

Project 1: Employee Management

System Date:

Description:

Create a Spring Boot application to manage employee records. The system should allow adding new employees, fetching details of existing employees, updating employee details, and deleting employees.

Entities Fields:

- Employee: id (Primary Key, auto-generated), name, email, department, salary.

Features:

- Create an Employee: POST request to /employees with employee details in the request body.
- Get All Employees: GET request to /employees.
- Get Employee by ID: GET request to /employees/{id}.
- Update an Employee: PUT request to /employees/{id} with the employee details to be updated in the request body.
- Delete an Employee: DELETE request to /employees/{id}.

Model:

```
package com.example.demo;
import jakarta.persistence.*;
@Entity
public class Model {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    private String name; private
    String email; private String
    department; private double
    salary;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }
}
```

```

public void setEmail(String email) {
    this.email = email;
}

public String getDepartment() {
    return department;
}

public void setDepartment(String department) {
    this.department = department;
}

public double getSalary() {
    return salary;
}

public void setSalary(double salary) {
    this.salary = salary;
}
}

```

Controller:

```

package com.example.demo;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
public class Controller {
    @Autowired
    private Serv serv;

    @GetMapping("/employee")
    public List<Model> display(){
        return serv.get_all();
    }

    @GetMapping("/employee/{id}")
    public Model display(@PathVariable Integer id){
        return serv.get(id);
    }

    @PostMapping("/employee")
    public ResponseEntity<Model> create(@RequestBody Model m){
        return serv.create(m);
    }

    @PutMapping("/employee/{id}")
    public ResponseEntity<Model> update(@RequestBody Model m, @PathVariable Integer id){ return
        serv.update(m, id);
    }

    @DeleteMapping("/employee/{id}")
}

```

```

public String delete(@PathVariable Integer id){
    return serv.delete(id);
}
}

Repo:
package com.example.demo;
import org.springframework.data.jpa.repository.JpaRepository;

public interface Repo extends JpaRepository<Model, Integer> {
}

Service
package com.example.demo;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class Serv {
    @Autowired private
    Repo repo;

    public List<Model> get_all() {
        return repo.findAll();
    }

    public ResponseEntity<Model> create(Model m) {
        repo.save(m);
        return ResponseEntity.ok(m);
    }

    public Model get(Integer id) {
        return repo.findById(id).orElse(null);
    }

    public String delete(Integer id) {
        repo.deleteById(id);
        return "deleted successfully";
    }

    public ResponseEntity<Model> update(Model m, Integer id) {
        m.setId(id);
        repo.save(m);
        return ResponseEntity.ok(m);
    }
}

```

Application Property:

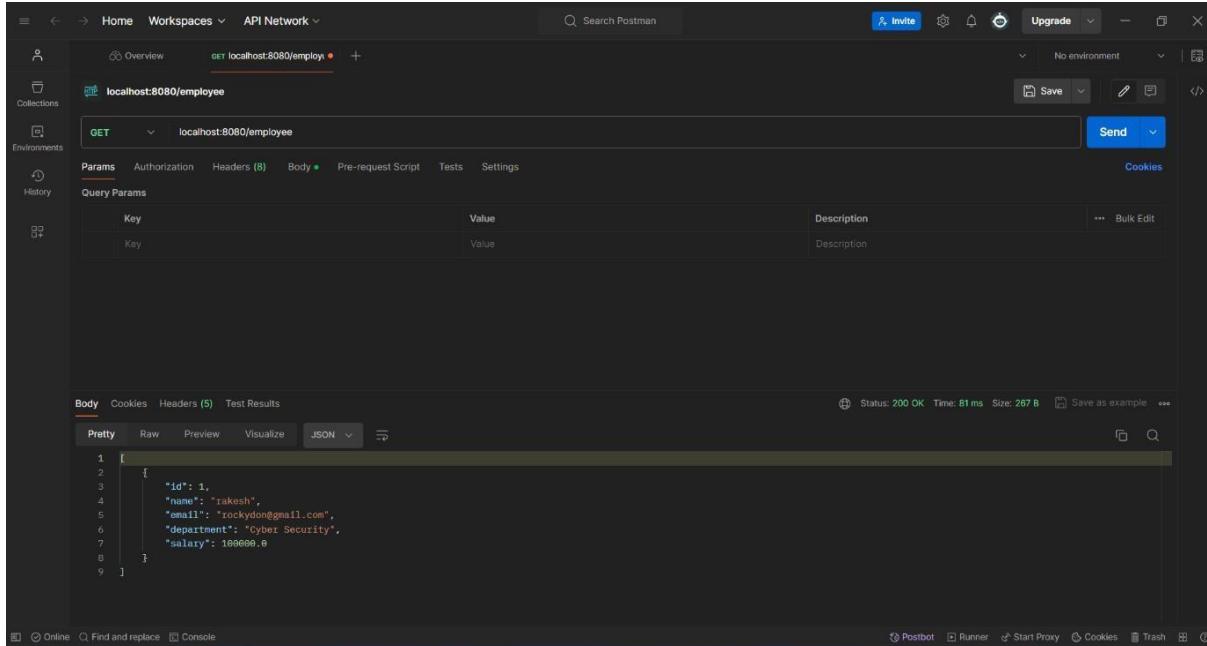
```

spring.application.name=ex-1
spring.datasource.url=jdbc:mysql://localhost:3306/ex1
spring.datasource.username=root

```

```
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.database=mysql
spring.jpa.generate-ddl=true
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

GET:

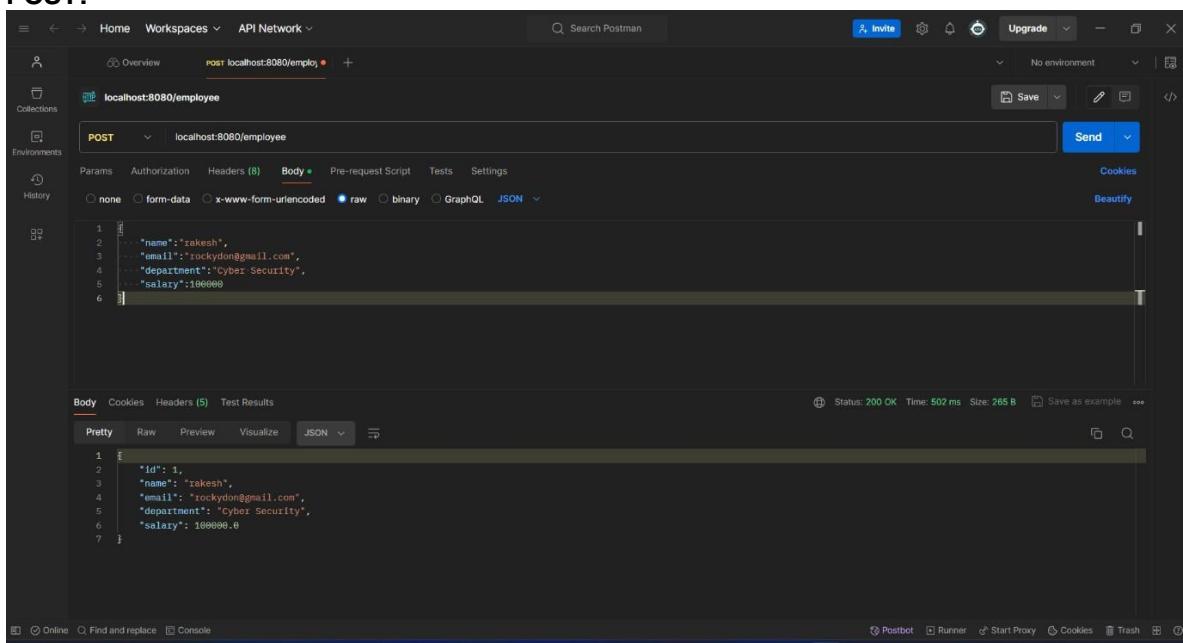


The screenshot shows the Postman interface with a GET request to `localhost:8080/employee`. The response body is a JSON object with the following structure:

```
1 {
2   "id": 1,
3   "name": "rakesh",
4   "email": "rockydon@gmail.com",
5   "department": "Cyber Security",
6   "salary": 100000.0
7 }
```

The status bar at the bottom indicates a `200 OK` status with a time of `81 ms` and a size of `267 B`.

POST:



The screenshot shows the Postman interface with a POST request to `localhost:8080/employee`. The request body is a JSON object with the following structure:

```
1 {
2   "name": "rakesh",
3   "email": "rockydon@gmail.com",
4   "department": "Cyber Security",
5   "salary": 100000
6 }
```

The status bar at the bottom indicates a `200 OK` status with a time of `502 ms` and a size of `265 B`.

PUT:

The screenshot shows the Postman interface with a PUT request to `localhost:8080/employee/1`. The request body is a JSON object:

```
1 {  
2   ... "name": "takesh",  
3   ... "email": "rockydon@gmail.com",  
4   ... "department": "Cyber Security",  
5   ... "salary": 200000  
6 }
```

The response status is 200 OK with a response body:

```
1 {  
2   "id": 1,  
3   "name": "takesh",  
4   "email": "rockydon@gmail.com",  
5   "department": "Cyber Security",  
6   "salary": 200000.0  
7 }
```

DELETE:

The screenshot shows the Postman interface with a DELETE request to `localhost:8080/employee/1`. A query parameter `id` is set to 1. The response status is 200 OK with a response body:

```
1 deleted successfully
```

Database:

The screenshot shows the MySQL Workbench interface. In the Navigator pane, under the SCHEMAS section, there is a schema named 'ex1' which contains a table named 'model'. The table 'model' has four columns: id, department, email, and name, with a salary column added later. A query is run in the Query Grid:

```
1 create database ex1;
2 use ex1;
3 select * from model;
```

The Result Grid shows the following data:

	id	department	email	name	salary
▶	2	Cyber Security	rockyon@gmail.com	rakesh	200000
*					

In the Output pane, the action is recorded as:

#	Time	Action
1	16:00:01	select * from model LIMIT 0,1000

Message: 1 row(s) returned
Duration / Fetch: 0.000 sec / 0.000 sec

Result:

Thus, the application is implemented in Spring boot.

Project 2: Book Library

Management System Date:

Description:

Develop a Spring Boot application to manage a library's book inventory. This system should allow the library to add new books, fetch details about available books, update details on books, and remove books from the inventory.

Entities Fields:

- Book: id (Primary Key, auto-generated), title, author, isbn, publishedYear, genre.

Features:

- Add a Book: POST request to /books with book details in the request body.
- Get All Books: GET request to /books.
- Get Book by ID: GET request to /books/{id}.
- Update a Book: PUT request to /books/{id} with the book details to be updated in the request body.
- Delete a Book: DELETE request to /books/{id}.

Model:

```
package com.example.demo;

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

@Entity
@Table(name="New")
public class Modelclass {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    private String title;
    private String author;
    private int isbn;
    private int publishedYear;
    private String genre; public
    int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
}
```

```

}
public String getAuthor() {
    return author;
}
public void setAuthor(String author) {
    this.author = author;
}
public int getIsbn() {
    return isbn;
}
public void setIsbn(int isbn) {
    this.isbn = isbn;
}
public int getPublishedYear() {
    return publishedYear;
}
public void setPublishedYear(int publishedYear) {
    this.publishedYear = publishedYear;
}
public String getGenre() {
    return genre;
}
public void setGenre(String genre) {
    this.genre = genre;
}
}

```

Controller:

```

package com.example.demo;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired; 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.RestController;

@RestController
public class Controller {
    @Autowired
    service s;
    @GetMapping("/get")
    public List<Modelclass> getdata(){
        return s.getall();
    }
    @PostMapping("/save")
    public String save(@RequestBody Modelclass m) {
        s.add(m);
        return "added";
    }
    @PutMapping("/update/{id}")

```

```

public String update(@PathVariable int id, @RequestBody Modelclass model) {
    s.update(id, model);
    return "Updated";
}

{@DeleteMapping("/delete/{id}")}
public String delete(@PathVariable int id) {
    s.delete(id);
    return "Deleted";
}
}

```

Repo:

```

package com.example.demo;

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

interface Repo extends JpaRepository<Modelclass, Integer> {

}

```

Service:

```

package com.example.demo;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class service {
    @Autowired
    Repo r;
    public String add(Modelclass m) {
        r.save(m);
        return "Added";
    }
    public List<Modelclass> getall(){
        return r.findAll();
    }

    public void update(int id, Modelclass model) { if
(r.existsById(id)) {
        model.setId(id);
        r.save(model);
    }
}

    public void delete(int id) {
        r.deleteById(id);
    }
}

```

```
Application Property: spring.application.name=demo-2  
spring.datasource.url=jdbc:mysql://localhost:3306/lab  
spring.datasource.username=root  
spring.datasource.password=root  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.jpa.database=mysql  
spring.jpa.generate-ddl=true  
spring.jpa.show-sql=true  
spring.jpa.hibernate.ddl-auto=update
```

GET :

The screenshot shows the Postman application interface. At the top, there are navigation links for Home, Workspaces, and Explore, along with a search bar labeled "Search Postman". On the right side, there are buttons for "Sign In" and "Create Account". The main workspace is titled "History" and shows a list of recent requests. A specific request, "GET localhost:8080/get", is selected and expanded. The request details panel shows the method as "GET", the URL as "localhost:8080/get", and the body content type as "application/json". The body content is a JSON object representing a book:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
{
  "id": "03",
  "title": "break",
  "author": "wal",
  "isbn": "936",
  "publishedYear": "2019",
  "genre": "rom"
}
```

Below the request details, there are tabs for Body, Cookies, Headers (5), and Test Results. The Body tab is active, showing the JSON response from the server. The response status is 200 OK, time is 29 ms, and size is 597 B. The response content is identical to the request body, indicating a successful read operation.

