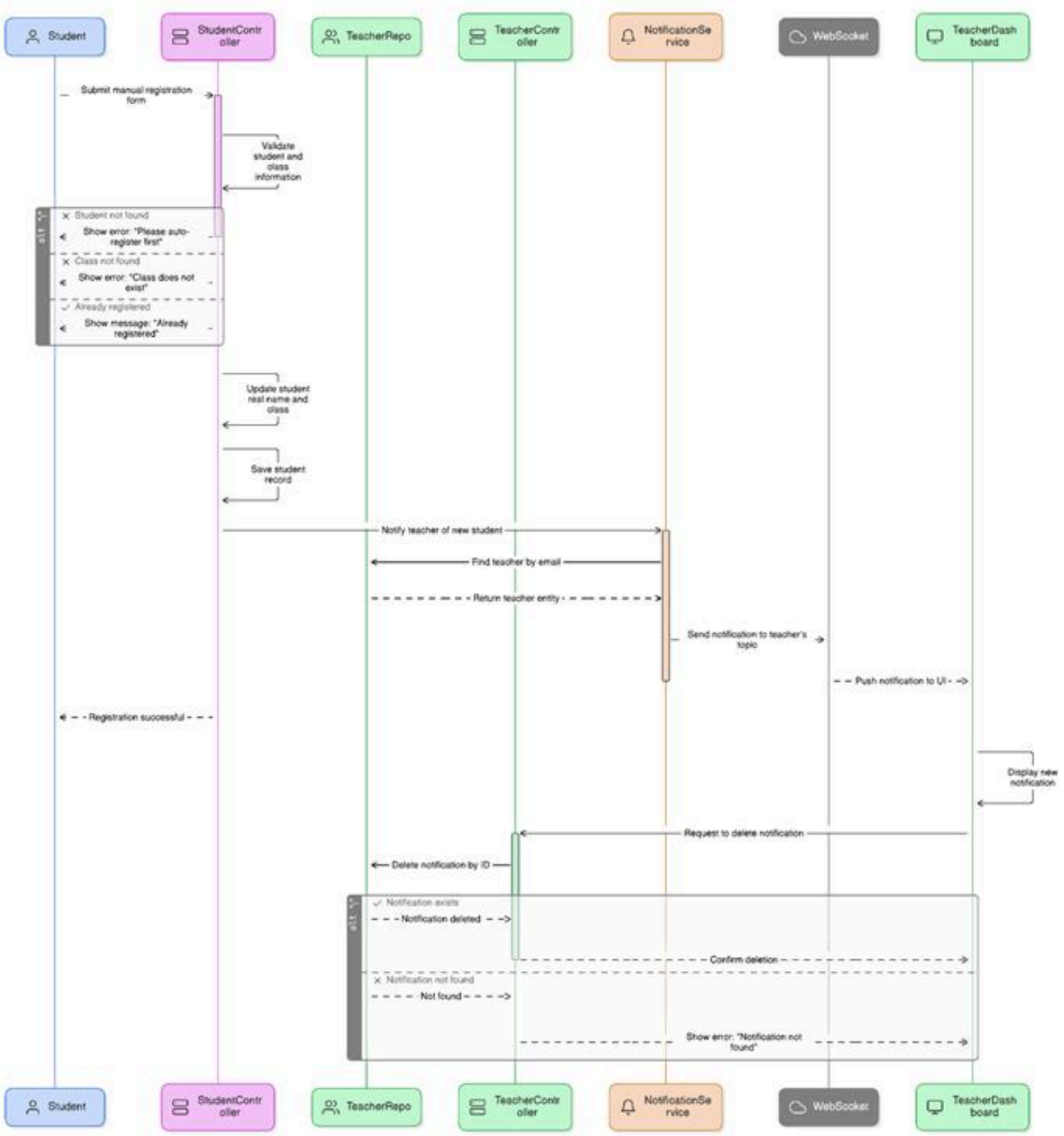
3. **Notification Entity**
   - ➢ **Description and Purpose:**
     Data Transfer Object (DTO) sent to the frontend over WebSocket or REST. Contains only necessary fields: message and timestamp.
   - ➢ **Component Type:** DTO Class – typically placed under src/main/java/.../dto/NotificationDTO.java.
4. **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.
5. **Teacher Controller**
   - ➢ **Description and Purpose:**
     Exposes REST API endpoints for teacher-related operations such as registering teachers, locking/unlocking worlds, teleporting students, retrieving student status logs, accessing teacher profile, and managing notifications. It coordinates with services like TeacherService, StudentService, and NotificationService.
   - ➢ **Component Type:** Controller Class – typically placed under src/main/java/.../controller/TeacherController.java.

