Q1) Spring Boot Application – Department & Employee Management System
You are developing a Spring Boot application to manage Departments and their Employees in an office management system.
The system allows storing department details along with the employees working in each department.
Each Department can have multiple Employees

Each Employee belongs to only one Department
The application must be implemented using:

Spring Boot
Spring Data JPA
JPQL
RESTful APIs
The application must follow a layered architecture.
Entity Layer
File: Department.java (Model Class)
Variables:

private Long departmentId;
private String departmentName;
private String departmentDescription;
private String location;
private List<Employee> employees;
Description:
Represents the Department entity in the system.
Establishes a One-to-Many relationship with Employee:
One department can have multiple employees.
Relationship is mapped by the field name department.

Uses appropriate Cascade Type.
Uses @JsonIgnoreProperties on department to avoid infinite recursion during JSON serialization.
Includes:

Getters and setters for all fields.
File: Employee.java (Model Class)
Variables:

private Long employeeId;
private String employeeName;
private String designation;
private String bio;
private Department department;
Description:
Represents an Employee entity in the system.

Establishes a Many-to-One relationship with Department:
Multiple employees can belong to one department.
Uses @JoinColumn with column name department_id.
Uses @JsonIgnoreProperties on employees to avoid infinite recursion during JSON serialization.
Includes:

Getters and setters for all fields.
Repository Layer
File: DepartmentRepository.java
public interface DepartmentRepository extends JpaRepository<Department, Long>
Description:
Provides CRUD operations for Department entities.

Includes a custom JPQL query to retrieve Department based on departmentName.
Method:
List<Department> findAllByDepartmentName(String departmentName);
File: EmployeeRepository.java
public interface EmployeeRepository extends JpaRepository<Employee, Long>
Description:
Provides CRUD operations for Employee entities.

Includes a custom JPQL query to retrieve Employee based on employeeName.
Method:
List<Employee> findEmployeesByName(String name);

```
Service Layer
File: DepartmentService.java
Methods:
public Department addDepartment(Department department)

Saves a new department into the database.
Assigns the department reference to each employee before saving.
Returns the saved department entity.
public Department getDepartmentById(Long departmentId)
Retrieves a department by ID.

Returns the department or null.
public List<Department> getAllDepartments(String departmentName)

Retrieves departments based on department name.
Calls the JPQL query method findAllByDepartmentName from DepartmentRepository.
public Department updateDepartment(Long departmentId, Department department)
Updates department details if the department exists.

Returns null if the department ID is not found.
File: EmployeeService.java
Methods:
public Employee addEmployee(Employee employee)

Saves a new employee into the database.
Returns the saved employee entity.
public Employee getEmployeeById(Long employeeId)
Retrieves an employee by ID.

Returns null if not found.
public List<Employee> getAllEmployees()

Retrieves all employees from the database.
public Employee updateEmployee(Long employeeId, Employee employee)
Updates employee details if the employee exists.

Returns null if the employee ID is not found.
Controller Layer
File: DepartmentController.java
Base URL: /departments
Endpoints:

Create Department
Endpoint: POST /departments
Method Signature:
public Department createDepartment(@RequestBody Department department)
Accepts department details in request body.
Saves and returns the department entity.
Get Department by ID
Endpoint: GET /departments/{departmentId}
Method Signature:
public Department getDepartment(@PathVariable Long departmentId)
Retrieves department details by ID.
Get Departments by Name
Endpoint: GET /departments/getAllDepartments/{departmentName}
Method Signature:
public List<Department> getAllDepartments(@PathVariable String departmentName)
Retrieves departments based on department name.
Update Department
Endpoint: PUT /departments/{departmentId}
Method Signature:
public Department updateDepartment(@PathVariable Long departmentId,
                                   @RequestBody Department department)
Updates department details based on department ID.
File: EmployeeController.java
Base URL: /employees
Endpoints:

Create Employee
```

```
Endpoint: POST /employees
Method Signature:
public Employee createEmployee(@RequestBody Employee employee)
Accepts employee details in request body.
Saves and returns the employee entity.
Get Employee by ID
Endpoint: GET /employees/{employeeId}
Method Signature:
public Employee getEmployee(@PathVariable Long employeeId)
Retrieves employee details by ID.
Get All Employees
Endpoint: GET /employees
Method Signature:
public List<Employee> getAllEmployees()
Retrieves all employees.
Update Employee
Endpoint: PUT /employees/{employeeId}
Method Signature:
public Employee updateEmployee(@PathVariable Long employeeId,
                               @RequestBody Employee employee)
Updates employee details based on employee ID.
6:24
Solution
Entity Layer
Department.java
package com.example.departmentemployee.entity;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import jakarta.persistence.*;
import java.util.List;
@Entity
@Table(name = "department")
public class Department {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long departmentId;
   private String departmentName;
   private String departmentDescription;
   private String location;
   @OneToMany(mappedBy = "department", cascade = CascadeType.ALL)
   @JsonIgnoreProperties("department")
   private List<Employee> employees;
   public Department() {}
   // Getters and Setters
   public Long getDepartmentId() {
       return departmentId;
   }
   public void setDepartmentId(Long departmentId) {
       this.departmentId = departmentId;
   }
   public String getDepartmentName() {
       return departmentName;
   }
   public void setDepartmentName(String departmentName) {
       this.departmentName = departmentName;
   }
   public String getDepartmentDescription() {
       return departmentDescription;
   }
   public void setDepartmentDescription(String departmentDescription) {
       this.departmentDescription = departmentDescription;
   }
   public String getLocation() {
       return location;
   }
   public void setLocation(String location) {
       this.location = location;
   }
   public List<Employee> getEmployees() {
       return employees;
```

```
    }
    public void setEmployees(List<Employee> employees) {
        this.employees = employees;
    }
}
Employee.java
package com.example.departmentemployee.entity;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import jakarta.persistence.*;
@Entity
@Table(name = "employee")
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long employeeId;
    private String employeeName;
    private String designation;
    private String bio;
    @ManyToOne
    @JoinColumn(name = "department_id")
    @JsonIgnoreProperties("employees")
    private Department department;
    public Employee() {}
    // Getters and Setters
    public Long getEmployeeId() {
        return employeeId;
    }
    public void setEmployeeId(Long employeeId) {
        this.employeeId = employeeId;
    }
    public String getEmployeeName() {
        return employeeName;
    }
    public void setEmployeeName(String employeeName) {
        this.employeeName = employeeName;
    }
    public String getDesignation() {
        return designation;
    }
    public void setDesignation(String designation) {
        this.designation = designation;
    }
    public String getBio() {
        return bio;
    }
    public void setBio(String bio) {
        this.bio = bio;
    }
    public Department getDepartment() {
        return department;
    }
    public void setDepartment(Department department) {
        this.department = department;
    }
}
Repository Layer
DepartmentRepository.java
package com.example.departmentemployee.repository;
import com.example.departmentemployee.entity.Department;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import java.util.List;
public interface DepartmentRepository extends JpaRepository<Department, Long> {
    @Query("SELECT d FROM Department d WHERE d.departmentName = :departmentName")
    List<Department> findAllByDepartmentName(String departmentName);
}
EmployeeRepository.java
package com.example.departmentemployee.repository;
import com.example.departmentemployee.entity.Employee;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import java.util.List;
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
   @Query("SELECT e FROM Employee e WHERE e.employeeName = :name")
   List<Employee> findEmployeesByName(String name);
}
```

Service Layer

DepartmentService.java

```
package com.example.departmentemployee.service;
import com.example.departmentemployee.entity.Department;
import com.example.departmentemployee.entity.Employee;
import com.example.departmentemployee.repository.DepartmentRepository;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;
@Service
public class DepartmentService {
   private final DepartmentRepository departmentRepository;
   public DepartmentService(DepartmentRepository departmentRepository) {
       this.departmentRepository = departmentRepository;
   }
   public Department addDepartment(Department department) {
       if (department.getEmployees() != null) {
           for (Employee e : department.getEmployees()) {
               e.setDepartment(department);
           }
       }
       return departmentRepository.save(department);
   }
   public Department getDepartmentById(Long departmentId) {
       Optional<Department> dept = departmentRepository.findById(departmentId);
       return dept.orElse(null);
   }
   public List<Department> getAllDepartments(String departmentName) {
       return departmentRepository.findAllByDepartmentName(departmentName);
   }
   public Department updateDepartment(Long departmentId, Department department) {
       Optional<Department> existing = departmentRepository.findById(departmentId);
       if (existing.isPresent()) {
           Department d = existing.get();
           d.setDepartmentName(department.getDepartmentName());
           d.setDepartmentDescription(department.getDepartmentDescription());
           d.setLocation(department.getLocation());
           return departmentRepository.save(d);
       }
       return null;
   }
}
```

EmployeeService.java

```
package com.example.departmentemployee.service;
import com.example.departmentemployee.entity.Employee;
import com.example.departmentemployee.repository.EmployeeRepository;
import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;
@Service
public class EmployeeService {
   private final EmployeeRepository employeeRepository;
   public EmployeeService(EmployeeRepository employeeRepository) {
       this.employeeRepository = employeeRepository;
   }
   public Employee addEmployee(Employee employee) {
       return employeeRepository.save(employee);
   }
   public Employee getEmployeeById(Long employeeId) {
       Optional<Employee> emp = employeeRepository.findById(employeeId);
       return emp.orElse(null);
   }
```

```
    public List<Employee> getAllEmployees() {
        return employeeRepository.findAll();
    }
    public Employee updateEmployee(Long employeeId, Employee employee) {
        Optional<Employee> existing = employeeRepository.findById(employeeId);
        if (existing.isPresent()) {
            Employee e = existing.get();
            e.setEmployeeName(employee.getEmployeeName());
            e.setDesignation(employee.getDesignation());
            e.setBio(employee.getBio());
            e.setDepartment(employee.getDepartment());
            return employeeRepository.save(e);
        }
        return null;
    }
}
```

Controller Layer

DepartmentController.java

```
package com.example.departmentemployee.controller;
import com.example.departmentemployee.entity.Department;
import com.example.departmentemployee.service.DepartmentService;
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
@RequestMapping("/departments")
public class DepartmentController {
    private final DepartmentService departmentService;
    public DepartmentController(DepartmentService departmentService) {
        this.departmentService = departmentService;
    }
    @PostMapping
    public Department createDepartment(@RequestBody Department department) {
        return departmentService.addDepartment(department);
    }
    @GetMapping("/{departmentId}")
    public Department getDepartment(@PathVariable Long departmentId) {
        return departmentService.getDepartmentById(departmentId);
    }
    @GetMapping("/getAllDepartments/{departmentName}")
    public List<Department> getAllDepartments(@PathVariable String departmentName) {
        return departmentService.getAllDepartments(departmentName);
    }
    @PutMapping("/{departmentId}")
    public Department updateDepartment(
            @PathVariable Long departmentId,
            @RequestBody Department department) {
        return departmentService.updateDepartment(departmentId, department);
    }
}
```

EmployeeController.java

```
package com.example.departmentemployee.controller;
import com.example.departmentemployee.entity.Employee;
import com.example.departmentemployee.service.EmployeeService;
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
@RequestMapping("/employees")
public class EmployeeController {
    private final EmployeeService employeeService;
    public EmployeeController(EmployeeService employeeService) {
        this.employeeService = employeeService;
    }
    @PostMapping
    public Employee createEmployee(@RequestBody Employee employee) {
        return employeeService.addEmployee(employee);
    }
    @GetMapping("/{employeeId}")
    public Employee getEmployee(@PathVariable Long employeeId) {
```

```
        return employeeService.getEmployeeById(employeeId);
    }
    @GetMapping
    public List<Employee> getAllEmployees() {
        return employeeService.getAllEmployees();
    }
    @PutMapping("/{employeeId}")
    public Employee updateEmployee(
            @PathVariable Long employeeId,
            @RequestBody Employee employee) {
        return employeeService.updateEmployee(employeeId, employee);
    }
}
```

Main Class

DepartmentEmployeeApplication.java

```
package com.example.departmentemployee;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class DepartmentEmployeeApplication {
    public static void main(String[] args) {
        SpringApplication.run(DepartmentEmployeeApplication.class, args);
    }
}
```

6:24

Q2) Develop a Spring Boot RESTful application to manage Retailers and the Food Items they supply in a supply-chain system.