POST:

The screenshot shows the Postman application interface. In the top navigation bar, 'localhost:8080/get' is selected. The main workspace displays a POST request to 'localhost:8080/save'. The 'Body' tab is active, showing a JSON payload:

```
1
2
3 {
4     "id": "03",
5     "title": "break",
6     "author": "Wal",
7     "isbn": "536",
8     "publishedYear": "2010",
9     "genre": "se"
10 }
```

The status bar at the bottom indicates a successful response: Status: 200 OK, Time: 68 ms, Size: 168 B.

PUT:

The screenshot shows the Postman application interface. In the top navigation bar, 'PUT localhost:8080/get' is selected. The main workspace displays a PUT request to 'localhost:8080/update/2'. The 'Body' tab is active, showing a JSON payload:

```
1
2
3 {
4     "id": "03",
5     "title": "beza",
6     "author": "en",
7     "isbn": "98",
8     "publishedYear": "200",
9     "genre": "a"
10 }
```

The status bar at the bottom indicates a successful response: Status: 200 OK, Time: 652 ms, Size: 170 B.

DELETE:

The screenshot shows the Postman application interface. The left sidebar displays a history of API requests made to 'localhost:8080'. The main workspace shows a single DELETE request to 'localhost:8080/delete/3'. The 'Body' tab is selected, showing the raw JSON payload: '1'. The 'Headers' tab shows 'Content-Type: application/json'. The 'Tests' tab contains the following script:

```
if (response.status === 200) {  
    console.log('Deleted');  
}
```

The status bar at the bottom right indicates: Status: 200 OK Time: 56 ms Size: 170 B Save Response.

DATABASE:

Result:

Thus, the application is implemented in Spring boot.

Project 3: Staff Administration

Platform Date:

Description:

Create an interactive web platform designed for staff management, enabling users to perform key actions such as creating, viewing, modifying, and removing staff profiles. Each profile should contain details such as Employee ID, full name, division, and earnings.

Key Features:

- A RESTful API that facilitates CRUD (Create, Read, Update, Delete) functionalities for staff profiles.
- Utilize JPA (Java Persistence API) repositories for seamless interaction with a MySQL database.
- Implement basic authentication mechanisms to safeguard the access to API endpoints.

Controller_Staff:

```
package com.gowtham.OneToOneMany;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired; 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.RestController;

public class Controller_Staff {
    @Autowired
    Serve_Staff s;

    @PostMapping("/create")
    public String create(@RequestBody Model_Staff m) {
        s.add(m);
        return "created sucessfully";
    }

    @GetMapping("/get")
    public List<Model_Staff> display(){
        return s.get_all();
    }

    @PutMapping("/update/{id}")
    public String update_user(@PathVariable int id, @RequestBody Model_Staff m){
        m.setEmp_id(id);
    }
}
```

```

        return s.update_user(m);
    }

    @DeleteMapping("/delete/{id}")
    public String delete(@PathVariable int id) {
        s.delete(id);
        return "deleted Sucessfully";
    }
}

```

Model_Staff:

```

package com.gowtham.OneToMany;

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

@Entity
@Table(name="Staff_detils")
public class Model_Staff {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int emp_id;
    private String name; private
    String department; private
    Long Salary; public int
    getEmp_id() {
        return emp_id;
    }
    public void setEmp_id(int emp_id) {
        this.emp_id = emp_id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getDepartment() {
        return department;
    }
    public void setDepartment(String department) {
        this.department = department;
    }
    public Long getSalary() {
        return Salary;
    }
    public void setSalary(Long salary) {
        Salary = salary;
    }
}

```

Repo_Admin:

```
package com.gowtham.OneToOneMany;

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

public interface Repo_Admin extends JpaRepository<Model_Staff,Integer>{

}
```

Serve_Staff:

```
package com.gowtham.OneToOneMany;

import java.util.List;

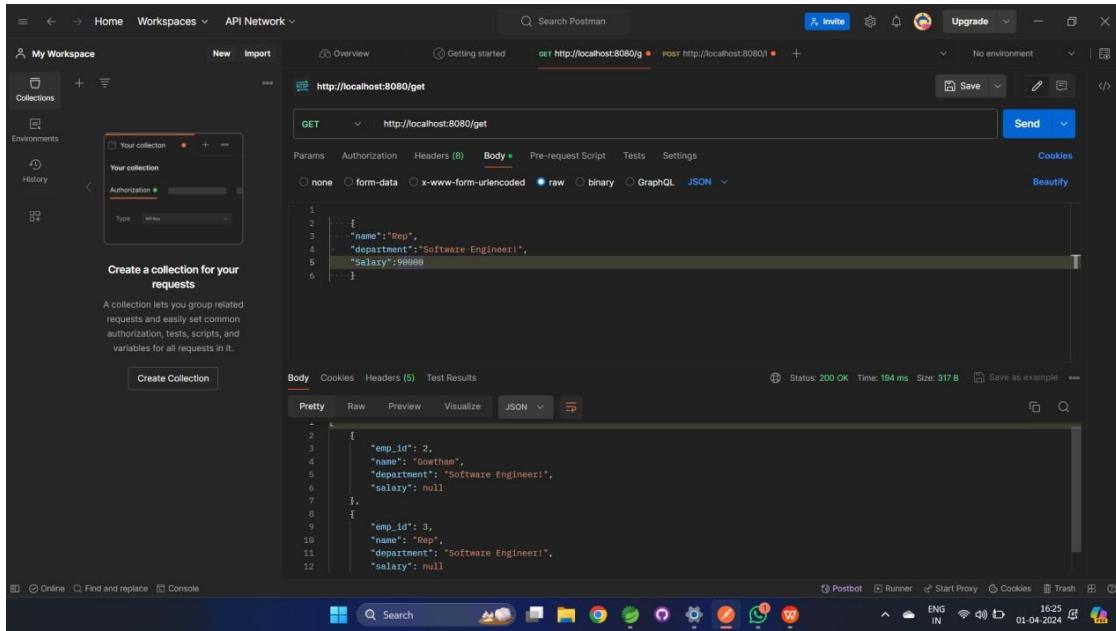
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
@Service
public class Serve_Staff {
    @Autowired
    Repo_Admin r;

    public void add(Model_Staff m)
    {
        r.save(m);
    }
    public List<Model_Staff>get_all(){
        return r.findAll();
    }

    public void delete(int id) {
        r.deleteById(id);
    }

    public String update_user(Model_Staff m){
        r.save(m);
        return "Sucess";
    }
}
```

GET:



My Workspace

Overview Getting started GET http://localhost:8080/get post http://localhost:8080/1 +

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1
2 ...
3 ...
4 ...
5 ...
6 ...
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "emp_id": 2,
4     "name": "Gowtham",
5     "department": "Software Engineer!",
6     "salary": null
7   },
8   {
9     "emp_id": 3,
10    "name": "Raj",
11    "department": "Software Engineer!",
12    "salary": null
13 }
14 ]
```

Status: 200 OK Time: 194 ms Size: 317 B Save as example

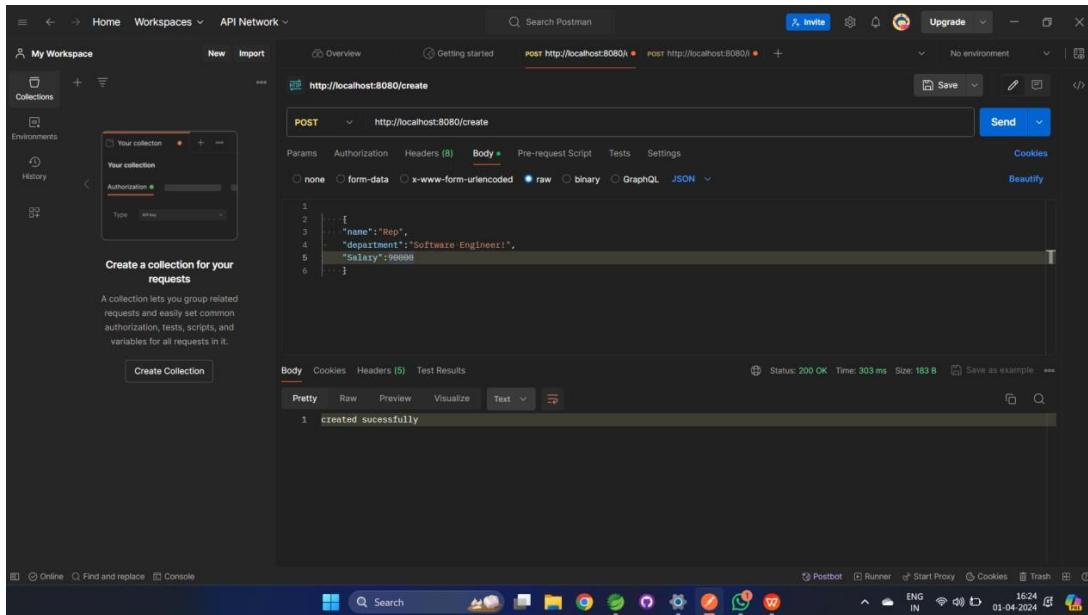
Postbot Runner Start Proxy Cookies Trash

Online Find and replace Console

Search

16:25 IN 01-04-2024

POST:



My Workspace

Overview Getting started POST http://localhost:8080/1 post http://localhost:8080/1 +

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 ...
2 ...
3 ...
4 ...
5 ...
6 ...
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text

```
1 created sucessfully
```

Status: 200 OK Time: 303 ms Size: 183 B Save as example

Postbot Runner Start Proxy Cookies Trash

Online Find and replace Console

Search

16:24 IN 01-04-2024

UPDATE:

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar displays a collection named 'Your collection'. The main workspace shows a 'PUT' request to 'http://localhost:8080/update/3'. The 'Body' tab is selected, showing the following JSON payload:

```
1
2 | ...
3 |   "name": "Rep_Bharath",
4 |   "department": "Software Engineer!",
5 |   "Salary": 90000
6 | ...
```

The 'Test Results' section indicates a successful response: Status: 200 OK, Time: 103 ms, Size: 169 B.

DELETE:

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar displays a collection named 'Your collection'. The main workspace shows a 'DELETE' request to 'http://localhost:8080/delete/3'. The 'Body' tab is selected, showing the same JSON payload as the update request:

```
1
2 | ...
3 |   "name": "Rep_Bharath",
4 |   "department": "Software Engineer!",
5 |   "Salary": 90000
6 | ...
```

The 'Test Results' section indicates a successful response: Status: 200 OK, Time: 80 ms, Size: 183 B. The response body contains the message: 'deleted Successfully'.

DB:

The screenshot shows the MySQL Workbench interface. In the top navigation bar, 'File', 'Edit', 'View', 'Query', 'Database', 'Server', 'Tools', 'Scripting', and 'Help' are visible. The 'Query' tab is selected. The 'Navigator' pane on the left lists 'SCHEMAS' (crud, sakila, staff_admin), 'Tables' (staff_auth, staff_detials), 'Views', 'Stored Procedures', and 'Functions'. The 'Information' pane at the bottom shows 'Schema: staff_admin'. The main area displays a 'Result Grid' for the query 'SELECT * FROM staff_admin.staff_detials;'. The grid has columns: emp_id, salary, department, name. One row is shown: emp_id 50000, salary 100000, department Software Engineer, name Gowtham. Below the grid, the 'Output' pane shows the query executed at 16:30:25 and returning 1 row(s). The system tray at the bottom right shows the date as 01-04-2024.

Project staff Authentication:

Controller_Auth:

```
package com.gowtham.Staff_Auth;

import org.springframework.beans.factory.annotation.Autowired;
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.RequestBody; import
org.springframework.web.bind.annotation.RestController;

import jakarta.transaction.Transactional;

@RestController
public class Controller_Auth {
    @Autowired
    Serve_Auth s;

    @PostMapping("/register")
    public String create(@RequestBody Model_Auth m) {
        s.add(m);
        return "created sucessfully";
    }
    @Transactional
    @GetMapping("/get/{user_name}/{password}")
    public String get_By(@PathVariable String user_name,@PathVariable String password ) { return
        s.get_By(user_name,password);
    }
}
```

```
        }  
    }
```

Model_Auth:

```
package com.gowtham.Staff_Auth;  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
import jakarta.persistence.Table;  
  
@Entity  
@Table(name="Staff_Auth")  
public class Model_Auth {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private int id;  
    private String user;  
    private String pass;  
    public int getId() {  
        return id;  
    }  
    public void setId(int id) {  
        this.id = id;  
    }  
    public String getUser() {  
        return user;  
    }  
    public void setUser(String user) {  
        this.user = user;  
    }  
    public String getPass() {  
        return pass;  
    }  
    public void setPass(String pass) {  
        this.pass= pass;  
    }  
}
```

Repo_Auth:

```
package com.gowtham.Staff_Auth;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
  
public interface Repo_Auth extends JpaRepository<Model_Auth,Integer>{
```

```
    @Query("Select m from Model_Auth m Where m.user = :user_name")
    public Model_Auth get_By(String user_name);
}
```

Serve_Auth:

```
package com.gowtham.Staff_Auth;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class Serve_Auth {
    @Autowired
    Repo_Auth r;

    public void add(Model_Auth m)
    {
        r.save(m);
    }
    public String get_By(String user_name,String password) {
        Model_Auth a=r.get_By(user_name);
        if(a==null) {
            return "user_name does not match";
        }
        else if(a.getPass().equals(password)) {
            return "Login";
        }
        else {
            return "Password is wrong";
        }
    }
}
```

POST:

The screenshot shows the Postman application interface. In the top navigation bar, there are tabs for Home, Workspaces, API Network, Overview, Getting started, and a search bar. Below the navigation is a collection named "Your collection" with an "Authorization" step. The main workspace shows a POST request to "http://localhost:8080/register". The "Body" tab is selected, showing raw JSON input:

```
1
2   ...
3   ...
4   "use1": "Rep",
5   "pass": "Pass"
6   ...
```

Below the body, the response status is 200 OK, Time: 785 ms, Size: 183 B. The response body contains the message "created sucessfully". The bottom of the screen shows the Windows taskbar with various icons.

Get (if user name and password is correct):

The screenshot shows the Postman application interface. In the top navigation bar, there are tabs for Home, Workspaces, API Network, Overview, Getting started, and a search bar. Below the navigation is a collection named "Your collection" with an "Authorization" step. The main workspace shows a GET request to "http://localhost:8080/get/Rep/Pass". The "Body" tab is selected, showing raw JSON input:

```
1
2   ...
3   ...
4   "use1": "Rep",
5   "pass": "Pass"
6   ...
```

Below the body, the response status is 200 OK, Time: 326 ms, Size: 168 B. The response body contains the message "Login". The bottom of the screen shows the Windows taskbar with various icons.

Get (if user name is correct and password is wrong):

The screenshot shows the Postman application interface. In the top navigation bar, the URL is set to `http://localhost:8080/get/Rep/wrong_pass`. The request method is `GET`. The `Body` tab is selected, showing a raw JSON payload:

```
1
2 ...
3   ...
4     "usez": "Rep",
5     "pass": "Pass"
6   ...
7 }
```

Below the request, the response status is `200 OK`, time `22 ms`, and size `181 B`. The response body contains the message `Password is wrong`.

Get (if user name and password also wrong):

The screenshot shows the Postman application interface. In the top navigation bar, the URL is set to `http://localhost:8080/get/undefined/wrong_pass`. The request method is `GET`. The `Body` tab is selected, showing a raw JSON payload:

```
1
2 ...
3   ...
4     "usez": "Rep",
5     "pass": "Pass"
6   ...
7 }
```

Below the request, the response status is `200 OK`, time `18 ms`, and size `188 B`. The response body contains the message `user_name does not match`.

DB :

The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the schema structure, including the **staff_auth** table under the **Tables** section.
- Query Editor:** Displays the query `SELECT * FROM staff_admin.staff_auth;` and its results.
- Result Grid:** Shows the data from the **staff_auth** table:

ID	Pass	User
1	P@ssword	Gowtham
2	Pass	Rep

- Action Output:** Shows the execution log with two entries:

#	Time	Action	Message	Duration / Fetch
1	16:30:25	SELECT * FROM staff_admin.staff_auth LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
2	16:38:25	SELECT * FROM staff_admin.staff_auth LIMIT 0, 1000	2 row(s) returned	0.016 sec / 0.000 sec

Result:

Thus, the application is implemented in Spring boot.

Project 4: Product

Catalog System Date:

Description:

Create a simple product catalog system where users can add new products, view a list of products, update product details, and delete products. Each product should have attributes like ID, name, category, and price.

Key Features:

- CRUD operations for product management.
- Use of JPA to query the database effectively.

Model:

```
package com.example.demo;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

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

    private String name;
    private String category;
    private double price;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getCategory() {
        return category;
    }

    public void setCategory(String category) {
        this.category = category;
    }

    public double getPrice() {
        return price;
    }
}
```

```

        }
    public void setPrice(double price) {
        this.price = price;
    }
}

```

Controller:

```

package com.example.demo;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired; 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;

@RestController
@RequestMapping("/products")
public class ProductController {
    @Autowired
    private ProductService productService;

    @GetMapping
    public List<Product> getAllProducts() {
        return productService.getAllProducts();
    }

    @GetMapping("/{id}")
    public Product getProductById(@PathVariable Long id) {
        return productService.getProductById(id);
    }

    @PostMapping("/insert")
    public Product addProduct(@RequestBody Product product) {
        return productService.addProduct(product);
    }

    @PutMapping("/{id}")
    public Product updateProduct(@PathVariable Long id, @RequestBody Product updatedProduct) { return
        productService.updateProduct(id, updatedProduct);
    }

    @DeleteMapping("/{id}")
    public void deleteProduct(@PathVariable Long id) {
        productService.deleteProduct(id);
    }
}

```

Repo:

```
package com.example.demo;

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

@Repository
public interface ProductRepository extends JpaRepository<Product, Long> { }
```

Service

```
package com.example.demo;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class ProductService {
    @Autowired
    private ProductRepository productRepository;

    public List<Product> getAllProducts() {
        return productRepository.findAll();
    }

    public Product getProductById(Long id) {
        return productRepository.findById(id).orElse(null);
    }

    public Product addProduct(Product product) {
        return productRepository.save(product);
    }

    public Product updateProduct(Long id, Product updatedProduct) { if
        (productRepository.existsById(id)) {
            updatedProduct.setId(id);
            return productRepository.save(updatedProduct);
        }
        return null;
    }

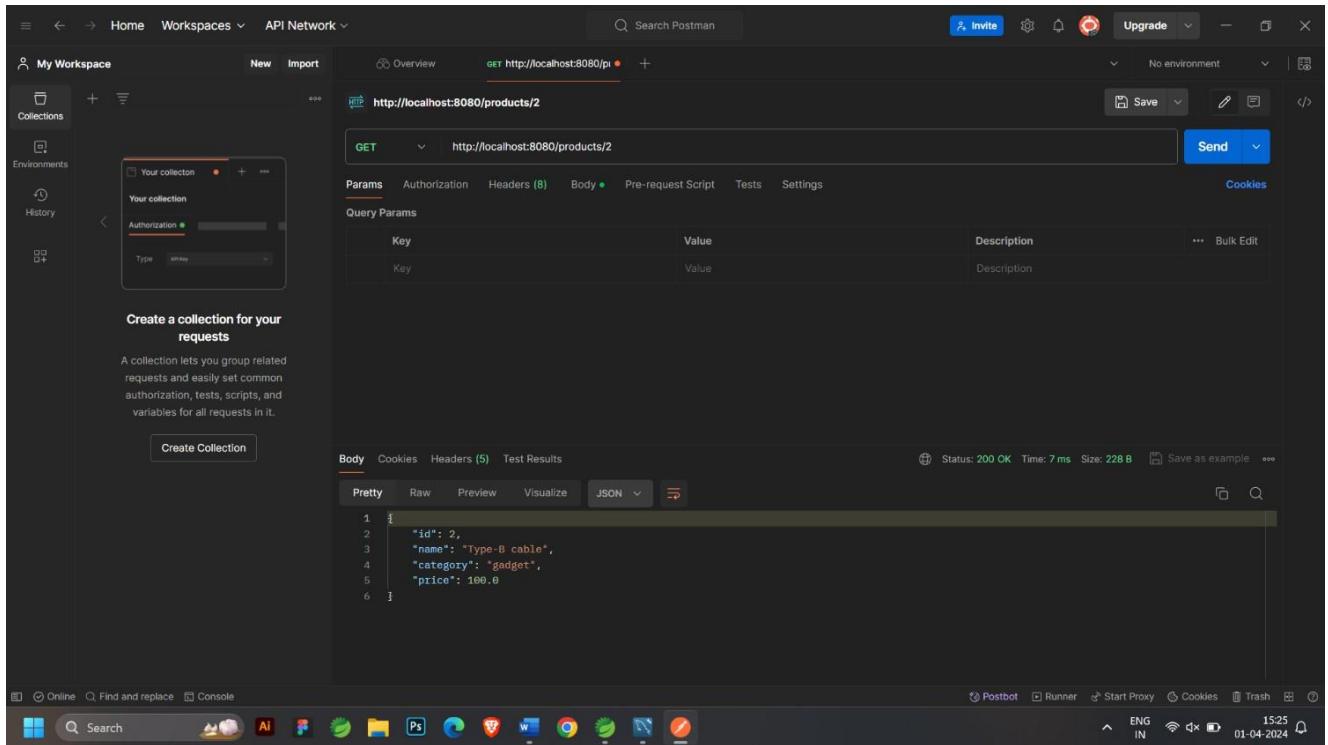
    public void deleteProduct(Long id) {
        productRepository.deleteById(id);
    }
}
```

Application Property:

```
spring.application.name=ProductCatalogSystem
spring.datasource.url=jdbc:mysql://localhost:3306/example9
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.database=mysql
```