Object-Oriented Components

- Class Diagram

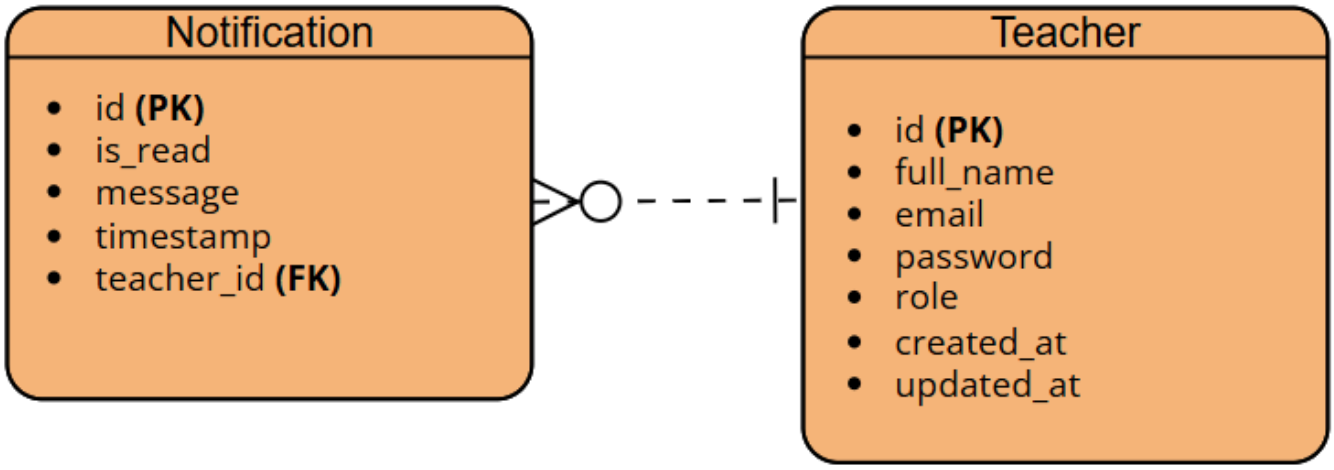


- Sequence Diagram



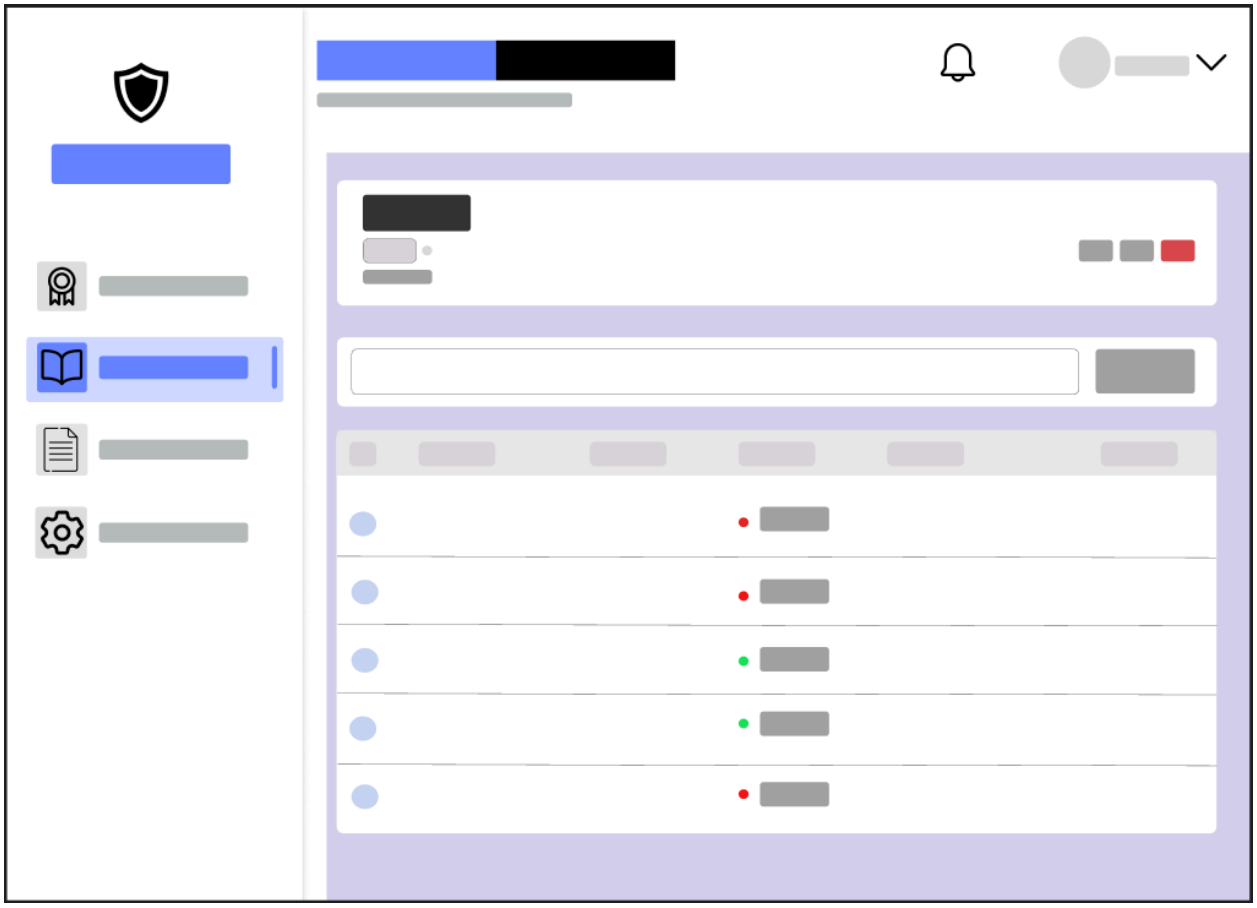
Data Design

- ERD or schema



Transaction 4.2 Track Online/Offline Status of Students

User Interface Design



Front-end component(s)

1. Clas.jsx

➢ **Description and purpose**: This is the main React component that handles the complete class-management system, including real-time student-status tracking for Transaction 4.2. It establishes a WebSocket connection to receive live online/offline status updates from students and displays visual indicators (green/red dots) next to each student's name in the students table. The component also provides access to detailed status history through expandable sections in the student-detail view.

