Project Documentation: Teaching Zone

1. Introduction

1.1 Project Overview

This document provides comprehensive documentation for the Teaching Zone (LMS) project. The system is a desktop application developed using JavaFX, designed to facilitate online learning by providing distinct interfaces and functionalities for Administrators, Teachers, and Students. It allows for user management, course creation, content delivery (videos), assignment handling, and basic system administration.

1.2 Purpose and Scope

The primary purpose of this LMS is to offer a platform for educational institutions or individual instructors to manage courses, users, and learning materials. The scope includes:

- User authentication (Login/Registration) with role-based access control.
- Admin capabilities: User management (add/delete/update), data management, system settings configuration, and report generation.
- Teacher capabilities: Course creation, adding course videos, assignment creation, and viewing submissions.
- Student capabilities: Browsing courses, enrolling in courses, viewing course content (videos), viewing assignments, and submitting assignments.

The system utilizes a PostgreSQL database for data persistence.

1.3 Target Audience

This documentation is intended for:

- **Developers:** To understand the system architecture, codebase, and database schema for maintenance or future development.
- **System Administrators:** To understand user management, system configuration, and database setup.
- **Project Managers:** To get an overview of the system's features and structure.

1.4 Technologies Used

- **Programming Language:** Java (JDK 11 or higher implied by `module-info.java`)
- **UI Framework:** JavaFX
- **Database:** PostgreSQL

- **Build Tool:** Maven (indicated by `pom.xml`)
- **Database Connectivity:** JDBC (PostgreSQL Driver)

2. System Architecture

2.1 Overview

The system follows a structure somewhat similar to the Model-View-Controller (MVC) pattern, adapted for JavaFX applications.

- **Model:** While there isn't a strict 'Model' layer with dedicated data objects passed around extensively (except for inner classes like `ManageUsersController.User`), the `Database` class acts as the primary data access layer, interacting directly with the PostgreSQL database. The `Info` class holds global session state. Helper classes like `UserManager` encapsulate business logic related to specific entities.
- **View:** The views are defined using FXML files (`login.fxml`, `AdminDashboard.fxml`, `ManageUsers.fxml`, etc.) located in the `src/main/resources/com/projecttest/projecttest` directory. These files define the layout and components of the user interface.
- **Controller:** Each FXML file is associated with a Controller class (e.g., `LoginController`, `AdminDashboardController`) located in `src/main/java/com/projecttest/projecttest`. These controllers handle user input, interact with the `Database` class or `UserManager` for data operations, update the view, and manage navigation between different screens.

2.2 Key Components

- **`Main.java`:** The application entry point, responsible for launching the JavaFX application and loading the initial login screen (`login.fxml`).
- **FXML Files:** Define the structure and layout of the user interfaces.
- **Controller Classes:** Contain the logic for handling UI events, interacting with the backend (Database/UserManager), and updating the UI.
- **`Database.java`:** A utility class providing static methods for all database operations (connecting, querying, inserting, updating, deleting data) via JDBC.
- **`UserManager.java` / `AbstractUserManager.java` / `UserOperations.java`:** Demonstrate object-oriented principles like abstraction, inheritance, and polymorphism for user management logic, separating it partially from the `ManageUsersController`.
- **`Info.java`:** A simple static class to hold the currently logged-in user's username and role globally.

2.3 Data Flow Examples

- **Login:** User enters credentials in `login.fxml` -> `LoginController` captures input -> `LoginController` calls `Database.loginUser()` -> `Database` queries the `users` table -> If successful, `Database` returns user data (including role) -> `LoginController` stores role/username in `Info` class -> `LoginController` loads the appropriate dashboard FXML (Admin, Teacher, or Student) based on the role.

- **User Management (Admin):** Admin navigates to Manage Users screen (`ManageUsers.fxml`) -> `ManageUsersController` loads existing users from `Database` -> Admin adds/deletes a user via the UI -> `ManageUsersController` calls `UserManager.addUser()` or `UserManager.removeUser()` -> `UserManager` interacts with `Database` to perform the operation -> `ManageUsersController` refreshes the user list in the UI.

