✍by Prabhat Kumar

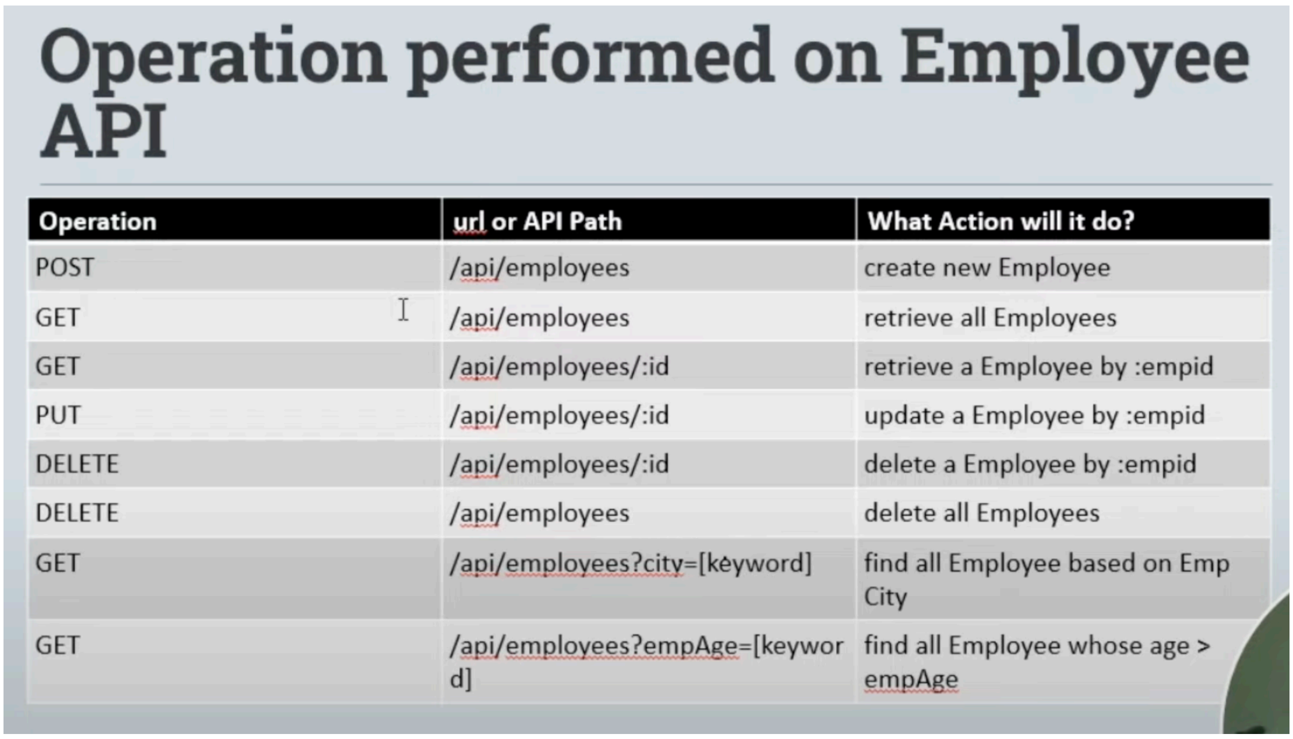
REST-API-CRUD-Operation

Overview of Employee Rest CRUD API

1. empid
2. empName
3. empSalary
4. empAge
5. empCity

Technology Used

1. Java8 (Version should be 8 or greater)
2. Tool : Spring Tool Suite, Postman
3. Spring Boot (Dependency: Spring web, Spring Data JPA, MySql Driver)
4. Created using: Maven



Operation performed on Employee API

Operation	url or API Path	What Action will it do?
POST	/api/employees	create new Employee
GET	/api/employees	retrieve all Employees
GET	/api/employees/:id	retrieve a Employee by :empid
PUT	/api/employees/:id	update a Employee by :empid
DELETE	/api/employees/:id	delete a Employee by :empid
DELETE	/api/employees	delete all Employees
GET	/api/employees?city=[keyword]	find all Employee based on Emp City
GET	/api/employees?empAge=[keyword]	find all Employee whose age > empAge