➢ **Component type**: React Page Component (typically placed in src/components/ClassComponent.jsx)

2. WebSocket Status-Tracking System

➢ **Description and purpose**: A real-time communication system that connects to the student-status WebSocket endpoint (/topic/student-status) to receive live updates about student online/offline status for Transaction 4.2. It automatically updates the UI when students come online or go offline, maintaining the studentStatuses state object that maps Roblox IDs to current online status. The system includes reconnection logic and error handling for robust real-time tracking.

➢ **Component type**: WebSocket integration within ClassComponent (useEffect hook with STOMP client)

3. getOnlineStatus Function

➢ **Description and purpose**: A utility function that determines the current online/offline status of a student for Transaction 4.2 display purposes. It prioritizes real-time WebSocket data from studentStatuses state, falls back to the stored student.online property from the database, and defaults to "offline" if no status information is available. This ensures consistent status display across the application.

➢ **Component type**: Utility Function within ClassComponent

4. fetchStudentStatusHistory Function

➢ **Description and purpose**: An API-integration function that retrieves the complete online/offline status history for a specific student to support Transaction 4.2. It makes a GET request to ${TEACHER_API_URL}/student-status-history/${studentId} and populates the statusHistory state with timestamped records of when the student went online or offline. The function includes loading states and error handling for a smooth user experience.

➢ **Component type**: API-integration Function within ClassComponent

5. Status-History Display Component

➢ **Description and purpose**: An expandable UI component within the student-detail view that presents the historical online/offline status data for Transaction 4.2. It renders as a “View History” button that expands to show a table with timestamps and status changes (online/offline), with appropriate colour coding (green for online, red for offline). The component includes loading states, empty states, and scrollable content for large history datasets.

➢ **Component type**: Inline JSX Component within ClassComponent’s student-detail view

6. Real-time Status Indicators

➢ **Description and purpose**: Visual status indicators displayed throughout the student interface for Transaction 4.2, showing current online/offline status with coloured dots (green for online, red for offline) next to student names. These indicators automatically update in real time based on WebSocket messages and provide immediate visual feedback about student availability without requiring page refreshes.

➢ **Component type**: Inline JSX Components within ClassComponent’s student list and detail views

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. **Teacher Entity**

- **Description and Purpose:**

  Represents a teacher user in the system with authentication credentials, assigned role, and tracked locked worlds. It facilitates managing teacher-specific features like class assignment and system access.

- **Component Type:**

  Java Class annotated with @Entity (typically stored in the entity or model package).

3. **Notification Entity**

- ➢ **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.

4. **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.

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. Teacher Controller

- ➢ **Description and Purpose:**

  Exposes REST API endpoints for teacher-related operations such as registering teachers, locking/unlocking worlds, teleporting students, retrieving student status logs, accessing teacher profile, and managing notifications. It coordinates with services like TeacherService, StudentService, and NotificationService.

- ➢ **Component Type:** Controller Class – typically placed under src/main/java/.../controller/TeacherController.java.

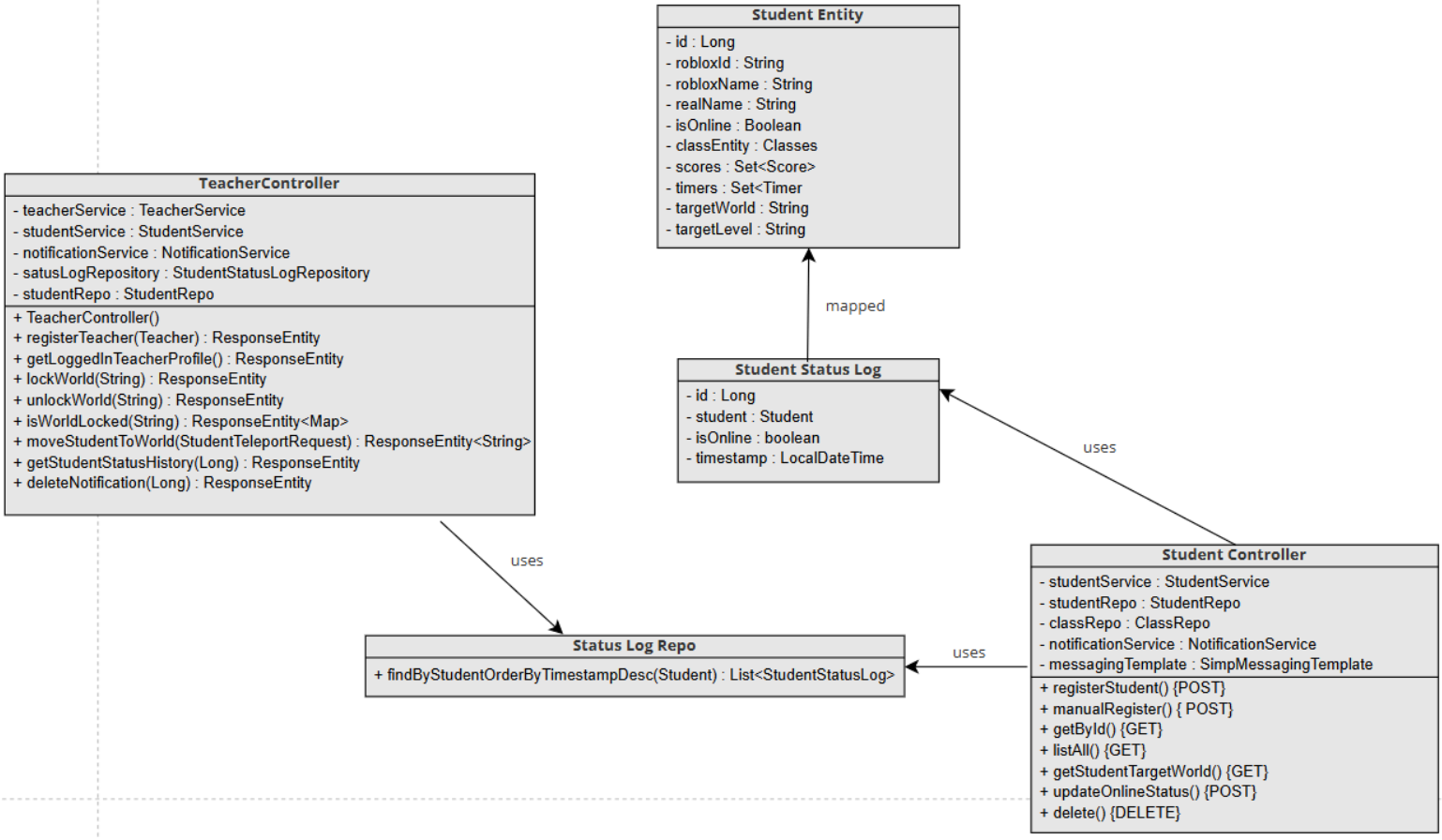
7. **StudentStatusLog Entity**
   - ➢ **Description and Purpose:**