The application must demonstrate:

Layered architecture (Entity → Repository → Service → Controller)

One-to-Many and Many-to-One JPA relationships

Spring Data JPA with JPQL

REST API development and testing

JSON serialization handling

Important Notes for Learners

This assessment follows partial test-case grading.

Each endpoint and feature is validated independently.

You can earn marks even if some endpoints are incomplete.

Follow package names, class names, method names, and URLs exactly.

Do NOT rename or delete any files.

Do NOT change method signatures.

Do NOT add extra endpoints beyond what is specified.

All responses must be JSON compatible.

Global exception handling is mandatory for some cases.

Entity Layer

1. Retailer.java

Purpose

Represents a retailer in the supply-chain system.

Fields (Must be EXACT):

```
private Long retailerId;
private String retailerName;
private String storeLocation;
private List<FoodItem> foodItems = new ArrayList<>();
```

Relationship Rules

One Retailer → Many FoodItems

Mapped by retailer field in FoodItem

Must use appropriate cascade type

Must use JSON annotation to prevent infinite recursion

Mandatory Requirements

Getter and Setter for all fields

Retailer must be serializable to JSON

Food items must appear under retailer when fetched

2. FoodItem.java

Purpose

Represents a food item supplied by a retailer.

Fields (Must be EXACT):

private Long itemId;
private String itemName;
private String type;
private Retailer retailer;
Relationship Rules
• Many FoodItems → One Retailer
• Must use:
@JoinColumn(name = "retailer_id")
Mandatory Requirements

Getter and Setter for all fields
Retailer object must be included in JSON response
Repository Layer
3. RetailerRepository.java
public interface RetailerRepository extends JpaRepository<Retailer, Long>
Custom JPQL Query
List<Retailer> findRetailerSortedAsc();
Behavior

Returns all retailers
Sorted by retailerName in ascending order
Custom JPQL Query
Long countFoodItemsByRetailerId(Long retailerId);
Behavior

Returns the total number of food items
Associated with the given retailer ID
4. FoodItemRepository.java
public interface FoodItemRepository extends JpaRepository<FoodItem, Long>
Mandatory JPQL Queries
FoodItem getFoodItemThroughJpqlQuery(Long itemId);
List<FoodItem> findFoodItemsSortedAsc();
List<FoodItem> findFoodItemsByRetailerId(@Param("retailerId") Long retailerId);
Service Layer
5. RetailerService.java
Variable:

private RetailerRepository retailerRepository;
Methods to Implement

public Retailer addRetailer(Retailer retailer)
public Retailer getRetailerById(Long retailerId)
public List<Retailer> getAllRetailers()
Must use JPQL sorting
public Retailer updateRetailer(Long retailerId, Retailer retailer)
Updates the retailer details for the given ID and returns the updated retailer.
public long countFoodItemsByRetailerId(Long retailerId)
Must throw exception if retailer does not exist
6. FoodItemService.java
Methods to Implement

public FoodItem addFoodItem(FoodItem foodItem)
public FoodItem getFoodItemById(Long itemId)
public List<FoodItem> getAllFoodItems()
public FoodItem updateFoodItem(Long itemId, FoodItem foodItem)
public List<FoodItem> getFoodItemsByRetailerId(Long retailerId)
Sorted by item name
Return empty list if retailer exists but has no items
Throw exception if retailer does not exist
Controller Layer
7. RetailerController.java
Available API Endpoints
POST /retailers
Add a new retailer to the system.
GET /retailers/{retailerId}
Retrieve retailer details using the retailer ID.
GET /retailers

Fetch all retailers.

Retailers should be returned in sorted order.
PUT /retailers/{retailerId}
Update the details of an existing retailer using the retailer ID.
GET /retailers/{retailerId}/food-items/count
Return the total number of food items associated with a specific retailer.
8. FoodItemController.java
Available API Endpoints
POST /food-items
Add a new food item to the system.
GET /food-items/{itemId}
Retrieve food item details using the food item ID.
GET /food-items
Fetch all food items.

Food items should be returned in sorted order.
PUT /food-items/{itemId}
Update the details of an existing food item using the food item ID.
GET /food-items/retailer/{retailerId}
Retrieve all food items associated with a specific retailer.
Global Exception Handling (Mandatory)
public ResponseEntity<Map<String, Object>> handleEntityNotFoundException(
EntityNotFoundException ex, WebRequest request)

Handles EntityNotFoundException and returns a 404 Not Found response with structured error details.
public ResponseEntity<Map<String, Object>> handleGlobalException(
Exception ex, WebRequest request)

Handles all unhandled exceptions and returns a 500 Internal Server Error response with error details.
A Global Exception Handler must:

Catch EntityNotFoundException
Return HTTP 404
JSON format:

```
{
 "status": 404,
 "error": "Not Found",
 "message": "Retailer not found with ID: {id}"
}
```

Final Hint
Implement the project layer by layer:

Entities & relationships
Repository JPQL queries
Services with exception handling
Controllers & endpoints
Global exception handler
If each endpoint returns exact JSON structure and status codes, all automated test cases will pass successfully.
6:25
Solution
entity/Retailer.java

```
package com.edutech.supplychainapplication.entity;
import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;
import com.fasterxml.jackson.annotation.JsonManagedReference;
@Entity
public class Retailer {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long retailerId;
   private String retailerName;
   private String storeLocation;
   @OneToMany(mappedBy = "retailer", cascade = CascadeType.ALL)
   @JsonManagedReference
```

```
    private List<FoodItem> foodItems = new ArrayList<>();
    public Long getRetailerId() {
        return retailerId;
    }
    public void setRetailerId(Long retailerId) {
        this.retailerId = retailerId;
    }
    public String getRetailerName() {
        return retailerName;
    }
    public void setRetailerName(String retailerName) {
        this.retailerName = retailerName;
    }
    public String getStoreLocation() {
        return storeLocation;
    }
    public void setStoreLocation(String storeLocation) {
        this.storeLocation = storeLocation;
    }
    public List<FoodItem> getFoodItems() {
        return foodItems;
    }
    public void setFoodItems(List<FoodItem> foodItems) {
        this.foodItems = foodItems;
    }
}
```

entity/FoodItem.java

```
package com.edutech.supplychainapplication.entity;
import javax.persistence.*;
import com.fasterxml.jackson.annotation.JsonBackReference;
@Entity
public class FoodItem {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long itemId;
   private String itemName;
   private String type;
   @ManyToOne
   @JoinColumn(name = "retailer_id")
   @JsonBackReference
   private Retailer retailer;
   public Long getItemId() {
       return itemId;
   }
   public void setItemId(Long itemId) {
       this.itemId = itemId;
   }
   public String getItemName() {
       return itemName;
   }
   public void setItemName(String itemName) {
       this.itemName = itemName;
   }
   public String getType() {
       return type;
   }
   public void setType(String type) {
       this.type = type;
   }
   public Retailer getRetailer() {
       return retailer;
   }
   public void setRetailer(Retailer retailer) {
       this.retailer = retailer;
   }
}
```