```
spring.jpa.generate-ddl=true  
spring.jpa.show-sql=true  
spring.jpa.hibernate.ddl-auto=update
```

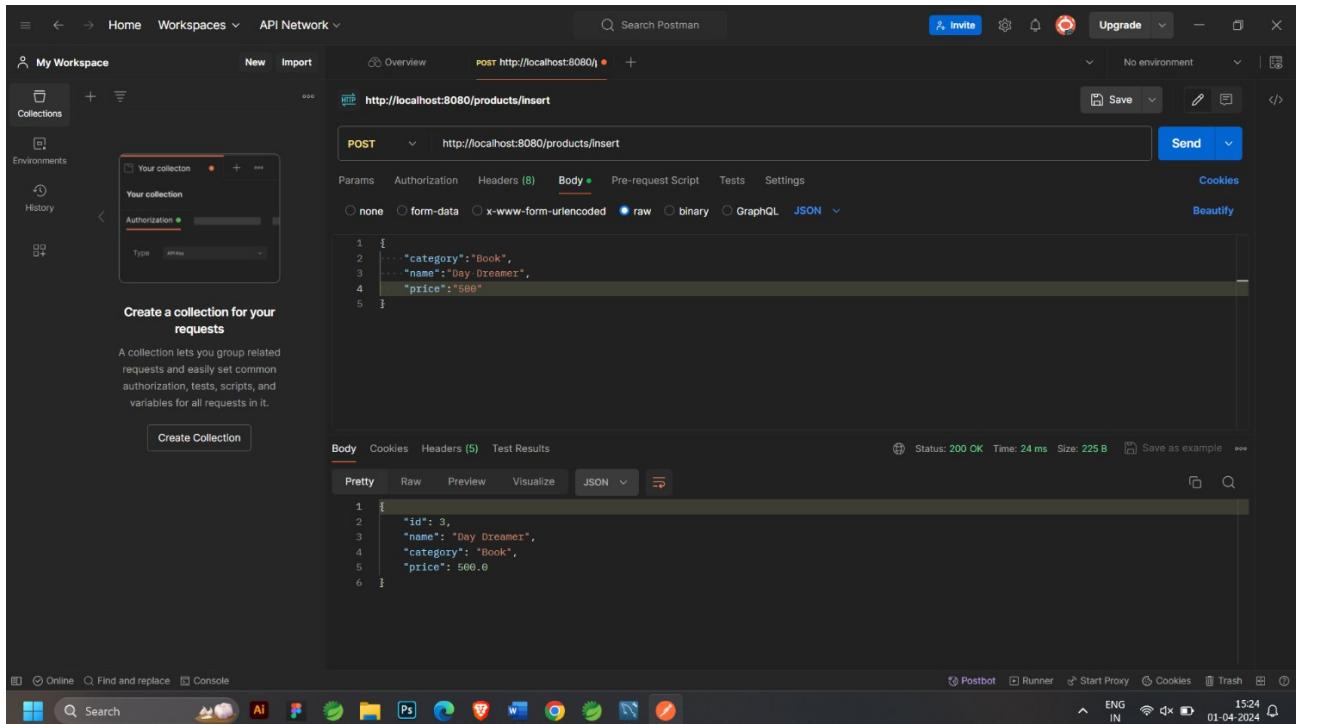
GET :



The screenshot shows the Postman interface with a collection named "Your collection". A GET request is made to `http://localhost:8080/products/2`. The response status is 200 OK, and the JSON body contains the following data:

```
1 {  
2   "id": 2,  
3   "name": "Type-B cable",  
4   "category": "gadget",  
5   "price": 100.0  
6 }
```

POST:



The screenshot shows the Postman interface with a collection named "Your collection". A POST request is made to `http://localhost:8080/products/insert`. The request body is a JSON object:

```
1 {  
2   ... "category": "Book",  
3   ... "name": "Day Dreamer",  
4   ... "price": "500"  
5 }
```

The response status is 200 OK, and the JSON body contains the inserted product details:

```
1 {  
2   "id": 3,  
3   "name": "Day Dreamer",  
4   "category": "Book",  
5   "price": 500.0  
6 }
```

PUT:

The screenshot shows the MySQL Workbench interface. In the top-left, the Navigator pane lists several databases, including 'example9'. The main Query Editor window displays the following SQL query:

```
CREATE DATABASE example9;
USE example9;
SELECT * FROM example9.products;
```

The results grid shows the following data:

ID	Category	Name	Price
2	gadget	Type-B cable	100
3	Book	Day Dreamer	500

In the bottom pane, the 'Action Output' section shows the history of actions taken on the database:

#	Time	Action	Message	Duration / Fetch
1	15:09:46	SELECT * FROM example9.products	0 row(s) returned	0.000 sec / 0.000 sec
2	15:11:06	SELECT * FROM example9.products	1 row(s) returned	0.000 sec / 0.000 sec
3	15:12:19	SELECT * FROM example9.products	2 row(s) returned	0.000 sec / 0.000 sec
4	15:12:46	SELECT * FROM example9.products	2 row(s) returned	0.000 sec / 0.000 sec
5	15:13:03	SELECT * FROM example9.products	2 row(s) returned	0.000 sec / 0.000 sec
6	15:13:25	SELECT * FROM example9.products	1 row(s) returned	0.000 sec / 0.000 sec

DELETE:

DATABASE:

The screenshot shows the Postman application interface. The left sidebar shows 'My Workspace' with a collection named 'Your collection'. The main window shows a DELETE request to 'http://localhost:8080/products/2'. The 'Body' tab contains the following JSON payload:

```
{  
  "category": "Book",  
  "name": "Day Dreamer",  
  "price": "400"  
}
```

The 'Test Results' tab shows the response details:

Body	Cookies	Headers [4]	Test Results
Pretty	Raw	Preview	Visualize

The status bar at the bottom indicates 'Status: 200 OK'.

Result:

Thus, the application is implemented in Spring boot.

Project 5: Online Recipe

Repository Date:

Objective: Develop a Spring Boot application to manage a digital collection of recipes. The application should enable users to execute basic CRUD (Create, Read, Update, Delete) operations on recipes, such as adding new recipes, updating existing recipe details, deleting recipes, and retrieving a list of all recipes or a single recipe by its ID.

Entities Fields:

- Recipe (id, name, chef, yearOfCreation, cuisineType)

Endpoints:

- POST /recipes: Add a new recipe
- GET /recipes: Retrieve all recipes
- GET /recipes/{id}: Fetch a recipe by ID
- PUT /recipes/{id}: Update details of a recipe
- DELETE /recipes/{id}: Remove a recipe by ID

Recipe.java: (Entity)

```
package com.example.demo; import
jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import java.util.Date;

@Entity
public class Recipe {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String chef;
    private Date yearOfCreation;
    private String cuisineType;

    public Recipe() {
    }

    public Recipe(String name, String chef, Date yearOfCreation, String cuisineType) { this.name
        = name;
        this.chef = chef;
        this.yearOfCreation = yearOfCreation;
        this.cuisineType = cuisineType;
    }

    public Long getId() {
        return id;
    }
}
```

```

public void setId(Long id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getChef() {
    return chef;
}

public void setChef(String chef) {
    this.chef = chef;
}

public Date getYearOfCreation() {
    return yearOfCreation;
}

public void setYearOfCreation(Date yearOfCreation) {
    this.yearOfCreation = yearOfCreation;
}

public String getCuisineType() {
    return cuisineType;
}

public void setCuisineType(String cuisineType) {
    this.cuisineType = cuisineType;
}
}

```

RecipeController.java :

```

package com.example.demo;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

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

@RestController
@RequestMapping("/recipes")
public class RecipeController {
    @Autowired
    private RecipeRepository recipeRepository;

    @PostMapping("/post")
    public Recipe addRecipe(@RequestBody Recipe recipe) {
        return recipeRepository.save(recipe);
    }
}

```

```

    }

    @GetMapping("/{id}")
    public Optional<Recipe> getRecipeById(@PathVariable Long id) {
        return recipeRepository.findById(id);
    }

    @PutMapping("/{id}")
    public Recipe updateRecipe(@PathVariable Long id, @RequestBody Recipe updatedRecipe) {
        updatedRecipe.setId(id); // Ensure ID consistency
        return recipeRepository.save(updatedRecipe);
    }

    @DeleteMapping("/{id}")
    public String deleteRecipe(@PathVariable Long id) {
        recipeRepository.deleteById(id);
        return "Value Deleted";
    }
}

```

RecipeRepository.java:

```

package com.example.demo;

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

public interface RecipeRepository extends JpaRepository<Recipe, Long> {
}

```

Application.properties:

```

spring.application.name=Project5
spring.datasource.url=jdbc:mysql://localhost:3306/Project5
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.database=mysql
spring.jpa.generate-ddl=true
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update

```

POST:

The screenshot shows the Postman application interface. The left sidebar displays 'My Workspace' with a collection named 'My first collection'. The main workspace shows a POST request to 'localhost:8080/recipes/post'. The request body is set to 'raw' JSON and contains the following data:

```
1 {
2   "name": "Lasagna",
3   "chef": "Massimo DiCaprio",
4   ... "yearOfCreation": "2004-05-10",
5   "cuisineType": "Italian"
6 }
```

The response status is 200 OK with a time of 17 ms and a size of 287 B. The response body is identical to the request body.

GET:

The screenshot shows the Postman application interface. The left sidebar displays 'My Workspace' with a collection named 'My first collection'. The main workspace shows a GET request to 'localhost:8080/recipes/1'. The request body is set to 'raw' JSON and contains the following data:

```
1 {
2   "name": "Lasagna",
3   "chef": "Massimo DiCaprio",
4   ... "yearOfCreation": "2004-05-10",
5   "cuisineType": "Italian"
6 }
```

The response status is 200 OK with a time of 14 ms and a size of 287 B. The response body is identical to the request body.

PUT:

The screenshot shows the Postman application interface. The left sidebar displays 'My Workspace' with a collection named 'My first collection'. The main workspace shows a PUT request to 'localhost:8080/recipes/1'. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2   "name": "Lasagna",
3   "label": "Italian Dish",
4   "yearOfCreation": "2004-06-18",
5   "cuisineType": "Italian"
6 }
```

The 'Test Results' tab shows a successful response with status 200 OK, time 14 ms, and size 285 B. The bottom status bar indicates the system is online and shows the date 01-04-2024.

DELETE:

The screenshot shows the Postman application interface. The left sidebar displays 'My Workspace' with a collection named 'My first collection'. The main workspace shows a DELETE request to 'localhost:8080/recipes/1'. The 'Body' tab is selected, showing a JSON payload identical to the PUT request:

```
1 {
2   "name": "Lasagna",
3   "label": "Italian Dish",
4   "yearOfCreation": "2004-06-18",
5   "cuisineType": "Italian"
6 }
```

The 'Test Results' tab shows a successful response with status 200 OK, time 13 ms, and size 177 B. The bottom status bar indicates the system is online and shows the date 01-04-2024.

DATABASE:

The screenshot shows the MySQL Workbench interface. In the top navigation bar, 'File', 'Edit', 'View', 'Query', 'Database', 'Server', 'Tools', 'Scripting', and 'HELP' are visible. The 'Navigator' pane on the left lists 'Schemas' (fries, grocery, project5, temp), 'Tables' (chef, recipe), and other objects like 'Views', 'Stored Procedures', 'Functions', 'Logs', 'sys', 'tarsoen', 'thanh', 'thanh1', 'thanh2', and 'world'. The 'SQL Editor' tab is selected, displaying the query: 'SELECT * FROM projects.recipe;'. Below the editor is a 'Result Grid' showing one row of data from the 'recipe' table:

ID	Chef	Cuisine Type	Name	Year of Creation
2	The...	Italian	Laza...	2004-05-10 05:00:00

The 'Information' pane on the right displays the 'Table: recipe' details, including columns: id (bigint), chef (varchar(25)), cuisine_type (varchar(25)), name (varchar(25)), and year_of_creation (varchar(25)). The 'Object Info' and 'Session' tabs are also visible. The bottom status bar shows the date as 01-04-2020 and time as 18:00.

Result:

Thus, the application is implemented in Spring boot.

Project 6: Customer Relationship

Management Tool Date:

Objective: Develop a Spring Boot application designed to streamline customer management processes for a growing business. This application should offer capabilities to register new customers, update customer profiles, delete customer records, and fetch detailed information about a single customer or an overview of all customers.