2.4 UML Class Diagram

(See attached `class_diagram.png` or `uml_diagrams_en.pdf` for a visual representation of the classes and their relationships.)

3. User Roles and Permissions

The system defines three primary authenticated user roles, plus an unauthenticated state:

- **Admin:** Has the highest level of privileges. Admins can manage users (add, delete, update details via Data Management), configure system settings, manage general data, and view reports. They have access to the Admin Dashboard.
- **Teacher:** Can manage courses (create, add videos), manage assignments (create, view submissions), and perform actions within their assigned courses. They have access to the Teacher Dashboard.
- **Student:** Can view available courses, enroll in courses, view course content (videos), view assignments, and submit assignments. They have access to the Student Dashboard.
- **General User (Unauthenticated):** Can only access the Login and Registration screens.

Permissions are enforced primarily through navigation logic within the controllers (e.g., `LoginController` directs users to different dashboards based on their role retrieved from the `Database`).

(See attached `use_case_diagram_en.png` or `uml_diagrams_en.pdf` for a visual representation of user interactions.)

4. Core Features and Workflows

This section details the main functionalities and how users interact with them.

4.1 User Authentication

- **Registration:**
  - Accessed from the Login screen (`login.fxml` -> `RegisterController`).
  - User provides username, email, password, and selects a role (Admin, Teacher, Student).
  - `RegisterController` calls `Database.registerUser()`.
  - `Database` checks for existing username and inserts the new user if unique.
  - Status message is displayed on the registration screen.
- **Login:**
  - User provides username and password on `login.fxml`.

- `LoginController` calls `Database.loginUser()`.
- `Database` verifies credentials against the `users` table.
- If valid, user data (including role) is returned.
- `LoginController` stores username and role in `Info` class.
- `LoginController` navigates the user to the corresponding dashboard (`AdminDashboard.fxml`, `Mesarw3Teacher.fxml`, or `haythamSTUDENT.fxml`).
- (See attached `login_sequence_diagram_en.png` or `uml_diagrams_en.pdf` for a detailed sequence.)

4.2 Admin Dashboard (`AdminDashboardController`)

Provides access to administrative functions:

- **Manage Users (`ManageUsersController`):**
  - Displays a table of all users loaded via `Database`.
  - Allows adding new users (validating input, checking email format/uniqueness) via `UserManager.addUser()`.
  - Allows deleting selected users via `UserManager.removeUser()`.
  - Uses an inner class `ManageUsersController.User` to represent user data in the TableView.
- **Data Management (`DataManagementController`):**
  - Allows searching for users.
  - Displays user details upon selection from a list.
  - Allows updating user details (username, email, password, role) directly via `Database` update queries, including checks for username conflicts.
- **System Settings (`SystemSettingsController`):**
  - Interface for managing system-wide settings (implementation details based on `SystemSettings.fxml` and its controller).
  - Likely interacts with the `settings` table via `Database.saveSetting()`.
- **Reports (`ReportsController`):**
  - Interface for viewing system reports (implementation details based on `Reports.fxml` and its controller).
  - Likely involves querying various database tables.
  - Uses an inner class `ReportsController.ReportItem` for display.
- **Logout:** Navigates back to the Login screen.

4.3 Teacher Dashboard (`TeacherController`)

Provides functionalities for teachers:

- **Course Management:**
  - Adding new courses (`Database.addCourse()`), including title, description, instructor ID validation, and requirements.
  - Adding videos to courses (`Database.addCourseVideo()`), including metadata like title, description, URL/path, duration, and order.

- **Assignment Management:**
  - Adding assignments to courses (`Database.addAssignment()`), including title, description, due date, and max grade.
  - Viewing assignment submissions (requires corresponding database query methods, likely in `Database` class but not fully shown in provided snippets).
- **Logout:** Navigates back to the Login screen.