repository/RetailerRepository.java

```
package com.edutech.supplychainapplication.repository;
import com.edutech.supplychainapplication.entity.Retailer;
```

```
import org.springframework.data.jpa.repository.*;
import org.springframework.data.repository.query.Param;
import java.util.List;
public interface RetailerRepository extends JpaRepository<Retailer, Long> {
   @Query("SELECT r FROM Retailer r ORDER BY r.retailerName ASC")
   List<Retailer> findRetailerSortedAsc();
   @Query("SELECT COUNT(f) FROM FoodItem f WHERE f.retailer.retailerId = :retailerId")
   Long countFoodItemsByRetailerId(@Param("retailerId") Long retailerId);
}
repository/FoodItemRepository.java
package com.edutech.supplychainapplication.repository;
import com.edutech.supplychainapplication.entity.FoodItem;
import org.springframework.data.jpa.repository.*;
import org.springframework.data.repository.query.Param;
import java.util.List;
public interface FoodItemRepository extends JpaRepository<FoodItem, Long> {
   @Query("SELECT f FROM FoodItem f WHERE f.itemId = :itemId")
   FoodItem getFoodItemThroughJpqlQuery(@Param("itemId") Long itemId);
   @Query("SELECT f FROM FoodItem f ORDER BY f.itemName ASC")
   List<FoodItem> findFoodItemsSortedAsc();
   @Query("SELECT f FROM FoodItem f WHERE f.retailer.retailerId = :retailerId ORDER BY f.itemName
ASC")
   List<FoodItem> findFoodItemsByRetailerId(@Param("retailerId") Long retailerId);
}
service/RetailerService.java
package com.edutech.supplychainapplication.service;
import com.edutech.supplychainapplication.entity.Retailer;
import com.edutech.supplychainapplication.repository.RetailerRepository;
import org.springframework.stereotype.Service;
import javax.persistence.EntityNotFoundException;
import java.util.List;
@Service
public class RetailerService {
   private final RetailerRepository retailerRepository;
   public RetailerService(RetailerRepository retailerRepository) {
       this.retailerRepository = retailerRepository;
   }
   public Retailer addRetailer(Retailer retailer) {
       return retailerRepository.save(retailer);
   }
   public Retailer getRetailerById(Long retailerId) {
       return retailerRepository.findById(retailerId)
               .orElseThrow(() ->
                       new EntityNotFoundException("Retailer not found with ID: " + retailerId));
   }
   public List<Retailer> getAllRetailers() {
       return retailerRepository.findRetailerSortedAsc();
   }
   public Retailer updateRetailer(Long retailerId, Retailer retailer) {
       Retailer existing = getRetailerById(retailerId);
       existing.setRetailerName(retailer.getRetailerName());
       existing.setStoreLocation(retailer.getStoreLocation());
       return retailerRepository.save(existing);
   }
   public long countFoodItemsByRetailerId(Long retailerId) {
       getRetailerById(retailerId);
       return retailerRepository.countFoodItemsByRetailerId(retailerId);
   }
}
service/FoodItemService.java
package com.edutech.supplychainapplication.service;
import com.edutech.supplychainapplication.entity.FoodItem;
import com.edutech.supplychainapplication.repository.FoodItemRepository;
import com.edutech.supplychainapplication.repository.RetailerRepository;
import org.springframework.stereotype.Service;
import javax.persistence.EntityNotFoundException;
import java.util.List;
@Service
public class FoodItemService {
```

```
    private final FoodItemRepository foodItemRepository;
    private final RetailerRepository retailerRepository;
    public FoodItemService(FoodItemRepository foodItemRepository,
                           RetailerRepository retailerRepository) {
        this.foodItemRepository = foodItemRepository;
        this.retailerRepository = retailerRepository;
    }
    public FoodItem addFoodItem(FoodItem foodItem) {
        return foodItemRepository.save(foodItem);
    }
    public FoodItem getFoodItemById(Long itemId) {
        FoodItem item = foodItemRepository.getFoodItemThroughJpqlQuery(itemId);
        if (item == null) {
            throw new EntityNotFoundException("FoodItem not found with ID: " + itemId);
        }
        return item;
    }
    public List<FoodItem> getAllFoodItems() {
        return foodItemRepository.findFoodItemsSortedAsc();
    }
    public FoodItem updateFoodItem(Long itemId, FoodItem foodItem) {
        FoodItem existing = getFoodItemById(itemId);
        existing.setItemName(foodItem.getItemName());
        existing.setType(foodItem.getType());
        existing.setRetailer(foodItem.getRetailer());
        return foodItemRepository.save(existing);
    }
    public List<FoodItem> getFoodItemsByRetailerId(Long retailerId) {
        retailerRepository.findById(retailerId)
                .orElseThrow(() ->
                        new EntityNotFoundException("Retailer not found with ID: " + retailerId));
        return foodItemRepository.findFoodItemsByRetailerId(retailerId);
    }
}
controller/RetailerController.java
package com.edutech.supplychainapplication.controller;
import com.edutech.supplychainapplication.entity.Retailer;
import com.edutech.supplychainapplication.service.RetailerService;
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
public class RetailerController {
    private final RetailerService retailerService;
    public RetailerController(RetailerService retailerService) {
        this.retailerService = retailerService;
    }
    @PostMapping("/retailers")
    public Retailer addRetailer(@RequestBody Retailer retailer) {
        return retailerService.addRetailer(retailer);
    }
    @GetMapping("/retailers/{retailerId}")
    public Retailer getRetailer(@PathVariable Long retailerId) {
        return retailerService.getRetailerById(retailerId);
    }
    @GetMapping("/retailers")
    public List<Retailer> getAllRetailers() {
        return retailerService.getAllRetailers();
    }
    @PutMapping("/retailers/{retailerId}")
    public Retailer updateRetailer(@PathVariable Long retailerId,
                                   @RequestBody Retailer retailer) {
        return retailerService.updateRetailer(retailerId, retailer);
    }
    @GetMapping("/retailers/{retailerId}/food-items/count")
    public long countFoodItems(@PathVariable Long retailerId) {
        return retailerService.countFoodItemsByRetailerId(retailerId);
    }
}
controller/FoodItemController.java
```

```
package com.edutech.supplychainapplication.controller;
import com.edutech.supplychainapplication.entity.FoodItem;
import com.edutech.supplychainapplication.service.FoodItemService;
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
public class FoodItemController {
   private final FoodItemService foodItemService;
   public FoodItemController(FoodItemService foodItemService) {
       this.foodItemService = foodItemService;
   }
   @PostMapping("/food-items")
   public FoodItem addFoodItem(@RequestBody FoodItem foodItem) {
       return foodItemService.addFoodItem(foodItem);
   }
   @GetMapping("/food-items/{itemId}")
   public FoodItem getFoodItem(@PathVariable Long itemId) {
       return foodItemService.getFoodItemById(itemId);
   }
   @GetMapping("/food-items")
   public List<FoodItem> getAllFoodItems() {
       return foodItemService.getAllFoodItems();
   }
   @PutMapping("/food-items/{itemId}")
   public FoodItem updateFoodItem(@PathVariable Long itemId,
                                  @RequestBody FoodItem foodItem) {
       return foodItemService.updateFoodItem(itemId, foodItem);
   }
   @GetMapping("/food-items/retailer/{retailerId}")
   public List<FoodItem> getFoodItemsByRetailer(@PathVariable Long retailerId) {
       return foodItemService.getFoodItemsByRetailerId(retailerId);
   }
}
exception/GlobalExceptionHandler.java
package com.edutech.supplychainapplication.exception;
import org.springframework.http.*;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.context.request.WebRequest;
import javax.persistence.EntityNotFoundException;
import java.util.HashMap;
import java.util.Map;
@RestControllerAdvice
public class GlobalExceptionHandler {
   @ExceptionHandler(EntityNotFoundException.class)
   public ResponseEntity<Map<String, Object>> handleEntityNotFound(
           EntityNotFoundException ex, WebRequest request) {
       Map<String, Object> response = new HashMap<>();
       response.put("status", 404);
       response.put("error", "Not Found");
       response.put("message", ex.getMessage());
       return new ResponseEntity<>(response, HttpStatus.NOT_FOUND);
   }
}
6:25
Q3) You are developing a Spring Boot application to manage Events and their Speakers.
The system allows storing event details along with the speakers participating in each event. Each
event can have multiple speakers, while each speaker is associated with only one event.
 The application must be implemented using Spring Boot, Spring Data JPA, JPQL, and RESTful APIs,
following a layered architecture.
Entity Layer
File: Event.java (Model Class)
Variables:

private Long eventId;
private String eventName;
private String eventDescription;
private Date date;
private List<Speaker> speakers = new ArrayList<>();
Description:
```

Represents the Event entity in the system.
Implements a primary key eventId with auto-increment strategy.
Stores event details such as event name, description, and date.
Establishes a One-to-Many relationship with Speaker.
One event can have multiple speakers.
Relationship is mapped by the field name event.
Use appropriate Cascade Type.
Uses correct annotation to avoid infinite recursion during JSON serialization.
Includes:

Getters and setters for all fields.
File: Speaker.java (Model Class)
Variables:

private Long speakerId;
private String speakerName;
private String expertise;
private String bio;
private Event event;
Description:

Represents a Speaker entity in the system.
Establishes a Many-to-One relationship with Event.
Multiple speakers can belong to one event.
Uses Join Column with column name event_id.
Includes:

Getters and setters for all fields.
Repository Layer
File: EventRepository.java
public interface EventRepository extends JpaRepository<Event, Long>
Description:

Provides CRUD operations for Event entities.
Includes a custom JPQL query to fetch Event based on eventName.
List<Event> findAllByEventName(String eventName);
File: SpeakerRepository.java
public interface SpeakerRepository extends JpaRepository<Speaker, Long>
Description:

Provides CRUD operations for Speaker entities.
Includes a custom JPQL query to fetch Speaker based on speakerName.
List<Speaker> findBySpeakerName(String speakerName);
Service Layer
File: EventService.java
Methods:
1. public Event addEvent(Event event)

Saves a new event into the database.
Returns the saved event entity.
2. public Event getEventById(Long eventId)

Retrieves an event by ID.
orElse null.
3. public List<Event> getAllEvents(String eventName)

Retrieves events based on event name using JPQL query.
Calls the JPQL query method defined in EventRepository.
4. public Event updateEvent(Long eventId, Event event)

Updates event details if the event exists.
Returns null if the event ID is not found.
File: SpeakerService.java
Methods:
1. public Speaker addSpeaker(Speaker speaker)

Saves a new speaker into the database.
Returns the saved speaker entity.

2. public Speaker getSpeakerById(Long speakerId)

Retrieves a speaker by ID.
orElse null.
3. public List<Speaker> getAllSpeakers(String speakerName)

Retrieves speakers based on speaker name using JPQL query.
Calls the JPQL query method defined in SpeakerRepository.
4. public Speaker updateSpeaker(Long speakerId, Speaker speaker)

Updates speaker details if the speaker exists.
Returns null if the speaker ID is not found.
Controller Layer
File: EventController.java
Base URL: /events
Endpoints:
1. Create Event

Endpoint: POST /events
Method Signature:
o public Event addEvent(@RequestBody Event event)

Accepts event details in request body.
Saves and returns the event entity.
1. Get Event by ID

Endpoint: GET /events/{eventId}
Method Signature:
o public Event getEventById(@PathVariable Long eventId)

Retrieves event details by ID.
2. Get Events by Name

Endpoint: GET /events/getAllEvents/{eventName}
Method Signature:
o public ResponseEntity<?> getAllEvents(@PathVariable String eventName)

Retrieves events by event name.
Returns HTTP status OK with body as event List.
3. Update Event

Endpoint: PUT /events/{eventId}
Method Signature:
o public Event updateEvent(@PathVariable Long eventId, @RequestBody Event event)

Updates event details based on event ID.
File: SpeakerController.java
Base URL: /speakers
Endpoints:
1. Create Speaker

Endpoint: POST /speakers
Method Signature:
o public Speaker addSpeaker(@RequestBody Speaker speaker)

Accepts speaker details in request body.
Saves and returns the speaker entity.
1. Get Speaker by ID

Endpoint: GET /speakers/{speakerId}
Method Signature:
o public Speaker getSpeakerById(@PathVariable Long speakerId)

Retrieves speaker details by ID.
2. Get Speakers by Name

Endpoint: GET /speakers/getAllSpeaker/{speakerName}
Method Signature:
o public ResponseEntity<?> getAllSpeaker(@PathVariable String speakerName)

```
Retrieves speakers by speaker name.
Returns HTTP status OK with body as speaker List.
3. Update Speaker

Endpoint: PUT /speakers/{speakerId}
Method Signature:
o  public Speaker updateSpeaker(@PathVariable Long speakerId, @RequestBody Speaker speaker)

Updates speaker details based on speaker ID.
6:25
package com.edutech.eventsapplication.controller;
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.*;
import com.edutech.eventsapplication.entity.Event;
import com.edutech.eventsapplication.service.EventService;
@RestController
@RequestMapping("/events")
public class EventController {
   @Autowired
   private EventService eventService;
   @PostMapping
   public Event addEvent(@RequestBody Event event) {
       return eventService.addEvent(event);
   }
   @GetMapping("/{eventId}")
   public Event getEventById(@PathVariable Long eventId) {
       return eventService.getEventById(eventId);
   }
   @GetMapping("getAllEvents/{eventName}")
       public ResponseEntity<?> getAllEvents(@PathVariable("eventName") String eventName) {
       List<Event> event = eventService.getAllEvents(eventName);
       if (event == null || event.isEmpty() || event.size()<0) {
           return ResponseEntity.status(HttpStatus.NOT_FOUND)
                   .body("No Event found with the eventName: " + eventName);
       }
       return ResponseEntity.ok(event);
   }
   @PutMapping("/{eventId}")
   public Event updateEvent(@PathVariable Long eventId, @RequestBody Event event) {
       return eventService.updateEvent(eventId, event);
   }
}
package com.edutech.eventsapplication.controller;
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.*;
import com.edutech.eventsapplication.entity.Speaker;
import com.edutech.eventsapplication.service.SpeakerService;
@RestController
@RequestMapping("/speakers")
public class SpeakerController {
   @Autowired
   private SpeakerService speakerService;
   @PostMapping
   public Speaker addSpeaker(@RequestBody Speaker speaker) {
       return speakerService.addSpeaker(speaker);
   }
   @GetMapping("/{speakerId}")
   public Speaker getSpeakerById(@PathVariable Long speakerId) {
       return speakerService.getSpeakerById(speakerId);
   }
      @GetMapping("getAllSpeaker/{speakerName}")
       public ResponseEntity<?> getAllSpeaker(@PathVariable("speakerName") String speakerName) {
```

```
        List<Speaker> speaker = speakerService.getAllSpeakers(speakerName);
        if (speaker == null || speaker.isEmpty() || speaker.size()<0) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND)
                    .body("No Speaker found with the speakerName: " + speakerName);
        }
        return ResponseEntity.ok(speaker);
    }
    @PutMapping("/{speakerId}")
    public Speaker updateSpeaker(@PathVariable Long speakerId, @RequestBody Speaker speaker) {
        return speakerService.updateSpeaker(speakerId, speaker);
    }
}
package com.edutech.eventsapplication.entity;
import com.fasterxml.jackson.annotation.JsonIgnore;
import javax.persistence.*;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
@Entity
public class Event {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long eventId;
    private String eventName;
    private String eventDescription;
    private Date date;
    @OneToMany(mappedBy = "event", cascade = CascadeType.ALL)
    @JsonIgnore
    private List<Speaker> speakers = new ArrayList<>();
    public Long getEventId() {
        return eventId;
    }
    public void setEventId(Long eventId) {
        this.eventId = eventId;
    }
    public String getEventName() {
        return eventName;
    }
    public void setEventName(String eventName) {
        this.eventName = eventName;
    }
    public String getEventDescription() {
        return eventDescription;
    }
    public void setEventDescription(String eventDescription) {
        this.eventDescription = eventDescription;
    }
    public Date getDate() {
        return date;
    }
    public void setDate(Date date) {
        this.date = date;
    }
    public List<Speaker> getSpeakers() {
        return speakers;
    }
    public void setSpeakers(List<Speaker> speakers) {
        this.speakers = speakers;
    }
}
package com.edutech.eventsapplication.repository;
import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;
import com.edutech.eventsapplication.entity.Speaker;
@Repository
public interface SpeakerRepository extends JpaRepository<Speaker, Long> {
    @Query("SELECT c FROM Speaker c WHERE c.speakerName =:speakerName")
```

```
    List<Speaker> findBySpeckerName(String speakerName);
}
package com.edutech.eventsapplication.service;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.edutech.eventsapplication.entity.Event;
import com.edutech.eventsapplication.repository.EventRepository;
@Service
public class EventService {
    @Autowired
    private EventRepository eventRepository;
    public Event addEvent(Event event) {
        return eventRepository.save(event);
    }
    public Event getEventById(Long eventId) {
        return eventRepository.findById(eventId).orElse(null);
    }
    public List<Event> getAllEvents(String eventName) {
        return eventRepository.findAllByEventName(eventName);
    }
    public Event updateEvent(Long eventId, Event event) {
        Event existingEvent = eventRepository.findById(eventId).orElse(null);
        if (existingEvent != null) {
            event.setEventId(existingEvent.getEventId());
            return eventRepository.save(event);
        } else {
            return null;
        }
    }
}
package com.edutech.eventsapplication.service;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.edutech.eventsapplication.entity.Speaker;
import com.edutech.eventsapplication.repository.SpeakerRepository;
@Service
public class SpeakerService {
    @Autowired
    private SpeakerRepository speakerRepository;
    public Speaker addSpeaker(Speaker speaker) {
        return speakerRepository.save(speaker);
    }
    public Speaker getSpeakerById(Long speakerId) {
        return speakerRepository.findById(speakerId).orElse(null);
    }
    public List<Speaker> getAllSpeakers(String speakerName) {
        return speakerRepository.findBySpeckerName(speakerName);
    }
    public Speaker updateSpeaker(Long speakerId, Speaker speaker) {
        Speaker existingSpeaker = speakerRepository.findById(speakerId).orElse(null);
        if (existingSpeaker != null) {
            speaker.setSpeakerId(existingSpeaker.getSpeakerId());
            return speakerRepository.save(speaker);
        }
        return null;
    }
}
```

