Employee Management Application

Abstract:

This project, titled **"Employee Management Application,"** is designed to facilitate the management of employee records within an organization using modern web technologies. The application leverages **Spring Boot** as the backend framework, integrated with **Java Persistence API (JPA)** for managing the data model. The frontend is developed using **HTML, CSS, and Bootstrap** alongside the **Thymeleaf** template engine, ensuring a seamless user experience. The system is built to interact with a **MySQL** database, depending on the configuration, to store and retrieve employee data.

Key functionalities include creating, reading, updating, and deleting employee records, with validation and error handling implemented at both the backend and frontend levels. The project also includes detailed documentation for setting up, configuring, and running the application, with **Swagger** integration for API documentation. This ensures that the application is not only functional but also well-documented for ease of use and future development.

1. Project Overview

- **Project Title:** Building an Employee Management Application with Spring Boot ,HTML, CSS, and Bootstrap or Thymeleaf Framework.

- **Description:** This project is aimed at developing a full-fledged employee management system with a Spring Boot backend and a Thymeleaf-based frontend. The application allows users to perform CRUD (Create, Read, Update, Delete) operations on employee data. It features robust backend development using Spring Boot and JPA for database interactions, and a user-friendly frontend developed with HTML, CSS, JavaScript, and Thymeleaf.

2. Prerequisites:

Before starting the project, ensure you have the following software installed:

- **Java Development Kit (JDK) 11 or higher**
- **Apache Maven 3.6+**
- **MySQL**
- **Integrated Development Environment (IDE):** Eclipse, Spring Tool Suite (STS) • **Git (optional, for version control)**
- **Postman or Command Prompt.**

3. Project Setup

3.1 Clone the Repository

To begin, clone the project repository to your local machine:

```
git clone <repository-url>
cd employee-management-app
```

3.2 Database Setup

You can choose either MySQL as your database. Configure the application.properties file to set up the database connection.

- **MySQL Configuration:**

```
spring.datasource.url=jdbc:mysql://localhost:3306/user_db_emp1
spring.datasource.username=root
spring.datasource.password=your-password
spring.jpa.hibernate.ddl-auto=update spring.jpa.show-sql=true
```

3.3 Build and Run the Project

Build the project using Maven:

- mvn clean install

After successfully building the project, run it using:

- mvn spring-boot:run

3.4.pom.xml on added:

- Lombok
- Spring web
- Spring Data JPA
- Spring boot DevTools
- MySQL Driven
- Thymeleaf
- Validation

The application will be accessible at http://localhost:8080.

4. Backend Development (Spring Boot and JPA)

4.1 JPA Entity and Data Model

The application uses JPA for database interactions. Below is an example of the `User` entity:

Sample coding:

User.java

```
package org.user.app.model;

import jakarta.persistence.Column; import
jakarta.persistence.Entity; import
jakarta.persistence.GeneratedValue; import
jakarta.persistence.GenerationType; import
jakarta.persistence.Id; import
jakarta.validation.constraints.Email; import
jakarta.validation.constraints.NotBlank;
import lombok.Data;
import lombok.NoArgsConstructor;


@Entity
@Data
@NoArgsConstructor public
class User
{



        @Id
        @GeneratedValue(strategy=GenerationType.IDENTITY)
private long id;


        @NotBlank(message="Name is mandatory")
@Column(name="UserFirstName")
```

```
        private String UserFirstName;

        @NotBlank(message="Name is mandatory")
@Column(name="UserLastName")
        private String UserLastName;

        @NotBlank(message="Email is mandatory")
        @Column(name="UserEmailId")
        @Email(message="Incorrect email format")
private String UserEmailId;
}
```

4.2 RESTful API Implementation

The backend provides RESTful APIs to handle JPA operations on employee data.