4.4 Student Dashboard (`StudentController`)

Provides functionalities for students:

- **Course Browsing/Enrollment:**
  - Viewing available courses (requires database query).
  - Enrolling in a course (`Database.enrollStudentInCourse()`), which also handles creating a basic student profile if one doesn't exist for the user ID.
- **Course Content:**
  - Viewing content of enrolled courses.
  - Watching course videos (`Database.getCourseVideos()`).
- **Assignments:**
  - Viewing assignments for enrolled courses (`Database.getAssignments()`).
  - Submitting assignments (`Database.submitAssignment()`), including submission text and timestamp.
- **Logout:** Navigates back to the Login screen.

5. Class Descriptions

This section provides brief descriptions of the main Java classes in the project.

- **`Main`:** The entry point for the JavaFX application. It extends `javafx.application.Application` and its `start()` method loads the initial `login.fxml` view.
- **`Database`:** A crucial utility class containing static methods to handle all interactions with the PostgreSQL database. It manages the database connection (using hardcoded credentials - **potential security risk**) and provides methods for user registration, login, adding/retrieving courses, videos, assignments, enrollments, settings, etc. It uses JDBC and `PreparedStatement` to interact with the database.
- **`Info`:** A simple class holding static variables (`currentUser`, `currentRole`) to maintain global session state about the logged-in user. This is a basic approach to session management.
- **Controller Classes:**
  - `LoginController`: Handles the logic for the login screen (`login.fxml`), including user input validation, calling `Database.loginUser()`, storing user info, and navigating to the appropriate dashboard or showing error messages.
  - `RegisterController`: Manages the registration screen (`register.fxml`), handling user input, calling `Database.registerUser()`, and providing feedback.
  - `AdminDashboardController`: Controls the main admin dashboard (`AdminDashboard.fxml`), primarily handling navigation to other admin-specific sections (Manage Users, Data Management,

Settings, Reports).
    - `ManageUsersController`: Controls the user management screen (`ManageUsers.fxml`) for admins. It displays users in a `TableView`, handles adding new users (using `UserManager`), deleting selected users (using `UserManager`), and loading user data from the `Database`.
    - `DataManagementController`: Controls the data management screen (`DataManagement.fxml`) for admins. Allows searching, viewing, and updating user details directly via the `Database` class.
    - `SystemSettingsController`: Controls the system settings screen (`SystemSettings.fxml`). Interacts with `Database.saveSetting()`.
    - `ReportsController`: Controls the reports screen (`Reports.fxml`). Likely queries the database to generate and display reports.
    - `TeacherController`: Controls the teacher dashboard (`Mesarw3Teacher.fxml`). Handles teacher-specific actions like course/assignment management (details depend on the FXML and associated methods).
    - `StudentController`: Controls the student dashboard (`haythamSTUDENT.fxml`). Handles student-specific actions like viewing/enrolling in courses and viewing/submitting assignments.
- **`UserOperations` (Interface):** Defines a contract for user operations (add, remove, get details), promoting polymorphism.
- **`AbstractUserManager` (Abstract Class):** Provides a base implementation for user management, including a reference to the `Database` object and a common `userExists` method. It defines an abstract `formatUserDetails` method.
- **`UserManager`:** Concrete implementation of `UserOperations` and extends `AbstractUserManager`. It handles the business logic for adding and removing users, delegating database interactions to the `Database` class. It demonstrates inheritance, abstraction, and polymorphism.
- **Inner Classes:**
    - `ManageUsersController.User`: A simple static inner class used as a data structure to hold user information for display in the `TableView` within `ManageUsersController`.
    - `ReportsController.ReportItem`: Likely a similar data structure for displaying items in the reports view.