6:26

Q4) Build retail application using JPQL

You are developing a Spring Boot application to manage Suppliers and the Products they provide in a retail management system.

The system allows storing supplier details along with the products supplied by each supplier.

Each supplier can supply multiple products, while each product is associated with only one supplier.

The application must be implemented using Spring Boot, Spring Data JPA, JPQL, and RESTful APIs, following a layered architecture.

Entity Layer

File: Supplier.java (Model Class)
Variables:

private Long supplierId;
private String supplierName;
private String contact;
private List<Product> products = new ArrayList<>();
Description:
Represents the Supplier entity in the system.
Establishes a One-to-Many relationship with Product:
One supplier can have multiple products.
Relationship is mapped by the field name supplier.

Uses appropriate Cascade Type.
Uses proper annotation to avoid infinite recursion during JSON serialization.
Includes:

Getters and setters for all fields.
File: Product.java (Model Class)
Variables:

private Long productId;
private String productName;
private double price;
private String category;
private Supplier supplier;
Description:
Represents the Product entity in the system.
Establishes a Many-to-One relationship with Supplier:
Multiple products can belong to one supplier.

Uses JoinColumn with column name supplier_id.
Includes:

Getters and setters for all fields.
Repository Layer
File: SupplierRepository.java
public interface SupplierRepository extends JpaRepository<Supplier, Long>
Description:
Provides CRUD operations for Supplier entities.

Includes a custom JPQL query to retrieve suppliers sorted by supplier name in descending order.
Method:
List<Supplier> getSupplierListDesc();
File: ProductRepository.java
public interface ProductRepository extends JpaRepository<Product, Long>
Description:
Provides CRUD operations for Product entities.

Includes a custom JPQL query to retrieve products sorted by product name in descending order.
Method:
List<Product> getProductListDesc();
Service Layer
File: SupplierService.java
Methods:
public Supplier addSupplier(Supplier supplier)

Saves a new supplier into the database.
Returns the saved supplier entity.
public Supplier getSupplierById(Long supplierId)
• Retrieves supplier details based on supplier ID.
• Uses
.orElseThrow(() -> new EntityNotFoundException("Supplier not found with id"))
if the supplier does not exist.
public List<Supplier> getAllSuppliers()

Retrieves all suppliers sorted by supplier name.
Calls the JPQL query method getSupplierListDesc().
public Supplier updateSupplier(Long supplierId, Supplier supplier)

• Retrieves the existing supplier using
.orElseThrow(() -> new EntityNotFoundException("Supplier not found with id")).
• Updates supplier details and saves the changes.
• Returns the updated supplier entity.
File: ProductService.java
Methods:
public Product addProduct(Product product)

Saves a new product into the database.
Returns the saved product entity.
public Product getProductById(Long productId)
• Retrieves a product based on product ID.
• Uses
.orElseThrow(() -> new EntityNotFoundException("Product not found with id"))
if the product does not exist.
public List<Product> getAllProducts()

Retrieves all products sorted by product name.
Calls the JPQL query method getProductListDesc().
public Product updateProduct(Long productId, Product product)
• Retrieves the existing product using
.orElseThrow(() -> new EntityNotFoundException("Product not found with id")).
• Updates product details and saves the changes.
• Returns the updated product entity.
Controller Layer
File: SupplierController.java
Base URL: /suppliers
Endpoints:
1. Register Supplier

Endpoint: POST /suppliers
Method:
o public Supplier addSupplier(@RequestBody Supplier supplier)

Accepts supplier details in the request body.
Saves and returns the supplier entity.
1. Get Supplier by ID

Endpoint: GET /suppliers/{supplierId}
Method:
o public Supplier getSupplierById(@PathVariable Long supplierId)

Retrieves supplier details by ID.
2. Get All Suppliers

Endpoint: GET /suppliers
Method:
o public List<Supplier> getAllSuppliers()

Retrieves all suppliers sorted by supplier name.
3. Update Supplier

Endpoint: PUT /suppliers/{supplierId}
Method:
o public Supplier updateSupplier(@PathVariable Long supplierId,
o @RequestBody Supplier supplier)

Updates supplier details if the supplier exists.
File: ProductController.java
Base URL: /products
Endpoints:
1. Add Product

Endpoint: POST /products
Method:
o public Product addProduct(@RequestBody Product product)

Accepts product details in the request body.
Saves and returns the product entity.

1. Get Product by ID

Endpoint: GET /products/{productId}
Method:
o public Product getProductById(@PathVariable Long productId)

Retrieves product details by ID.
2. Get All Products

Endpoint: GET /products
Method:
o public List<Product> getAllProducts()

Retrieves all products sorted by product name.
3. Update Product

Endpoint: PUT /products/{productId}
Method:
o public Product updateProduct(@PathVariable Long productId,
o @RequestBody Product product)

Updates product details if the product exists.
6:28
Solution

entity/Supplier.java

```
package com.edutech.retailapplication.entity;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

@Entity
public class Supplier {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long supplierId;

    private String supplierName;
    private String contact;

    @OneToMany(mappedBy = "supplier", cascade = CascadeType.ALL)
    @JsonIgnoreProperties("supplier")
    private List<Product> products = new ArrayList<>();

    public Long getSupplierId() {
        return supplierId;
    }

    public void setSupplierId(Long supplierId) {
        this.supplierId = supplierId;
    }

    public String getSupplierName() {
        return supplierName;
    }

    public void setSupplierName(String supplierName) {
        this.supplierName = supplierName;
    }

    public String getContact() {
        return contact;
    }

    public void setContact(String contact) {
        this.contact = contact;
```

```
    }

    public List<Product> getProducts() {
        return products;
    }

    public void setProducts(List<Product> products) {
        this.products = products;
    }
}
```

---

entity/Product.java

```
package com.edutech.retailapplication.entity;

import javax.persistence.*;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

@Entity
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long productId;

    private String productName;
    private double price;
    private String category;

    @ManyToOne
    @JoinColumn(name = "supplier_id")
    @JsonIgnoreProperties("products")
    private Supplier supplier;

    public Long getProductId() {
        return productId;
    }

    public void setProductId(Long productId) {
        this.productId = productId;
    }

    public String getProductName() {
        return productName;
    }

    public void setProductName(String productName) {
        this.productName = productName;
    }

    public double getPrice() {
        return price;
    }

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

    public String getCategory() {
        return category;
    }

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

    public Supplier getSupplier() {
        return supplier;
    }
```

```
    public void setSupplier(Supplier supplier) {
        this.supplier = supplier;
    }
}
```

---

repository/SupplierRepository.java

```
package com.edutech.retailapplication.repository;

import com.edutech.retailapplication.entity.Supplier;
import org.springframework.data.jpa.repository.*;
import java.util.List;

public interface SupplierRepository extends JpaRepository<Supplier, Long> {

    @Query("SELECT s FROM Supplier s ORDER BY s.supplierName DESC")
    List<Supplier> getSupplierListDesc();
}
```

---

repository/ProductRepository.java

```
package com.edutech.retailapplication.repository;

import com.edutech.retailapplication.entity.Product;
import org.springframework.data.jpa.repository.*;
import java.util.List;

public interface ProductRepository extends JpaRepository<Product, Long> {

    @Query("SELECT p FROM Product p ORDER BY p.productName DESC")
    List<Product> getProductListDesc();
}
```

---

service/SupplierService.java

```
package com.edutech.retailapplication.service;

import com.edutech.retailapplication.entity.Supplier;
import com.edutech.retailapplication.repository.SupplierRepository;
import javax.persistence.EntityNotFoundException;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class SupplierService {

    private final SupplierRepository supplierRepository;

    public SupplierService(SupplierRepository supplierRepository) {
        this.supplierRepository = supplierRepository;
    }

    public Supplier addSupplier(Supplier supplier) {
        return supplierRepository.save(supplier);
    }

    public Supplier getSupplierById(Long supplierId) {
        return supplierRepository.findById(supplierId)
                .orElseThrow(() -> new EntityNotFoundException("Supplier not found with id"));
    }

    public List<Supplier> getAllSuppliers() {
        return supplierRepository.getSupplierListDesc();
    }

    public Supplier updateSupplier(Long supplierId, Supplier supplier) {
        Supplier existing = supplierRepository.findById(supplierId)
                .orElseThrow(() -> new EntityNotFoundException("Supplier not found with id"));

        existing.setSupplierName(supplier.getSupplierName());
        existing.setContact(supplier.getContact());
        return supplierRepository.save(existing);
```

```
    }
}
```

