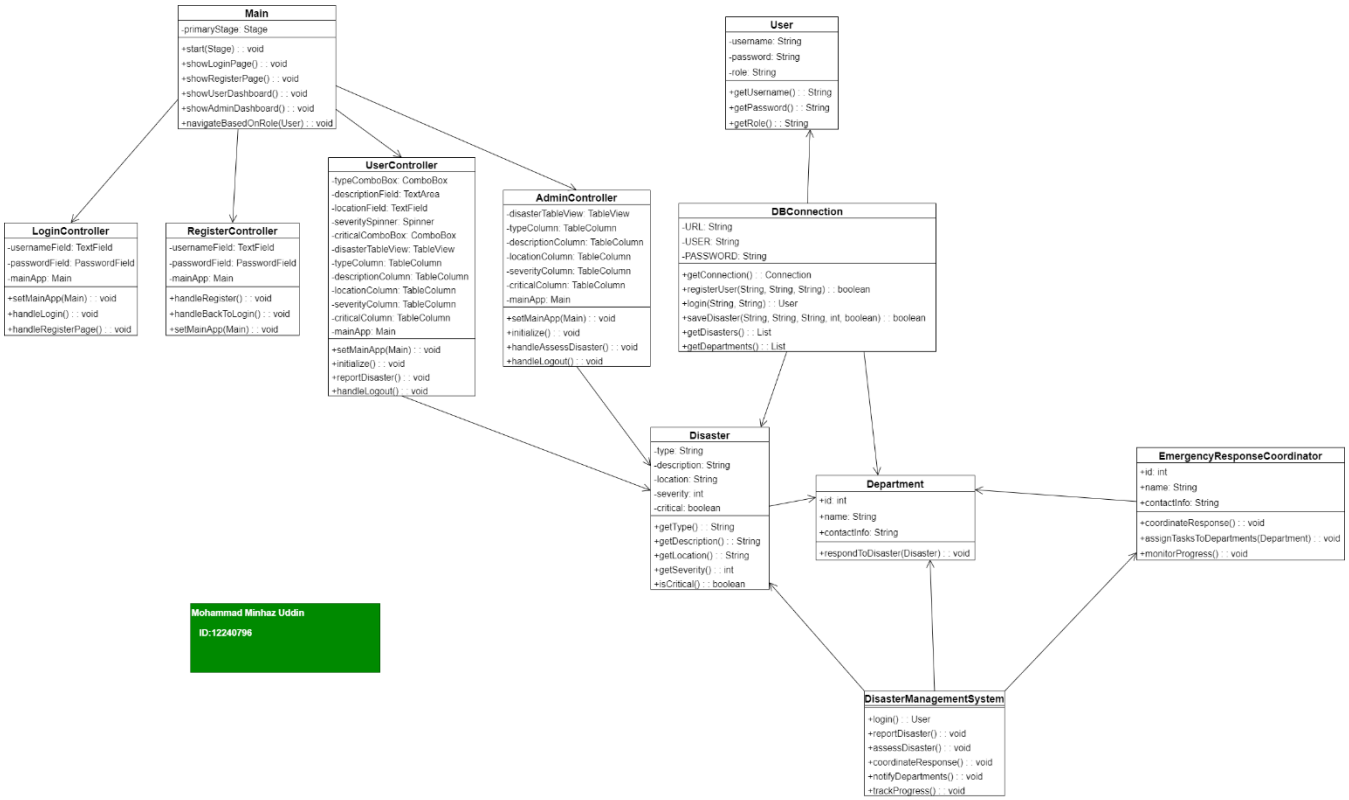
Disaster Response System (DRS) Prototype Report

Introduction

The Disaster Response System (DRS) is designed to manage disaster events such as fires, floods, earthquakes, and hurricanes. Users may report catastrophes, rate their severity, and alert the appropriate agencies for a coordinated response using the system. Effective communication between various agencies, including police enforcement, fire departments, hospitals, and other emergency services, is guaranteed by the DRS. The information flow inside the system is managed by administrators and users using an intuitive interface.

UML Class Diagrams



The UML diagrams provided represent the structure of the **Disaster Response System (DRS)**. The primary application logic, user interface controllers, and business logic pertaining to users, disasters, and the several departments involved in disaster response are all included in the system. A thorough explanation of the class diagrams, including relationships, data types, methods, and properties, is provided below.

Main Class Diagram

- **Main**: Handles the primary user interface logic and application navigation.
  - `+start(Stage): void`: Starts the application.
  - `+showLoginPage(): void`: Navigates to the login page.
  - `+showRegisterPage(): void`: Navigates to the registration page.
  - `+showUserDashboard(): void`: Displays the user dashboard.
  - `+showAdminDashboard(): void`: Displays the admin dashboard.
  - `+navigateBasedOnRole(User): void`: Directs users based on their role (Admin/User).
  - `-primaryStage: Stage`: The primary stage for the application interface.

Controllers

- **LoginController**: Manages the login functionality.
  - `+setMainApp(Main): void`: Sets the main application reference.
  - `+handleLogin(): void`: Validates login credentials.
  - `+handleRegisterPage(): void`: Navigates to the registration page.
  - `-usernameField: TextField`: Holds the entered username.
  - `-passwordField: PasswordField`: Holds the entered password.
  - `-mainApp: Main`: Reference to the main application.
- **RegisterController**: Handles user registration.
  - `+handleRegister(): void`: Registers a new user.
  - `+handleBackToLogin(): void`: Navigates back to the login page.
  - `+setMainApp(Main): void`: Sets the main application reference.
  - `-usernameField: TextField`: Holds the username.
  - `-passwordField: PasswordField`: Holds the password.
  - `-mainApp: Main`: Reference to the main application.
- **UserController**: Manages disaster reporting for users.
  - `+setMainApp(Main): void`: Sets the main application reference.
  - `+initialize(): void`: Initializes the disaster report form.
  - `+reportDisaster(): void`: Submits the disaster report.
  - `+handleLogout(): void`: Logs the user out.
  - UI Components:
    - `-typeComboBox: ComboBox<String>`
    - `-descriptionField: TextArea`
    - `-locationField: TextField`
    - `-severitySpinner: Spinner<Integer>`
    - `-criticalComboBox: ComboBox<String>`
    - `-disasterTableView: TableView<Disaster>`
- **AdminController**: Allows the admin to assess reported disasters.
  - `+setMainApp(Main): void`: Sets the main application reference.
  - `+initialize(): void`: Initializes the disaster assessment form.
  - `+handleAssessDisaster(): void`: Assesses selected disasters.
  - `+handleLogout(): void`: Logs the admin out.
  - UI Components:
    - `-disasterTableView: TableView<Disaster>`

