Project Overview

This project is a JavaFX GUI application designed to manage user accounts with multiple roles and permissions. It implements functionalities such as user authentication, role management, and account setup within a secure and user-friendly interface. The application supports roles like **Admin**, **Student**, and **Instructor**, each with specific access and capabilities.

Key Features

- **User Authentication**: Secure login system with password hashing and validation.
- **Role Management**: Users can have multiple roles; admins can assign or revoke roles.
- **Account Setup**: New users complete their profiles with personal information.
- **Invitation System**: Admins can invite new users via unique invitation codes.
- **Password Reset**: One-time passwords with expiration for secure account recovery.
- **User Interface**: Intuitive GUI using JavaFX for a smooth user experience.

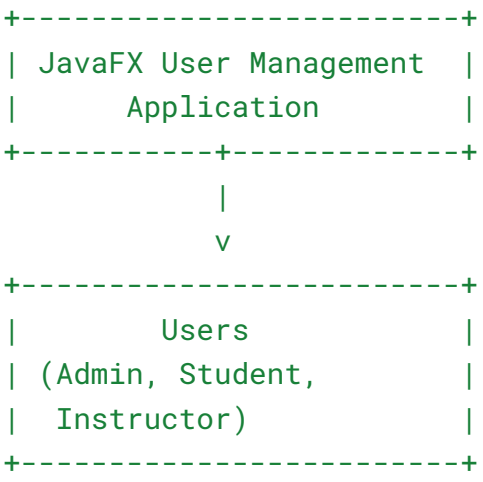
Requirements (User Stories)

1. **First User Setup**:
    - As a new application, when no users exist, the first user to log in becomes an **Admin**.
    - The first user must specify a username and password to create the Admin account.
2. **User Authentication**:
    - As a user, I want to log in with my username and password to access my account.
    - If I have multiple roles, I should be able to select which role to use during the session.
3. **Account Completion**:
    - As a new user, I must complete my account setup by providing my email and name before accessing the system.
4. **Admin Functionalities**:
    - As an Admin, I can invite new users by generating a one-time invitation code and assigning roles.
    - I can reset user accounts, forcing them to set a new password upon next login.
    - I can delete user accounts with a confirmation prompt.
    - I can view a list of all user accounts with their details.
    - I can add or remove roles from existing users.
5. **User Account Creation**:

    - As a user with an invitation code, I can create an account by specifying a username and password.
6. **Password Management**:
    - As a user, I must enter my password twice when creating or resetting it to ensure accuracy.
    - If my password is a one-time password, I must create a new password upon next login.
7. **Logout Functionality**:
    - As a user, I can log out of the application from any role-specific home page.

Architecture

Context Diagram



Major Architectural Components

1. **Model Layer**:
    - Represents the data structures (`User`, `Role`, `InvitationCode`, `Name`).
    - Handles business logic and data manipulation.
2. **View Layer**:
    - Contains FXML files defining the UI layout.
    - Presents information to the user and captures user input.
3. **Controller Layer**:
    - Processes user input from the View.
    - Interacts with the Model to update the application state.
4. **Utility Classes**:

    - Provide common functionalities like password hashing (`PasswordUtil`), code generation (`CodeGenerator`), and data persistence (`DataManager`).
5. **Data Persistence**:
    - Manages saving and loading user data and invitation codes.
    - Uses serialization to persist data in files.

Design

Use Case Diagram

```
[User]
   |
   +---------------------------------+
   |                                 |
[First Login]                  [Use Invitation Code]
   |                                 |
[Create Admin Account]         [Create User Account]
   |                                 |
[Finish Account Setup]         [Finish Account Setup]
   |                                 |
[Select Role (if multiple)]          |
   |                                 |
[Access Role-Specific Home Page]     |
```

Class Diagram

User Class Diagram

sql
Copy code

```
+-----------------+
|      User       |
+-----------------+
| - username      |
| - passwordHash  |
| - salt          |
| - email         |
| - isOneTimePwd  |
```

```
| - otpExpiry     |
| - name: Name    |
| - roles: Set<Role> |
+-----------------+
| + getters/setters |
| + addRole(Role)    |
| + removeRole(Role) |
+-----------------+
```

Related Classes

Name Class:

diff

Copy code

```
+-----------------+
|      Name       |
+-----------------+
| - firstName     |
| - middleName    |
| - lastName      |
| - preferredName |
+-----------------+
| + getDisplayName() |
+-----------------+
```

Role Enum:

diff

Copy code

```
+-----------------+
|     Role        |
+-----------------+
| ADMIN           |
| STUDENT         |
| INSTRUCTOR      |
+-----------------+
```

InvitationCode Class:
sql
Copy code

```
+----------------------+
|   InvitationCode     |
+----------------------+
| - code               |
| - roles: Set<Role>   |
| - isUsed             |
+----------------------+
| + getters/setters    |
+----------------------+
```

-

Class Responsibility Collaborator (CRC) Cards

User Class

- **Responsibilities**:
  - Store user information (username, password hash, email, roles).
  - Authenticate users using hashed passwords.
  - Manage user roles (add or remove roles).
  - Handle one-time password logic and expiration.
- **Collaborators**:
  - `Name`: Represents the user's name details.
  - `Role`: Enum representing user roles.
  - `PasswordUtil`: Utility for password hashing and salt generation.
  - `DataManager`: Handles data persistence.

AdminHomeController

- **Responsibilities**:
  - Handle admin-specific actions (inviting users, resetting accounts, deleting accounts).
  - Navigate to different admin functionalities.
- **Collaborators**:
  - `User`: Perform actions on user accounts.
  - `InvitationCode`: Generate and manage invitation codes.
  - `DataManager`: Save and load user data.