6. Database Schema

While a full schema diagram is not available, the structure can be inferred from the queries within the `Database.java` class. Key tables likely include:

- **`users`:** Stores user login information.
    - Columns: `id` (Primary Key), `username`, `email`, `password`, `role`.
    - **Note:** Passwords are stored in plain text (`Database.registerUser` and `Database.loginUser` handle passwords directly without hashing), which is a significant security vulnerability and should be addressed by implementing password hashing (e.g., using bcrypt).
- **`students`:** Stores additional details for users with the 'Student' role.
    - Columns: `id` (Primary Key), `user_id` (Foreign Key to `users.id`), `student_id`, `organisation`, `level`, `gpa`, `full_name`.

- **`courses`:** Stores information about courses.
    - Columns: `id` (Primary Key), `title`, `description`, `instructor_id` (Foreign Key to `users.id` where role='Teacher'), `requirements`.
- **`course_teachers`:** Maps teachers to courses (supporting multiple teachers per course, although current logic seems to imply one primary instructor).
    - Columns: `course_id` (Foreign Key), `teacher_id` (Foreign Key).
- **`course_videos`:** Stores details about videos associated with courses.
    - Columns: `id` (Primary Key), `course_id` (Foreign Key), `title`, `description`, `video_url`, `video_file_path`, `duration`, `order_number`.
- **`course_enrollments`:** Maps students to the courses they are enrolled in.
    - Columns: `id` (Primary Key), `student_id` (Foreign Key to `students.id`), `course_id` (Foreign Key to `courses.id`).
- **`assignments`:** Stores information about assignments for courses.
    - Columns: `id` (Primary Key), `course_id` (Foreign Key), `title`, `description`, `due_date`, `max_grade`.
- **`assignment_submissions`:** Stores student submissions for assignments.
    - Columns: `id` (Primary Key), `assignment_id` (Foreign Key), `student_id` (Foreign Key to `students.id`), `submission_date`, `submission_text`, `submission_file` (present in old method).
- **`settings`:** Stores system configuration settings.
    - Columns: `setting_name` (Primary Key), `setting_value`, `updated_at`.

Relationships are primarily managed through foreign keys linking tables like `users` to `students`, `courses` to `users` (instructors), `courses` to `course_videos`, `students` to `course_enrollments`, etc.

7. Setup and Installation

To set up and run this project, the following steps are generally required:

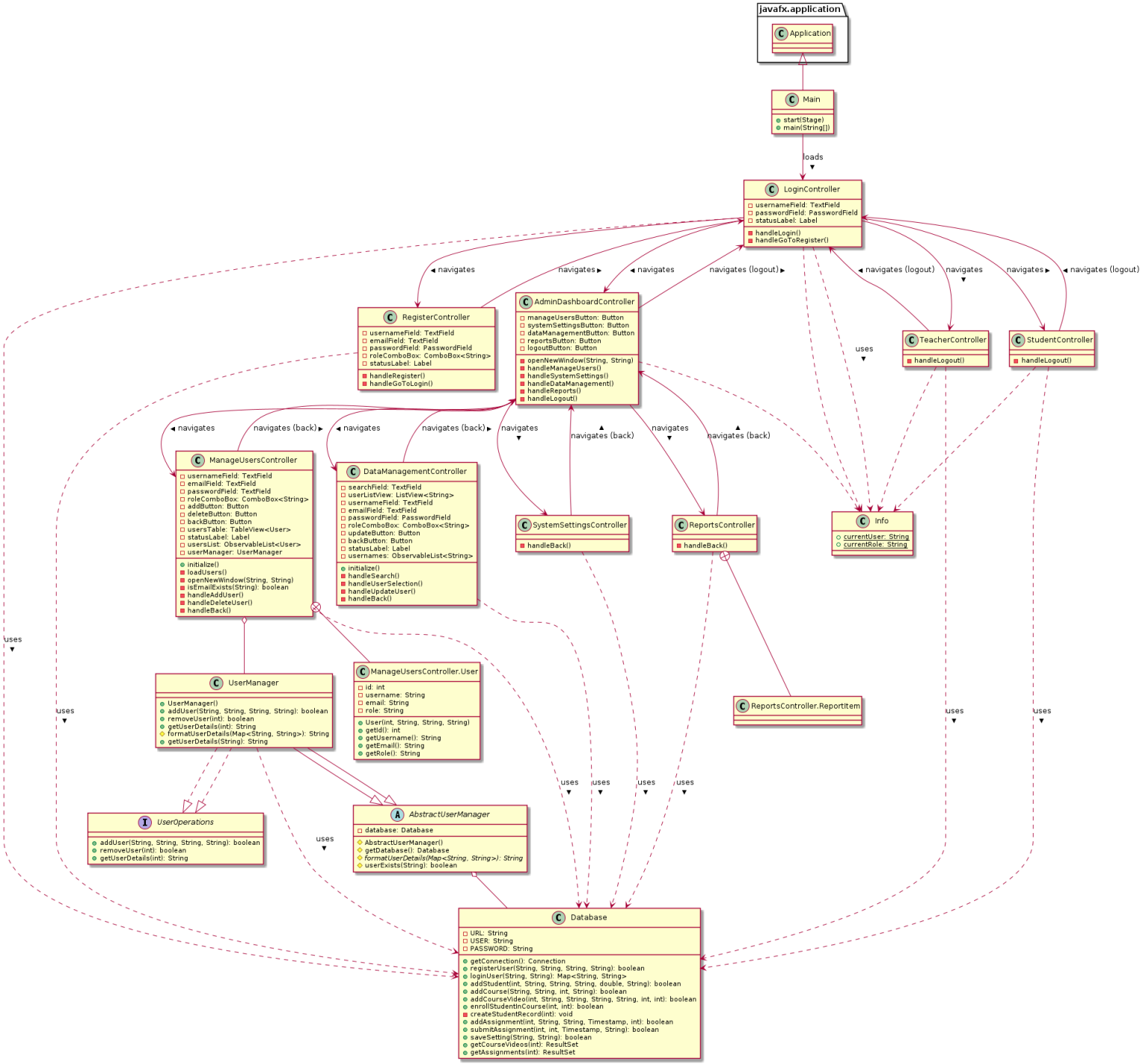
1. **Prerequisites:**
    - Java Development Kit (JDK) compatible with the project (likely JDK 11+).
    - Apache Maven build tool.
    - PostgreSQL database server.
2. **Database Setup:**
    - Create a PostgreSQL database (named `oop2` according to `Database.java`).
    - Create a PostgreSQL user (named `postgres` with password `ym` according to `Database.java` - **Note:** Using default superuser with a simple password is not recommended for production).
    - Execute the necessary SQL statements to create the tables and relationships outlined in the Database Schema section. The project includes an `export.sql` file and `oop2.backup` which might contain the schema and/or initial data; these should be used to set up the database structure.
3. **Build:**
    - Navigate to the project's root directory (`ProjectTest/ProjectTest`) in a terminal.
    - Run `mvn clean install` to compile the code and download dependencies.
4. **Run:**
    - The application can likely be run using Maven: `mvn javafx:run`.