Business Logic

- **Disaster**: Represents a disaster report.
  - `+getType(): String`
  - `+getDescription(): String`
  - `+getLocation(): String`
  - `+getSeverity(): int`
  - `+isCritical(): boolean`
  - Attributes:
    - `-type: String`
    - `-description: String`
    - `-location: String`
    - `-severity: int`
    - `-critical: boolean`
- **User**: Represents a user in the system (Admin or regular User).
  - `+getUsername(): String`
  - `+getPassword(): String`
  - `+getRole(): String`
  - Attributes:
    - `-username: String`
    - `-password: String`
    - `-role: String`

Database Connection and Department Classes

- **DBConnection**: Manages database interactions.
  - `+getConnection(): Connection`
  - `+registerUser(String, String, String): boolean`
  - `+login(String, String): User`
  - `+saveDisaster(String, String, String, int, boolean): boolean`
  - `+getDisasters(): List<Disaster>`
  - `+getDepartments(): List<Department>`
  - Attributes:
    - `-URL: String`
    - `-USER: String`
    - `-PASSWORD: String`
- **DisasterManagementSystem**: Manages disaster-related workflows.
  - `+login(): User`
  - `+reportDisaster(): void`
  - `+assessDisaster(): void`
  - `+coordinateResponse(): void`
  - `+notifyDepartments(): void`
  - `+trackProgress(): void`
- **Department**: Abstract class for different departments (Fire, Hospital, etc.).
  - `+id: int`
  - `+name: String`
  - `+contactInfo: String`
  - `+respondToDisaster(Disaster): void`
- **EmergencyResponseCoordinator**: Manages disaster response coordination.
  - `+id: int`
  - `+name: String`
  - `+contactInfo: String`
  - `+coordinateResponse(): void`
  - `+assignTasksToDepartments(Department): void`
  - `+monitorProgress(): void`

Functional Requirements

The Disaster Response System implements several key functions:

1. **Disaster Reporting**: Users can report different kinds of disasters, including fires, floods, earthquakes, and hurricanes.
2. **Quick Disaster Assessment**: The system accurately assesses reported disasters based on severity and provides a prioritized response.
3. **Department Coordination**: Various departments (Fire, Hospital, Law Enforcement) are notified and coordinate responses effectively.

Implementation of Two Creative Features

Login and Logout Features:

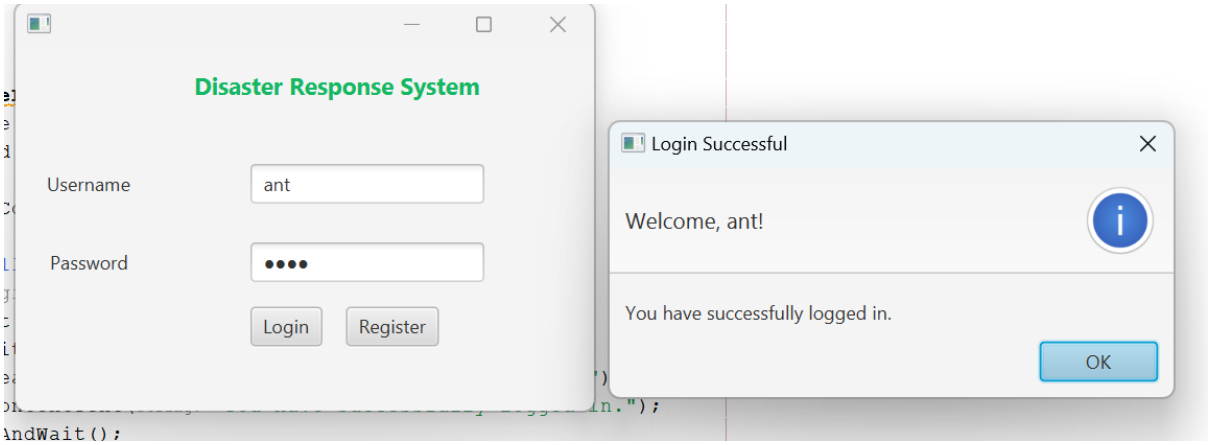
1. **Login**: Users are authenticated using a username and password, enabling secure access to disaster reporting and administrative features.(For Admin Username=admin Password = 123)
2. **Logout**: Once users log out, the system terminates the session securely, preventing unauthorized access. A confirmation message indicates the success of the logout.

Test Plan and Test Data

Test Case 1: Login Functionality

- **Input:**
  Username = "ant", Password = "1234"
- **Expected Output:**
  User is successfully logged in.
- **Actual Output:**
  User login is successful, and user session is active.
- **Pass/Fail:**
  Pass

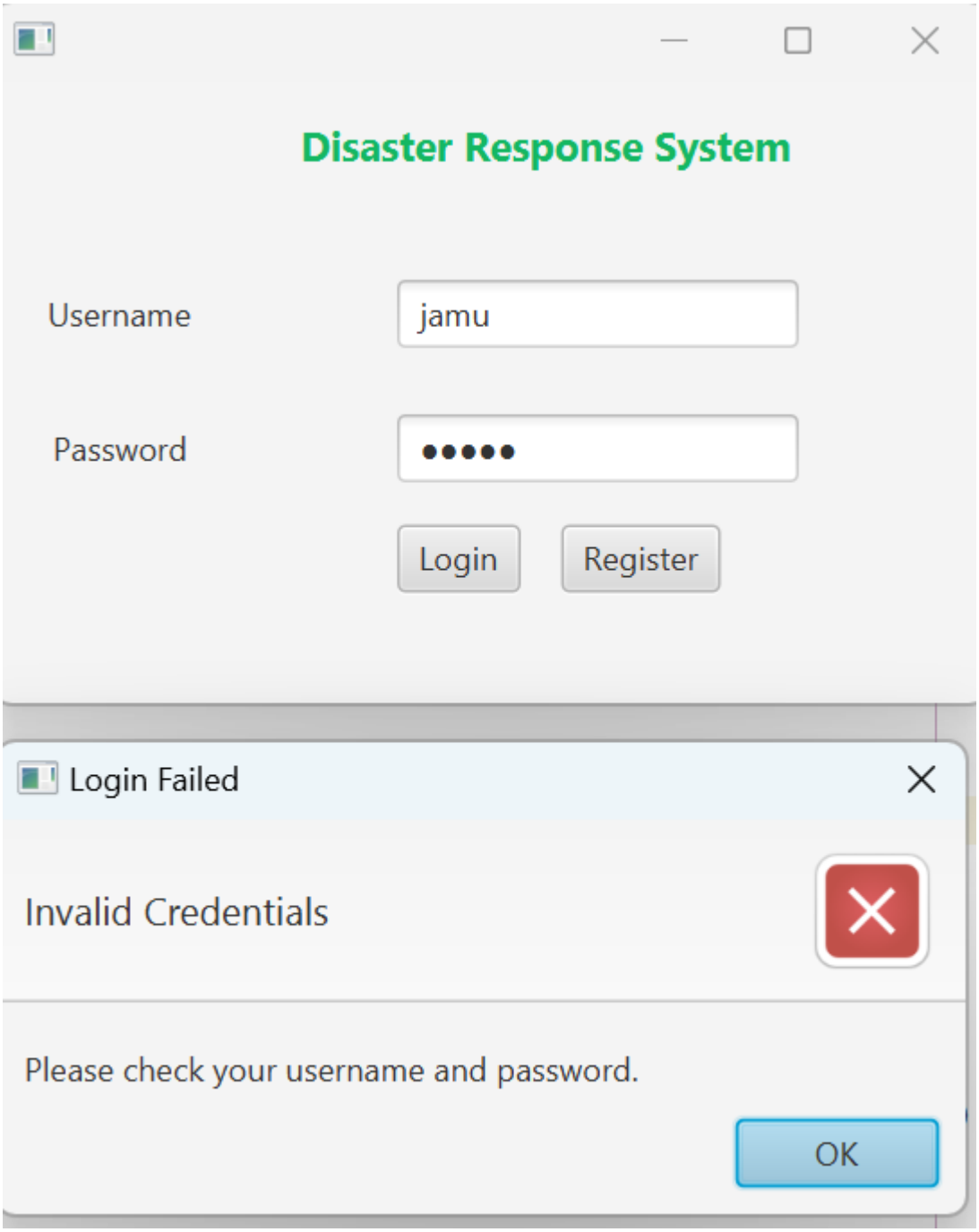
Screenshot of NetBeans:



Test Case 2: Invalid Login Functionality

- **Input:**
  Username = "invalidUser", Password = "wrongpassword"
- **Expected Output:**
  Error message: "Invalid login!"
- **Actual Output:**
  Error message displayed: "Invalid login!"
- **Pass/Fail:**
  Pass

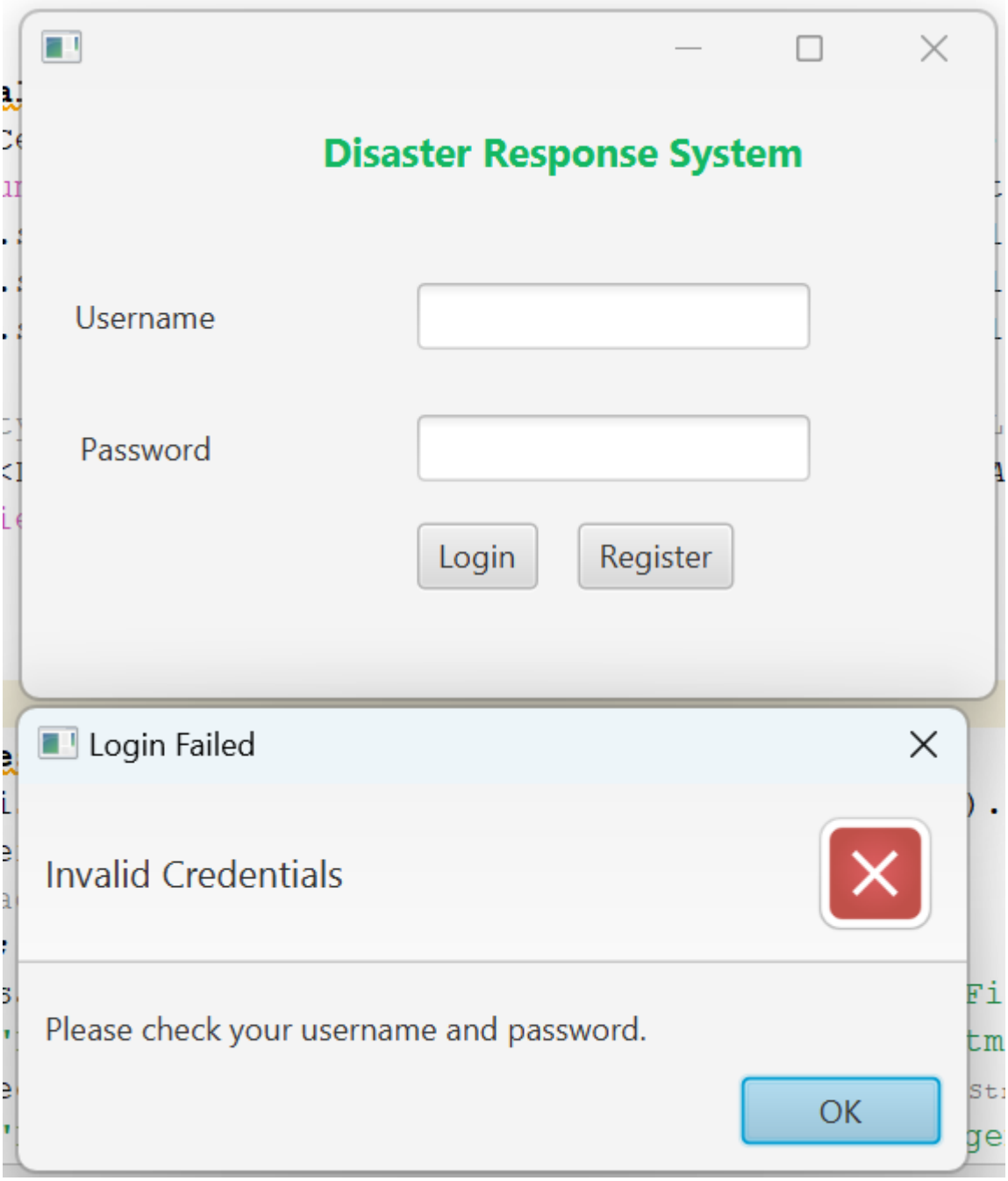
Screenshot of NetBeans:



Test Case 3: Empty Login Fields

- **Input:**
  Username = "", Password = ""
- **Expected Output:**
  Error message: "Please fill in all fields"
- **Actual Output:**
  Error message displayed: "Please fill in all fields"
- **Pass/Fail:**
  Pass