     Represents a log entry for a student's online status. Each entry includes a reference to a Student, a boolean indicating if the student is online, and a timestamp of the status change. Used for tracking the history of a student's status over time.
   - ➢ **Component Type:** Entity Class – typically placed under src/main/java/.../entity/StudentStatusLog.java.
8. **StudentStatusLog Repository**
   - ➢ **Description and Purpose:**

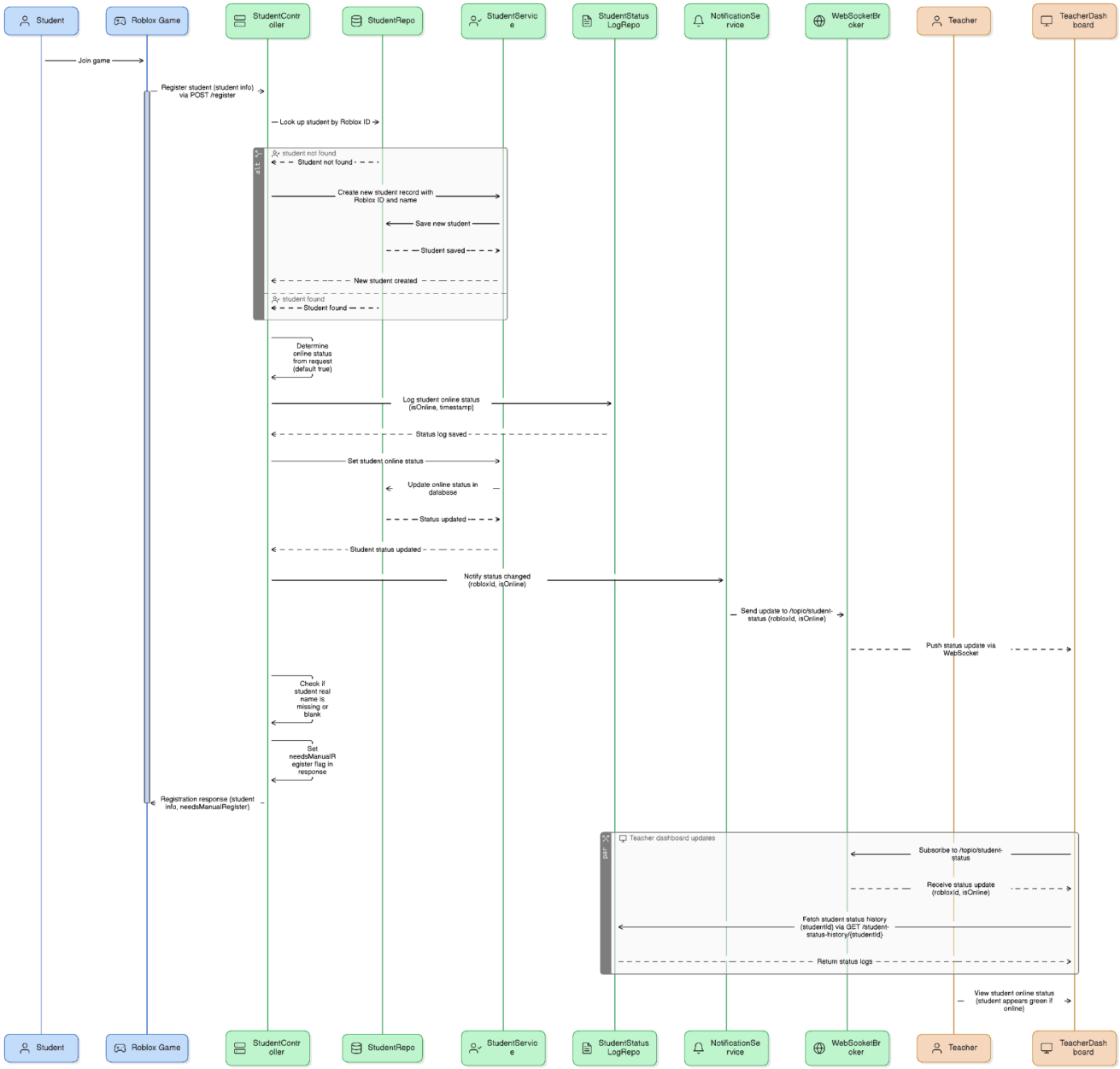
     Provides database access for StudentStatusLog entities, including a method to retrieve status logs for a given student ordered by timestamp in descending order.
   - ➢ **Component Type:** Repository Interface