Work Flow of Employee CRUD REST API

Here's the high-level flow of your project for performing CRUD operations on an Employee entity using Spring Boot and JPA:

1. EmployeeModel (Entity Layer):
- Create an Employee class with attributes: empId, empName, empSalary, empAge, empCity.
- Annotate the class with `@Entity` and define the primary key using `@Id`.
- Use appropriate annotations like `@Column` to map attributes to database columns.

2. EmployeeRepository:
- Create an interface `EmployeeRepository` that extends `JpaRepository<Employee, Long>`.
- This repository interface provides CRUD methods out of the box.

3. EmployeeController:
- Create a class `EmployeeController` with methods for handling CRUD operations.
- Use `@Autowired` to inject `EmployeeRepository`.
- Implement methods for creating, reading, updating, and deleting employees using appropriate HTTP methods (`@PostMapping`, `@GetMapping`, `@PutMapping`, `@DeleteMapping`).

4. Application.Properties:
- Configure database properties like URL, username, password, and dialect.
- Set other Spring Boot properties as needed.

5. Postman:
- Use Postman to test the API endpoints.
- Use different requests (GET, POST, PUT, DELETE) to interact with the CRUD operations on employee data.

6. Run the Application:
- Run the Spring Boot application.
- The application initializes the database, exposes REST endpoints, and listens for incoming requests.

Remember to handle exceptions, validation, and any other necessary details based on your specific project requirements. This is a very simplified overview of the project structure and flow. Actual implementation might require additional configurations and considerations.

Here's a very concise flow of your CRUD operations for the Employee REST API using Postman:

1. Create (POST):
- Use Postman to send a POST request with employee details.
- API endpoint receives data and adds a new employee to the database.

2. Read (GET):
- Use Postman to send a GET request to the API endpoint.
- API retrieves a list of all employees or a specific employee based on empid.

3. Update (PUT/PATCH):
- Use Postman to send a PUT/PATCH request with updated employee details and empid.
- API endpoint finds the employee by empid and updates the provided fields.

4. Delete (DELETE):
- Use Postman to send a DELETE request with empid.
- API endpoint locates the employee by empid and removes them from the database.

Your Postman requests interact with the API endpoints, allowing you to perform CRUD operations on the Employee resource. Make sure your API handles validation, error responses, and proper status codes for a complete implementation.

Application Layer

Package: com.Sharma
Class: EmployeeApicrud2Application.java

Code:

```
package com.Sharma;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class EmployeeApicrud2Application {

	public static void main(String[] args) {
		SpringApplication.run(EmployeeApicrud2Application.class, args);
	}

}
```