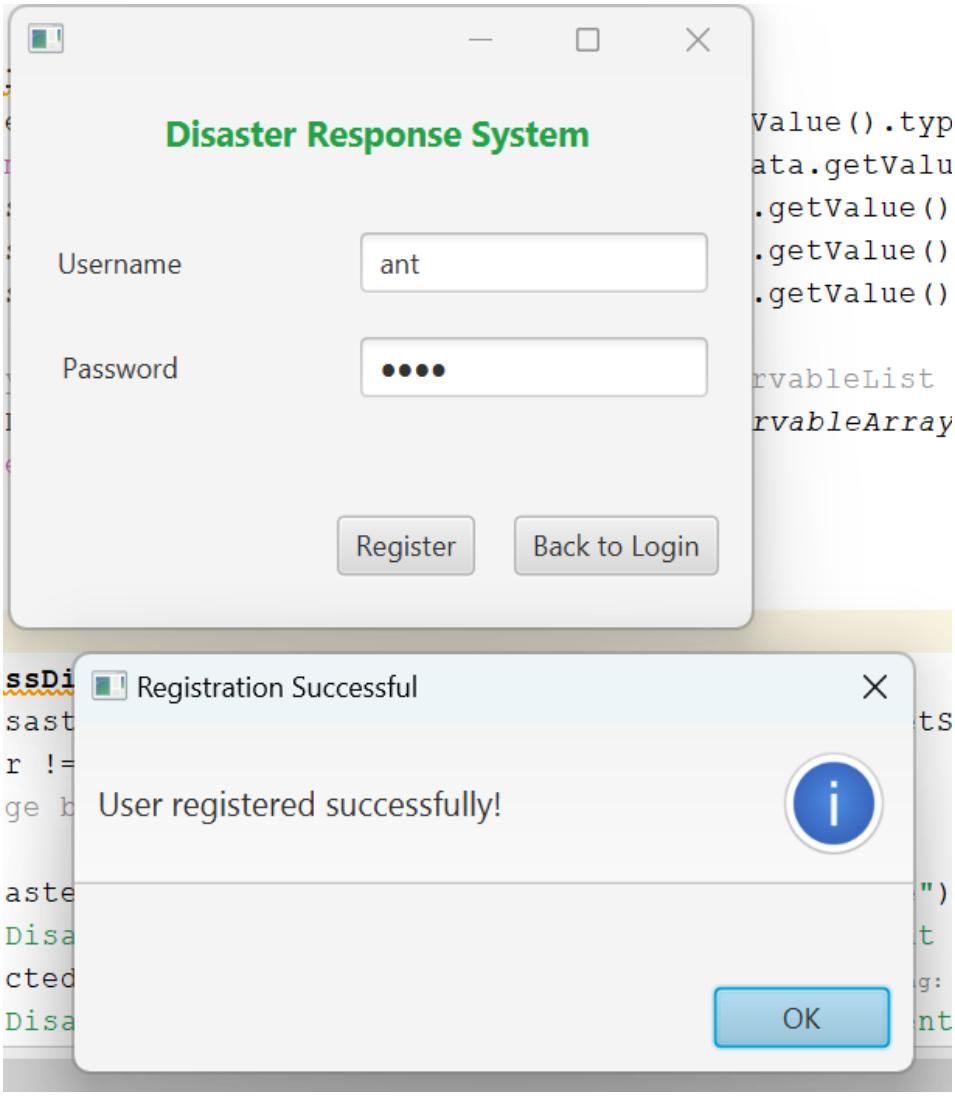
Screenshot of NetBeans:



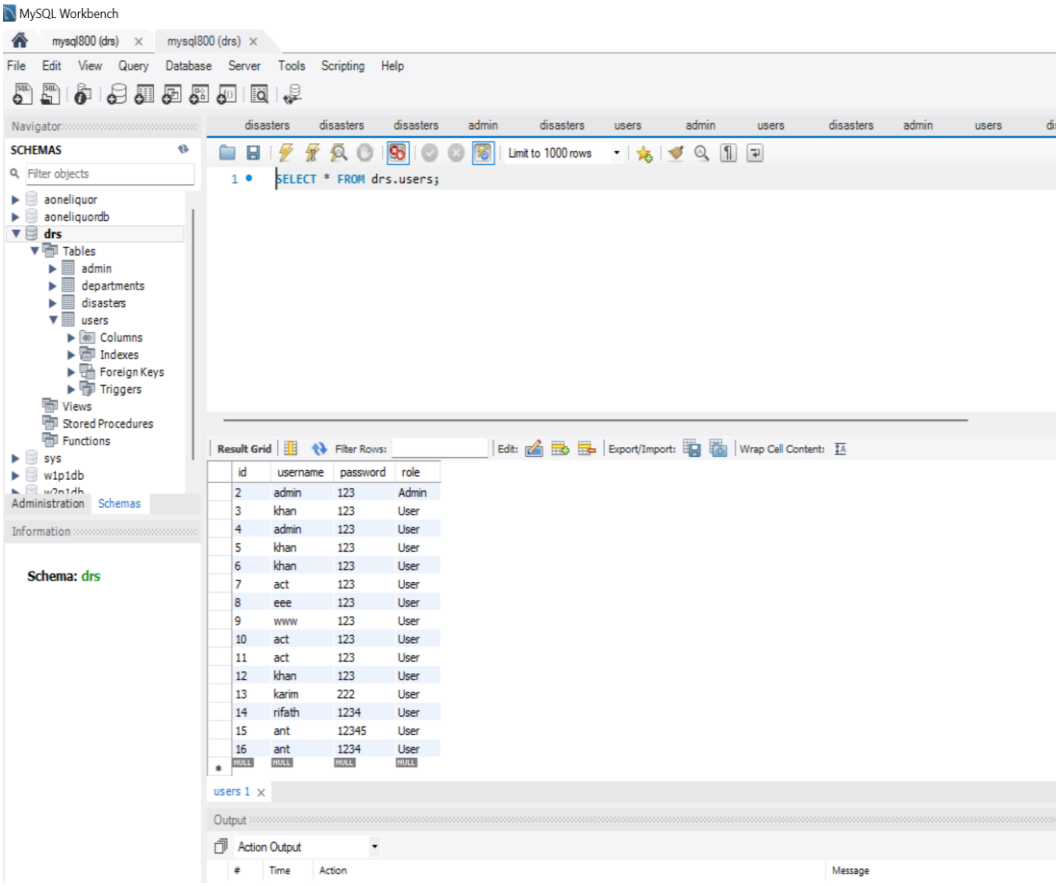
Test Case 4: Successful Registration

- **Input:**
  Username = "ant", Password = "1234"
- **Expected Output:**
  Registration successful and user is redirected to the login page.
- **Actual Output:**
  User registration successful, redirected to login page.
- **Pass/Fail:**
  Pass

Screenshot of NetBeans:



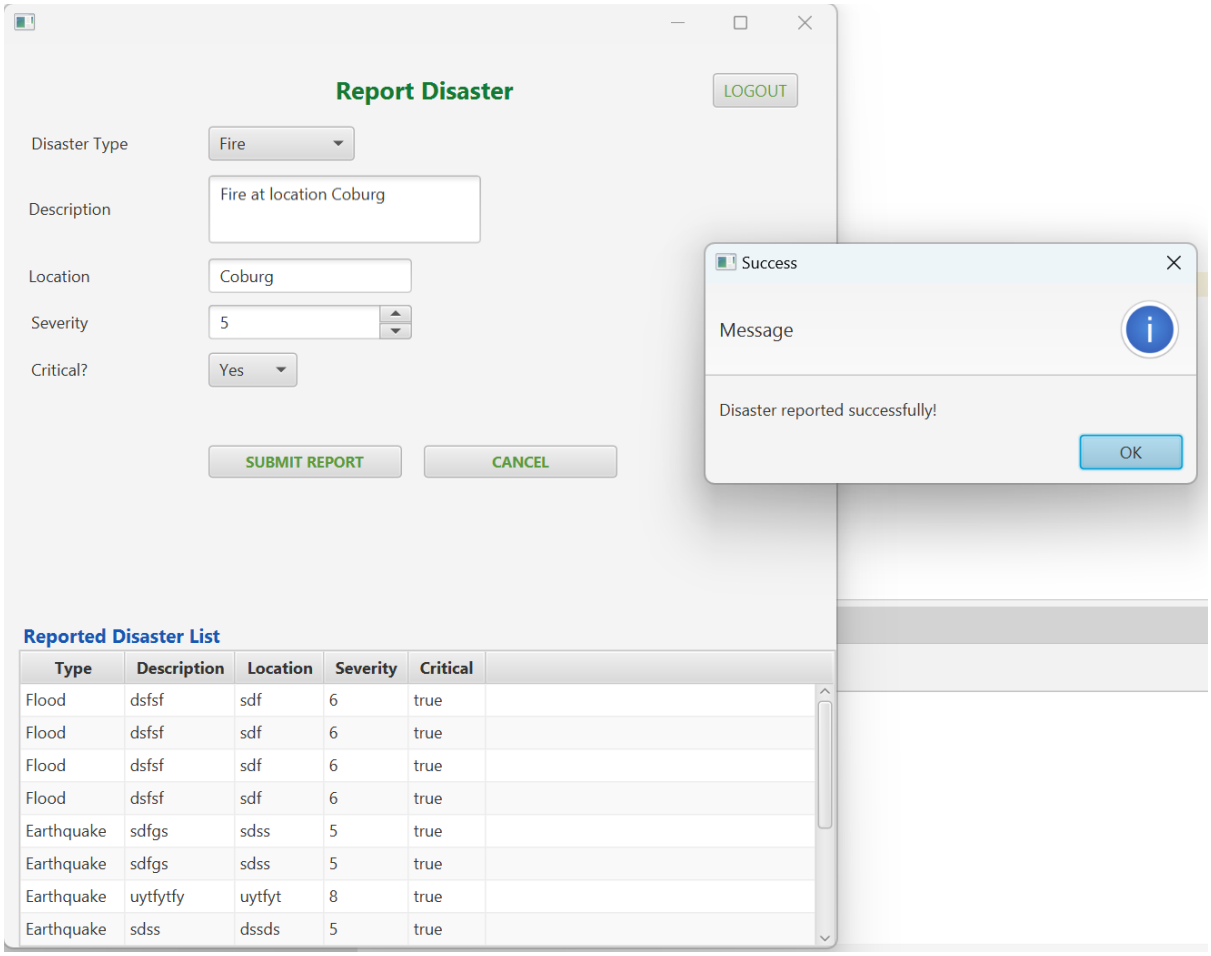
Screenshot of Database User Table:

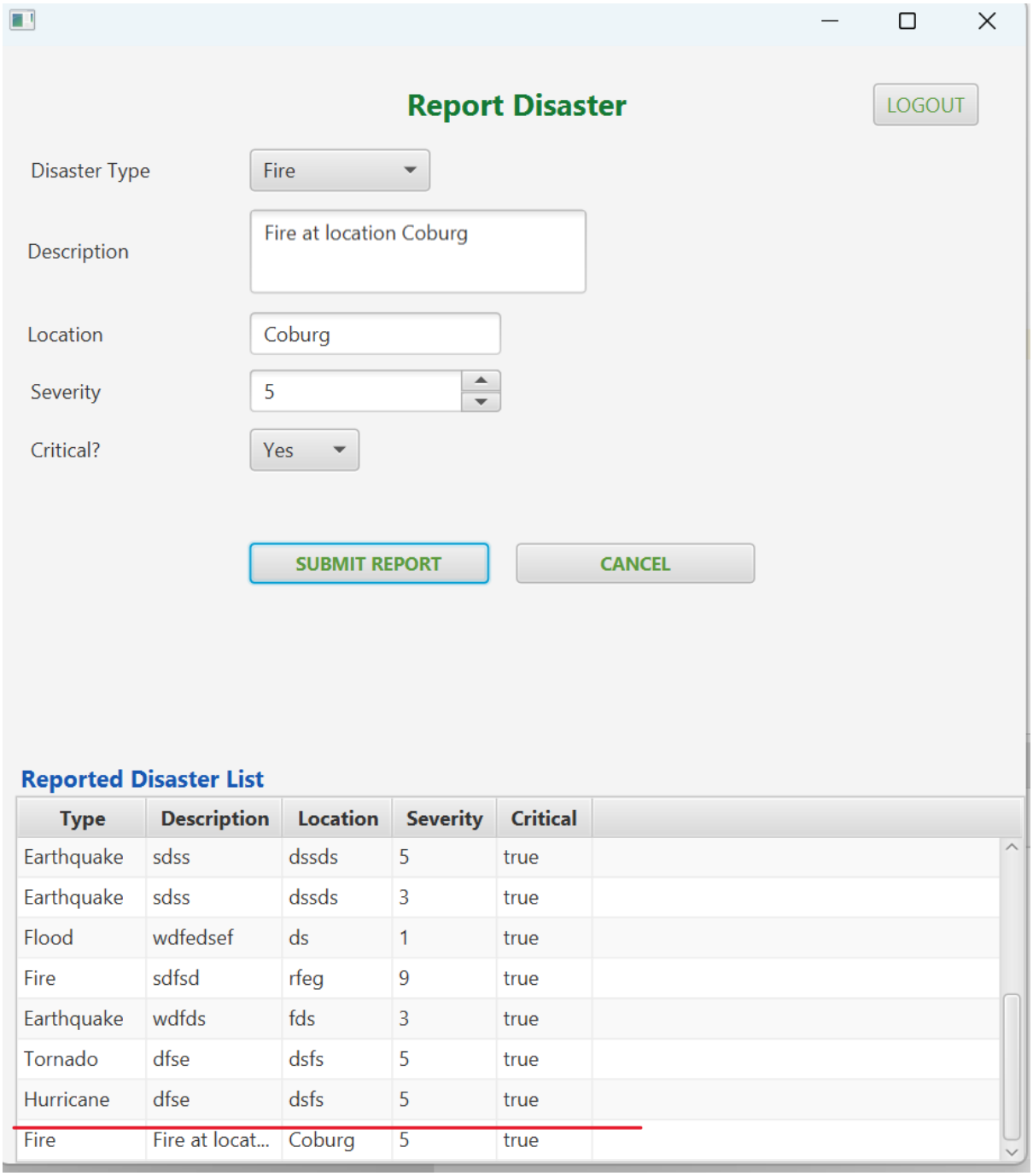


Test Case 5: Disaster Report Submission (Valid Data)