---

service/ProductService.java

```
package com.edutech.retailapplication.service;

import com.edutech.retailapplication.entity.Product;
import com.edutech.retailapplication.repository.ProductRepository;
import javax.persistence.EntityNotFoundException;
import org.springframework.stereotype.Service;
import java.util.List;

@Service
public class ProductService {

    private final ProductRepository productRepository;

    public ProductService(ProductRepository productRepository) {
        this.productRepository = productRepository;
    }

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

    public Product getProductById(Long productId) {
        return productRepository.findById(productId)
                .orElseThrow(() -> new EntityNotFoundException("Product not found with id"));
    }

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

    public Product updateProduct(Long productId, Product product) {
        Product existing = productRepository.findById(productId)
                .orElseThrow(() -> new EntityNotFoundException("Product not found with id"));

        existing.setProductName(product.getProductName());
        existing.setPrice(product.getPrice());
        existing.setCategory(product.getCategory());
        return productRepository.save(existing);
    }
}
```

---

controller/SupplierController.java

```
package com.edutech.retailapplication.controller;

import com.edutech.retailapplication.entity.Supplier;
import com.edutech.retailapplication.service.SupplierService;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/suppliers")
public class SupplierController {

    private final SupplierService supplierService;

    public SupplierController(SupplierService supplierService) {
        this.supplierService = supplierService;
    }

    @PostMapping
    public Supplier addSupplier(@RequestBody Supplier supplier) {
        return supplierService.addSupplier(supplier);
    }

    @GetMapping("/{supplierId}")
```

```
    public Supplier getSupplierById(@PathVariable Long supplierId) {
        return supplierService.getSupplierById(supplierId);
    }

    @GetMapping
    public List<Supplier> getAllSuppliers() {
        return supplierService.getAllSuppliers();
    }

    @PutMapping("/{supplierId}")
    public Supplier updateSupplier(@PathVariable Long supplierId,
                                   @RequestBody Supplier supplier) {
        return supplierService.updateSupplier(supplierId, supplier);
    }
}
```

---

controller/ProductController.java

```
package com.edutech.retailapplication.controller;

import com.edutech.retailapplication.entity.Product;
import com.edutech.retailapplication.service.ProductService;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/products")
public class ProductController {

    private final ProductService productService;

    public ProductController(ProductService productService) {
        this.productService = productService;
    }

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

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

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

    @PutMapping("/{productId}")
    public Product updateProduct(@PathVariable Long productId,
                                 @RequestBody Product product) {
        return productService.updateProduct(productId, product);
    }
}
```

6:29

Q5)

Develop a Spring Boot RESTful application for an Insurance Management System that manages Clients and their Insurance Policies.

The application demonstrates entity relationships (One-to-Many / Many-to-One), JPQL queries, CRUD operations, and REST API development using Spring Boot and JPA.

Important Notes for Learners

- Read all instructions carefully before starting.
- Follow the exact package name and class names as specified.
- Do not rename, delete, or modify any provided files.
- Implement code only in the required classes.
- API paths, method signatures, and JPQL queries must match exactly.
- Partial implementation may still fetch partial marks based on test coverage.

Package Structure (Mandatory)
All classes must be placed inside:
com.edutech.insuranceapplication
Files to Complete
Entity Classes
- entity/Client.java
- entity/Policy.java

Repository Interfaces
- repository/ClientRepository.java
- repository/PolicyRepository.java

Service Classes
- service/ClientService.java
- service/PolicyService.java

Controller Classes
- controller/ClientController.java
- controller/PolicyController.java

Client Entity (Client.java)
Purpose
Represents an insurance client who can subscribe to multiple policies.
Fields (Must be EXACT)
- private Long clientId; (Auto-generated, Primary Key)
- private String clientName;
- private Date dateOfBirth;
- private String contactNumber;
- private List<Policy> policies;

Relationship
- OneToMany with Policy

Requirements
- Generate constructors, getters, and setters
- Maintain proper bidirectional mapping

Policy Entity (Policy.java)
Purpose
Represents an insurance policy subscribed by a client.
Fields (Must be EXACT)
- private Long policyId; (Auto-generated, Primary Key)
- private String policyName;
- private String coverageType;
- private Client client;

Relationship
- ManyToOne with Client

Requirements
- Generate constructors, getters, and setters
- Policy must always be linked to one client

Repository Layer
PolicyRepository
- Must extend JpaRepository<Policy, Long>

Mandatory JPQL Method
List<Policy> findAllPolicySearchByPolicyName(String policyName);
- Uses JPQL to fetch policies by policy name

ClientRepository
- Must extend JpaRepository<Client, Long>

Mandatory JPQL Method
List<Client> findAllGetByClientName(String clientName);
- Uses JPQL to fetch clients by client name

Optional
Optional<Client> findByContactNumber(@Param("contactNumber") String contactNumber);
- Uses JPQL to find a client using their contact number

Service Layer
PolicyService
Implement the following methods:
- Policy addPolicy(Policy policy)
- Policy getPolicyById(Long policyId)
- List<Policy> getAllPolicies(String policyName)

- Policy updatePolicy(Long policyId, Policy policy)

_____

ClientService
Implement the following methods:

- Client addClient(Client client)
- Client getClientById(Long clientId)
- List<Client> getAllClients(String clientName)
- Client updateClientByContactNumber(String contactNumber, Client client)

_____

Controller Layer
PolicyController APIs

- POST /policies
- Add a new insurance policy
- GET /policies/{policyId}
- Fetch a policy by ID along with its client details
- GET /policies/getPolicySearch/{policyName}
- Retrieve all policies matching the policy name
- PUT /policies/{policyId}
- Update an existing policy

Method Signature

```
public Policy addPolicy(Policy policy);
public Policy getPolicyById(Long policyId);
public ResponseEntity<?> getAllPoliciesSearch(String policyName);
public Policy updatePolicy(Long policyId, Policy policy);
```

_____

ClientController APIs

- POST /clients
- Add a new client
- GET /clients/{clientId}
- Fetch client details with all associated policies
- GET /clients/getClientSearch/{clientName}
- Retrieve all clients matching the client name
- PUT /clients/contact/{contactNumber}
- Update client details using contact number

Method Signature

```
public Client addClient(Client client);
public Client getClientById(Long clientId);
public ResponseEntity<?> getAllClients(String clientName);
public ResponseEntity<Client> updateClientByContactNumber(String contactNumber, Client client);
```

_____

Constraints and Rules

- Each Policy belongs to only one Client
- Each Client can have multiple Policies
- JPQL queries are mandatory
- Use Spring Data JPA default methods where applicable
- No external databases or custom SQL queries
- API responses must include associated entities

(edited)
Aliza Shaikh 7:50 PM

SOLUTION

```
package com.edutech.insuranceapplication.controller;

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

import java.util.List;

import com.edutech.insuranceapplication.entity.Client;
import com.edutech.insuranceapplication.service.ClientService;
@RestController
@RequestMapping("/clients")
public class ClientController {

    @Autowired
    private ClientService clientService;
```

```
    @PostMapping
    public Client addClient(@RequestBody Client client) {
        return clientService.addClient(client);
    }


    @GetMapping("/{clientId}")
    public Client getClientById(@PathVariable Long clientId) {
        return clientService.getClientById(clientId);
    }


    @GetMapping("/getClientSearch/{clientName}")
    public ResponseEntity<List<Client>> getAllClients(@PathVariable String clientName) {
        return ResponseEntity.ok(clientService.getAllClients(clientName));
    }

    @PutMapping("/{clientId}")
    public Client updateClient(
            @PathVariable Long clientId,
            @RequestBody Client client) {

        return clientService.updateClient(clientId, client);
    }

    @PutMapping("/contact/{contactNumber}")
    public ResponseEntity<Client> updateClientByContactNumber(
            @PathVariable String contactNumber,
            @RequestBody Client client) {

        return ResponseEntity.ok(
                clientService.updateClientByContactNumber(contactNumber, client)
        );
    }
}
package com.edutech.insuranceapplication.controller;

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

import java.util.List;

import com.edutech.insuranceapplication.entity.Policy;
import com.edutech.insuranceapplication.service.PolicyService;

@RestController
@RequestMapping("/policies")
public class PolicyController {

    @Autowired
    private PolicyService policyService;

    @PostMapping
    public Policy addPolicy(@RequestBody Policy policy) {
        return policyService.addPolicy(policy);
    }

    @GetMapping("/{policyId}")
    public Policy getPolicyById(@PathVariable Long policyId) {
        return policyService.getPolicyById(policyId);
    }

    @GetMapping("/getPolicySearch/{policyName}")
    public ResponseEntity<List<Policy>> getAllPolicies(@PathVariable String policyName) {
        return ResponseEntity.ok(policyService.getAllPolicies(policyName));
    }
```

```
    @PutMapping("/{policyId}")
    public Policy updatePolicy(@PathVariable Long policyId,
                               @RequestBody Policy policy) {
        return policyService.updatePolicy(policyId, policy);
    }
}

package com.edutech.insuranceapplication.entity;

import javax.persistence.*;
import java.util.Date;
import java.util.List;
import java.util.ArrayList;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

@Entity
public class Client {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long clientId;

    private String clientName;

    @Temporal(TemporalType.DATE)
    private Date dateOfBirth;

    private String contactNumber;

    @OneToMany(mappedBy = "client")
    @JsonIgnore
    private List<Policy> policies = new ArrayList<>();

    public Client() {}

    public Long getClientId() {
        return clientId;
    }

    public void setClientId(Long clientId) {
        this.clientId = clientId;
    }

    public String getClientName() {
        return clientName;
    }

    public void setClientName(String clientName) {
        this.clientName = clientName;
    }

    public Date getDateOfBirth() {
        return dateOfBirth;
    }

    public void setDateOfBirth(Date dateOfBirth) {
        this.dateOfBirth = dateOfBirth;
    }

    public String getContactNumber() {
        return contactNumber;
    }

    public void setContactNumber(String contactNumber) {
        this.contactNumber = contactNumber;
    }
```

```
    public List<Policy> getPolicies() {
        return policies;
    }

    public void setPolicies(List<Policy> policies) {
        this.policies = policies;
    }
}

package com.edutech.insuranceapplication.entity;

import javax.persistence.*;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

@Entity
public class Policy {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long policyId;

    private String policyName;
    private String coverageType;

    @ManyToOne
    @JoinColumn(name = "client_id")
    @JsonIgnoreProperties("policies")
    private Client client;

    public Policy() {}

    public Long getPolicyId() {
        return policyId;
    }

    public void setPolicyId(Long policyId) {
        this.policyId = policyId;
    }

    public String getPolicyName() {
        return policyName;
    }

    public void setPolicyName(String policyName) {
        this.policyName = policyName;
    }

    public String getCoverageType() {
        return coverageType;
    }

    public void setCoverageType(String coverageType) {
        this.coverageType = coverageType;
    }

    public Client getClient() {
        return client;
    }

    public void setClient(Client client) {
        this.client = client;
    }
}

package com.edutech.insuranceapplication.repository;

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

```
import org.springframework.data.repository.query.Param;

import com.edutech.insuranceapplication.entity.Client;

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

public interface ClientRepository extends JpaRepository<Client, Long> {

    @Query("SELECT c FROM Client c WHERE c.clientName LIKE %:clientName%")
    List<Client> findAllGetByClientName(@Param("clientName") String clientName);

    @Query("SELECT c FROM Client c WHERE c.contactNumber = :contactNumber")
    Optional<Client> findByContactNumber(@Param("contactNumber") String contactNumber);
}
package com.edutech.insuranceapplication.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import com.edutech.insuranceapplication.entity.Policy;

import java.util.List;

public interface PolicyRepository extends JpaRepository<Policy, Long> {

    @Query("SELECT p FROM Policy p WHERE p.policyName LIKE %:policyName%")
    List<Policy> findAllPolicySearchByPolicyName(@Param("policyName") String policyName);
}


package com.edutech.insuranceapplication.service;

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

import javax.persistence.EntityNotFoundException;
import java.util.List;

import com.edutech.insuranceapplication.entity.Client;
import com.edutech.insuranceapplication.repository.ClientRepository;
@Service
public class ClientService {