Controller Layer

Package: com.Sharma.Controller
Class: EmployeeController.java

Code:

```
package com.Sharma.controller;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
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.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.Sharma.Repository.EmployeeRepository;
import com.Sharma.model.Employee;




@RestController   // use to define the class as a controller in restful web service
@RequestMapping("/api")     // used to map a web request to a method in controller layer/
class
public class EmployeeController {

        @Autowired    // used to inject a bean into class OR used for automatic dependency
injection.
        EmployeeRepository employeeRepository;



        @PostMapping("/employees")
        public String createNewEmployee(@RequestBody Employee employee) {

// here @RequestBody indicates that whatever data we are sending in the body it's(the data
we are sending) mapping should be done properly with the object
                // @RequestBody will bind the request body to a method parameter in a
controller class

                employeeRepository.save(employee);

                return "Employee Created in Database";
        }


        @GetMapping("/employees")
        public ResponseEntity<List<Employee>> getAllEmployees(){
                List<Employee>empList = new ArrayList<>();
                employeeRepository.findAll().forEach(empList::add);
                return new ResponseEntity<List<Employee>>(empList, HttpStatus.OK);
        }


//
//      @GetMapping("/employees {empid}")
//      public ResponseEntity<Employee> getEmployeeById(@PathVariable long empid){
//              Optional<Employee> emp = employeeRepository.findById(empid);
//              if(emp.isPresent()) {
//                      return new ResponseEntity<Employee>(emp.get(),
HttpStatus.FOUND);
//              } else {
//                      return new ResponseEntity<Employee>(HttpStatus.NOT_FOUND);
//              }
//
//      }

  @GetMapping("/employees/{id}")
        public ResponseEntity<Employee> getEmployeeById(@PathVariable Long id) {
           Optional<Employee> employee = employeeRepository.findById(id);
           if (employee.isPresent()) {
              return new ResponseEntity<>(employee.get(), HttpStatus.OK);
           } else {
              return new ResponseEntity<>(HttpStatus.NOT_FOUND);
           }
        }
```

```
	@PutMapping("/employees/{empid}")
	public String updateEmployeeById(@PathVariable long empid, @RequestBody
Employee employee) {
		Optional<Employee> emp = employeeRepository.findById(empid);
		if(emp.isPresent()) {
			Employee existEmp = emp.get();
			existEmp.setEmp_age(employee.getEmp_age());
			existEmp.setEmp_city(employee.getEmp_city());
			existEmp.setEmp_name(employee.getEmp_name());
			existEmp.setEmp_salary(employee.getEmp_salary());
			employeeRepository.save(existEmp);
			return "Employee Details against Id" + " " + empid + " " + "updated";

		} else {
			return "Employee doesn't exist for empid" + " " + empid;
		}
	}

//	@PutMapping("/employees/{id}")
//	public ResponseEntity<String> updateEmployeeById(@PathVariable Long id,
@RequestBody Employee updatedEmployee) {
//	    Optional<Employee> existingEmployee = employeeRepository.findById(id);
//
//	    if (existingEmployee.isPresent()) {
//	        Employee employeeToUpdate = existingEmployee.get();
//
//	        // Update the employee's information with the data from the updatedEmployee
object
//	        employeeToUpdate.setName(updatedEmployee.getName());
//	        employeeToUpdate.setRole(updatedEmployee.getRole());
//	        // Set other properties you want to update
//
//	        employeeRepository.save(employeeToUpdate);
//
//	        return new ResponseEntity<>("Employee Updated in Database",
HttpStatus.OK);
//	    } else {
//	        return new ResponseEntity<>("Employee not found",
HttpStatus.NOT_FOUND);
//	    }
//	}
```

```
	@DeleteMapping("/employees/{empid}")
	public String deleteEmployeeByEmpId(@PathVariable Long empid) {
		employeeRepository.deleteById(empid);
		return "Employee Deleted Successfully";
	}



	@DeleteMapping("/employees")
	public String deleteAllEmployee() {
		employeeRepository.deleteAll();
		return "Employee Deleted Successfully...";
	}
}
```