Object-Oriented Components

- Class Diagram

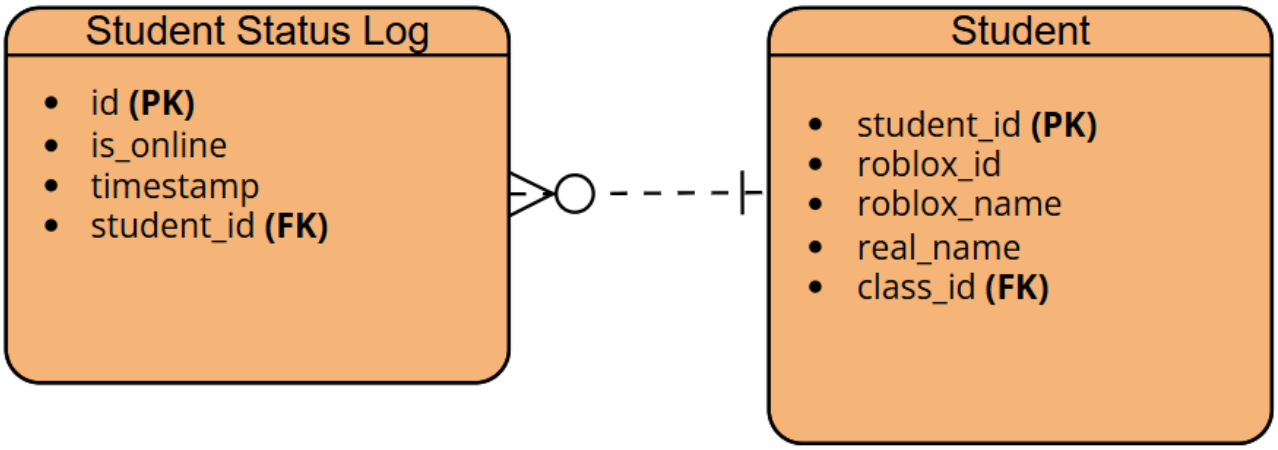


- Sequence Diagram



Data Design

- ERD or schema



Transaction 4.3 Real-Time Student Progress Notification

User Interface Design

Front-end component(s)

1. TeacherDashboard.jsx

➢ **Description and purpose**: This is the main dashboard component that handles the complete teacher interface, including the notification management system for Transaction 4.1. It provides a notification bell icon in the header that displays unread notification count, a dropdown menu showing recent notifications, and a full notification modal. Each notification includes a "Delete" button that permanently removes the notification from the system by calling the deleteNotification function.

➢ **Component type**: React Page Component (typically placed in src/components/TeacherDashboard.jsx)

2. Notification Dropdown UI

➢ **Description and purpose**: A dropdown component that appears when clicking the notification bell icon, displaying recent notifications with options to mark as read or delete. Each notification item shows the title, message, timestamp, and includes a delete option for Transaction 4.1. The dropdown provides quick access to notification management without opening the full modal.

➢ **Component type**: Inline JSX Component within TeacherDashboard's render method

3. Notification Modal

➢ **Description and purpose**: A full-screen modal that displays all notifications with enhanced management capabilities for Transaction 4.1. It shows notifications in a scrollable list format, each with individual "Mark as read" and "Delete" buttons. The modal provides bulk actions like "Mark all as read" and gives teachers a comprehensive view of all their notifications with the ability to permanently delete any notification.

➢ **Component type**: Inline JSX Component within TeacherDashboard's render method

4. Notification State Management

➢ **Description and purpose**: A state management system that tracks notifications array, unread count, and handles real-time updates via WebSocket. For Transaction 4.1, it manages the removal of deleted notifications from the local state, ensuring the UI immediately reflects the deletion without requiring a page refresh. It also handles updating the unread count when deleted notifications were previously unread.