    @Autowired
    private ClientRepository clientRepository;

    public Client addClient(Client client) {
        return clientRepository.save(client);
    }

    public Client getClientById(Long clientId) {
        return clientRepository.findById(clientId)
                .orElseThrow(() -> new EntityNotFoundException("Client not found"));
    }

    public List<Client> getAllClients(String clientName) {
        return clientRepository.findAllGetByClientName(clientName);
    }

    public Client updateClient(Long clientId, Client client) {

        Client existing = clientRepository.findById(clientId)
                .orElseThrow(() -> new EntityNotFoundException("Client not found"));

        existing.setClientName(client.getClientName());
        existing.setDateOfBirth(client.getDateOfBirth());
        existing.setContactNumber(client.getContactNumber());
```

```
        return clientRepository.save(existing);
    }
    public Client updateClientByContactNumber(String contactNumber, Client client) {

        Client existing = clientRepository.findByContactNumber(contactNumber)
                .orElseThrow(() -> new EntityNotFoundException("Client not found"));

        existing.setClientName(client.getClientName());
        existing.setDateOfBirth(client.getDateOfBirth());

        return clientRepository.save(existing);
    }
}


package com.edutech.insuranceapplication.service;

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

import javax.persistence.EntityNotFoundException;
import java.util.List;

import com.edutech.insuranceapplication.entity.Policy;
import com.edutech.insuranceapplication.repository.PolicyRepository;

@Service
public class PolicyService {

    @Autowired
    private PolicyRepository policyRepository;

    public Policy addPolicy(Policy policy) {
        return policyRepository.save(policy);
    }

    public Policy getPolicyById(Long policyId) {
        return policyRepository.findById(policyId)
                .orElseThrow(() -> new EntityNotFoundException("Policy not found"));
    }

    public List<Policy> getAllPolicies(String policyName) {
        return policyRepository.findAllPolicySearchByPolicyName(policyName);
    }

    public Policy updatePolicy(Long policyId, Policy policy) {
        Policy existing = policyRepository.findById(policyId)
                .orElseThrow(() -> new EntityNotFoundException("Policy not found"));

        existing.setPolicyName(policy.getPolicyName());
        existing.setCoverageType(policy.getCoverageType());

        return policyRepository.save(existing);
    }
}
Aliza Shaikh  8:29 PM

Q6) Build a Springboot online platform for a recruitment agency using JPQL

You are tasked with developing a Spring Boot application for a recruitment agency.
The agency wants a system where they can maintain records of Jobs and Applicants.

Each Job can have multiple Applicants, and an Applicant can apply for multiple Jobs.
You are required to develop Create and Retrieve operations for both Job and Applicant
entities using JPA and MySQL. The relationship between the two entities should be
accurately represented and managed in the database.

Your application must meet the following requirements:
```

```
Entities:

Job Entity:
Attributes:
- id: Unique identifier and should be auto generated. (type Long)
- title: Title of the job. (type String)
- description: Description of the job. (type String)
- applicants: List of applicants associated with the job. (type List<Applicant>)
Relationship: OneToMany with Applicant.

Constructors:
- public Job()
- public Job(String title, String description)

Getters and Setters

-------------------------------------

Applicant Entity:
Attributes:
- id: Unique identifier and should be auto generated. (type Long)
- name: Name of the applicant. (type String)
- resumeLink: Link to the applicant's resume. (type String)
- Job job

Relationship:
- ManyToOne with Job
- Join the column using "job_id"

Constructors:
- public Applicant()
- public Applicant(String name, String resumeLink, Job job)

Generate constructors, getters, and setters for the entity classes as per standard Java
conventions.

Repositories:

- JobRepository should extend JpaRepository
- ApplicantRepository should extend JpaRepository

Add JPQL query in ApplicantRepository to find all applicants by job id.

Method signature:
List<Applicant> retrieveApplicantsByJobId(Long jobId);

Services:

JobService:
- saveJob(Job job)
- Optional<Job> getJobById(Long id)
- List<Job> getAllJobs()

ApplicantService:
- saveApplicant(Applicant applicant)
- Optional<Applicant> getApplicantById(Long id)
- List<Applicant> getAllApplicants()
- List<Applicant> getApplicantsByJobId(Long jobId)

Controllers:

JobController:
- POST /jobs : add a new job
- GET /jobs/{id} : fetch a job by id with applicants
- GET /jobs : retrieve all jobs with applicants

ApplicantController:
- POST /applicants : add a new applicant
```

- GET /applicants/{id} : get applicant by id
- GET /applicants : fetch all applicants
- GET /job/{jobId} : fetch all applicants by jobId

Note:
- Handle entity relationships correctly
- Return correct JSON responses
- Implement JPQL query exactly as specified

Aliza Shaikh 8:36 PM

SOLUTION

entity/Job.java

```
package com.edutech.entity;

import javax.persistence.*;
import java.util.List;
import java.util.ArrayList;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

@Entity
public class Job {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;
    private String description;

    @OneToMany(mappedBy = "job", cascade = CascadeType.ALL)
    @JsonIgnoreProperties("job")
    private List<Applicant> applicants = new ArrayList<>();

    public Job() {}

    public Job(String title, String description) {
        this.title = title;
        this.description = description;
    }

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

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    public String getDescription() { return description; }
    public void setDescription(String description) { this.description = description; }

    public List<Applicant> getApplicants() { return applicants; }
    public void setApplicants(List<Applicant> applicants) { this.applicants = applicants; }
}
```

_____

entity/Applicant.java

```
package com.edutech.entity;

import javax.persistence.*;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

@Entity
public class Applicant {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String resumeLink;
```

```
    @ManyToOne
    @JoinColumn(name = "job_id")
    @JsonIgnoreProperties("applicants")
    private Job job;

    public Applicant() {}

    public Applicant(String name, String resumeLink, Job job) {
        this.name = name;
        this.resumeLink = resumeLink;
        this.job = job;
    }

    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 getResumeLink() { return resumeLink; }
    public void setResumeLink(String resumeLink) { this.resumeLink = resumeLink; }

    public Job getJob() { return job; }
    public void setJob(Job job) { this.job = job; }
}
```

---

repository/JobRepository.java

```
package com.edutech.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import com.edutech.entity.Job;

public interface JobRepository extends JpaRepository<Job, Long> {
}
```

---

repository/ApplicantRepository.java

```
package com.edutech.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import com.edutech.entity.Applicant;
import java.util.List;

public interface ApplicantRepository extends JpaRepository<Applicant, Long> {

    @Query("SELECT a FROM Applicant a WHERE a.job.id = :jobId")
    List<Applicant> retrieveApplicantsByJobId(@Param("jobId") Long jobId);
}
```

---

service/JobService.java

```
package com.edutech.service;

import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;

import com.edutech.entity.Job;
import com.edutech.repository.JobRepository;

@Service
public class JobService {

    private final JobRepository jobRepository;

    public JobService(JobRepository jobRepository) {
        this.jobRepository = jobRepository;
    }
```

```
    public Job saveJob(Job job) {
        return jobRepository.save(job);
    }

    public Optional<Job> getJobById(Long id) {
        return jobRepository.findById(id);
    }

    public List<Job> getAllJobs() {
        return jobRepository.findAll();
    }
}
```

---

service/ApplicantService.java

```
package com.edutech.service;

import org.springframework.stereotype.Service;
import java.util.List;
import java.util.Optional;

import com.edutech.entity.Applicant;
import com.edutech.repository.ApplicantRepository;

@Service
public class ApplicantService {

    private final ApplicantRepository applicantRepository;

    public ApplicantService(ApplicantRepository applicantRepository) {
        this.applicantRepository = applicantRepository;
    }

    public Applicant saveApplicant(Applicant applicant) {
        return applicantRepository.save(applicant);
    }

    public Optional<Applicant> getApplicantById(Long id) {
        return applicantRepository.findById(id);
    }

    public List<Applicant> getAllApplicants() {
        return applicantRepository.findAll();
    }

    public List<Applicant> getApplicantsByJobId(Long jobId) {
        return applicantRepository.retrieveApplicantsByJobId(jobId);
    }
}
```

---

controller/JobController.java

```
package com.edutech.controller;

import org.springframework.web.bind.annotation.*;
import org.springframework.http.ResponseEntity;
import java.util.List;

import com.edutech.entity.Job;
import com.edutech.service.JobService;

@RestController
@RequestMapping("/jobs")
public class JobController {

    private final JobService jobService;

    public JobController(JobService jobService) {
        this.jobService = jobService;
    }
```

```
    @PostMapping
    public ResponseEntity<Job> createJob(@RequestBody Job job) {
        return ResponseEntity.ok(jobService.saveJob(job));
    }

    @GetMapping
    public ResponseEntity<List<Job>> getAllJobs() {
        return ResponseEntity.ok(jobService.getAllJobs());
    }

    @GetMapping("/{id}")
    public ResponseEntity<Job> getJobById(@PathVariable Long id) {
        return jobService.getJobById(id)
                .map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }
}
```

---

controller/ApplicantController.java

```
package com.edutech.controller;

import org.springframework.web.bind.annotation.*;
import org.springframework.http.ResponseEntity;
import java.util.List;

import com.edutech.entity.Applicant;
import com.edutech.service.ApplicantService;

@RestController
@RequestMapping("/applicants")
public class ApplicantController {

    private final ApplicantService applicantService;

    public ApplicantController(ApplicantService applicantService) {
        this.applicantService = applicantService;
    }

    @PostMapping
    public ResponseEntity<Applicant> createApplicant(@RequestBody Applicant applicant) {
        return ResponseEntity.ok(applicantService.saveApplicant(applicant));
    }

    @GetMapping
    public ResponseEntity<List<Applicant>> getAllApplicants() {
        return ResponseEntity.ok(applicantService.getAllApplicants());
    }

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

    @GetMapping("/job/{jobId}")
    public ResponseEntity<List<Applicant>> getApplicantsByJobId(@PathVariable Long jobId) {
        return ResponseEntity.ok(applicantService.getApplicantsByJobId(jobId));
    }
}
```

Aliza Shaikh 8:46 PM

Q7) Build a Springboot online platform for a telecommunications company using JPQL
Objective
Develop a Spring Boot RESTful application to manage Customers and the Services they have subscribed to.
The application demonstrates One-to-Many / Many-to-One relationships, Spring Data JPA, JPQL, and REST API development using a layered architecture.

Important Notes for Learners

- Read all instructions carefully before starting.
- Follow the exact package name, class names, and method signatures.
- Do not rename, delete, or modify any provided files.
- Write code only in the required classes.
- API paths and response structures must match the test cases.

Package Structure (Mandatory)

All classes must be placed inside: com.edutech

Files to Complete

Entity Layer

- entity/Customer.java
- entity/Service.java

Repository Layer

- repository/CustomerRepository.java
- repository/ServiceRepository.java

Service Layer

- service/CustomerService.java
- service/ServiceService.java

Controller Layer

- controller/CustomerController.java
- controller/ServiceController.java

Customer Entity (Customer.java)

Purpose

Represents a customer who can subscribe to multiple services.

Fields (Must be EXACT)

- private Long id;
- private String name;
- private String email;
- private List<Service> services;

Relationship

- OneToMany relationship with Service
- Mapped by field name customer
- Use appropriate Cascade Type
- Use JSON annotation to avoid infinite recursion

Requirements

- Default constructor
- Getters and setters for all fields

Service Entity (Service.java)

Purpose

Represents a service subscribed by a customer.

Fields (Must be EXACT)

- private Long id; (GenerationType.AUTO)
- private String name;
- private String description;
- private Customer customer;

Relationship

- ManyToOne relationship with Customer
- Use @JoinColumn(name = "customer_id", nullable = false)

Requirements

- Default constructor
- Getters and setters for all fields

Repository Layer

CustomerRepository

Must extend:

JpaRepository<Customer, Long>

Mandatory JPQL Method

List<Customer> retrieveCustomersByServiceId(Long serviceId);

- Retrieves customers based on service ID
- Uses JPQL JOIN between Customer and Service

Additional Method (Required by Tests)

Optional<Customer> findByEmail(String email);

ServiceRepository

Must extend:

```
JpaRepository<Service, Long>
List<Service> findByCustomerId(@Param("customerId") Long customerId);
```

```
Service Layer
CustomerService
Variable:
private final CustomerRepository customerRepository;
Repository must be injected using constructor injection
public CustomerService(CustomerRepository customerRepository)
Methods to implement:
•       List<Customer> getAllCustomers()
•       Optional<Customer> getCustomerById(Long id)
•       Customer createCustomer(Customer customer)
•       Customer updateCustomerByEmail(String email, Customer customer)
•       public List<com.edutech.entity.Service> getServicesByCustomerEmail(String email)
```

```
ServiceService
Repository must be injected using constructor injection.
Variable:
private final ServiceRepository serviceRepository;
private final CustomerRepository customerRepository;
Methods to implement:
•       public ServiceService(ServiceRepository serviceRepository, CustomerRepository
customerRepository)
•       List<Service> getAllServices()
•       Optional<Service> getServiceById(Long id)
•       Service createService(Service service)
•       List<Service> getServicesByCustomerId(Long customerId)
•       public List<com.edutech.entity.Service> updateServicesByCustomerId(Long customerId,
com.edutech.entity.Service updatedService)
```

```
Controller Layer
CustomerController
Base URL: /customers
Endpoints (Bulleted – No Table)
GET /customers
Retrieve all customers
Returns the list of customers with HTTP 200 OK
GET /customers/{id}
Retrieve customer by ID
Returns the customer if found, else HTTP 404 NOT FOUND
POST /customers
Create a new customer
Returns the saved customer with HTTP 200 OK
PUT /customers/email/{email}
Update customer using email address
Returns the updated customer with HTTP 200 OK
GET /customers/email/{email}/services
Retrieve all services for a customer using email
Returns the list of services with HTTP 200 OK (empty list if no services exist)
Variable:
private final CustomerService customerService;
Method Signatures:
•       public CustomerController(CustomerService customerService)
•       public List<Customer> getAllCustomers();
•       public Customer getCustomerById(Long id);
•       public Customer createCustomer(Customer customer);
•       public Customer updateCustomerByEmail(String email, Customer customer);
•       public List<Service> getServicesByCustomerEmail(String email);
```

```
ServiceController
Base URL: /services
POST /services
Create a new service
Returns saved service with HTTP 200 OK
GET /services
Retrieve all services
Returns list of services with HTTP 200 OK
GET /services/{id}
```

Retrieve service by ID
Returns service if found, else 404 NOT FOUND
PUT /services/customer/{customerId}
Update all services for a given customer
Returns updated list of services with HTTP 200 OK
GET /services/customer/{customerId}
Retrieve all services for a given customer
Returns empty list if no services exist with HTTP 200 OK
Variable:
private final ServiceService serviceService;
Method Signature:
- public ServiceController(ServiceService serviceService)
- public List<Service> getAllServices();
- public Optional<Service> getServiceById(Long id);
- public Service createService(Service service);
- public List<Service> updateServicesByCustomerId(Long customerId, Service service);
- public List<Service> getServicesByCustomerId(Long customerId);

_____
Constraints and Rules
- One customer can have multiple services
- Each service must belong to exactly one customer
- Use Spring Data JPA for persistence
- Use JPQL only where specified
- No global exception handler required
- No database configuration changes
- Do not alter API paths or response formats

_____
Final Hint
If:
- Entity relationships are correctly mapped
- JPQL query in CustomerRepository is implemented
- REST endpoints match the test cases
- Services correctly delegate to repositories

Then all automated test cases will pass, including update and JPQL-based retrieval tests.
Aliza Shaikh 9:21 PM

Q8) Content Management using JPQL

You are developing a Spring Boot application for a content management system (CMS). This application will facilitate content creators to publish, update, and manage articles across various categories.

Each article can belong to one category, while a category might have multiple articles.

Requirements:

Entities:

1. Article Entity
Attributes:
a. articleId: Unique identifier and should be auto-generated. (type Long)
b. title: Title of the article. (type String)
c. content: Content of the article. (type String)
d. publicationDate: Date of publication. (type Date)
e. authorName: Name of the content creator. (type String)
f. category: category of article (type Category).
Relationship: ManyToOne with Category.

2. Category Entity:
Attributes:
a. categoryId: Unique identifier and should be auto-generated. (type Long)
b. categoryName: Name of the category. (type String)
c. articles: List of articles under this category. (type List<Article>)
Relationship: OneToMany with Article.
Mapped by "category"

-> generate getters, and setters for the entity classes as per standard Java conventions.
-> For example: getArticleId(),setArticleId(Long articleId) etc.

Repositories:

1. ArticleRepository: Should extend JpaRepository.
2. CategoryRepository: Should extend JpaRepository.

Add jpql query in the ArticleRepository to sort all articles by authorName and using id sort in descending order using findAllBasedOnSorting method.
The method signature should be:
-> List<Article> retrieveArticlesByAuthorName(String authorName);
-> List<Article> sortByAuthorNameAndId();

Add jpql query in the CategoryRepository to sort all articles by authorName and using id in descending order.
The method signature should be:
List<Category> sortByCategoryNameAndId();

You need to add above method signature in ArticleRepository and add JPQL query to find all articles by authorName for other methods, you can use the default methods provided by JpaRepository.

Services:

CategoryService:
Variable:
private CategoryRepository categoryRepository;
Offers methods to
1. Category addCategory(Category category): Adds a new category to the database.
2. void deleteCategory(Long categoryId): Delete Category and its child (articles) based on ONDELETECASCADE settings.
3. List<Category> getAllCategories(): Returns a list of all categories.

ArticleService:
Variable:
private ArticleRepository articleRepository;
Provides methods to
1. Article addArticle(Article article): Adds a new article to the database.
2. List<Article> getAllArticles(): Returns a list of all articles.

Controllers:

CategoryController: Manages HTTP requests related to the Category entity:
1. POST /categories: to add a new category.
details: The api should accept Category object in the request body and return the created category.

2. DELETE /categories/{categoryId}: to delete a Category by their ID and the corresponding Articles too
details: The api should delete the category and the articles associated with it specified by the categoryId.

3. GET /categories: to retrieve based on filter categories.
details: The api should return a categories.

Variables:
private CategoryService categoryService;

Method Signature:
Category addCategory(Category category);
void deleteCategory(Long categoryId);
List<Category> getAllCategories();

ArticleController: Manages HTTP requests related to the Article entity
1. POST /articles: to create a new article.
details: The api should accept Article object in the request body and return the created article.

2. GET /articles: to retrieve all articles.
details: The api should return a list of all articles and its associated category.

Method Signature:

```
Article addArticle(Article article);
List<Article> getAllArticles();

Note:
• Make sure to handle the relationship between entities correctly
• Make sure you return the correct response as mentioned in the details section of the API.
• Make sure you implement the JPQL query method in the ArticleRepository as mentioned in the
repository section.
9:21
Q9)
Tourism Management using JPQL
You are developing a Spring Boot application to manage Tourists and their Travel Bookings.
The system allows storing tourist details along with the bookings they have made. Each tourist can
have multiple bookings, while each booking is associated with only one tourist.
The application must be implemented using Spring Boot, Spring Data JPA, JPQL, and RESTful APIs,
following a layered architecture.
_____
Entity Layer
File: Tourist.java (Model Class)
Variables:
•       private Long touristId;
•       private String fullName;
•       private String email;
•       private String passportNumber;
•       private List<Booking> bookings = new ArrayList<>();
Description:
•       Represents the Tourist entity in the system.
•       Establishes a One-to-Many relationship with Booking:
o       One tourist can have multiple bookings.
o       Relationship is mapped by the field name tourist.
o       Uses appropriate Cascade Type with orphanRemoval.
•       Use proper annotation to avoid infinite recursion during JSON serialization.
Includes:
•       Getters and setters for all fields.
_____
File: Booking.java (Model Class)
Variables:
•       private Long bookingId;
•       private Date bookingDate;
•       private String destinationName;
•       private double packagePrice;
•       private int duration;
•       private Tourist tourist;
Description:
•       Represents a Booking entity in the system.
•       Establishes a Many-to-One relationship with Tourist:
o       Multiple bookings can belong to one tourist.
o       Uses Join Column with column name tourist_id.
Includes:
•       Getters and setters for all fields.
_____
Repository Layer
File: TouristRepository.java
public interface TouristRepository extends JpaRepository<Tourist, Long>
Description:
•       Provides CRUD operations for Tourist entities.
•       Includes a custom JPQL query to retrieve Tourist sorted by fullName in descending order.
Method Signature:
List<Tourist> findAllByTouristIdAndName();
_____
File: BookingRepository.java
public interface BookingRepository extends JpaRepository<Booking, Long>
Description:
•       Provides CRUD operations for Booking entities.
•       Includes custom JPQL queries:
o       To retrieve all Booking based on destinationName.
o       To retrieve all Booking sorted by destinationName in descending order.
Method Signatures:
List<Booking> retrieveAllBookingsByDestination(String destinationName);
```

