Employee Management CRUD Project Documentation

1. Introduction

The **Employee Management System** is a simple CRUD (Create, Read, Update, Delete) web application built using **Spring Boot, JPA, and REST APIs**. This project provides an API to manage employee details, including adding, retrieving, updating, and deleting employees.

2. Technologies Used

- **Spring Boot** (for building REST APIs)
- **Spring Data JPA** (for database interaction)
- **H2 Database / MySQL** (for storing employee details)
- **Maven** (for project management)
- **Lombok (optional)** (for reducing boilerplate code)
- **Postman / Swagger UI** (for testing APIs)

3. Project Setup

3.1. Generating the Project

This project was generated using Spring Initializr with the following dependencies:

- **Spring Web**
- **Spring Data JPA**
- **H2 Database (or MySQL)**

3.2. Cloning the Project

```
git clone https://github.com/your-repository/employee-crud.git
cd employee-crud
```

3.3. Configuring Application Properties

For H2 Database

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
```

For MySQL (If using MySQL)

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

4. Project Structure

```
css
CopyEdit
employee-crud/
├── src/main/java/org/employee/employee_CRUD/
│   ├── Controller/
│   │   ├── EmployeeController.java
│   ├── Dao/
│   │   ├── EmployeeDao.java
│   ├── Repository/
│   │   ├── EmployeeRepository.java
│   ├── dto/
│   │   ├── Employee.java
│   ├── EmployeeCrudApplication.java
├── src/main/resources/
│   ├── application.properties
```

```
|— pom.xml
```

5. Entity Class

```
java
CopyEdit
@Entity
public class Employee {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int emp_id;
	private String emp_name;
	private String emp_email;
	private String emp_password;
	private long mobileno;

	// Getters and Setters
}
```

6. Repository Interface

```
@Repository
public interface EmployeeRepository extends JpaRepository<Employee,
Integer> {
}
```

7. DAO Class

```
@Repository
public class EmployeeDao {

	@Autowired
	private EmployeeRepository repo;

	// Save Employee
```

```
	public void saveEmployee(Employee emp) {
		repo.save(emp);
	}

	// Find Employee by ID
	public Optional<Employee> findById(int id) {
		return repo.findById(id);
	}

	// Get all Employees
	public List<Employee> findAll() {
		return repo.findAll();
	}

	// Delete Employee by ID
	public void deleteById(int id) {
		repo.deleteById(id);
	}

	// Update Employee
	public void update(int id, Employee emp) {
		if (repo.existsById(id)) {
			Employee existingEmployee =
repo.findById(id).orElse(null);
			if (existingEmployee != null) {
				existingEmployee.setEmp_name(emp.getEmp_name());

	existingEmployee.setEmp_email(emp.getEmp_email());

	existingEmployee.setEmp_password(emp.getEmp_password());
				existingEmployee.setMobileno(emp.getMobileno());
				repo.save(existingEmployee);
			}
		} else {
			throw new RuntimeException("Employee with ID " + id +
" not found.");
		}
	}
```

```
}
```

8. REST Controller (API Layer)

```
@RestController
@RequestMapping("/employees")
public class EmployeeController {

      @Autowired
      private EmployeeDao empDao;

      // Save a single employee
      @PostMapping("/save")
    public String saveEmployee(@RequestBody Employee employee) {
        empDao.saveEmployee(employee);
        return "Employee saved successfully!";
    }

      // Find Employee by ID
    @GetMapping("/find/{id}")
    public ResponseEntity<?> findById(@PathVariable int id) {
        Optional<Employee> employee = empDao.findById(id);
        return employee.map(ResponseEntity::ok).orElseGet(() ->
ResponseEntity.status(HttpStatus.NOT_FOUND).body("Employee Not
Found"));
    }

      // Get All Employees
    @GetMapping("/findAll")
    public ResponseEntity<List<Employee>> findAll() {
        return ResponseEntity.ok(empDao.findAll());
    }
```

```
// Delete Employee
    @DeleteMapping("/delete/{id}")
    public ResponseEntity<String> deleteEmployeeById(@PathVariable int
id) {
        try {
            empDao.deleteById(id);
            return ResponseEntity.ok("Employee deleted
successfully.");
        } catch (Exception e) {
            return
ResponseEntity.status(HttpStatus.NOT_FOUND).body("Employee Not
Found");
        }
    }

      // Update Employee
    @PutMapping("/update/{id}")
    public ResponseEntity<String> updateEmployee(@PathVariable int id,
@RequestBody Employee emp) {
        try {
            empDao.update(id, emp);
 return ResponseEntity.ok("Employee updated successfully.");
        } catch (RuntimeException e) {
            return
ResponseEntity.status(HttpStatus.NOT_FOUND).body(e.getMessage());
        }
    }
}
```

9. API Endpoints

Method	Endpoint	Description
POST	/employees/save	Add a new employee
POST	/employees/saveAll	Add multiple employees
GET	/employees/find/{id}	Get employee by ID
GET	/employees/findAll	Get all employees

GET	/employees/count	Get total count of employees
DELETE	/employees/delete/{id}	Delete employee by ID
DELETE	/employees/deleteAll	Delete all employees
PUT	/employees/update/{id}	Update employee details

10. Running the Project

10.1. Using Maven

```
mvn spring-boot:run
```

10.2. Running in IDE

- Open project in **Eclipse/IntelliJ**
- Run `EmployeeCrudApplication.java` as a **Spring Boot Application**

11. Testing the API

Using Postman

- **POST**: http://localhost:8080/employees/save
- **GET**: http://localhost:8080/employees/find/1
- **PUT**: http://localhost:8080/employees/update/1
- **DELETE**: http://localhost:8080/employees/delete/1

12. Conclusion

This **Employee Management CRUD API** is built using **Spring Boot, JPA, and REST principles**. It provides a structured way to manage employee records efficiently.