Employee Management System - Spring Boot CRUD

Overview

This is a Spring Boot application that provides CRUD (Create, Read, Update, Delete) operations for managing employees in a **MySQL database**.
The project uses **Spring Data JPA** for database interactions, **Spring Web** for REST APIs, and **MySQL** as the persistence layer.

Features

- Create a new employee record.
- Get all employees.
- Get a single employee by ID.
- Update employee details.
- Delete employee by ID.
- Fully RESTful API design.
- MySQL database integration.
- Configurable via `application.properties`.

Tech Stack

- **Java 17+** (or compatible version in your setup)
- **Spring Boot 3+**
- **Spring Data JPA**

- **MySQL**
- **Maven**
- **Postman** (for testing)

Database Schema

Table: `employees`

Column	Type	Constraints
id	BIGINT	Primary Key, Auto-Increment
first_name	VARCHAR(255)	NOT NULL
last_name	VARCHAR(255)	NOT NULL
email	VARCHAR(255)	UNIQUE, NOT NULL
department	VARCHAR(255)	NULL allowed

Project Structure

css
CopyEdit

```
src/main/java/com/example/employeecrud
    ├── controller
    │     └── EmployeeController.java
    ├── model
    │     └── Employee.java
    ├── repository
    │     └── EmployeeRepository.java
    ├── service
    │     └── EmployeeService.java
    └── EmployeeCrudApplication.java

src/main/resources
    ├── application.properties
```

`pom.xml`

Configuration

In `src/main/resources/application.properties`:

properties
CopyEdit
```
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/employee_db
spring.datasource.username=root
spring.datasource.password=system
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8
Dialect
```

Entity Class

```
@Entity
@Table(name = "employees")
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "first_name", nullable = false)
    private String firstName;

    @Column(name = "last_name", nullable = false)
    private String lastName;

    @Column(nullable = false, unique = true)
    private String email;

    @Column
    private String department;
    // Getters, setters, constructors...
}
```

Repository

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

Service

```
@Service
public class EmployeeService {
    @Autowired
    private EmployeeRepository employeeRepository;

    public Employee saveEmployee(Employee employee) {
        return employeeRepository.save(employee);
    }

    public List<Employee> getAllEmployees() {
        return employeeRepository.findAll();
    }

    public Optional<Employee> getEmployeeById(Long id) {
        return employeeRepository.findById(id);
    }

    public void deleteEmployee(Long id) {
        employeeRepository.deleteById(id);
    }
}
```

Controller

```
@RestController
@RequestMapping("/api/employees")
public class EmployeeController {
    @Autowired
    private EmployeeService employeeService;

    @PostMapping
    public Employee createEmployee(@RequestBody Employee employee) {
        return employeeService.saveEmployee(employee);
    }

    @GetMapping
    public List<Employee> getAllEmployees() {
        return employeeService.getAllEmployees();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Employee> getEmployeeById(@PathVariable
Long id) {
        return employeeService.getEmployeeById(id)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
    }

    @PutMapping("/{id}")
    public ResponseEntity<Employee> updateEmployee(@PathVariable
Long id, @RequestBody Employee updatedEmployee) {
        return employeeService.getEmployeeById(id).map(employee -> {
            employee.setFirstName(updatedEmployee.getFirstName());
            employee.setLastName(updatedEmployee.getLastName());
            employee.setEmail(updatedEmployee.getEmail());
            employee.setDepartment(updatedEmployee.getDepartment());
            return
ResponseEntity.ok(employeeService.saveEmployee(employee));
        }).orElse(ResponseEntity.notFound().build());
    }
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteEmployee(@PathVariable Long
id) {
        employeeService.deleteEmployee(id);
        return ResponseEntity.noContent().build();
    }
}
```

Testing with Postman

1️⃣ Create Employee (POST)

URL:
`POST http://localhost:8080/api/employees`

Body (JSON):

```
{
    "firstName": "Alice",
    "lastName": "Smith",
    "email": "alice.smith@example.com",
    "department": "IT"
}
```

2️⃣ Get All Employees (GET)

URL:
`GET http://localhost:8080/api/employees`

3️⃣ Get Employee by ID (GET)

URL:
`GET http://localhost:8080/api/employees/1`

4️⃣ Update Employee (PUT)

URL:
`PUT http://localhost:8080/api/employees/1`

Body (JSON):

```json
{
    "firstName": "Alice",
    "lastName": "Johnson",
    "email": "alice.johnson@example.com",
    "department": "Finance"
}
```

5️⃣ Delete Employee (DELETE)

URL:
`DELETE http://localhost:8080/api/employees/1`

Running the Application

1. Start MySQL server and ensure `employee_db` database exists.

2. Update credentials in `application.properties`.
3. Run the Spring Boot application:

   ```
   mvn spring-boot:run
   ```
4. Test APIs using Postman.