➢ **Component type**: State Management Logic within TeacherDashboard

Back-end component(s)

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

     Represents the database table for storing persistent notifications sent to teachers. Contains fields like id, message, timestamp, isRead, and teacher reference for tracking notification history and status.
   - ➢ **Component Type:**

     Java Class annotated with @Entity, located in the Entity package.
2. **NotificationRepo**
   - ➢ **Description and Purpose:**

     Provides data access for Notification entities including CRUD operations, finding notifications by teacher, and managing notification persistence for real-time messaging system.
   - ➢ **Component Type:**

     Java Interface extending JpaRepository<Notification, Long>, located in the Repository package.
3. **NotificationService**
   - ➢ **Description and Purpose:**

     Contains business logic for real-time notification delivery using WebSocket messaging, persistent notification storage, and teacher-student progress communication including game completion and status changes.
   - ➢ **Component Type:**

     Java Class annotated with @Service, located in the Service package.
4. **NotificationController**
   - ➢ **Description and Purpose:**

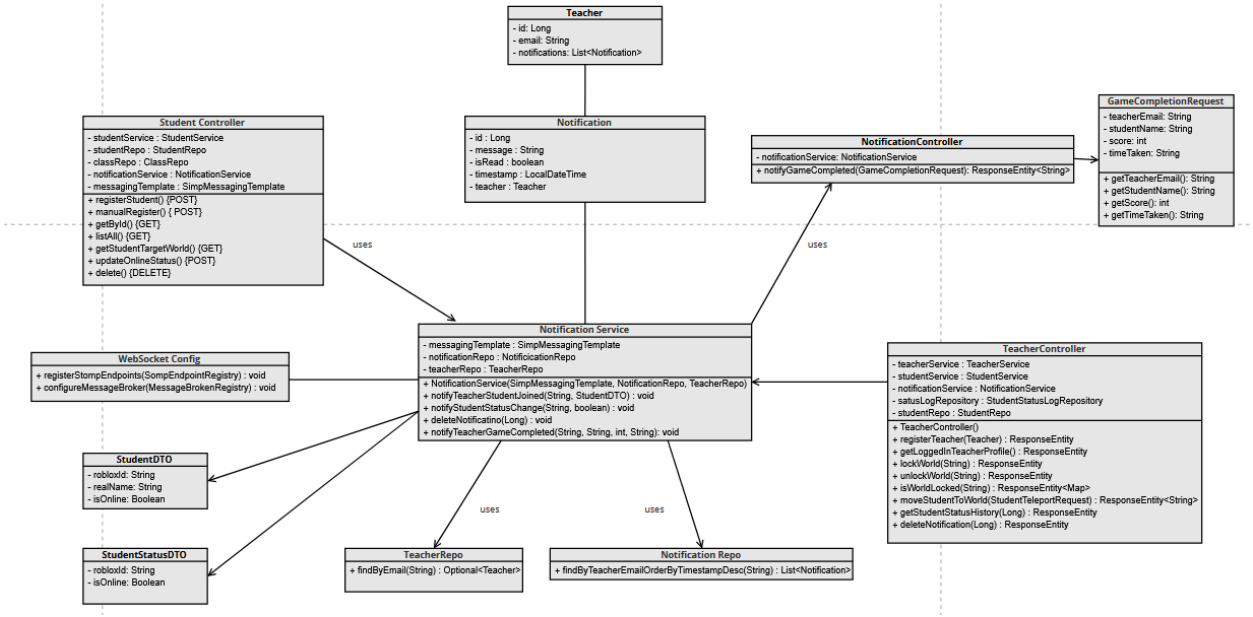
     Handles HTTP requests for notification operations including game completion notifications from Roblox clients and notification management endpoints for teachers.
   - ➢ **Component Type:**

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

5. **Teacher Entity**
   - ➢ **Description and Purpose:**
     Required for notification ownership and teacher identification in real-time messaging system. Links notifications to specific teachers for targeted message delivery.
   - ➢ **Component Type:**
     Java Class annotated with @Entity, located in the Entity package.
6. **GameCompletionRequest DTO**
   - ➢ **Description and Purpose:**
     Data Transfer Object for receiving game completion notifications from Roblox clients including teacher email, student name, score, and time taken information.
   - ➢ **Component Type:**
     Java POJO (Plain Old Java Object) used in HTTP requests, located in the DTOs package.
7. **StudentDTO**
   - ➢ **Description and Purpose:**
     Data Transfer Object used in student registration notifications to teachers, containing student information for real-time updates.
   - ➢ **Component Type:**
     Java POJO used in notification messaging, located in the DTOs package.
8. **StudentStatusDTO**
   - ➢ **Description and Purpose:**
     Data Transfer Object for broadcasting student online/offline status changes to teachers through WebSocket messaging for real-time student monitoring.
   - ➢ **Component Type:**
     Java POJO used in status change notifications, located in the DTOs package.

