# **Design Document for User Management Application**

## **Table of Contents**

1. Introduction
2. Architecture Overview
3. Components Design
   1. 3.1 Backend Design
   2. 3.2 Frontend Design
4. Database Design
5. API Design
6. Security Considerations
7. Setup Instructions
8. Assumptions and Dependencies

## **1. Introduction**

This document outlines the design of a User Management Application that provides user authentication (login/logout) and displays user profile information. The application is built using Spring Boot for the backend and Angular for the frontend.

## **2. Architecture Overview**

The application follows a client-server architecture where:

* **Frontend**: Angular application communicates with the backend via RESTful APIs.
* **Backend**: Spring Boot application processes requests, interacts with the database, and handles business logic.
* **Database**: A relational database (PostgreSQL) stores user data.

### **Architecture Diagram**

## **3. Components Design**

### **3.1 Backend Design**

#### **Project Structure**

src/main/java/com/example/ usermanagement  
│  
├── controller  
│ ├── UserController.java  
│  
├── service  
│ ├── UserService.java  
│  
├── model  
│ ├── User.java   
│  
├── repository  
│ ├── UserRepository.java  
│  
└── security  
 ├── SecurityConfig.java

├── JwtAuthenticationFilter.java  
 └── JwtUtil.java

#### **Key Classes**

* **UserController**: Handles HTTP requests for user-related operations (login, profile retrieval).
* **UserService**: Contains business logic for user authentication and profile management.
* **UserRepository**: Interacts with the database to perform CRUD operations on user data.
* **JwtUtil**: Utility class for generating and validating JWT tokens.
* **JwtAuthenticationFilter**: This Filter intercept the requests and validates the token from request header.

### **3.2 Frontend Design**

#### **Project Structure**

src/app  
│  
├── components  
│ ├── login  
│ │ ├── login.component.ts  
│ │ └── login.component.html  
│ ├── dashboard  
│ │ ├── dashboard.component.ts  
│ │ └── dashboard.component.html

│ ├── register  
│ │ ├── register.component.ts  
│ │ └── register.component.html

│ ├── root  
│ │ ├── app.component.ts  
│ │ └── app.component.html  
│  
├── services  
│ └── auth.service.ts

├── module  
│ └── app.module.ts

├── guards  
│ └── auth.guard.ts

└── routing  
 └── app-routing.module.ts

**Key Components**

* **LoginComponent**: Provides the user interface for login and calls the authentication service.
* **DashboardComponent**: Displays the logged-in user's profile information.
* **RegisterComponent**: Registers a new user for the application.

## **4. Database Design**

### **Schema**

CREATE TABLE users (  
 id SERIAL PRIMARY KEY,  
 username VARCHAR(50) UNIQUE NOT NULL,  
 password VARCHAR(255) NOT NULL,  
 email VARCHAR(100),  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);

### **Entity Classes**

* **User.java**: Maps to the users table and contains fields for user attributes.

## **5. API Design**

### **Endpoints**

|  |  |  |
| --- | --- | --- |
| **HTTP Method** | **Endpoint** | **Description** |
| POST | /api/login | Authenticate user and return JWT |
| GET | /api/user | Fetch user profile (protected) |
| POST | /api/register | Create/Save new user in the applicaton |
| POST | /api/logout | Invalidate user session (optional) |

### **Sample Request/Response**

#### **Login**

**Request**:

{  
 "username": "user1",  
 "password": "password123"  
}

**Response**:

{  
 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."  
}

## **6. Security Considerations**

* **Password Storage**: Passwords are hashed using BCrypt before storing them in the database.
* **JWT Authentication**: Secure user sessions using JWT for stateless authentication.

## **7. Setup Instructions**

### **Backend**

1. Clone the backend repository.
2. Update application.properties with the database connection details.
3. Run the application using Maven: ./mvnw spring-boot:run.

### **Frontend**

1. Clone the frontend repository.
2. Install dependencies: npm install.
3. Run the application: ng serve.

## **8. Assumptions and Dependencies**

* **Database**: PostgreSQL is used; ensure it is installed and configured.
* **Framework Versions**: Spring Boot (3.3.5), Angular (14.x), PostgreSQL.
* **Environment**: Development setup is assumed to be on local machines.