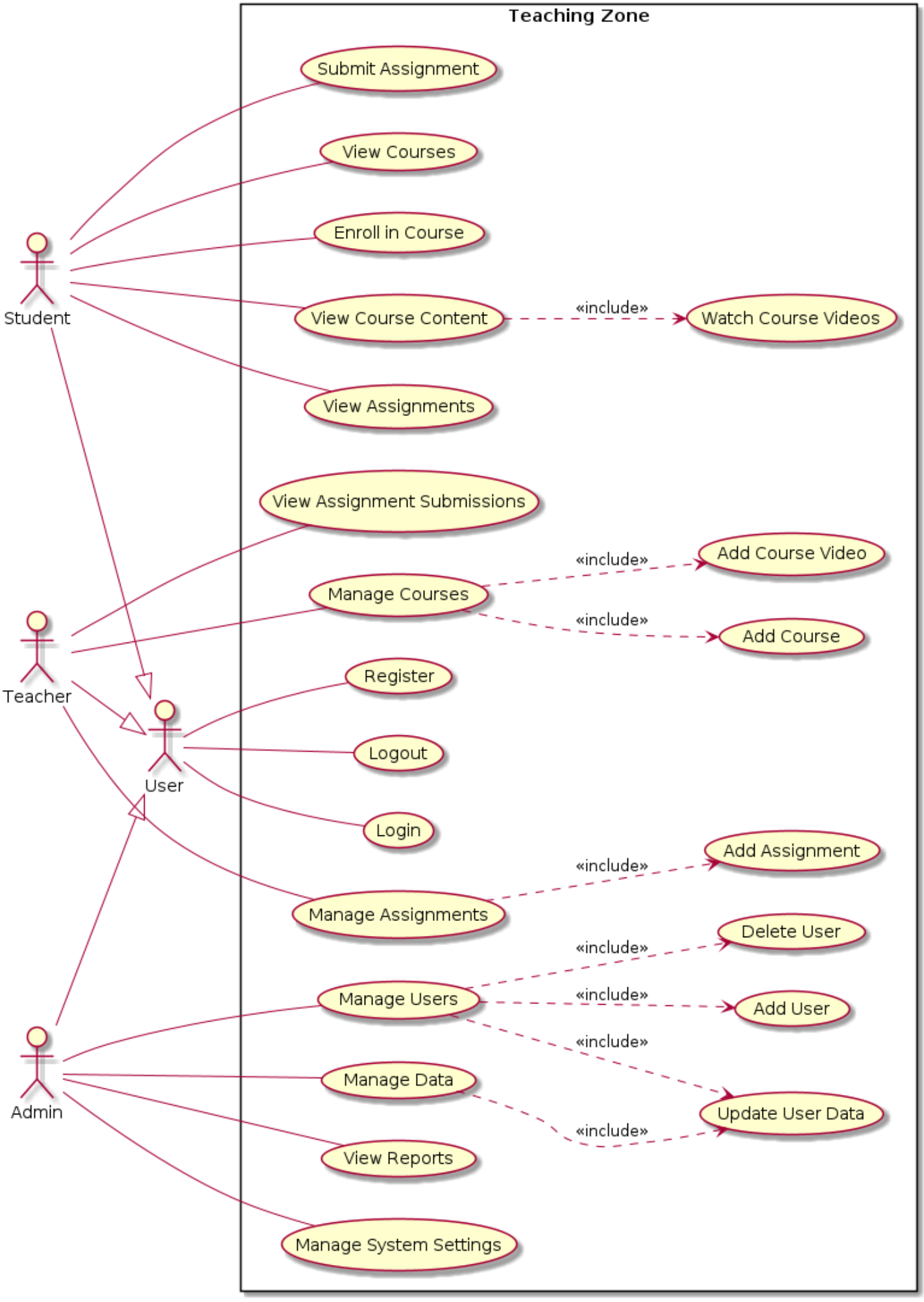
- Alternatively, it can be run from an IDE (like IntelliJ IDEA or Eclipse) by executing the `main` method in the `com.projecttest.projecttest.Main` class.