Entities Fields:

Customer (id, firstName, lastName, email, loyaltyPoints, address)

Key Features:

- POST /customers: Register a new customer
- GET /customers: Retrieve a list of all customers
- GET /customers/{id}: Access details of a specific customer by ID
- PUT /customers/{id}: Update a customer's profile
- DELETE /customers/{id}: Delete a customer record

Additional Features to Consider:

- Implement pagination and sorting for displaying customers
- Ensure robust validation for all data inputs

Model Class:

```
package com.example.Exercise6;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
@Entity
@Table(name="Customer")

public class Model {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    private String firstname;
    private String lastname;
    private String email;
    private int loyaltyPoints;
    private String address;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
}
```

```

    }
    public String getFirstname() {
        return firstname;
    }
    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }
    public String getLastname() {
        return lastname;
    }
    public void setLastname(String lastname) { this.lastname
        = lastname;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public int getLoyaltyPoints() {
        return loyaltyPoints;
    }
    public void setLoyaltyPoints(int loyaltyPoints) {
        this.loyaltyPoints = loyaltyPoints;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
}

}

```

Controller class:

```

package com.example.Exercise6;

import java.util.List;

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;

```

*@RestController
@RequestMapping("/customers")*

```

public class Controller {
    @Autowired
    private serv customerService;

    @PostMapping("/create")
    public ResponseEntity<Model> createCustomer( @RequestBody Model customer) {
        Model createdCustomer = customerService.createCustomer(customer);
        return new ResponseEntity<>(createdCustomer, HttpStatus.CREATED);
    }

    @GetMapping("/get")
    public ResponseEntity<List<Model>> getAllCustomers() {
        List<Model> customers = customerService.getAllCustomers();
        return new ResponseEntity<>(customers, HttpStatus.OK);
    }

    @GetMapping("/{id}")
    public ResponseEntity<Model> getCustomerById( @PathVariable int id) { Model
        customer = customerService.getCustomerById(id);
        if (customer != null) {
            return new ResponseEntity<>(customer, HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }

    @PutMapping("/{id}")
    public ResponseEntity<Model> updateCustomer( @PathVariable int id, @RequestBody Model updatedCustomer) {
        Model customer = customerService.updateCustomer(id, updatedCustomer); if
        (customer != null) {
            return new ResponseEntity<>(customer, HttpStatus.OK);
        } else {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteCustomer( @PathVariable int id) {
        customerService.deleteCustomer(id);
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    }
}

```

Service Class:

```

package com.example.Exercise6;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

```

```

@Service
public class serv {
    @Autowired
    Repo customerRepo;
    public Model createCustomer(Model customer) { return
        customerRepo.save(customer);
    }
    public List<Model> getAllCustomers(){
        return customerRepo.findAll();
    }

    public Model getCustomerById(int id) { return
        customerRepo.findById(id).orElse(null);
    }
    public Model updateCustomer(int id, Model updatedCustomer) {
        Model existingCustomer = customerRepo.findById(id).orElseGet(null); if(existingCustomer
        != null) {
            return customerRepo.save(existingCustomer);
        }
        return null;
    }
    public void deleteCustomer(int id) {
        customerRepo.deleteById(id);
    }
}

```

Repo Interface:

```

package com.example.Exercise6;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository; @Repository
public interface Repo extends JpaRepository<Model,Integer>{

}

```

application.properties:

```

spring.application.name=Exercise6 spring.datasource.url=
jdbc:mysql://localhost:3306/ex6
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.database= mysql
spring.jpa.generate-ddl=true
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto= update

```

PostMapping:

The screenshot shows the Postman interface. At the top, it says "localhost:8080/customers/create". Below that, under "Body", there is a JSON payload:

```
1 {  
2   "firstname": "Rohan",  
3   "lastname": "ram",  
4   "email": "example507@gmail.com",  
5   "loyaltyPoints": 6,  
6   "address": "Skcet"  
7 }  
8 }
```

At the bottom, the response is shown with status 201 Created, time 24 ms, and size 281 B.

Database:

The screenshot shows MySQL Workbench. In the left sidebar, under "SCHEMAS", the "customer" schema is selected. In the main area, a query window displays the results of the following SQL statement:

```
1 * +-----+-----+-----+-----+-----+  
2 | id  | address| email | firstame| lastname| loyalty_points|  
3 +-----+-----+-----+-----+-----+-----+  
4 | 1   | Skcet  | example123@gmail.com | tree  | harsh  | 9           |  
5 | 2   | Skcet  | example456@gmail.com | praveen | kumar  | 12          |  
6 | 3   | Skcet  | example46@gmail.com  | Raj    | kumar  | 34          |  
7 | 4   | Skcet  | example22@gmail.com  | Krishna | kumar  | 22          |  
8 | 5   | Skcet  | example506@gmail.com | Ravi   | raj    | 7           |  
9 | 6   | Skcet  | example507@gmail.com | Rohan  | ram    | 6           |  
10|     | Skcet  |                         |        |        |             |
```

Below the result grid, the "Output" pane shows the execution history of the query:

#	Time	Action	Message	Duration / Fetch
9	12:25:30	SELECT * FROM pr.adt LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
10	12:25:32	SELECT * FROM pruser LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
11	12:26:05	SELECT * FROM pr.adt LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
12	15:53:14	create database ex6	1 rows affected	0.031 sec
13	16:07:42	SELECT * FROM ex6.customer LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
14	16:12:17	SELECT * FROM ex6.customer LIMIT 0, 1000	6 row(s) returned	0.016 sec / 0.000 sec

GetMapping:

The screenshot shows the Postman interface with a collection named "My Workspace". A GET request is being made to `localhost:8080/customers/get`. The response status is 200 OK, time 18 ms, size 853 B. The response body is a JSON array containing two customer objects:

```
[{"id": 1, "firstname": "sree", "lastname": "harish", "email": "example123@gmail.com", "loyaltyPoints": 9, "address": "Skcet"}, {"id": 2, "firstname": "praveen", "lastname": "kumar", "email": "example456@gmail.com"}]
```

PutMapping:

The screenshot shows the Postman interface with a collection named "My Workspace". A PUT request is being made to `localhost:8080/customers/3`. The response status is 200 OK, time 24 ms, size 276 B. The response body is a JSON object representing the updated customer:

```
{"id": 3, "firstname": "Raj", "lastname": "kumar", "email": "example456@gmail.com", "loyaltyPoints": 34, "address": "Skcet"}
```

DeleteMapping:

The screenshot shows the Postman interface. The URL is `localhost:8080/customers/3`. The method is set to `DELETE`. The body is a JSON object with `id: 3` and `firstname: "Ashok"`. The response status is `204 No Content`.

After Deleting :

The screenshot shows the MySQL Workbench interface. The schema is `k7`. The `customer` table is selected. A query `SELECT * FROM ex6.customer` is run, showing the following data:

	<code>id</code>	<code>address</code>	<code>email</code>	<code>firstname</code>	<code>lastname</code>	<code>loyalty_points</code>
1	Sketet		example123@gmail.com	sree	harish	9
2	Sketet		example456@gmail.com	praveen	kumar	12
3	Sketet		example22@gmail.com	Krishna	kumar	22
4	Sketet		example506@gmail.com	Ravi	raj	7
5	Sketet		example507@gmail.com	Rohan	ram	6
6	Sketet					

Result:

Thus, the application is implemented in Spring boot.

Project 7: Campus Club

Management System Date:

Objective: Develop a web-based application to manage member information within various clubs at a college campus.

Entities and Relationships:

Member - Represents the members in various clubs on campus.

Attributes: id (Primary Key), name, email, club (Many-to-One relationship with Club entity)

Club - Represents the different clubs available on campus.

Attributes: id (Primary Key), name, members (One-to-Many relationship with Member entity)

Relational Mapping:

The Member entity has a Many-to-One relationship with the Club entity, indicating that multiple members can be part of one club.

Basic Operations:

- Create, Read, Update, and Delete members.
- Create, Read, Update, and Delete clubs.
- Enroll members into a club.

Member.java:

```
package com.example.pipe; import
jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
@Entity
public class Member {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;
    @ManyToOne
    @JoinColumn(name = "club_id")
    private Club club;
    public Club getClub() {
        return club;
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
```

```

        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public void setClub(Club club) {
        this.club = club;
    }
}
// Constructors, getters, and setters
}

```

Club.java:

```

package com.example.pipe;

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

import jakarta.persistence.CascadeType;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.OneToMany;

@Entity
public class Club {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    @OneToMany(mappedBy = "club", cascade = CascadeType.ALL)
    private List<Member> members = new ArrayList<>();
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public List<Member> getMembers() {
        return members;
    }
}

```

```
    public void setMembers(List<Member> members) {
        this.members = members;
    }
}
```

MemberService.java:

```
package com.example.pipe;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class MemberService {
    @Autowired
    private MemberRepository memberRepository;

    public List<Member> getAllMembers() {
        return memberRepository.findAll();
    }

    // Other CRUD operations for members
}
```

ClubService.java:

```
package com.example.pipe;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class ClubService {
    @Autowired
    private ClubRepository clubRepository;

    public List<Club> getAllClubs() {
        return clubRepository.findAll();
    }

    // Other CRUD operations for clubs

    public void enrollMember(Long clubId, Member member) {
        Club club = clubRepository.findById(clubId).orElseThrow(() -> new RuntimeException("Club not found"));
        member.setClub(club);
        club.getMembers().add(member);
        clubRepository.save(club);
    }
}
```

MemberController.java:

```
package com.example.pipe;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
```

```

import org.springframework.web.bind.annotation.GetMapping; import
org.springframework.web.bind.annotation.RequestMapping; import
org.springframework.web.bind.annotation.RestController;
@RestController
@RequestMapping("/api/members")
public class MemberController {
    @Autowired
    private MemberService memberService;
    @GetMapping
    public List<Member> getAllMembers() {
        return memberService.getAllMembers();
    }
    // Other CRUD endpoints for members
}
ClubController.java:
package com.example.pipe;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired; 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.RequestBody; import
org.springframework.web.bind.annotation.RequestMapping; import
org.springframework.web.bind.annotation.RestController;
@RestController
@RequestMapping("/api/clubs")
public class ClubController {
    @Autowired
    private ClubService clubService;
    @GetMapping
    public List<Club> getAllClubs() {
        return clubService.getAllClubs();
    }
    @PostMapping("/{clubId}/enroll")
    public void enrollMember(@PathVariable Long clubId, @RequestBody Member member) {
        clubService.enrollMember(clubId, member);
    }
    // Other CRUD endpoints for clubs
}
MemberRepository.java:
package com.example.pipe;
import org.springframework.data.jpa.repository.JpaRepository;
public interface MemberRepository extends JpaRepository<Member, Long> {
}
ClubRepository.java:
package com.example.pipe;

import org.springframework.data.jpa.repository.JpaRepository;
public interface ClubRepository extends JpaRepository<Club, Long> {
}
Application.properties:
spring.application.name=lab
spring.datasource.url= jdbc:mysql://localhost:3306/modellab
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

```

```

spring.jpa.database= mysql
spring.jpa.generate-ddl=true
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto= update

```

OUTPUT:

The screenshot shows the MySQL Workbench interface with the 'Schemas' tree on the left. A context menu is open over the 'member' table in the 'club' schema, with the 'Apply SQL Script' option selected. The 'Review SQL Script' dialog is displayed, showing the following SQL code:

```

1   INSERT INTO `modellab`.`member` ('id','email','name','club_id') VALUES ('1','surya','surya','50')
2   INSERT INTO `modellab`.`member` ('id','email','name','club_id') VALUES ('1','praj...','praga...','50')
3   INSERT INTO `modellab`.`member` ('id','email','name','club_id') VALUES ('1','mrthy...','mrthy...','50')
4   INSERT INTO `modellab`.`member` ('id','email','name','club_id') VALUES ('1','praj...','praj...','50')
5   INSERT INTO `modellab`.`member` ('id','email','name','club_id') VALUES ('1','udh...','udh...','50')

```

The 'Result Grid' shows the following data:

	id	email	name	club_id
101	surya	surya	50	
102	praj...	praga...	50	
103	mrthy...	mrthy...	50	
104	praj...	praj...	50	
105	udh...	udh...	50	

The 'Output' pane shows the following log entries:

- 12 15:20:27 use labexercise
- 13 15:20:40 show tables
- 14 15:21:13 use club
- 15 15:23:35 create database modellab
- 16 15:24:31 SELECT * FROM modellab.club LIMIT 0, 1000
- 17 15:24:34 SELECT * FROM modellab.member LIMIT 0, 1000

The 'Duration / Fetch' column indicates execution times for each query.

The screenshot shows the MySQL Workbench interface with the 'Schemas' tree on the left. A context menu is open over the 'member' table in the 'club' schema, with the 'SELECT * FROM club.member' option selected. The results are displayed in the 'Result Grid'.

	id	email	name	club_id
102	praj...	praga	502	
103	mrthy...	mrthy	501	
104	praj...	prajeth	502	
105	udh...	udh...	502	

MySQL Workbench

Local instance MySQL83 (digit) - W.x Local instance MySQL83 (lab...)