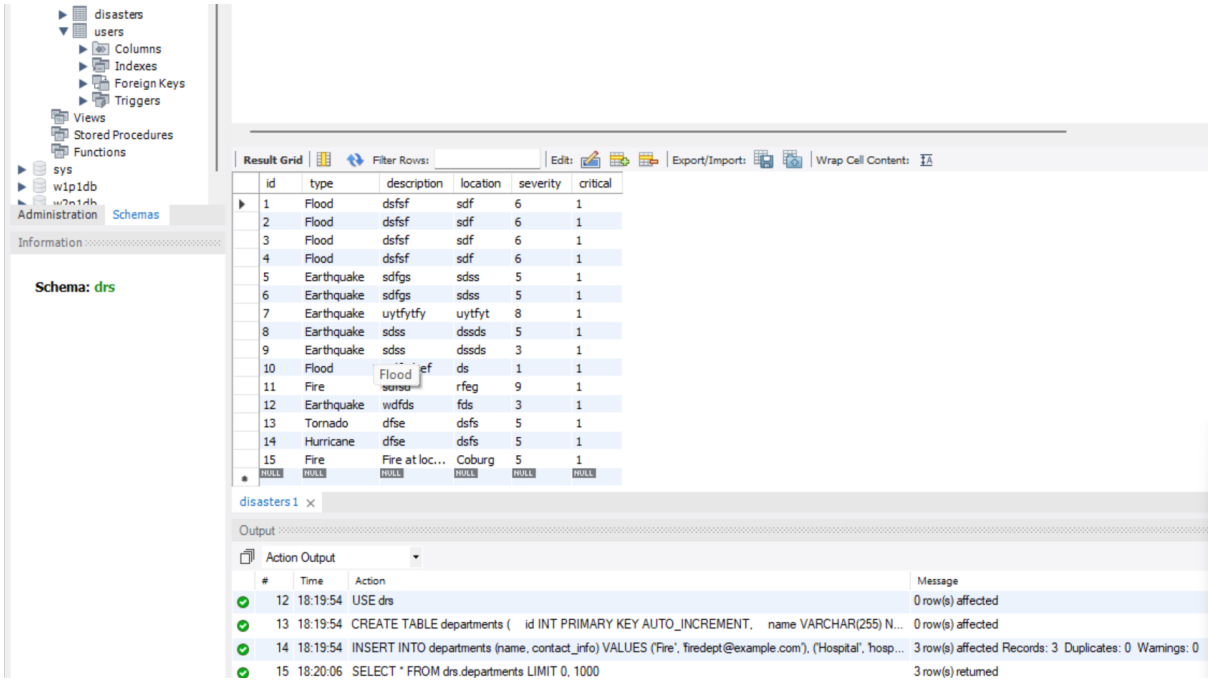
- **Input:**
  Disaster Type = "Fire", Description = "Fire at location Coburg", Location = "Coburg", Severity = 5, Critical = Yes
- **Expected Output:**
  Disaster report submitted successfully.
- **Actual Output:**
  Disaster report submitted and visible in the list.
- **Pass/Fail:**
  Pass

Screenshot of NetBeans:




Report Disaster
LOGOUT
Disaster Type
Fire
Description
Fire at location Coburg
Location
Coburg
Severity
5
Critical?
Yes
SUBMIT REPORT
CANCEL
Reported Disaster List
Type
Description
Location
Severity
Critical
Earthquake
sdss
dssds
5
true
Earthquake
sdss
dssds
3
true
Flood
wdfedsef
ds
1
true
Fire
sdfsd
rfeg
9
true
Earthquake
wdfds
fds
3
true
Tornado
dfse
dsfs
5
true
Hurricane
dfse
dsfs
5
true
Fire
Fire at locat...
Coburg
5
true

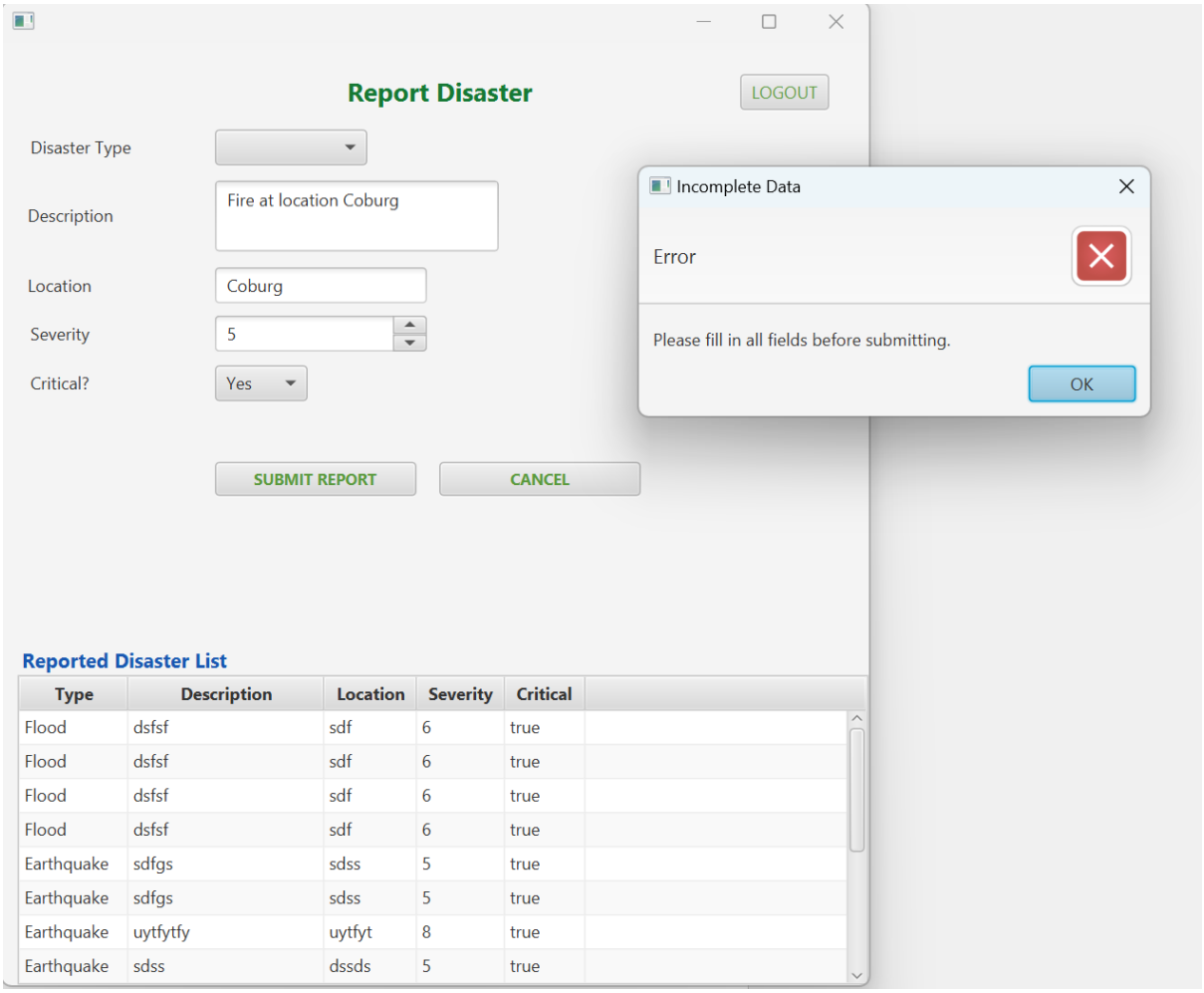
Screenshot of Database Disaster Table:



Test Case 6: Disaster Report Submission (Missing Type)

- **Input:**
  Disaster Type = "", Description = "Fire at location X", Location = "Location X", Severity = 5, Critical = Yes
- **Expected Output:**
  Error message: " Please fill in all fields before submitting "
- **Actual Output:**
  Error message displayed: "Please fill in all fields before submitting"
- **Pass/Fail:**
  Pass

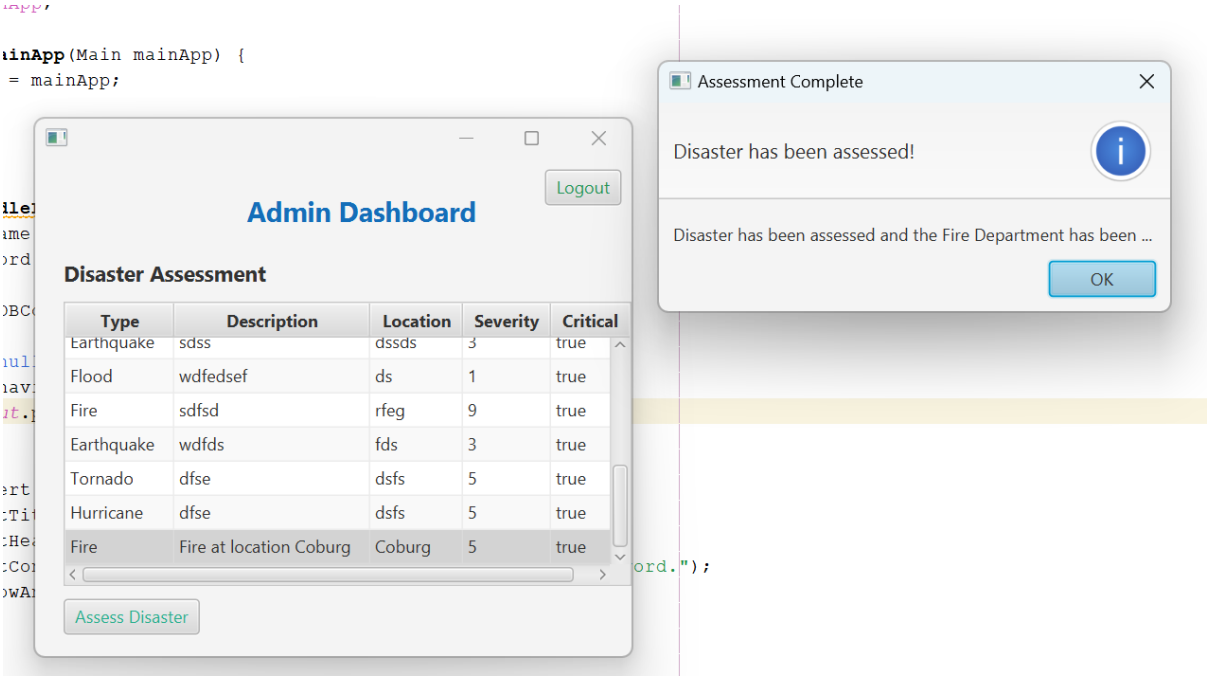
Screenshot of NetBeans:



Test Case 7: Admin Assess Disaster

- **Input:**
  Select disaster type = "Fire"
- **Expected Output:**
  Disaster assessed and related department (Fire Department) notified.
- **Actual Output:**
  Disaster assessed and Fire Department notified.
- **Pass/Fail:**
  Pass

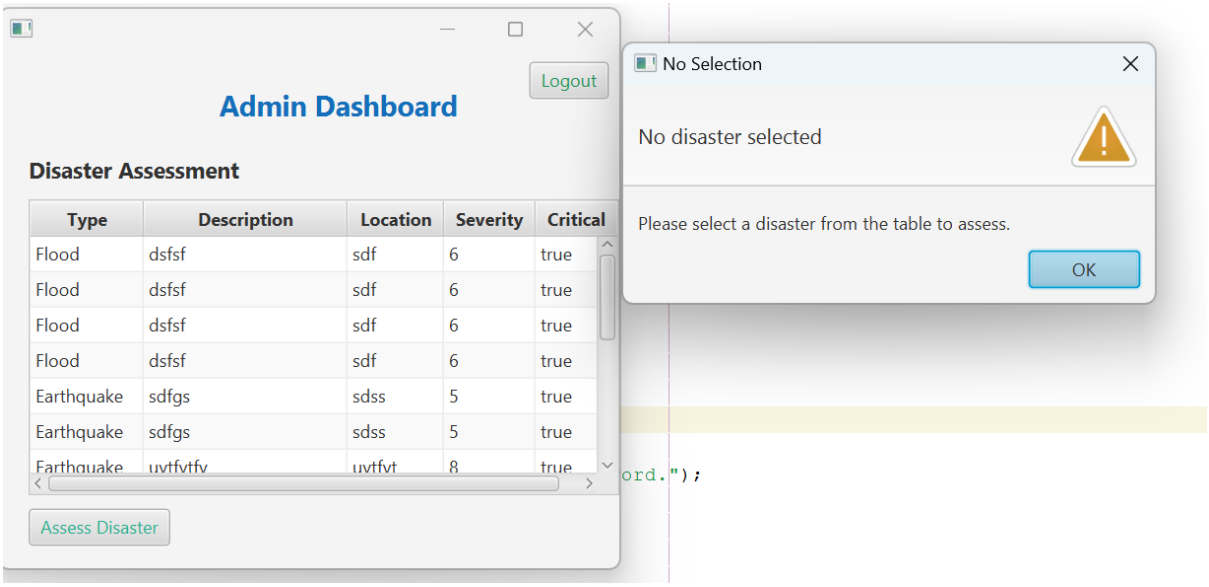
Screenshot of NetBeans:



Test Case 8: Admin Assess Disaster (No Selection)

- **Input:**
  No disaster selected
- **Expected Output:**
  Error message: "No disaster selected"
- **Actual Output:**
  Error message displayed: "No disaster selected"
- **Pass/Fail:**
  Pass