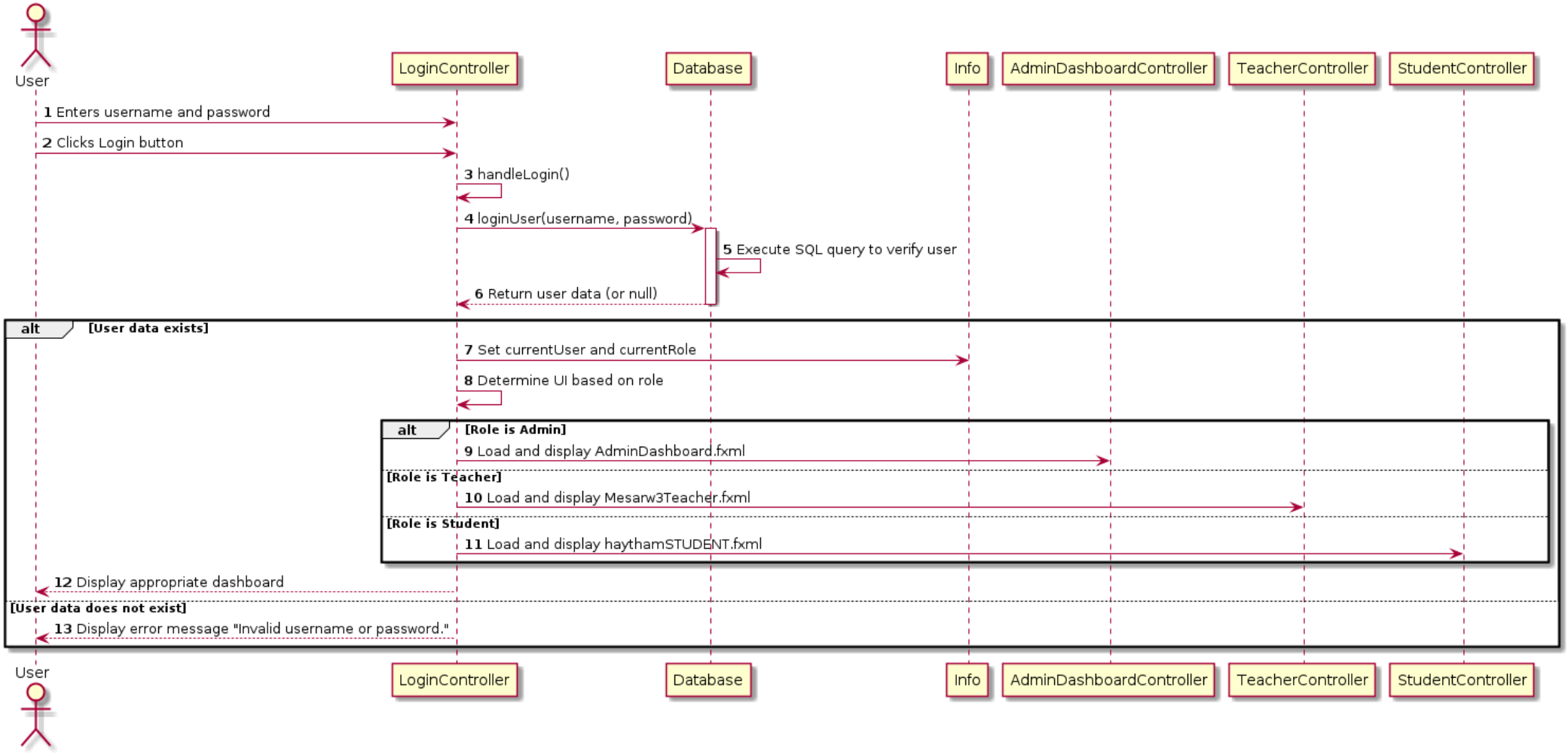
8. Conclusion

8.2 Potential Improvements / Future Work

- **Security:** Implement password hashing immediately. Avoid hardcoding database credentials; use configuration files or environment variables.
- **Error Handling:** Improve error handling and user feedback beyond simple status labels.
- **Database Design:** Review database schema for normalization and efficiency. Consider using an ORM (Object-Relational Mapper) like Hibernate or JPA for data persistence instead of raw JDBC.
- **Code Structure:** Refactor `Database.java` to be non-static or use a proper Data Access Object (DAO) pattern. Improve separation of concerns between controllers and business logic.
- **UI/UX:** Enhance the user interface design and user experience.
- **Features:** Add more advanced LMS features like quizzes, discussion forums, grading systems, file uploads for submissions, etc.
- **Testing:** Implement unit and integration tests.
- **Concurrency:** Ensure thread safety if any background tasks are added.