UserController:

```
package org.user.app.controller;

import org.springframework.validation.BindingResult;
import org.springframework.ui.Model;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller; import
org.springframework.web.bind.annotation.GetMapping; import
org.springframework.web.bind.annotation.PathVariable; import
org.springframework.web.bind.annotation.PostMapping; import
org.springframework.web.bind.annotation.RequestMapping; import
org.springframework.web.servlet.ModelAndView; import
org.user.app.model.User; import org.user.app.service.UserService;
import jakarta.validation.Valid;

@Controller
public class UserController {
  @Autowired
  private UserService userService;

  @GetMapping("/users")    public String userList(Model
model) {       model.addAttribute("users",
this.userService.getUsers());       return "index";
  }
```

```
  @GetMapping("/user")
  public String showAddUserForm(Model model) {
model.addAttribute("user", new User());
    return "add-user";
  }

  @PostMapping("/process")
  public String addUserProcess(@Valid User user, BindingResult result,
Model model) {
    if (result.hasErrors()) {
return "add-user";
    }
    this.userService.addUser(user);
return "redirect:users";
  }

  @GetMapping("/view/{id}")    public String
viewUser(@PathVariable("id") long id, Model model) {
model.addAttribute("user", this.userService.getUserById(id).get());
return "user";
  }

  @RequestMapping("/delete/{id}")
  public String deleteUser(@PathVariable("id") Long id) {
    this.userService.deleteUserById(id);
return "redirect:/users";
  }

  @GetMapping("/update/{id}")    public ModelAndView
showUpdateUserForm(@PathVariable("id") long id) {
    Optional<User> user = this.userService.getUserById(id);
ModelAndView modelAndView = new ModelAndView();
    if (user.isPresent()) {
modelAndView.setViewName("update-user");
modelAndView.addObject("user",
this.userService.getUserById(id).get());
    } else {
      modelAndView.setViewName("index");
    }
    return modelAndView;}
}

modelAndView.setViewName("update-user");
```

```
modelAndView.addObject("user",
this.userService.getUserById(id).get());
return modelAndView;}
else{
modelAndView.setViewName("index"); return
modelAndView;}
```

5. Frontend Development (HTML, Bootstrap, Thymeleaf)

5.1 Thymeleaf Template Setup

Thymeleaf is used as the template engine for the frontend. The templates are located in the `src/main/resources/templates` directory.

Index.html(sample code)

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">

    <title>Hello, world!</title>
  </head>
  <body>
    <h1>User Records</h1>
       <a  href="/user" type="button" class="btn btn-info">Add User</a>
    <div class="container">
             <table class="table">
               <thead>
                 <tr>
                      <th scope="col">UserFirstName</th>
                   <th scope="col">UserLastName</th>
                   <th scope="col">UserEmailId</th>
                      <th scope="col">Action</th>
                 </tr>
               </thead>
```

```
                <tbody>
                  <tr th:each="user:${users}">
                    <td th:text="${user.UserFirstName}"></td>
                    <td th:text="${user.UserLastName}"></td>
                    <td th:text="${user.UserEmailId}"></td>
                    <td>
    <a th:href="@{/update/{id}(id=${user.id})}" type="button" class="btn btn-
info">update</a>
    <a th:href="@{/view/{id}(id=${user.id})}" type="button" class="btn btn-
danger">view</a>
                     <a th:href="@{/delete/{id}(id=${user.id})}" type="button"
class="btn btn-info">Delete</a>
                      </td>
                  </tr>

                </tbody>
              </table>
        </div>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
crossorigin="anonymous"></script>

    <!-- Option 2: Separate Popper and Bootstrap JS -->
    <!--
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"
integrity="sha384-IQsoLXl5PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFTxbJ8NT4GN1R8p"
crossorigin="anonymous"></script>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js"
integrity="sha384-cVKIPhGWiC2Al4u+LWgxfKTRIcfu0JTxR+EQDz/bgldoEyl4H0zUF0QKbrJ0EcQF"
crossorigin= "anonymous"></script>
    -->
  </body>
</html>
```

6.Database and Data Model

6.1 MySQL Setup

Set up the database depending on your preference. Use the following commands to create the database:

- **MySQL:**

  CREATE DATABASE user_db_emp1;
  TABLE NAME: user.

6.2 JPA Entity Mapping

Map the `Employee` entity to the database using JPA annotations.

Example Mapping for MySQL:

@Entity

@Table(name = "user")

Using private modifier ,data type String.and getter and setters…….

7. API Documentation

7.1 Swagger Integration

Swagger is integrated into the project for API documentation. Add the Swagger dependency in your `pom.xml`:

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-boot-starter</artifactId>
    <version>3.0.0</version>
</dependency>
```

Accessing Swagger UI: After starting the application, Swagger UI is accessible at ~~`http://localhost:8080/swagger-ui/`~~.

7.2 API Endpoints Documentation

- **Create Employee:** Method: POST

  URL: `/api/user`

  - Request Body:

"UserFirstName": "Harini",

"UserLastName": "Ravi",

"UserEmailId": "harini@test.com",

- **Get Employee by ID:**
- **Method:** GET • **URL:** `/api/user/{id}`
- **Update Employee:**
- **Method:** PUT
- **URL:** `/api/user/{id}`
- **Request Body:** Same as create employee
- **Delete Employee:**
- **Method:** DELETE
- **URL:** `/api/user/{id}`

8. Deployment

10.1 Packaging the Application

Package the application into a JAR file using Maven:

- Mvn clean package

10.2 Deploying to a Server

9. Conclusion

This documentation provides a comprehensive guide to setting up, developing and deploying the Employee Management Application. It covers all aspects of the application, including backend and frontend development, database configuration, API documentation, validation, error handling, and testing. By following this documentation, developers can easily set up and run the application while ensuring that all required functionalities are properly implemented.