Authentication System

ARINDA MARVIN – 2022BSE020PS

Overview

This project is a basic authentication system that demonstrates the use of software design patterns and refactoring principles to create a secure, modular, and reusable software solution.

Repository

The complete source code for this project is available on GitHub:
`https://github.com/marvinarinda/ARINDA_MARVIN2022BSE020PS.git`

1. TODO

1.1. Prerequisites

- **Java Development Kit (JDK):** Version 8 or higher.
- **Apache Ant:** For building the project.
- **Apache Tomcat:** A servlet container to run the application.
- **MySQL Database:** A running MySQL server.

1.2. Procedures

1. **Database Setup:**
   - Create a database named `auth_db`.
   - Create a `users` table with the following schema:

```
CREATE TABLE users (
  id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(255) NOT NULL UNIQUE,
  password_hash VARCHAR(255) NOT NULL,
  password_salt VARCHAR(255) NOT NULL
);
```

   - Update the `db.properties` file in the `src` directory with your MySQL username and password.

2. **Build the Application:**

- Open a terminal and navigate to the project's root directory.
- Run the following command:

```
ant dist
```

3. **Deploy the Application:**
   - Copy the generated `ARINDA_MARVIN_2022BSE020PS.war` file from the `dist` directory to the `webapps` directory of your Tomcat installation.
   - Start the Tomcat server.
4. **Access the Application:**
   - Open your web browser and go to `http://localhost:8080/ARINDA_MARVIN_2022BSE020PS/`.

2. Design Patterns

2.1. Factory Method

The **Factory Method** pattern is used in the `UserFactory` class. The `createUser()` method acts as a factory for creating `User` objects. This pattern encapsulates the object creation logic, making it easy to change the implementation of the `User` class without affecting the rest of the application. It also promotes loose coupling between the `RegisterServlet` and the `User` class.

2.2. Data Access Object (DAO)

The `UserDAO` class implements the **Data Access Object (DAO)** pattern. This pattern separates the data access logic from the business logic. The `UserDAO` provides a clean API for accessing user data without exposing the underlying database implementation. This makes the application more modular and easier to maintain.

3. System Architecture

3.1. Database Credentials

The database credentials are a very important aspect of the project as they allow for the connection between the server and the database. The credentials for this project are as follows:

- **Database:** `auth_db`
- **Location:** `localhost`
- **Username:** `root`
- **Password:** (no password)

3.2. Client-Server Architecture

This application follows a **client-server architecture**:

- **Client:** The client is the user's web browser, which is used to access the application's HTML pages (`login.jsp`, `register.jsp`, etc.). The client sends requests to the server to log in, register, or perform other actions.
- **Server:** The server is the Apache Tomcat server, which hosts the Java servlets that handle the application's business logic. The server receives requests from the client, processes them, and sends responses back to the client.
- **Database:** The database is the MySQL server, which stores the application's data (in this case, the user accounts). The server communicates with the database to retrieve and store data.

This separation of concerns makes the application more scalable and maintainable.

4. Anti-patterns Avoided

- **Hard-coded Values:** Database credentials and configuration values are stored in a `db.properties` file instead of being hard-coded, making the application more configurable and secure.
- **Singleton Database Connection:** The singleton pattern for database connections was removed to prevent threading issues. A new connection is created per request (a connection pool could further improve performance).
- **Code Duplication:** CSS styles have been externalized into `style.css` to avoid duplication and improve UI maintainability.
- **Spaghetti Code:** Code is organized into logical layers (controller, model, util) to improve readability and maintainability.

5. Test Cases

5.1. Registration

- **Test Case 1:** Register a new user with a valid username and password. **Expected Result:** User created successfully and redirected to login page.
- **Test Case 2:** Register with a username that already exists. **Expected Result:** Registration fails; error message displayed.
- **Test Case 3:** Register with an invalid username (too short or contains special characters). **Expected Result:** Registration fails; error message displayed.
- **Test Case 4:** Register with an invalid password (too short). **Expected Result:** Registration fails; error message displayed.

5.2. Login

- **Test Case 5:** Log in with valid credentials. **Expected Result:** Login successful; redirected to success page.
- **Test Case 6:** Log in with valid username but invalid password. **Expected Result:** Login fails; error message displayed.
- **Test Case 7:** Log in with an invalid username. **Expected Result:** Login fails; error message displayed.
- **Test Case 8:** Log in with empty username or password. **Expected Result:** Login fails; error message displayed.

Appendix: Screenshots

This appendix contains screenshots demonstrating various parts and states of the authentication system.