javafx.application
Application
Main
start(Stage)
main(String[])
loads
LoginController
usernameField: TextField
passwordField: PasswordField
statusLabel: Label
handleLogin()
handleGoToRegister()
navigates
navigates (logout)
uses
RegisterController
usernameField: TextField
emailField: TextField
passwordField: PasswordField
roleComboBox: ComboBox<String>
statusLabel: Label
handleRegister()
handleGoToLogin()
AdminDashboardController
manageUsersButton: Button
systemSettingsButton: Button
dataManagementButton: Button
reportsButton: Button
logoutButton: Button
openNewWindow(String, String)
handleManageUsers()
handleSystemSettings()
handleDataManagement()
handleReports()
handleLogout()
TeacherController
handleLogout()
StudentController
handleLogout()
navigates (back)
ManageUsersController
usernameField: TextField
emailField: TextField
passwordField: TextField
roleComboBox: ComboBox<String>
addButton: Button
deleteButton: Button
backButton: Button
usersTable: TableView<User>
statusLabel: Label
usersList: ObservableList<User>
userManager: UserManager
initialize()
loadUsers()
openNewWindow(String, String)
isEmailExists(String): boolean
handleAddUser()
handleDeleteUser()
handleBack()
DataManagementController
searchField: TextField
userListView: ListView<String>
usernameField: TextField
emailField: TextField
passwordField: PasswordField
roleComboBox: ComboBox<String>
updateButton: Button
backButton: Button
statusLabel: Label
usernames: ObservableList<String>
initialize()
handleSearch()
handleUserSelection()
handleUpdateUser()
handleBack()
SystemSettingsController
handleBack()
ReportsController
handleBack()
Info
currentUser: String
currentRole: String
UserManager
UserManager()
addUser(String, String, String, String): boolean
removeUser(int): boolean
getUserDetails(int): String
formatUserDetails(Map<String, String>): String
getUserDetails(String): String
ManageUsersController.User
id: int
username: String
email: String
role: String
User(int, String, String, String)
getId(): int
getUsername(): String
getEmail(): String
getRole(): String
ReportsController.ReportItem
UserOperations
addUser(String, String, String, String): boolean
removeUser(int): boolean
getUserDetails(int): String
AbstractUserManager
database: Database
AbstractUserManager()
getDatabase(): Database
formatUserDetails(Map<String, String>): String
userExists(String): boolean
Database
URL: String
USER: String
PASSWORD: String
getConnection(): Connection
registerUser(String, String, String, String): boolean
loginUser(String, String): Map<String, String>
addStudent(int, String, String, String, double, String): boolean
addCourse(String, String, int, String): boolean
addCourseVideo(int, String, String, String, int, int): boolean
enrollStudentInCourse(int, int): boolean
createStudentRecord(int): void
addAssignment(int, String, String, Timestamp, int): boolean
submitAssignment(int, int, Timestamp, String): boolean
saveSetting(String, String): boolean
getCourseVideos(int): ResultSet
getAssignments(int): ResultSet


Teaching Zone
Submit Assignment
View Courses
Enroll in Course
View Course Content
«include»
Watch Course Videos
View Assignments
View Assignment Submissions
Manage Courses
«include»
Add Course Video
«include»
Add Course
Register
Logout
Login
Manage Assignments
«include»
Add Assignment
Manage Users
«include»
Delete User
«include»
Add User
«include»
Update User Data
Manage Data
«include»
View Reports
Manage System Settings
Student
Teacher
User
Admin


User
LoginController
Database
Info
AdminDashboardController
TeacherController
StudentController
1 Enters username and password
2 Clicks Login button
3 handleLogin()
4 loginUser(username, password)
5 Execute SQL query to verify user
6 Return user data (or null)
alt
[User data exists]
7 Set currentUser and currentRole
8 Determine UI based on role
alt
[Role is Admin]
9 Load and display AdminDashboard.fxml
[Role is Teacher]
10 Load and display Mesarw3Teacher.fxml
[Role is Student]
11 Load and display haythamSTUDENT.fxml
12 Display appropriate dashboard
[User data does not exist]
13 Display error message "Invalid username or password."
User
LoginController
Database
Info
AdminDashboardController
TeacherController
StudentController