THE CRUD SYSTEM

**Application Structure:**

* **AwesomeProjectProposal**:
  + **src**:
    - **main**:
      * **java**:
        + **com**:

**csis3275**:

**controller\_admin**:

Controller\_Admin.java

**controller\_member**:

HomeController\_Member.java

**dao**:

UserDAO.java

**model**:

User.java

**repository**:

UserRepository.java

**service**:

UserService.java

AwesomeProjectProposalApplication.java

* + - * **resources**:
        + **static**:

\_ajax

assets

demo.files

* + - * + **templates**:

**admin**:

index.html

**member**:

index.html

login.html

**A screenshot of a computer

Description automatically generated**

the user logs in:

1. **User Login:**
   * User accesses the login page (**login.html**).
   * User enters their **username** and **password**.
2. **Authentication:**
   * system checks the entered credentials against the stored user data in the **UserRepository**.
   * If credentials are valid, the system determines the user's role (e.g., Admin or Member).
3. **User Role Determination:**
   * If the user is an Admin, they are redirected to the Admin dashboard (**AdminTemplate**).
   * If the user is a Member, they are redirected to the Member homepage (**HomeController\_Member**).
4. **Admin Dashboard:**
   * Admin can view a list of all users.
   * Admin has the ability to create, update, or delete users using the **Controller\_Admin** methods.
   * The **Controller\_Admin** interacts with the **UserService** to perform CRUD operations.
   * The **UserService** communicates with the **UserRepository** to fetch or modify user data in the database.
5. **Member Homepage:**
   * Member can view their profile details.
   * Member has the option to update their profile using the **HomeController\_Member** method.
   * The **HomeController\_Member** interacts with the **UserService** to update user details.
   * The **UserService** communicates with the **UserRepository** to update user data in the database.
6. **Data Retrieval and Storage:**
   * The **UserRepository** is responsible for direct interactions with the database.
   * It retrieves user data based on queries (e.g., **findByUsername**) and can store updated user information.
7. **Frontend Rendering:**
   * Based on user actions and backend responses, the appropriate HTML templates (**AdminTemplate** or **MemberTemplate**) are rendered to display the required information or interfaces.

**Database Schema:**

* **User**:
  + id (Primary Key, Auto Increment)
  + username (Not Blank)
  + password (Not Blank)
  + role (e.g., "admin", "member")

**CRUD Operations:**

* **Controller\_Admin**:
  + **Create**: **createUser(User user)**
  + **Read**: **renderHomeAdmin(Model model)**
  + **Update**: **updateUser(User user)**
  + **Delete**: **deleteUser(@RequestParam int id)**
* **HomeController\_Member**:
  + **Read**: **renderHomeMember(Model model)**
  + **Update**: **updateProfile(User user)**
* **UserRepository**:
  + **Read by Username**: **findByUsername(String username)**
* **UserService**:
  + **Create**: **createUser(User user)**
  + **Read All**: **getAllUsers()**
  + **Read by ID**: **getUserById(int id)**
  + **Read by Username**: **getUserByUsername(String username)**
  + **Update**: **updateUser(User user)**
  + **Delete**: **deleteUser(int id)**