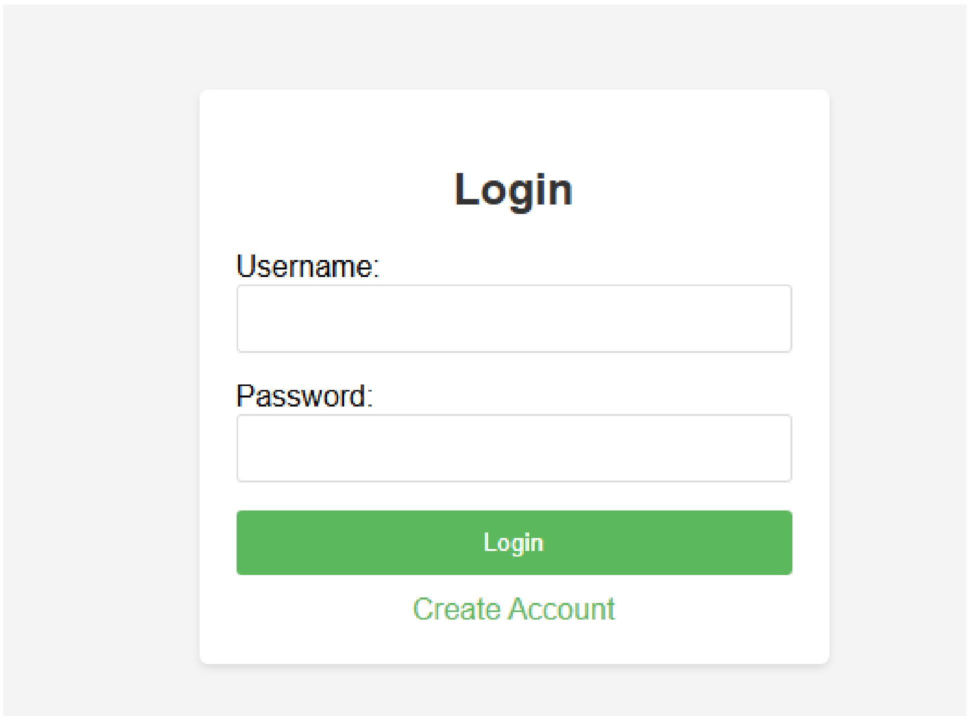
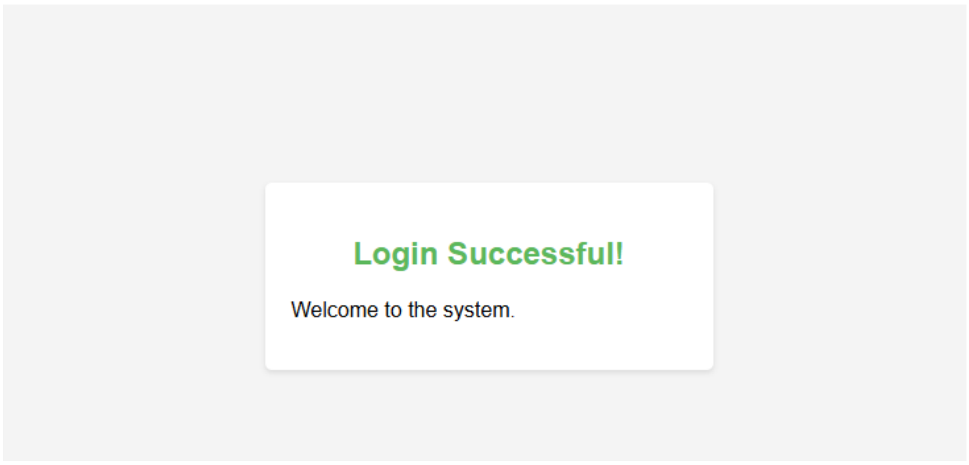


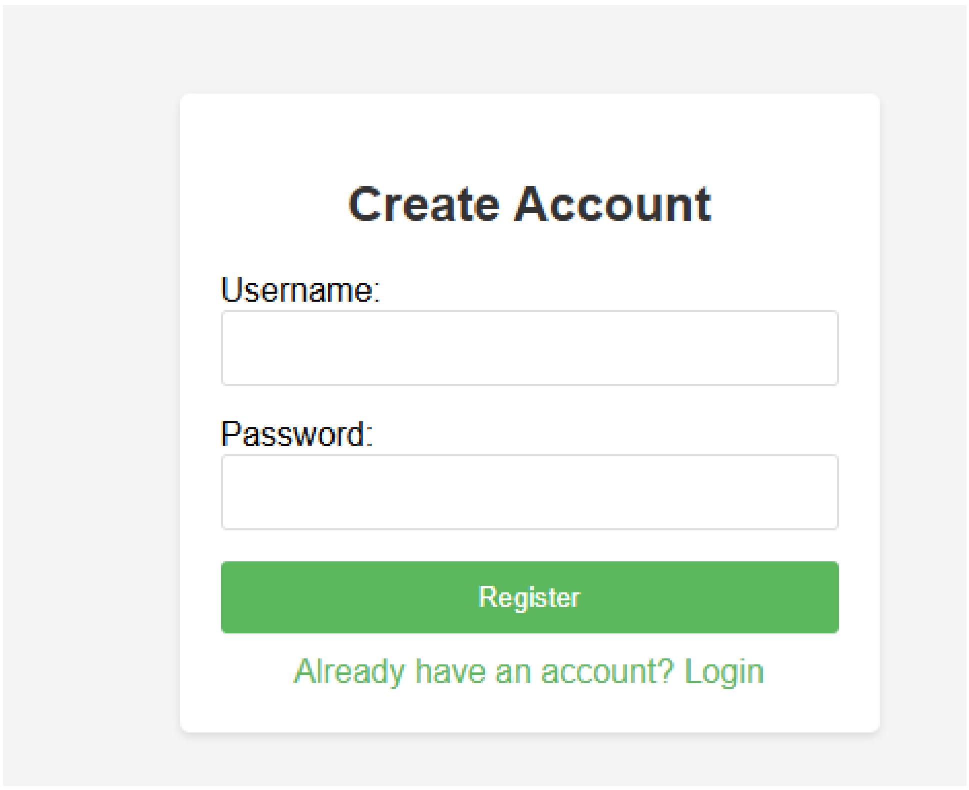
Figure 1: Login Page



Login Successful!