Object-Oriented Components

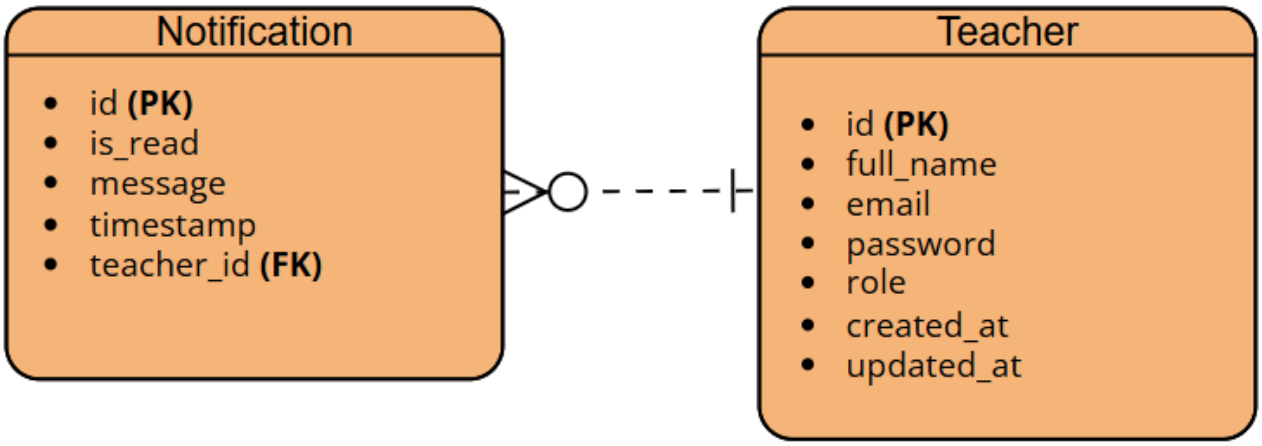
- Class Diagram



- Sequence Diagram

Data Design

- ERD or schema



Transaction 4.4 Real-Time Student Registration Notification

User Interface Design

Front-end component(s)

1. TeacherDashboard.jsx

➢ **Description and purpose**: This is the main dashboard component that handles the complete teacher interface, including the notification management system for Transaction 4.1. It provides a notification bell icon in the header that displays unread notification count, a dropdown menu showing recent notifications, and a full notification modal. Each notification includes a "Delete" button that permanently removes the notification from the system by calling the deleteNotification function.

➢ **Component type**: React Page Component (typically placed in src/components/TeacherDashboard.jsx)

2. Notification Dropdown UI

➢ **Description and purpose**: A dropdown component that appears when clicking the notification bell icon, displaying recent notifications with options to mark as read or delete. Each notification item shows the title, message, timestamp, and includes a delete option for Transaction 4.1. The dropdown provides quick access to notification management without opening the full modal.

➢ **Component type**: Inline JSX Component within TeacherDashboard's render method

3. Notification Modal

➢ **Description and purpose**: A full-screen modal that displays all notifications with enhanced management capabilities for Transaction 4.1. It shows notifications in a scrollable list format, each with individual "Mark as read" and "Delete" buttons. The modal provides bulk actions like "Mark all as read" and gives teachers a comprehensive view of all their notifications with the ability to permanently delete any notification.

➢ **Component type**: Inline JSX Component within TeacherDashboard's render method

4. Notification State Management

➢ **Description and purpose**: A state management system that tracks notifications array, unread count, and handles real-time updates via WebSocket. For Transaction 4.1, it manages the removal of deleted notifications from the local state, ensuring the UI immediately reflects the deletion without requiring a page refresh. It also handles updating the unread count when deleted notifications were previously unread.

➢ **Component type**: State Management Logic within TeacherDashboard

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. **Teacher Entity**
   - ➢ **Description and Purpose:**

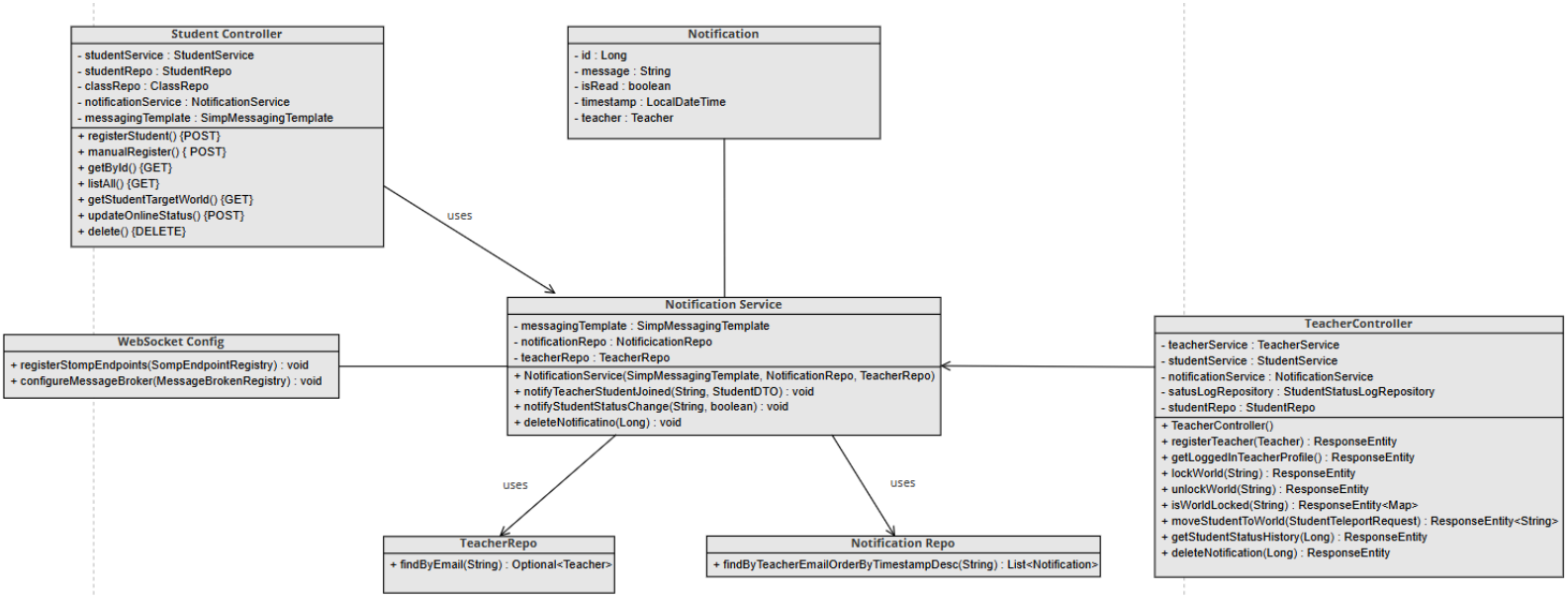
     Represents a teacher user in the system with authentication credentials, assigned role, and tracked locked worlds. It facilitates managing teacher-specific features like class assignment and system access.
   - ➢ **Component Type:**

     Java Class annotated with @Entity (typically stored in the entity or model package).