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

- Tables Fetching...
- Views Fetching...
- Stored Procedures Fetching...
- Functions Fetching...
- lab_1
 - Tables Fetching...
 - Views Fetching...
 - Stored Procedures Fetching...
 - Functions Fetching...
- lab
 - Tables Fetching...
 - Views Fetching...
 - Stored Procedures Fetching...
 - Functions Fetching...
- modellab
 - Tables
 - club
 - member
 - Views
 - Stored Procedures
 - Functions
 - onetomany
 - query1
 - query2
 - sakila
 - sem
 - sample1
 - sample2
 - sample3
 - sample4
 - simple1
 - simple2

club - Table club member member club member club SQLAdditions

1 • SELECT * FROM modellab.club;

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Result Grid | Filter Rows: | Edit | Export/Imports: | Write | Result Grid

	id	name
501	beni	
502	cyber	
503	hock...	
504	hoss	

club 1 × Output

Action Output

#	Time	Action	Message	Duration / Fetch
24	15:28:17	SELECT * FROM modellab.club LIMIT 0, 1000	0 row(s) returned	0.016 sec / 0.000 sec
25	15:28:20	SELECT * FROM modellab.member LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
26	15:28:41	INSERT INTO modellab.member ('id', 'email', 'name', 'club_id') VALUES ('101', 'surya@gmail.com', 'surya', '1')	1 row(s) inserted	0.000 sec / 0.000 sec
27	15:29:03	INSERT INTO modellab.member ('id', 'email', 'name', 'club_id') VALUES ('101', 'surya@gmail.com', 'surya', '1')	1 row(s) inserted	0.000 sec / 0.000 sec
28	15:29:53	INSERT INTO modellab.member ('id', 'email', 'name', 'club_id') VALUES ('101', 'surya@gmail.com', 'surya', '1')	1 row(s) inserted	0.000 sec / 0.000 sec
29	15:30:12	SELECT * FROM modellab.club LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
30	15:31:21	SELECT * FROM modellab.member LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

Object Info Session

MySQL Workbench

Local instance MySQL83 (digit) - W.x Local instance MySQL83 (lab...)

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

- Tables Fetching...
- Views Fetching...
- Stored Procedures Fetching...
- Functions Fetching...
- lab_1
 - Tables Fetching...
 - Views Fetching...
 - Stored Procedures Fetching...
 - Functions Fetching...
- lab
 - Tables Fetching...
 - Views Fetching...
 - Stored Procedures Fetching...
 - Functions Fetching...
- modellab
 - Tables
 - club
 - member
 - Views
 - Stored Procedures
 - Functions
 - onetomany
 - query1
 - query2
 - sakila
 - sem
 - sample1
 - sample2
 - sample3
 - sample4
 - simple1
 - simple2

member - Table club member member club member club SQLAdditions

1 • SELECT * FROM modellab.member;

Apply SQL Script to Database

Review SQL Script

Apply SQL Script

Applying SQL script to the database

The following tasks will now be executed. Please monitor the execution. Press Show Logs to see the execution logs.

Execute SQL Statements

SQL script was successfully applied to the database.

Result Grid | Filter Rows: | Edit | Export/Imports: | Write | Result Grid

	id	email	name	club_id
102	praveen	praveen	50	1
103	prathyush	prathyush	50	1
104	preeti	preeti	50	1
105	udhaya	udhaya	50	1

member 1 × Output

Action Output

#	Time	Action	Message	Duration / Fetch
25	15:28:20	SELECT * FROM modellab.member LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
26	15:28:41	INSERT INTO modellab.member ('id', 'email', 'name', 'club_id') VALUES ('101', 'surya@gmail.com', 'surya', '1')	1 row(s) inserted	0.000 sec / 0.000 sec
27	15:29:03	INSERT INTO modellab.member ('id', 'email', 'name', 'club_id') VALUES ('101', 'surya@gmail.com', 'surya', '1')	1 row(s) inserted	0.000 sec / 0.000 sec
28	15:29:53	INSERT INTO modellab.member ('id', 'email', 'name', 'club_id') VALUES ('101', 'surya@gmail.com', 'surya', '1')	1 row(s) inserted	0.000 sec / 0.000 sec
29	15:30:12	SELECT * FROM modellab.club LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
30	15:31:21	SELECT * FROM modellab.member LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

Object Info Session

Result:

Thus, the application is implemented in Spring boot.

Project 8: Patient and Appointment

Management System Date:

Objective: Develop a comprehensive system to manage patients and their appointments in a medical facility.

Entities and Relationships:

Patient - Represents patients visiting the medical facility.

Attributes: patientId (Primary Key), name, Date_of_appointment, contactNumber, doctor (Many-to-One relationship with Doctor entity), issue

Doctor - Represents doctors providing medical services.

Attributes: doctorId (Primary Key), name, specialty, appointments (One-to-Many relationship with Patient entity through appointments)

Appointment - Represents appointments between patients and doctors.

Attributes: appointmentId (Primary Key), date, time, patient (Many-to-One relationship with Patient entity), doctor (Many-to-One relationship with Doctor entity)

Relational Mapping:

The Appointment entity has a Many-to-One relationship with both the Patient and Doctor entities, indicating that a patient can have multiple appointments with possibly different doctors, and a doctor can have appointments with multiple patients.

Basic Operations:

- Create, Read, Update, and Delete patients.
- Create, Read, Update, and Delete doctors.
- Schedule, reschedule, and cancel appointments.

MODEL CLASS:

1) Patient.java

```
package com.example.PatientManagementSystem;
```

```
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Patient {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long patientId;
```

```

private String name; private
String dateOfBirth;
private String contactNumber;
private String issue;
    public Long getPatientId() {
        return patientId;
    }
    public void setPatientId(Long patientId) {
        this.patientId = patientId;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getDateOfBirth() {
        return dateOfBirth;
    }
    public void setDateOfBirth(String dateOfBirth) {
        this.dateOfBirth = dateOfBirth;
    }
    public String getContactNumber() {
        return contactNumber;
    }
    public void setContactNumber(String contactNumber) {
        this.contactNumber = contactNumber;
    }
    public String getIssue() {
        return issue;
    }
    public void setIssue(String issue) {
        this.issue = issue;
    }
}

```

2) Doctor.java

```

package com.example.PatientManagementSystem;

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

@Entity
public class Doctor {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long doctorId;
    private String name;
    private String specialty;
    // Getters and setters
    public Long getDoctorId() {
        return doctorId;
    }
}

```

```

    }
    public void setDoctorId(Long doctorId) {
        this.doctorId = doctorId;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getSpecialty() {
        return specialty;
    }
    public void setSpecialty(String specialty) {
        this.specialty = specialty;
    }
}

}

```

3)Appointment.java

```

package com.example.PatientManagementSystem;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;

@Entity
public class Appointment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long appointmentId;
    private String date;
    private String time;

    @ManyToOne @JoinColumn(name =
    "patient_id") private Patient patient;

    @ManyToOne @JoinColumn(name =
    "doctor_id") private Doctor doctor;

    public Long getAppointmentId() {
        return appointmentId;
    }

    public void setAppointmentId(Long appointmentId) {
        this.appointmentId = appointmentId;
    }

    public String getDate() {
        return date;
    }
}

```

```

    }

    public void setDate(String date) {
        this.date = date;
    }

    public String getTime() {
        return time;
    }

    public void setTime(String time) {
        this.time = time;
    }

    public Patient getPatient() {
        return patient;
    }

    public void setPatient(Patient patient) {
        this.patient = patient;
    }

    public Doctor getDoctor() {
        return doctor;
    }

    public void setDoctor(Doctor doctor) {
        this.doctor = doctor;
    }

}

```

SERVICE

1)PatientService.java

```
package com.example.PatientManagementSystem;
```

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
@Service
public class PatientService {
    @Autowired
    private PatientRepository patientRepository;

    public Patient createPatient(Patient patient) {
        return patientRepository.save(patient);
    }

    public List<Patient> getAllPatients() {
        return patientRepository.findAll();
    }

    public Patient getPatientById(Long id) {

```

```

        return patientRepository.findById(id).orElse(null);
    }

    public Patient updatePatient(Long id, Patient newPatientData) { Patient
        existingPatient = patientRepository.findById(id).orElse(null); if
        (existingPatient != null) {
            existingPatient.setName(newPatientData.getName()); existingPatient.setDateOfBirth(newPatientData.getDateOfBirth());
            existingPatient.setContactNumber(newPatientData.getContactNumber());
            existingPatient.setIssue(newPatientData.getIssue());
            return patientRepository.save(existingPatient);
        }
        return null;
    }

    public void deletePatient(Long id) {
        patientRepository.deleteById(id);
    }
}

```

2)DoctorService.java

```

package com.example.PatientManagementSystem;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class DoctorService {
    @Autowired
    private DoctorRepository doctorRepository;

    public Doctor createDoctor(Doctor doctor) {
        return doctorRepository.save(doctor);
    }

    public List<Doctor> getAllDoctors() {
        return doctorRepository.findAll();
    }

    public Doctor getDoctorById(Long id) {
        return doctorRepository.findById(id).orElse(null);
    }

    public Doctor updateDoctor(Long id, Doctor newDoctorData) { Doctor
        existingDoctor = doctorRepository.findById(id).orElse(null); if
        (existingDoctor != null) {
            existingDoctor.setName(newDoctorData.getName());
            existingDoctor.setSpecialty(newDoctorData.getSpecialty());
            return doctorRepository.save(existingDoctor);
        }
        return null;
    }

    public void deleteDoctor(Long id) {
        doctorRepository.deleteById(id);
    }
}

```

```
}
```

3)AppointmentService.java

```
package com.example.PatientManagementSystem;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

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

@Service
public class AppointmentService {
    @Autowired
    private AppointmentRepository appointmentRepository;

    public Appointment scheduleAppointment(Appointment appointment) {
        return appointmentRepository.save(appointment);
    }

    public List<Appointment> getAllAppointments() {
        return appointmentRepository.findAll();
    }

    public Appointment getAppointmentById(Long id) {
        Optional<Appointment> appointmentOptional = appointmentRepository.findById(id);
        return appointmentOptional.orElse(null);
    }

    public Appointment rescheduleAppointment(Long id, String newDate, String newTime) {
        Optional<Appointment> appointmentOptional = appointmentRepository.findById(id);
        if (appointmentOptional.isPresent()) {
            Appointment existingAppointment = appointmentOptional.get();
            existingAppointment.setDate(newDate);
            existingAppointment.setTime(newTime);
            return appointmentRepository.save(existingAppointment);
        }
        return null; // Or throw an exception indicating appointment not found
    }

    public void cancelAppointment(Long id) {
        appointmentRepository.deleteById(id);
    }
}
```

REPOSITORY

1)PatientRepository.java

```
package com.example.PatientManagementSystem;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
```

```
@Repository
public interface PatientRepository extends JpaRepository<Patient, Long> {
}
```

2)DoctorRepository.java

```
package com.example.PatientManagementSystem;

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

@Repository
public interface DoctorRepository extends JpaRepository<Doctor, Long> { }
```

3)AppointmentRepository.java

```
package com.example.PatientManagementSystem;

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

@Repository
public interface AppointmentRepository extends JpaRepository<Appointment, Long> { }
```

CONTROLLER

1)PatientController.java

```
package com.example.PatientManagementSystem;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/patient")
public class PatientController {
    @Autowired
    private PatientService patientService;

    @PostMapping
    public Patient createPatient(@RequestBody Patient patient) {
        return patientService.createPatient(patient);
    }

    @GetMapping
    public List<Patient> getAllPatients() {
        return patientService.getAllPatients();
    }

    @GetMapping("/{id}")
    public Patient getPatientById(@PathVariable Long id) {
        return patientService.getPatientById(id);
    }

    @PutMapping("/{id}")
    public Patient updatePatient(@PathVariable Long id, @RequestBody Patient newPatientData) {
        return patientService.updatePatient(id, newPatientData);
    }

    @DeleteMapping("/{id}")
    public void deletePatient(@PathVariable Long id) { }
```

```
        patientService.deletePatient(id);
    }
}
```

2)DoctorController.java

```
package com.example.PatientManagementSystem;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/doctors")
public class DoctorController {
    @Autowired
    private DoctorService doctorService;

    @PostMapping
    public Doctor createDoctor(@RequestBody Doctor doctor) {
        return doctorService.createDoctor(doctor);
    }

    @GetMapping
    public List<Doctor> getAllDoctors() {
        return doctorService.getAllDoctors();
    }

    @GetMapping("/{id}")
    public Doctor getDoctorById(@PathVariable Long id) {
        return doctorService.getDoctorById(id);
    }