Welcome to the system.

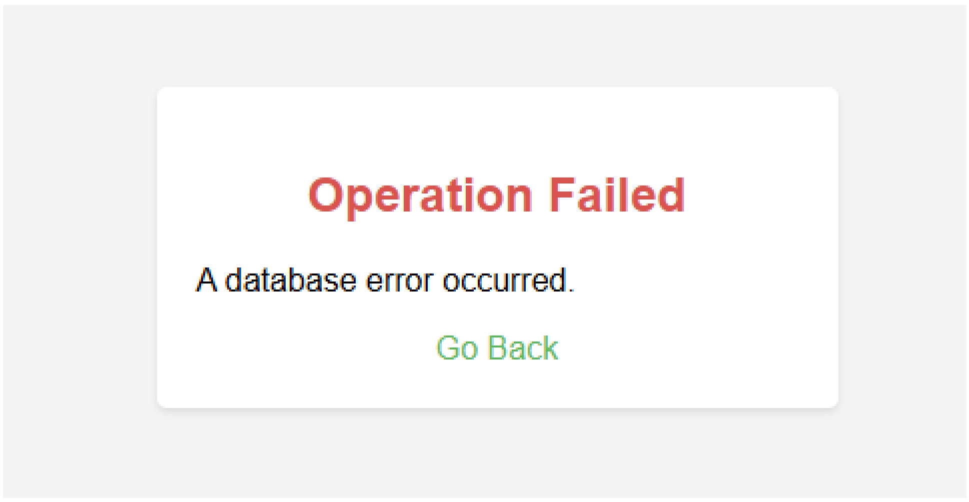
Figure 2: Login Successful


Create Account

Username:

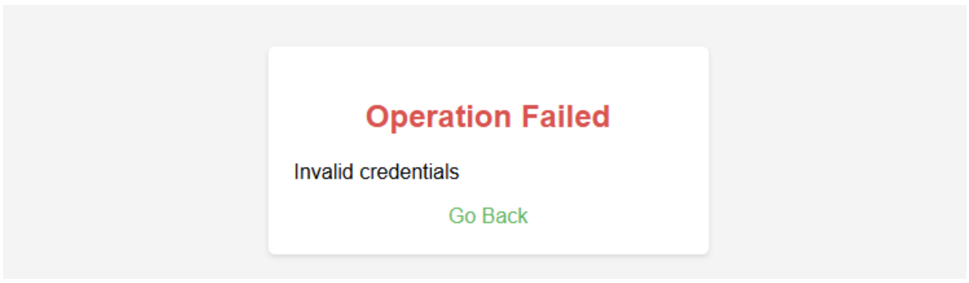
Password:

Figure 3: User Registration Page


Operation Failed

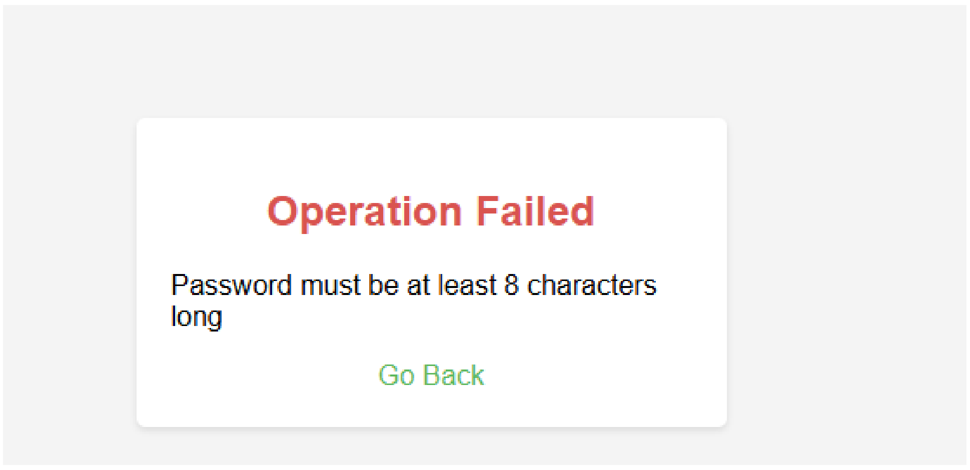
A database error occurred.

Figure 4: Database Connection Error Screen


Operation Failed

Invalid credentials

Figure 5: Invalid Credentials


Operation Failed

Password must be at least 8 characters long

Figure 6: Operation Failed Message