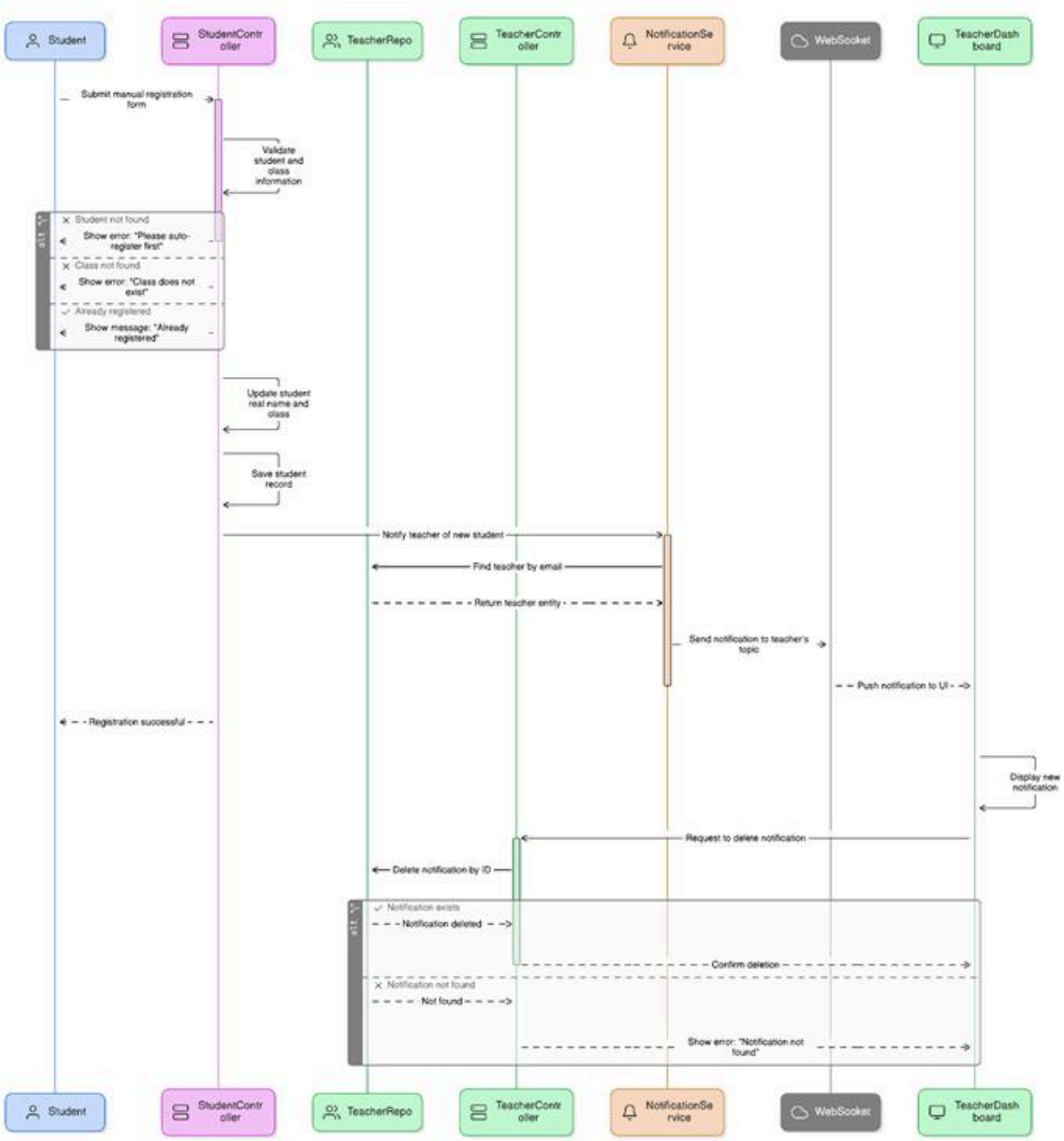
3. **Notification Entity**
   - ➢ **Description and Purpose:**
     Data Transfer Object (DTO) sent to the frontend over WebSocket or REST. Contains only necessary fields: message and timestamp.
   - ➢ **Component Type:** DTO Class – typically placed under src/main/java/.../dto/NotificationDTO.java.
4. **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.
5. **Teacher Controller**
   - ➢ **Description and Purpose:**
     Exposes REST API endpoints for teacher-related operations such as registering teachers, locking/unlocking worlds, teleporting students, retrieving student status logs, accessing teacher profile, and managing notifications. It coordinates with services like TeacherService, StudentService, and NotificationService.
   - ➢ **Component Type:** Controller Class – typically placed under src/main/java/.../controller/TeacherController.java.

Object-Oriented Components

- Class Diagram



- Sequence Diagram



Data Design

- ERD or schema