**Employee Management System Documentation**

**Introduction**

The Employee Management System is a web-based application designed to manage employee information efficiently. The system allows users to add, view, edit, and delete employee records through a user-friendly interface built with HTML, CSS, and JavaScript (or Thymeleaf). The backend is implemented using Spring Boot and JPA, with data stored in a relational database like MySQL.

**Technologies Used**

1. **Frontend:**
   * HTML, CSS, JavaScript (or Thymeleaf)
   * Axios (for HTTP requests)
2. **Backend:**
   * Spring Boot (RESTful APIs)
   * Spring Data JPA (data persistence)
3. **Database:**
   * MySQL/MongoDB
4. **Tools:**
   * Swagger (API documentation)
   * Maven (dependency management)
   * GitHub (version control)

**Project Architecture**

1. **Frontend:** User interface for employee data management.
2. **Backend:** REST APIs to handle CRUD operations.
3. **Database:** Data storage and JPA entity mappings.

**Project Features**

1. **Employee Management:**
   * Add new employee records.
   * View a list of employees with details (First Name, Last Name, Email, etc.).
   * Update existing employee information.
   * Delete employee records.
2. **Validation:**
   * Form validation on the frontend.
   * Validation for API requests on the backend.
3. **Error Handling:**
   * Display error messages for invalid inputs.
   * Handle server errors gracefully.

**Database Design**

* **Employee Table:**
  + id (Primary Key)
  + first\_name (VARCHAR)
  + last\_name (VARCHAR)
  + email (VARCHAR)
  + created\_at (TIMESTAMP)
  + updated\_at (TIMESTAMP)

**API Endpoints**

1. **GET /api/employees**
   * Fetch all employee records.
   * **Response:** Array of employee objects.
2. **GET /api/employees/{id}**
   * Fetch a specific employee by ID.
   * **Response:** Employee object.
3. **POST /api/employees**
   * Add a new employee record.
   * **Request Body:** JSON object with employee details.
   * **Response:** Status message.
4. **PUT /api/employees/{id}**
   * Update an existing employee record.
   * **Request Body:** JSON object with updated employee details.
   * **Response:** Status message.
5. **DELETE /api/employees/{id}**
   * Delete an employee record.
   * **Response:** Status message.

**Frontend Features**

1. **Employee List:**
   * Display all employee records in a table format with options to update or delete.
2. **Add/Edit Employee:**
   * Form to add or update employee details.
3. **Validation:**
   * Real-time validation for form fields like email and names.

**Swagger Integration**

Add dependency in pom.xml

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-boot-starter</artifactId>

<version>3.0.0</version>

</dependency>

Configure Swagger:

@Configuration

@EnableSwagger2

public class SwaggerConfig {

@Bean

public Docket api() {

return new Docket(DocumentationType.SWAGGER\_2)

.select()

.apis(RequestHandlerSelectors.any())

.paths(PathSelectors.any())

.build();

}

}