Project Documentation

GITHUB LINK
https://github.com/tushar11-prog/Bank_mangement

1. Overview

This project is a modern **web application** for a bank that facilitates **user registration (signup)**, **authentication (login)**, and a **dashboard** displaying personalized information. The system is designed to handle scalability, security, and efficiency.

---

2. Technology Stack

- **Frontend**:
  - React (JavaScript)
  - CSS for styling
- **Backend**:
  - Spring Boot (Java, REST APIs)
  - Spring Security (Password encoding)
  - Spring Data JPA (Database interaction)
  - MySQL (Relational database)

---

3. Features

Frontend

- **Signup Page**:
  - Collects user details: name, email, and password.
  - Communicates with the backend to register a user.
- **Login Page**:
  - Authenticates users by validating email and password.
  - Redirects authenticated users to the dashboard.
- **Dashboard**:
  - Displays:
    - Account balance.
    - Recent transactions.
    - Navigation features (extendable).

Backend

- **Authentication APIs**:
  - `/api/auth/signup` - Handles user registration.
  - `/api/auth/login` - Handles user authentication.
- **Dashboard API**:
  - `/api/dashboard` - Returns personalized dashboard data.
- **Password Encoding**:
  - Secure storage of passwords using **BCrypt**.

---

4. ER Diagram

Entities:

1. **User**
   - Represents user information.

Relationships:

- Each `User` has:
  - Unique **email** (primary key constraint).
  - One **name** and one **password**.

ER Diagram:

ER Diagram:
plaintext
Copy code

```
+------------------+
|     User         |
+------------------+
| id (PK)          |
| name             |
| email (Unique)   |
| password         |
+------------------+
```

Flow:

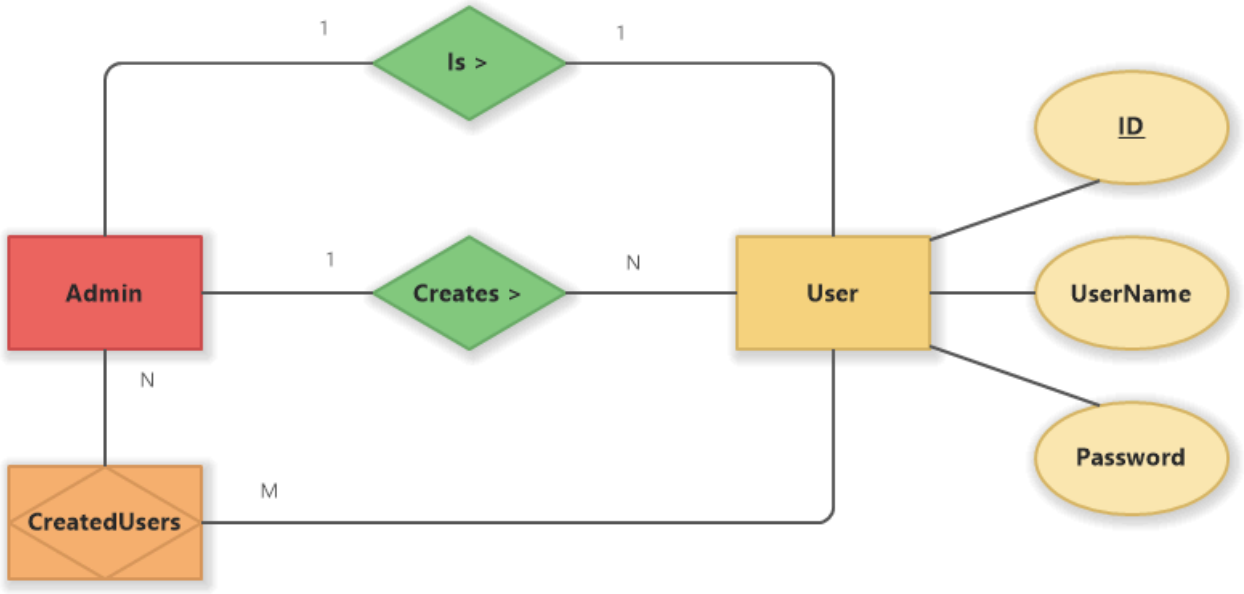
1. **Signup**: Creates a new `User` entry in the database.
2. **Login**: Verifies `email` and `password` against the `User` table.
3. **Dashboard**: Fetches personalized data for the authenticated user.