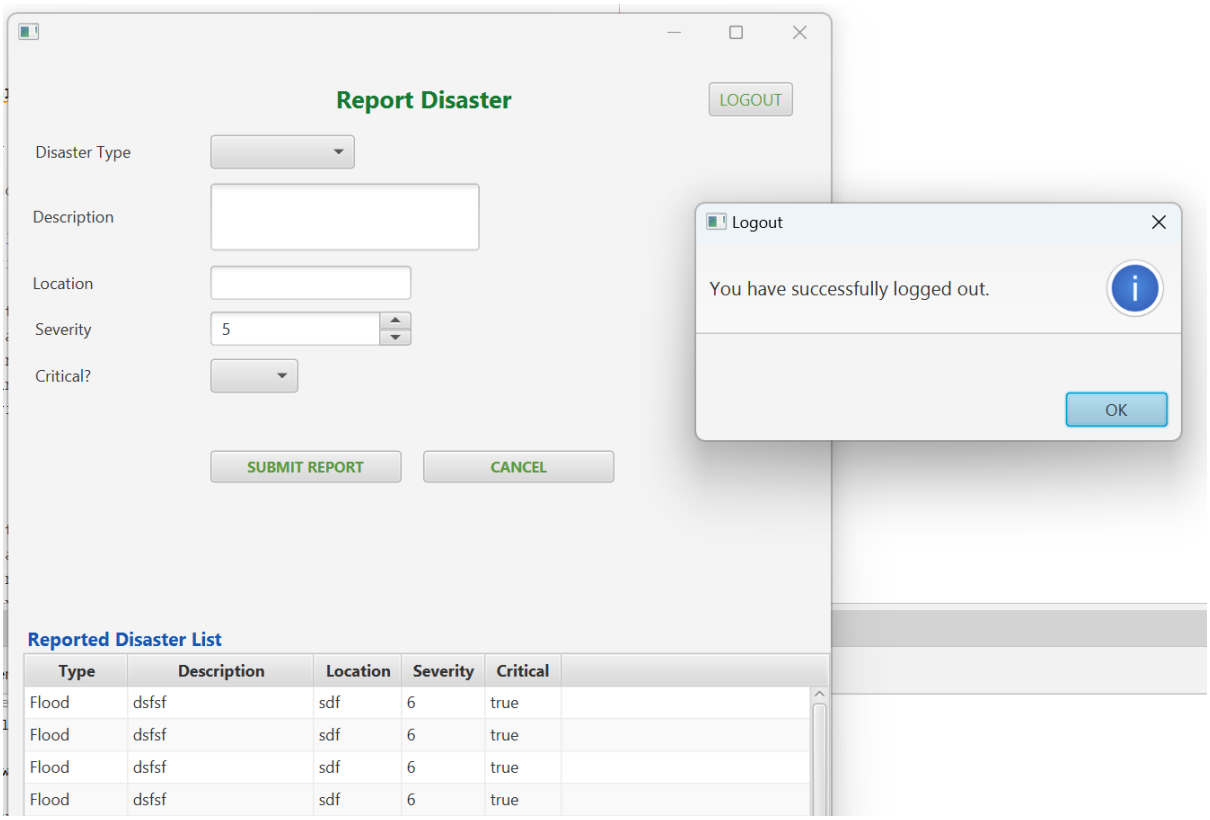
Screenshot of NetBeans:



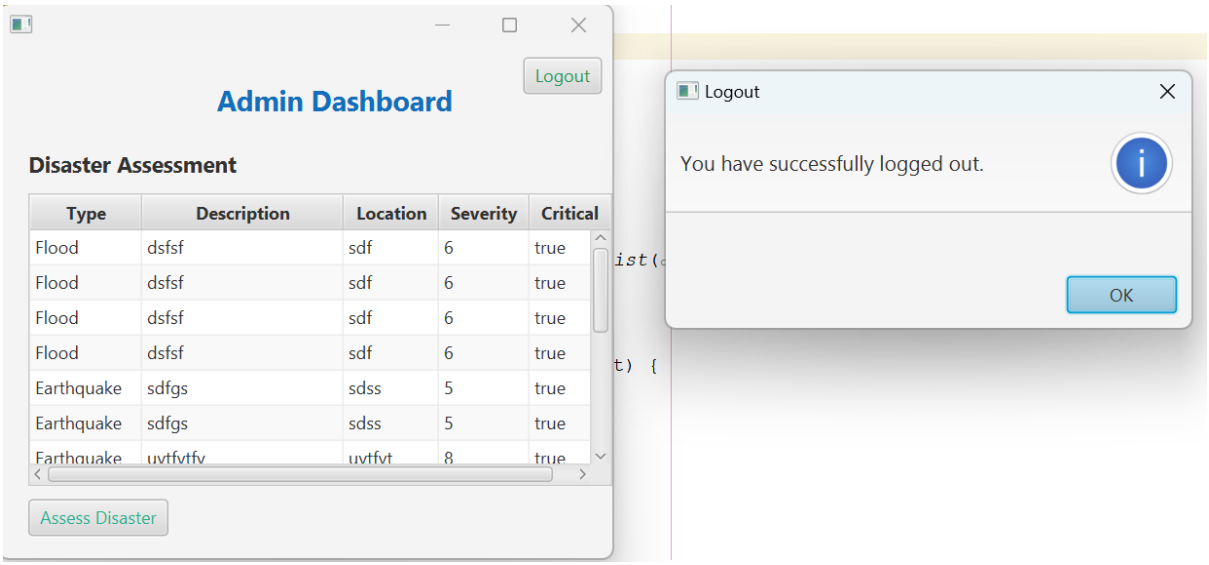
Test Case 9: User Logout

- **Input:**
  Click "Logout" button
- **Expected Output:**
  User is logged out and redirected to the login page.
- **Actual Output:**
  User logged out and redirected to the login page.
- **Pass/Fail:**
  Pass

Screenshot of User Logout:



Screenshot of Admin Logout:



Reflection on Requirements, UI Design, and Implementation

The development of the Disaster Response System (DRS) involved designing a system that can handle disaster reporting and coordination across multiple departments. By enabling users to report different kinds of catastrophes, the system satisfies the functional criteria. Administrators, or coordinators, may then evaluate these reports and designate responsibilities to agencies like Fire, Hospital, and Law Enforcement.

Requirements Specification Reflection

The requirements specified that the system should:

1. Allow disaster reporting by users.
2. Provide disaster assessment by administrators.
3. Coordinate responses with different departments.

The system effectively included these needs. But while the project was being developed, it became clear that further features were required, such improved user input validation and more thorough departmental notifications.

User Interface Design Reflection

Making the system easy to use and intuitive was the goal of the user interface design. ComboBox, TextArea, and TableView were used to create a simple interface for organising reactions and reporting catastrophes. The system was also made more modular by using distinct interfaces for administrators and users.

Implementation Reflection

Managing the role-based navigation (User vs. Admin) and combining the database connection with the JavaFX interface were the main implementation hurdles. A well-designed Model-View-Controller (MVC) architecture and meticulous user session management were the keys to the solution. Future upgrades could provide more thorough catastrophe analytics and improved security features like password encryption.

Conclusion

The Disaster Response System (DRS) meets the key functional requirements and offers a solid foundation for disaster management and coordination. The system is designed to scale, allowing for the addition of more departments and features in the future.

This project offered insightful information on database integration, disaster management procedures, and JavaFX program development. Subsequent revisions may concentrate on enhancing the user interface and including more sophisticated functions like catastrophe simulations and real-time warnings.