**Employee Management System** (Mini Project 1)

WorkFlow : [Link](https://drive.google.com/file/d/1VaPxPUdUnQ94HSA3OD1dM_YJkHx0-sO6/view?usp=sharing)

1. Introduction

This employee management system allows authorized users to manage employee data. Users are assigned roles with specific permissions, ensuring only authorized actions can be performed. The system utilizes CRUD (Create, Read, Update, Delete) operations for managing employee information. The frontend leverages routing to navigate between different functionalities.

2. Project Setup

2.1 Prerequisites

* Java version: 17
* Node.js version: 18.18.0
* npm version: 9.7.2
* IDE: IntelliJ IDEA, VS Code
* React.Js Version : ^18.2.0
* MySQL Workbench

2.2 Project Structure

**2.2.1 Frontend (React)**

The React application will likely have the following directory structure:

* **src** - Contains the source code for the React application.
  + **components** - Houses reusable React components for UI elements (e.g., EmployeeList, EmployeeDetails, etc.).
  + **service**- Utility functions used throughout the application (e.g., API calls, data formatting).
  + **App.js** - The main application entry point that renders the root component.
  + **index.js** - Initializes the React application and renders it to the DOM.
  + **styles** - Contains stylesheets for the application (e.g., style-mobile.css).
* **public** - Stores static assets used by the application (e.g., images, favicon.ico,project workflow video).
* **package.json** - Defines project dependencies and scripts (e.g., npm start to run the development server).

**Optional High-Level Diagram:**

EmployeeManagementFrontend/

├── src/

│   ├── components/

│   ├── service/

│   ├── App.js

│   ├── index.js

│   ├── styles/

├── public/

└── package.json

**2.2.2 Backend (Java Spring Boot)**

The Java backend using Spring Boot will likely have the following structure:

* **src/main/java** - Contains the main Java source code for the backend application.
  + **com.example.EmployeeManagementBackend**
    - **entity**- Java classes representing data models (e.g., Employee.java).
    - **config**: beans are created securityfilterchain
    - **dto**- Java classes representing data models (e.g., Employee.java).
    - **repository** - Interfaces and implementations for data access (e.g., EmployeeRepository.java).
    - **service** - Business logic layer for employee management operations (e.g., EmployeeService.java).
      * **Impl** - Implementation part of service layer.
    - **controller** - REST API controllers handling incoming requests (e.g., EmployeeController.java).
    - **security** - Security configuration classes.
    - **Exception**: Exception handling
    - **application** - Main application class with Spring Boot configuration (e.g., EmployeeManagementApplication.java).
* **src/main/resources** - Contains configuration files and resources.
  + **application.properties** - Spring Boot application configuration.
  + **sql** - Script to initialize the database with sample data.
* **pom.xml** - Defines project dependencies and build configuration for Maven.

**Optional High-Level Diagram:**

EmployeeManagementBackend/

├── src/

│   └── main/

│       ├── java/

│       │   └── com.example.employeemanagement/

│       │           ├── entity/

│       │           ├── dto/

│       │           ├── repository/

│       │           ├── service/

│       │           ├── impl/

│       │           ├── controller/

│       │           └── security/

│       │           └── exception/

│       │               └── application.java

│       └── resources/

│           ├── application.properties

│           └──sql

└── pom.xml

2.3 Setting Up the Development Environment

* Creating the project (using Spring Initializr for backend, create-react-app for frontend)
* Installing dependencies (using mvn install for backend, npm install for frontend)
* Running the application locally (commands to start both frontend - localhost:3000 and backend -localhost:8080), database - localhost:3306)

3. Backend API Documentation

3.1 Technology Stack

* Java framework: Spring Boot
* Database: MySQL
* Javascript Framework : React.js

3.2 API Reference

* Used Swagger to create API Documentation <http://localhost:8080/swagger-ui/index.html>

4. Frontend React Application

4.1 Component Structure

* Application contains 2 Roles - Admin  and User
* Admin can add, edit , delete , View all Employees
* Users can only view the Employee Details.
* React useState is used to manage state.

4.2 Data Fetching

* Axios Library used to fetch and send frontend data.

import axios from "axios";

const AUTH\_REST\_API\_URL = "http://localhost:8080/api/auth";

export const registerAPICall = (registerObj) =>

  axios.post(AUTH\_REST\_API\_URL + "/register", registerObj);

4.3 User Interface (UI) Components

* Employee Register and Login UI - Interacts api to login and register
* Employee CRUD - Maintained Separate Components for all
* React Router used for routing, react-hook-form used for form validation.

4.4 Styling

* Style.css and style-mobile.css used.

5. Deployment Instructions

5.1 Building the Application

* Frontend  - Execute below comments to run the application
  + npm install
  + Npm start
* Backend
  + mvn package

5.2 Server Configuration

* Document any server-side configuration required (web server setup, environment variables etc.).

6. Additional Considerations

* Testing: unit testing added for backend.
* Deployment Options: Heroku
* Version Control: GitHub

7. Conclusion

This documentation provides a comprehensive guide for the employee management system. It covers the system overview, user roles and permissions, frontend routing with CRUD operations, technology stack considerations, and security best practices. Additionally, it details the project structure for both the React frontend and Java Spring Boot backend, along with instructions for packaging the backend for deployment using Maven.