    @PutMapping("/{id}")
    public Doctor updateDoctor(@PathVariable Long id, @RequestBody Doctor newDoctorData) {
        return doctorService.updateDoctor(id, newDoctorData);
    }

    @DeleteMapping("/{id}")
    public void deleteDoctor(@PathVariable Long id) {
        doctorService.deleteDoctor(id);
    }
}
```

3)AppointmentController.java

```
package com.example.PatientManagementSystem;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/appointments")
public class AppointmentController {
    @Autowired
    private AppointmentService appointmentService;
```

```

    @PostMapping
    public Appointment scheduleAppointment(@RequestBody Appointment appointment) { return
        appointmentService.scheduleAppointment(appointment);
    }

    @GetMapping
    public List<Appointment> getAllAppointments() {
        return appointmentService.getAllAppointments();
    }

    @GetMapping("/{id}")
    public Appointment getAppointmentById(@PathVariable Long id) {
        return appointmentService.getAppointmentById(id);
    }

    @PutMapping("/{id}/reschedule")
    public Appointment rescheduleAppointment(@PathVariable Long id, @RequestParam String newDate, @RequestParam
String newTime) {
        return appointmentService.rescheduleAppointment(id, newDate, newTime);
    }

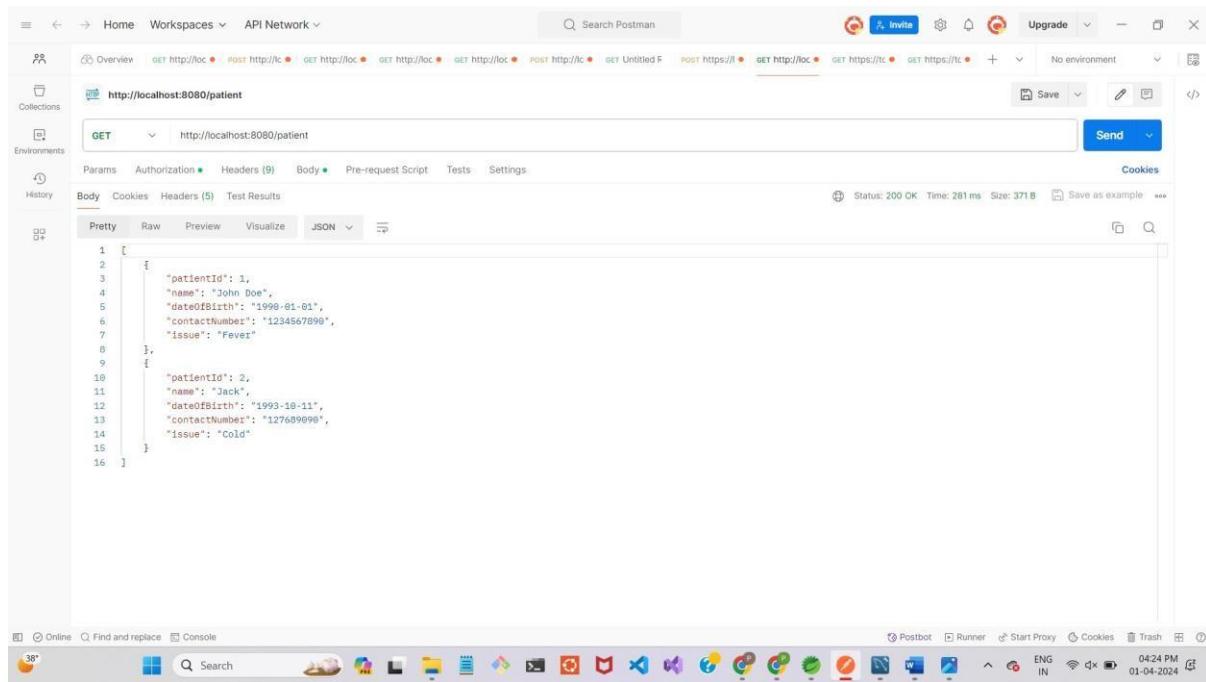
    @DeleteMapping("/{id}")
    public void cancelAppointment(@PathVariable Long id) {
        appointmentService.cancelAppointment(id);
    }
}

APPLICATION PROPERTY
spring.application.name=PatientManagementSystem

spring.datasource.url=jdbc:mysql://localhost:3306/sample9
spring.datasource.username=root spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.database=mysql
spring.jpa.generate-ddl=true
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update

```

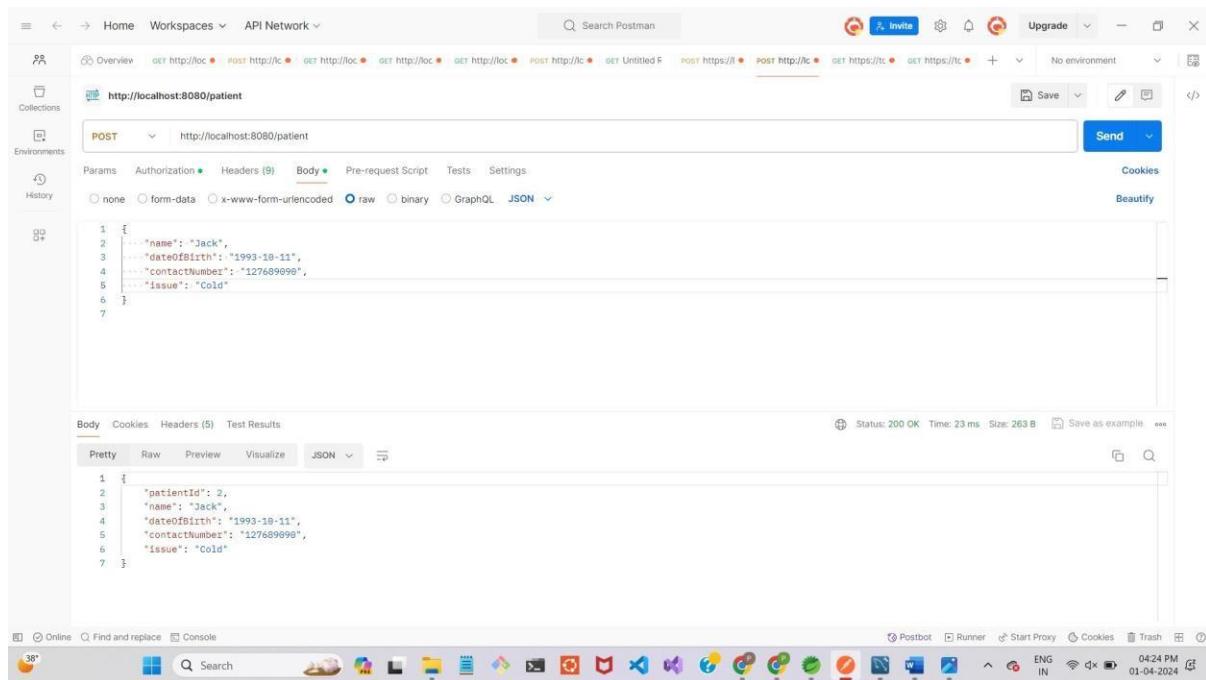
GET:



The screenshot shows the Postman interface with a successful GET request to `http://localhost:8080/patient`. The response status is 200 OK, time is 281 ms, and size is 371 B. The response body is a JSON array with two elements, each representing a patient record:

```
[{"patientId": 1, "name": "John Doe", "dateOfBirth": "1990-01-01", "contactNumber": "1234567890", "issue": "Fever"}, {"patientId": 2, "name": "Jack", "dateOfBirth": "1993-10-11", "contactNumber": "127689098", "issue": "Cold"}]
```

POST:



The screenshot shows the Postman interface with a successful POST request to `http://localhost:8080/patient`. The response status is 200 OK, time is 23 ms, and size is 263 B. The response body is a JSON object representing a single patient record:

```
{"patientId": 2, "name": "Jack", "dateOfBirth": "1993-10-11", "contactNumber": "127689098", "issue": "Cold"}
```

DATABA

SE: PATIENT:

The screenshot shows the MySQL Workbench interface with the database set to 'Local instance MySQL80'. The 'patient' table is selected in the Navigator. A query window displays the following SQL statement and results:

```
1 • SELECT * FROM sample9.patient;
```

patient_id	contact_number	date_of_birth	issue	name
1	1234567890	1990-01-01	Fever	John Doe
2	1234567890	1990-10-11	Cold	Jack
3	9876543211	2004-07-06	Thro...	Ravi
4	3456789566	1994-08-09	Fever	Raj

Table: patient

Columns:

patient_id	contact_number	date_of_birth	issue	name
1	1234567890	1990-01-01	Fever	John Doe
2	1234567890	1990-10-11	Cold	Jack
3	9876543211	2004-07-06	Thro...	Ravi
4	3456789566	1994-08-09	Fever	Raj

DOCTOR

The screenshot shows the MySQL Workbench interface with the database set to 'Local instance MySQL80'. The 'doctor' table is selected in the Navigator. A query window displays the following SQL statement and results:

```
1 • SELECT * FROM sample9.doctor;
```

doctor_id	name	specality
1	jimmy	dentist
2	joel	general
3	john	general
4	sara	eye
5	rose	general
6	Jeff	skin

Table: doctor

Columns:

doctor_id	name	specality
1	jimmy	dentist
2	joel	general
3	john	general
4	sara	eye
5	rose	general
6	Jeff	skin

APPOINTMENT

The screenshot shows the MySQL Workbench interface with the following details:

- File Bar:** File, Edit, View, Query, Database, Server, Tools, Scripting, Help.
- Navigator:** Local instance MySQL80, Schemas (sample5, sample6, sample9), Tables (appointment, doctor, patient).
- Query Editor:** Query 1 tab selected, SQL text: `SELECT * FROM sample9.appointment;`, Result Grid shows the following data:

appointment_id	date	time	doctor_id	patient_id
1	200..	9:30	1	1
2	200..	2:30	2	2
3	199..	1:30	3	3
4	199..	4:30	4	4

- Information:** Table: appointment, Columns: appointment_id, date, time, doctor_id, patient_id.
- Right Panel:** SQLAdditions, message: "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

Result:

Thus, the application is implemented in Spring boot.

Project 9: Managing User Entities

with JPA Date:

Create a simple CRUD application using Spring framework, Maven, and MySQL as a backend database. Implement the CRUD operations for a 'User' entity using JPA. Your application should have the following functionalities:

- Create a new user with attributes such as name, email, and age.
- Retrieve a list of all users from the database.
- Retrieve a specific user by their ID.
- Update the details of an existing user.
- Delete a user by their ID.

User Entity Fields:

- **id:** A unique identifier for each user. Typically, this would be an auto-generated field.
- **name:** The user's full name. This is a string.
- **email:** The user's email address. This field should be unique to ensure that each user can be individually identified and contacted. This is a string.
- **age:** The user's age. This could be represented by an integer.
- **createdAt:** The date and time when the user account was created. This can be automatically set when a new user record is created.
- **updatedAt:** The date and time when the user account was last updated. This field can be automatically updated every time the user's information is modified.
- **status:** The user's account status. This can indicate whether the account is active, suspended, or deleted. Enumerations can be used here to define possible states.

Model:

package com.example.spring;

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

@Entity
@Table(name="Marks")
public class Model {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    private String name,email;
    private int age;
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
}
```

```

public int getAge() {
    return age;
}
public void setAge(int age) {
    this.age = age;
}
public int getId() {
    return id;
}
public void setId(int id) {
    this.id = id;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
}
}

```

Controller:

```

package com.example.spring;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
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.RestController;

@RestController
public class Controller {
    @Autowired
    public service s;
    @GetMapping("/get")
    public List<Model> getdata(){
        return s.getall();
    }
    @GetMapping("/viewbyraw/{searchid}")
    public List<Model> viewbyraw(@PathVariable int searchid){
        return s.findid(searchid);
    }
    @PostMapping("/save")
    public String save(@RequestBody Model m) { s.add(m);
        return "added";
    }
    @PutMapping("/update/{id}")
    public ResponseEntity<Model> updateuser(@PathVariable int id,@RequestBody Model m){
        m.setId(id);
    }
}

```

```

        return s.updateuser(m);
    }
    @DeleteMapping("/delete/{id}")
    public String deleteuser(@PathVariable int id){ return
        s.delete(id);
    }
}