Entity Layer

Package: com.Sharma.Model
Class: Employee.java

Code:

```
package com.Sharma.model;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;




@Entity
@Table(name="Employee")
public class Employee {


	@Id    // used to mark a field as primary key of entity class
	@GeneratedValue(strategy = GenerationType.IDENTITY)   // @GeneratedValue is
used to specify how primary key value is generated
	private Long empid;



	@Column(name="emp_name")  // @Column is used to specify name & other
attribute in for a field in entity
	private String emp_name;

	@Column(name ="emp_salary")
	private Float emp_salary;

	@Column(name="emp_age")
	private int emp_age;

	@Column(name="emp_city")
	private String emp_city;
	public Long getEmpid() {
		return empid;
	}

//	Generate 'Getter & Setter' and 'Constructor using Fields' then 'toString'  from Source



	public void setEmpid(Long empid) {
		this.empid = empid;
	}

	public String getEmp_name() {
		return emp_name;
	}

	public void setEmp_name(String emp_name) {
		this.emp_name = emp_name;
	}

	public Float getEmp_salary() {
		return emp_salary;
	}

	public void setEmp_salary(Float emp_salary) {
		this.emp_salary = emp_salary;
	}

	public int getEmp_age() {
		return emp_age;
	}

	public void setEmp_age(int emp_age) {
		this.emp_age = emp_age;
	}

	public String getEmp_city() {
		return emp_city;
	}

	public void setEmp_city(String emp_city) {
		this.emp_city = emp_city;
	}
	public Employee(Long empid, String emp_name, Float emp_salary, int emp_age,
String emp_city) {
		super();
		this.empid = empid;
		this.emp_name = emp_name;
		this.emp_salary = emp_salary;
		this.emp_age = emp_age;
		this.emp_city = emp_city;
	}



   public Employee(){

   }

	@Override
	public String toString() {
		return "Employee [empid=" + empid + ", emp_name=" + emp_name + ",
emp_salary=" + emp_salary + ", emp_age="
				+ emp_age + ", emp_city=" + emp_city + "]";
	}




}
```

Repository Layer

Package: com.Sharma.Repository
Class: EmployeeRepository.java

Code:

```
package com.Sharma.Repository;

//import java.util.List;
//import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;

import com.Sharma.model.Employee;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {


//	Home work

//	Optional<List<Employee>> findEmployeeByEmpcity(String emp_city);

}
```

Resources

Application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/EmployeeDB?
                            useSSL=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=Prabhat72
spring.jpa.show-sql=false

# hibernate properties
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect

 # hibernate ddl auto
spring.jpa.hibernate.ddl-auto = none
server.port=8000
```

Pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/
xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>3.0.11-SNAPSHOT</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.sharma</groupId>
	<artifactId>EmployeeAPICRUD2</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>EmployeeAPICRUD2</name>
	<description>EMployee CRUD API using Spring Boot</description>
	<properties>
		<java.version>17</java.version>
	</properties>
	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-jpa</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>

		<dependency>
			<groupId>com.mysql</groupId>
			<artifactId>mysql-connector-j</artifactId>
			<scope>runtime</scope>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>
```

```
        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                        </plugin>
                </plugins>
        </build>
        <repositories>
                <repository>
                        <id>spring-milestones</id>
                        <name>Spring Milestones</name>
                        <url>https://repo.spring.io/milestone</url>
                        <snapshots>
                                <enabled>false</enabled>
                        </snapshots>
                </repository>
                <repository>
                        <id>spring-snapshots</id>
                        <name>Spring Snapshots</name>
                        <url>https://repo.spring.io/snapshot</url>
                        <releases>
                                <enabled>false</enabled>
                        </releases>
                </repository>
        </repositories>
        <pluginRepositories>
                <pluginRepository>
                        <id>spring-milestones</id>
                        <name>Spring Milestones</name>
                        <url>https://repo.spring.io/milestone</url>
                        <snapshots>
                                <enabled>false</enabled>
                        </snapshots>
                </pluginRepository>
               <pluginRepository>
                        <id>spring-snapshots</id>
                        <name>Spring Snapshots</name>
                        <url>https://repo.spring.io/snapshot</url>
                        <releases>
                                <enabled>false</enabled>
                        </releases>
                </pluginRepository>
        </pluginRepositories>
</project>
```

JSON Data :

```
{
	"emp_name": "Kirti",
	"emp_salary": 30000.0,
	"emp_age": 25,
	"emp_city": "Goa"
}
```

SQL Query:

```
create database if not exists EmployeeDB;
use EmployeeDB;

## drop table if exists employee;

CREATE TABLE Employee (
empid bigint NOT NULL AUTO_INCREMENT,
emp_name VARCHAR(50) DEFAULT NULL,
emp_salary float DEFAULT NULL,
emp_age integer DEFAULT NULL,
emp_city VARCHAR(50) DEFAULT NULL,
PRIMARY KEY (empid));


USE EmployeeDB ;
select * from Employee;
```