---

5. Frontend

Folder Structure

```
src/
|-- components/
|   |-- Login.js
|   |-- Signup.js
|   |-- Dashboard.js
|-- App.js
|-- styles.css
```



Frontend Workflow

1. **Signup Page**:
   - Collects user details.
   - Sends a POST request to `/api/auth/signup`.
   - Redirects to the Login page upon success.
2. **Login Page**:
   - Validates credentials with a POST request to `/api/auth/login`.
   - Redirects to the Dashboard on success.
3. **Dashboard**:
   - Fetches data using a GET request to `/api/dashboard`.

API Integration

- Uses `fetch` or `axios` for API calls.

Example from Login:

```
const handleSubmit = async (e) => {
  e.preventDefault();
  const response = await fetch("http://localhost:8080/api/auth/login",
{
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ email, password }),
  });
  if (response.ok) {
    const data = await response.json();
    console.log("Login Successful:", data);
  } else {
    alert("Login Failed");
  }
};
```

CSS Highlights

- Clean and responsive forms.
- Hover effects for buttons.

Example:

```
.form-container {
  max-width: 400px;
  margin: auto;
  background: #fff;
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
```

```
}
```

-

---

6. Backend

Folder Structure

```
src/main/java/com/bankapp/
|-- config/
|   |-- AppConfig.java
|-- controller/
|   |-- UserController.java
|   |-- DashboardController.java
|-- model/
|   |-- User.java
|-- repository/
|   |-- UserRepository.java
|-- service/
|   |-- UserService.java
```

Backend Workflow

1. **Signup Endpoint** (`POST /api/auth/signup`):
    - Accepts JSON payload: `{ name, email, password }`.
    - Hashes the password using **BCrypt**.
    - Saves user to the database.
    - Returns success/failure response.
2. **Login Endpoint** (`POST /api/auth/login`):
    - Accepts JSON payload: `{ email, password }`.
    - Validates credentials against the database.
    - Returns user details or a 401 Unauthorized error.
3. **Dashboard Endpoint** (`GET /api/dashboard`):

Returns:
json

```
{
```

```
  "balance": 5000.0,
  "transactions": ["Deposit $200", "Withdrawal $50", "Transfer $100"]
}
```

-

---

7. Backend Code Overview

Key Files

User Entity: Defines the `users` table.

```
@Entity
@Table(name = "users")
public class User { ... }
```

1.

User Repository: Database operations for the `User` entity.
java
Copy code

```
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByEmail(String email);
}
```

2.

User Service: Business logic for registration and login.

```
public User registerUser(User user) {
    user.setPassword(passwordEncoder.encode(user.getPassword()));
    return userRepository.save(user);
}
```

3. **Controllers**: API endpoints for frontend interaction.

Signup:

```
@PostMapping("/signup")
public ResponseEntity<String> signup(@RequestBody User user) { ... }
```

-

Login:

```
@PostMapping("/login")
public ResponseEntity<?> login(@RequestBody User user) { ... }
```

-

---

8. Running the Application

1. **Backend**:

Run the Spring Boot server:
bash
```
mvn spring-boot:run
```

   - Ensure it is accessible at `http://localhost:8080`.
2. **Frontend**:

Start the React development server:
bash
```
npm start
```

   - Ensure it is accessible at `http://localhost:3000`.

---

9. Testing

- **Signup**:
  - Use the Signup page to create a user.
  - Verify the data is stored in the `users` table.
- **Login**:
  - Test valid/invalid credentials on the Login page.
  - Verify session redirection to the Dashboard.
- **Dashboard**:
  - Fetch data from `/api/dashboard` and display it.

---

10. Future Enhancements

- **Frontend**:
    - Add a logout feature.
    - Enhance the dashboard with real-time charts and graphs.
- **Backend**:
    - Implement JWT-based authentication for secure session management.
    - Add more database tables for transactions and accounts.