```

Repo:

```

package com.example.spring;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository; import
org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;

@Repository
public interface Repo extends JpaRepository<Model, Integer>{
    @Query("select e from Model e where e.id = :searchid")
    List<Model> find(int searchid);
}

```

service:

```

package com.example.spring;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;

@Service
public class service {
    @Autowired
    public Repo r;
    public List<Model> getall() {
        return r.findAll();
    }
    public String add(Model m) {
        r.save(m);
        return "Added";
    }
    public List<Model> findid(int searchid){
        return r.find(searchid);
    }
    public ResponseEntity<Model> updateuser( Model m) {

```

```

        r.save(m);
        return ResponseEntity.ok(m);
    }
    public String delete(int id){
        r.deleteById(id);
        return "Successfully deleted";
    }
}

```

Application Property :

```

spring.application.name=UserService
spring.datasource.url=jdbc:mysql://localhost:3306/entity
spring.datasource.username=root
spring.datasource.password=mysql
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
#spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.database=mysql
spring.jpa.generate-ddl=true
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update

```

Get:

The screenshot shows the Postman application interface. The URL bar indicates a GET request to `localhost:8080/get`. The response body is displayed in a JSON editor, showing a list of five user entities. Each entity is represented as a JSON object with properties: `id`, `name`, `email`, and `age`. The entities have `id` values of 1, 2, 3, 4, and 5, and all have the same `name` value of "Yakshini" and `email` value of "yaks@gmail.com". The `age` value is consistently 19.

```

1 [
2   {
3     "id": 1,
4     "name": "Gobhika",
5     "email": "gobhi@gmail.com",
6     "age": 19
7   },
8   {
9     "id": 2,
10    "name": "Anjali",
11    "email": "anjul@gmail.com",
12    "age": 19
13  },
14  {
15    "id": 3,
16    "name": "Sivadharane",
17    "email": "siva@gmail.com",
18    "age": 19
19  },
20  {
21    "id": 4,
22    "name": "Yakshini",
23    "email": "yaks@gmail.com",
24    "age": 19
25  },
26  {
27    "id": 5,
28  }
]

```

Post:

The screenshot shows the Postman application interface. A POST request is being made to `localhost:8080/save`. The request body is set to `raw` JSON format and contains the following data:

```
1 {
2   ... "name": "safra",
3   ... "age": 19,
4   "email": "123@gmail.com"
5 }
```

The response status is `200 OK`, time `33 ms`, and size `168 B`. The response body is shown as `added`.

Put:

The screenshot shows the Postman application interface. A PUT request is being made to `localhost:8080/update/3`. The request body is set to `raw` JSON format and contains the following data:

```
1 {
2   ... "name": "safra",
3   ... "age": 19,
4   "email": "123@gmail.com"
5 }
```

The response status is `200 OK`, time `32 ms`, and size `220 B`. The response body is shown as a JSON object with fields `id`, `name`, `email`, and `age`.

Delete:

The screenshot shows the Postman interface. A DELETE request is made to `localhost:8080/delete/3`. The request body is a JSON object:

```
1 {  
2   ... "name": "safira",  
3   ... "age": 19,  
4   ... "email": "123@gmail.com"  
5 }
```

The response status is `200 OK` with a message `Successfully deleted`.

Database:

The screenshot shows MySQL Workbench. A query is run against the `entity.marks` table:

```
1 • SELECT * FROM entity.marks;
```

The result grid displays the following data:

ID	Name	Age	Email
1	Gobhila	19	gobh@gmail.com
2	Anjali	19	anjali@gmail.com
3	Sivadarane	19	siva@gmail.com
4	Yakshini	19	yaks@gmail.com
5	Jeslyn	19	jes@gmail.com

The execution output shows the following logs:

#	Time	Action	Message	Duration / Fetch
1	14:41:46	create database entity	1 row(s) affected	0.016 sec
2	15:51:43	SELECT * FROM entity.marks LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec

Result:

Thus, the application is implemented in Spring boot.

Project 10: Vehicle Dealership Inventory

Management System Date:

Design and implement a basic web application for a vehicle dealership using the Spring framework, Maven, and MySQL. Utilize JPA for performing CRUD operations on 'Vehicle' entities. The application should enable the following capabilities:

- Register a new vehicle with details like make, model, VIN (Vehicle Identification Number), and year of manufacture.
- Display a catalog of all vehicles in the dealership's inventory.
- Provide a feature to search for vehicles based on make or model.
- Allow for the modification of vehicle details.
- Facilitate the removal of a vehicle from the inventory.

Vehicle Entity Fields:

- **id:** A unique identifier for each vehicle in the inventory. This could be an auto-generated field .
- **make:** The manufacturer of the vehicle. This is a string representing the brand.
- **model:** The specific model of the vehicle. This is a string.
- **vin:** The Vehicle Identification Number, which is a unique code used to identify individual motor vehicles. This is a string.
- **yearOfManufacture:** The year the vehicle was manufactured. This could be represented by an integer.
- **color:** The color of the vehicle. This detail can be important for buyers and is a string.
- **price:** The sale price of the vehicle. This could be a decimal to accommodate cents, useful for pricing.

MODEL:

```
package com.example.Vehicle;

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

@Entity
@Table(name = "vehicles")
public class Vehicle {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
private Long id;

private String make;
private String model;
private String vin;
private int yearOfManufacture;
private String color;
private double price; public
Long getId() {
    return id;
}
public void setId(Long id) {
    this.id = id;
}
public String getMake() {
    return make;
}
public void setMake(String make) {
    this.make = make;
}
public String getModel() {
    return model;
}
public void setModel(String model) {
    this.model = model;
}
public String getVin() {
    return vin;
}
public void setVin(String vin) {
    this.vin = vin;
}
public int getYearOfManufacture() {
    return yearOfManufacture;
}
public void setYearOfManufacture(int yearOfManufacture) {
    this.yearOfManufacture = yearOfManufacture;
}
public String getColor() {
    return color;
}
public void setColor(String color) {
    this.color = color;
}
public double getPrice() {
    return price;
}
public void setPrice(double price) {
    this.price = price;
}

}
```

CONTROLLER:

```
package com.example.Vehicle;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController @RequestMapping("/vehicles")
@CrossOrigin(origins = "http://localhost:3000")
public class VehicleController {

    @Autowired
    private VehicleService vehicleService;

    @GetMapping
    public List<Vehicle> getAllVehicles() {
        return vehicleService.getAllVehicles();
    }

    @GetMapping("/{id}")
    public Vehicle getVehicleById(@PathVariable Long id) {
        return vehicleService.getVehicleById(id);
    }

    @PostMapping
    public Vehicle saveVehicle(@RequestBody Vehicle vehicle) {
        return vehicleService.saveVehicle(vehicle);
    }

    @PutMapping("/{id}")
    public Vehicle updateVehicle(@PathVariable Long id, @RequestBody Vehicle updatedVehicle) { return
        vehicleService.updateVehicle(id, updatedVehicle);
    }

    @DeleteMapping("/{id}")
    public void deleteVehicleById(@PathVariable Long id) {
        vehicleService.deleteVehicleById(id);
    }
}
```

REPO:

```
package com.example.Vehicle;

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

public interface VehicleRepository extends JpaRepository<Vehicle, Long>{
}
```

SERVICE

```
package com.example.Vehicle;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.List;
```

```

import java.util.Optional;

@Service
public class VehicleService {
    @Autowired
    private VehicleRepository vehicleRepository;

    public List<Vehicle> getAllVehicles() {
        return vehicleRepository.findAll();
    }

    public Vehicle saveVehicle(Vehicle vehicle) {
        return vehicleRepository.save(vehicle);
    }

    public Vehicle getVehicleById(Long id) {
        return vehicleRepository.findById(id).orElse(null);
    }

    public void deleteVehicleById(Long id) {
        vehicleRepository.deleteById(id);
    }

    public Vehicle updateVehicle(Long id, Vehicle updatedVehicle) {
        Optional<Vehicle> optionalVehicle = vehicleRepository.findById(id); if
        (optionalVehicle.isPresent()) {
            Vehicle existingVehicle = optionalVehicle.get();
            existingVehicle.setMake(updatedVehicle.getMake());
            existingVehicle.setModel(updatedVehicle.getModel());
            existingVehicle.setColor(updatedVehicle.getColor());
            existingVehicle.setVin(updatedVehicle.getVin());
            existingVehicle.setYearOfManufacture(updatedVehicle.getYearOfManufacture()); return
            vehicleRepository.save(existingVehicle);
        } else {
            return null; // Or throw an exception, depending on your error handling strategy
        }
    }
}

```

APPLICATION-PROPERTIES:

```

spring.application.name=Vehicle spring.datasource.url=
jdbc:mysql://localhost:3306/new
spring.datasource.username=root
spring.datasource.password=kasthuri@1102004
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.database= mysql
spring.jpa.generate-ddl=true
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto= update

```

GET:

The screenshot shows the Postman interface with a dark theme. On the left, the sidebar displays 'My Workspace' with a 'Collections' section containing 'Your collection'. Below it is a note about creating a collection for requests. A central panel shows a 'GET' request to 'localhost:8080/vehicles'. The 'Body' tab is selected, showing a raw JSON payload. The response tab shows a 200 OK status with a response body containing vehicle data.

```
1 {
2     "id": 1,
3     "make": "Japan",
4     "model": "Sedan",
5     "vin": "xxx",
6     "yearOfManufacture": 0,
7     "color": "black",
8     "price": 500000.0
}
```

POST:

The screenshot shows the Postman interface with a dark theme. The left sidebar is identical to the first screenshot. The central panel shows a 'POST' request to 'localhost:8080/vehicles'. The 'Body' tab is selected, showing a raw JSON payload. The response tab shows a 200 OK status with a response body containing vehicle data, identical to the GET response.

```
1 {
2     "id": 8,
3     "make": "Japan",
4     "model": "Sedan",
5     "vin": "xxx",
6     "yearOfManufacture": 0,
7     "color": "black",
8     "price": 800000.0
}
```

PUT:

The screenshot shows the Postman interface with a PUT request to `localhost:8080/vehicles/2`. The request body is set to raw JSON:

```
PUT /localhost:8080/vehicles/2
{
  "id": 2,
  "make": "India",
  "model": "SUV",
  "vin": "xxx",
  "yearOfManufacture": 2022,
  "color": "red",
  "price": 800000.0
}
```

The response status is 200 OK with 16 ms latency and 358 B size.

DELETE:

The screenshot shows the Postman interface with a DELETE request to `localhost:8080/vehicles/6`. The request body is set to raw JSON:

```
DELETE /localhost:8080/vehicles/6
{
  "id": 2,
  "make": "India",
  "model": "SUV",
  "vin": "xxx",
  "yearOfManufacture": 2022,
  "color": "red",
  "price": 800000.0
}
```

The response status is 200 OK with 58 ms latency and 212 B size.

DATABASE:

The screenshot shows the MySQL Workbench interface. In the top navigation bar, the title is "Local instance MySQL80". The menu includes File, Edit, View, Query, Database, Server, Tools, Scripting, and Help. The toolbar contains various icons for database management tasks.

The Navigator pane on the left lists Schemas: tb2, Views, Stored Procedures, Functions, dbc (Tables: demo, marks, Views, Stored Procedures, Functions), fullstack, new (Tables: tb2). The current schema is "new".

The main workspace displays a query editor with the SQL command: `1 • SELECT * FROM new.vehicles;`. Below the query is a Result Grid showing the following data:

	id	color	make	model	price	vin	year_of_manufacture
1	black	Japan	Sedan	500...	xxx	0	
2	red	India	SUV	400...	xxx	2022	
4	blue	China	Sedan	0	4	2020	
5	White	China	Sedan	0	yyy	2017	
7	White	Italy	Sedan	0	yyy	2018	

The Output pane shows the Action History:

#	Time	Action	Message	Duration / Fetch
9	12:33:17	use new	0 row(s) affected	0.000 sec
10	12:34:49	insert into train values(1,"12:00","12:10","Karthika Express")	1 row(s) affected	0.000 sec
11	12:35:00	SELECT * FROM new.train LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
12	12:45:10	create database fullstack	1 row(s) affected	0.015 sec
13	14:45:27	SELECT * FROM new.vehicles LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec

The system tray at the bottom right shows the date (01-04-2024), time (16:18), and various system icons.

Result:

Thus, the application is implemented in Spring boot.