```
List<Booking> findAllSortByBookIdAndName();
_____________________________________________
Service Layer
File: TouristService.java
Variable:
private TouristRepository touristRepository;
Methods:
public Tourist registerTourist(Tourist tourist)
•       Saves a new tourist into the database.
•       Returns the saved tourist entity.
public void deleteTourist(Long touristId)
•       Deletes a tourist based on tourist ID.
public List<Tourist> getAllTourists()
•       Retrieves all tourists sorted by full name.
•       Calls the JPQL query method defined in TouristRepository that is
findAllByTouristIdAndName().
_____________________________________________
File: BookingService.java
Variable:
private BookingRepository bookingRepository;
Methods:
public Booking createBooking(Booking booking)
•       Saves a new booking into the database.
•       Returns the saved booking entity.
public List<Booking> getAllBookings()
•       Retrieves all bookings sorted by destination name.
•       Calls the JPQL query method defined in BookingRepository that is
findAllSortByBookIdAndName().
_____________________________________________
Controller Layer
File: TouristController.java
Variable:
private TouristService touristService;
Base URL: /tourists
Endpoints:
1. Register Tourist
•       Endpoint: POST /tourists
•       Method Signature:
public Tourist registerTourist(@RequestBody Tourist tourist)
•       Accepts tourist details in request body.
•       Saves and returns the tourist entity.
_____________________________________________
2. Delete Tourist
•       Endpoint: DELETE /tourists/{touristId}
•       Method Signature:
public void deleteTourist(@PathVariable Long touristId)
•       Deletes a tourist based on tourist ID.
_____________________________________________
3. Get All Tourists
•       Endpoint: GET /tourists
•       Method Signature:
public List<Tourist> getAllTourists()
•       Retrieves all tourists sorted by full name.
_____________________________________________
File: BookingController.java
Variable:
private BookingService bookingService;
Base URL: /bookings
Endpoints:
1. Create Booking
•       Endpoint: POST /bookings
•       Method Signature:
public Booking createBooking(@RequestBody Booking booking)
•       Accepts booking details in request body.
•       Saves and returns the booking entity.
_____________________________________________
2. Get All Bookings
•       Endpoint: GET /bookings
•       Method Signature:
```

public List<Booking> getAllBookings()
- Retrieves all bookings sorted by destination name.

9:22
Q10) Seating Arrangements System using JPQL
You are developing a Spring Boot application to manage Guests and their Seats.
The system allows storing guest details along with the seats allocated to them.
Each guest can be assigned multiple seats, while each seat is associated with only one guest.
The application must be implemented using Spring Boot, Spring Data JPA, JPQL, and RESTful APIs, following a layered architecture.

_____

Entity Layer
File: Guest.java (Model Class)
Variables:
- private Long guestId;
- private String guestName;
- private String guestCategory;
- private List<Seat> seats;

Description:
- Represents the Guest entity in the system.
- Implements guestId as a primary key with auto-increment strategy.
- Stores guest details such as name and category.
- Establishes a One-to-Many relationship with Seat:
  - One guest can have multiple seats.
  - Relationship is mapped by the field name "guest" in the Seat entity.
  - Uses CascadeType.ALL.
- Uses @JsonIgnoreProperties("guest") to avoid infinite recursion during JSON serialization.

Includes:
- Default and parameterized constructors.
- Getters and setters for all fields.

_____

File: Seat.java (Model Class)
Variables:
- private Long seatId;
- private String seatNumber;
- private String seatType;
- private boolean isOccupied;
- private Guest guest;

Description:
- Represents the Seat entity in the system.
- Implements seatId as a primary key with auto-increment strategy.
- Stores seat details such as number, type, and occupancy status.
- Establishes a Many-to-One relationship with Guest:
  - Multiple seats can belong to one guest.
  - Uses @JoinColumn with column name "guest_id".
- Uses @JsonIgnoreProperties("seats") to prevent infinite recursion.

Includes:
- Default and parameterized constructors.
- Getters and setters for all fields.

_____

Repository Layer
File: GuestRepository.java
public interface GuestRepository extends JpaRepository<Guest, Long>
Description:
- Provides CRUD operations for Guest entities.
- Includes a custom JPQL query to fetch guests by guest name.

Method Signature:
List<Guest> findAllByGuestName(String guestName);

_____

File: SeatRepository.java
public interface SeatRepository extends JpaRepository<Seat, Long>
Description:
- Provides CRUD operations for Seat entities.
- Includes custom JPQL queries to:
  - Fetch seats based on seat number.
  - Fetch a seat based on seat ID.

Method Signatures:
List<Seat> findAllBySeatNumber(String seatNumber);
Seat findSeatById(Long seatId);

```
_____________________________________________
Service Layer
File: GuestService.java
Variable:
•       private GuestRepository guestRepository;
Methods:
1.      public Guest addGuest(Guest guest)
o       Saves a new guest into the database.
o       Ensures that each seat is correctly associated with the guest.
2.      public List<Guest> getAllGuests(String guestName)
o       Retrieves guests based on guest name using JPQL query.
o       Calls the JPQL method defined in the repository.
3.      public Guest updateGuest(Long guestId, Guest guest)
o       Updates guest details if the guest exists.
o       Returns null if the guest ID is not found.
4.      public void deleteGuest(Long guestId)
o       Deletes a guest by ID.
_____________________________________________
File: SeatService.java
Variable:
•       private SeatRepository seatRepository;
Methods:
1.      public Seat addSeat(Seat seat)
o       Saves a new seat into the database.
2.      public List<Seat> getAllSeats(String seatNumber)
o       Retrieves seats based on seat number using JPQL query.
3.      public Seat updateSeat(Long seatId, Seat seat)
o       Updates seat details if the seat exists.
o       Returns null if the seat ID is not found.
4.      public void deleteSeat(Long seatId)
o       Deletes a seat by ID.
_____________________________________________
Controller Layer
File: GuestController.java
Variable:
•       private GuestService guestService;
Base URL: /guests
Endpoints:
1.      Create Guest
o       End Point: POST /guests
o       Method Signature:
o       public Guest createGuest(@RequestBody Guest guest)
o       Accepts guest details in request body.
o       Saves and returns the guest entity.
2.      Get Guests by Name
o       End Point: GET /guests/{guestName}
o       Method Signature:
o       public List<Guest> getGuests(@PathVariable String guestName)
o       Retrieves guests based on guest name.
3.      Update Guest
o       End Point: PUT /guests/{guestId}
o       Method Signature:
o       public Guest updateGuest(@PathVariable Long guestId, @RequestBody Guest guest)
o       Updates guest details based on guest ID.
4.      Delete Guest
o       End Point: DELETE /guests/{guestId}
o       Method Signature:
o       public String deleteGuest(@PathVariable Long guestId)
o       Deletes a guest by ID.
o       Returns message:
o       "Guest with ID {guestId} has been deleted."
_____________________________________________
File: SeatController.java
Variable:
•       private SeatService seatService;
Base URL: /seats
Endpoints:
1.      Create Seat
o       End Point: POST /seats
```

o Method Signature:
o public Seat createSeat(@RequestBody Seat seat)
o Accepts seat details in request body.
o Saves and returns the seat entity.
2. Get Seats by Number
o End Point: GET /seats/{seatNumber}
o Method Signature:
o public List<Seat> getSeats(@PathVariable String seatNumber)
o Retrieves seats based on seat number.
3. Update Seat
o End Point: PUT /seats/{seatId}
o Method Signature:
o public Seat updateSeat(@PathVariable Long seatId, @RequestBody Seat seat)
o Updates seat details if seat exists.
4. Delete Seat
o End Point: DELETE /seats/{seatId}
o Method Signature:
o public String deleteSeat(@PathVariable Long seatId)
o Deletes a seat by ID.
o Returns message:
o "Seat with ID {seatId} has been deleted."
9:22
Q11) Bag Management System using JPQL
Build an Owner–Bag Management Application using Spring Boot and JPQL
You are developing a Spring Boot application to manage Owners and their Bags.
The system allows storing owner details along with the bags they own. Each owner can possess multiple bags, while each bag is associated with only one owner.
The application must be implemented using Spring Boot, Spring Data JPA, JPQL, and RESTful APIs, following a layered architecture.

_____

Entity Layer
File: Owner.java (Model Class)
Variables:
• private Long ownerId;
• private String ownerName;
• private Date dateOfBirth;
• private String contactNumber;
• private List<Bag> bags;
Description:
• Represents the Owner entity in the system.
• Use appropriate Temporal annotation mention TemporalType is Date for variable dateOfBirth.
• Establishes a One-to-Many relationship with Bag:
o One owner can own multiple bags.
o Relationship is mapped by the field name "owner" in the Bag entity.
o And mention cascade type also.
• Uses proper @JsonIgnoreProperties annotations "owner" to avoid infinite recursion during serialization.
Includes:
• Default and parameterized constructors.
• Getters and setters for all fields.

_____

File: Bag.java (Model Class)
Variables:
• private Long bagId;
• private String bagType;
• private String brand;
• private Owner owner;
Description:
• Represents a Bag entity in the system.
• Implements a primary key bagId with auto-increment strategy.
• Stores bag details such as type and brand.
• Establishes a Many-to-One relationship with Owner:
o Multiple bags can belong to one owner.
o Uses JoinColumn with column name "owner_id".
• Uses @JsonIgnoreProperties annotations on "bags" to prevent infinite recursion.
Includes:
• Default and parameterized constructors.
• Getters and setters for all fields.

_____

Repository Layer
File: OwnerRepository.java
public interface OwnerRepository extends JpaRepository<Owner, Long>
Description:
- Provides CRUD operations for Owner entities.
- Includes a custom JPQL query to fetch Owner by ownerName.

Method Signature:
List<Owner> findAllGetByOwnerName(String ownerName);

File: BagRepository.java
public interface BagRepository extends JpaRepository<Bag, Long>
Description:
- Provides CRUD operations for Bag entities.
- Includes a custom JPQL query to fetch Bag based on bagType.

Method Signature:
List<Bag> findAllBagSearchByBagType(String bagType);

Service Layer
File: OwnerService.java
Variable:
private OwnerRepository ownerRepository;
Methods:
1. public Owner addOwner(Owner owner)
   - Saves a new owner into the database.
   - Ensures that each bag is correctly associated with the owner.
2. public Owner getOwnerById(Long ownerId)
   - Retrieves an owner by ID.
   - orElse null.
3. public List<Owner> getAllOwners(String ownerName)
   - Retrieves owners based on owner name using JPQL query.

Note:
Here you need to call JPQL Queries method which you written in Repository that return values.

4. public Owner updateOwner(Long ownerId, Owner owner)
   - Updates owner details if the owner exists.
   - Returns null if the owner ID is not found.
5. public String deleteOwner(Long ownerId)
   - Deletes an owner by ID.
   - Returns appropriate success or failure message.
   - Check condition if owner details perform deletion operation and return the message
"Owner is deleted successfully !!"
or if owner details not found
"Owner not found with " + ownerId

File: BagService.java
Variable:
private BagRepository bagRepository;
Methods:
1. public Bag addBag(Bag bag)
   - Saves a new bag into the database.
2. public Bag getBagById(Long bagId)
   - Retrieves a bag by ID.
   - orElse null.
3. public List<Bag> getAllBags(String bagType)
   - Note: Here you need to call JPQL Queries method which you written in Repository that return values.
4. public Bag updateBag(Long bagId, Bag bag)
   - Updates bag details if the bag exists.
   - Returns null if the bag ID is not found.
5. public void deleteBag(Long bagId)
   - Deletes a bag by ID.

Controller Layer
File: OwnerController.java
Variable:
private OwnerService ownerService;
Base URL: /owners
Endpoints:
1. Create Owner
   - End Point: POST /owners

o Method Signature:
o public Owner createOwner(@RequestBody Owner owner)
o Accepts owner details in request body.
o Saves and returns the owner entity.
2. Get Owner by ID
o End Point: GET /owners/{ownerId}
o Method Signature:
o public Owner getOwnerById(@PathVariable Long ownerId)
o Retrieves owner details by ID.
3. Get Owners by Name
o End Point: GET /owners/getOwnerSearch/{ownerName}
o Method Signature:
o public List<Owner> getOwnersByName(@PathVariable String ownerName)
o Retrieves owners by owner name.
4. Update Owner
o End Point: PUT /owners/{ownerId}
o Method Signature:
o public Owner updateOwner(@PathVariable Long ownerId, @RequestBody Owner owner)
o Updates owner details based on owner ID.
5. Delete Owner
o End Point: DELETE /owners/{ownerId}
o Method Signature:
o public String deleteOwner(@PathVariable Long ownerId)
o Deletes an owner by ID.

_____

File: BagController.java
Variable:
private BagService bagService;
Base URL: /bags
Endpoints:
1. Create Bag
o End Point: POST /bags
o Method Signature:
o public Bag createBag(@RequestBody Bag bag)
o Accepts bag details in request body.
o Saves and returns the bag entity.
2. Get Bag by ID
o End Point: GET /bags/{bagId}
o Method Signature:
o public Bag getBagById(@PathVariable Long bagId)
o Retrieves bag details by ID.
3. Get Bags by Type
o End Point: GET /bags/getBagSearch/{bagType}
o Method Signature:
o public List<Bag> getBagsByType(@PathVariable String bagType)
o Retrieves bags based on bag type.
4. Update Bag
o End Point: PUT /bags/{bagId}
o Method Signature:
o public Bag updateBag(@PathVariable Long bagId, @RequestBody Bag bag)
o Updates bag details if bag exists.
5. Delete Bag
o End Point: DELETE /bags/{bagId}
o Method Signature:
o public String deleteBag(@PathVariable Long bagId)
o Deletes a bag by ID.
o Finally return message
"Bag with ID " + bagId + " has been deleted."