**22IT901**

**REST Application Development Using Spring Boot and JPA**

**(Lab Integrated)**

**UNIT 3**

**1. Develop a Web Application for Pagination and Sorting of Children Details Using RESTful APIs**

**Aim:**

To develop a web application for pagination and sorting of children details using RESTful APIs, implementing POST and GET operations.

**Algorithm:**

1. Set up a Spring Boot project with MySQL and JPA dependencies.

2. Create an entity class for `Child` that maps to a database table.

3. Implement sorting and pagination using Spring Data JPA's `Pageable` feature.

4. Create a service layer to handle pagination and sorting logic.

5. Create a controller layer for handling the API requests and responses.

6. Test the application using Postman.

**Project Structure:**

src/main/java/com/example/childrenapi/

├── controller

│ └── ChildController.java

├── model

│ └── Child.java

├── repository

│ └── ChildRepository.java

├── service

│ └── ChildService.java

└── ChildrenApiApplication.java

**Code:**

**Child.java (Model)**

package com.example.childrenapi.model;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

@Entity

public class Child {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private int age;

public Child() {}

public Child(String name, int age) {

this.name = name;

this.age = age;

}

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 int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

}

**ChildRepository.java (Repository)**

package com.example.childrenapi.repository;

import com.example.childrenapi.model.Child;

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

import org.springframework.stereotype.Repository;

@Repository

public interface ChildRepository extends JpaRepository<Child, Long> {}

**ChildService.java (Service Layer)**

package com.example.childrenapi.service;

import com.example.childrenapi.model.Child;

import com.example.childrenapi.repository.ChildRepository;

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

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.stereotype.Service;

@Service

public class ChildService {

@Autowired

private ChildRepository repository;

public Page<Child> getAllChildren(Pageable pageable) {

return repository.findAll(pageable);

}

public Child saveChild(Child child) {

return repository.save(child);

}

}

**ChildController.java (Controller)**

package com.example.childrenapi.controller;

import com.example.childrenapi.model.Child;

import com.example.childrenapi.service.ChildService;

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

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Sort;

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/api/children")

public class ChildController {

@Autowired

private ChildService service;

@PostMapping

public Child addChild(@RequestBody Child child) {

return service.saveChild(child);

}

@GetMapping

public Page<Child> getChildren(

@RequestParam int page,

@RequestParam int size,

@RequestParam String sortBy) {

return service.getAllChildren(PageRequest.of(page, size, Sort.by(sortBy)));

}

}

**Result:**

Thus The program to develop a Web Application for Pagination and Sorting of Children Details Using RESTful APIs was successfully executed.

**2. Create a Web Application for Managing Person Details Using JPA Methods via RESTful APIs**

**Aim:**

To develop a web application for managing person details using JPA methods via RESTful APIs, enabling POST and GET operations.

**Algorithm:**

1. Set up a Spring Boot project with JPA and MySQL dependencies.

2. Create a `Person` entity class that maps to a database table.

3. Create repository interfaces extending `JpaRepository` for CRUD operations.

4. Develop service and controller layers to handle POST and GET operations.

5. Test the application using Postman.

**Project Structure:**

src/main/java/com/example/personapi/

├── controller

│ └── PersonController.java

├── model

│ └── Person.java

├── repository

│ └── PersonRepository.java

├── service

│ └── PersonService.java

└── PersonApiApplication.java

**Code:**

**Person.java (Model)**

package com.example.personapi.model;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

@Entity

public class Person {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String firstName;

private String lastName;

private String email;

private int age;

public Person() {}

public Person(String firstName, String lastName, String email, int age) {

this.firstName = firstName;

this.lastName = lastName;

this.email = email;

this.age = age;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

}

**PersonRepository.java (Repository)**

package com.example.personapi.repository;

import com.example.personapi.model.Person;

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

import org.springframework.stereotype.Repository;

@Repository

public interface PersonRepository extends JpaRepository<Person, Long> {}

**PersonService.java(Service Layer)**

package com.example.personapi.service;

import com.example.personapi.model.Person;

import com.example.personapi.repository.PersonRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class PersonService {

@Autowired

private PersonRepository repository;

public Person savePerson(Person person) {

return repository.save(person);

}

public List<Person> getAllPersons() {

return repository.findAll();

}

}

**PersonController.java(Controller)**

package com.example.personapi.controller;

import com.example.personapi.model.Person;

import com.example.personapi.service.PersonService;

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

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/persons")

public class PersonController {

@Autowired

private PersonService service;

@PostMapping

public Person addPerson(@RequestBody Person person) {

return service.savePerson(person);

}

@GetMapping

public List<Person> getAllPersons() {

return service.getAllPersons();

}

}

**Result:**

Thus The program to create a Web Application for Managing Person Details Using JPA Methods via RESTful APIs.

**3. Retrieve Person Details Using JPQL with Conditions**

**Aim:**

To develop a web application that retrieves person details using JPQL with conditions for names starting or ending with specific patterns.

**Algorithm:**

1. Set up a Spring Boot project with JPA and MySQL dependencies.

2. Create a `Person` entity class and repository interface.

3. Define custom JPQL queries using the `@Query` annotation to search by name patterns.

4. Develop service and controller layers to handle JPQL queries.

5. Test the application using Postman.

**Project Structure:**

src/main/java/com/example/personjpql/

├── controller

│ └── PersonController.java

├── model

│ └── Person.java

├── repository

│ └── PersonRepository.java

├── service

│ └── PersonService.java

└── PersonJPQLApplication.java

**Code:**

PersonRepository.java` (Repository with JPQL Queries)

package com.example.personjpql.repository;

import com.example.personjpql.model.Person;

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

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

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

import java.util.List;

public interface PersonRepository extends JpaRepository<Person, Long> {

@Query("SELECT p FROM Person p WHERE p.firstName LIKE :prefix%")

List<Person> findByFirstNameStartingWith(@Param("prefix") String prefix);

@Query("SELECT p FROM Person p WHERE p.lastName LIKE %:suffix")

List<Person> findByLastNameEndingWith(@Param("suffix") String suffix);

}

**PersonService.java (Service Layer)**

package com.example.personjpql.service;

import com.example.personjpql.model.Person;

import com.example.personjpql.repository.PersonRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class PersonService {

@Autowired

private PersonRepository repository;

public List<Person> getPersonsByFirstNamePrefix(String prefix) {

return repository.findByFirstNameStartingWith(prefix);

}

public List<Person> getPersonsByLastNameSuffix(String suffix) {

return repository.findByLastNameEndingWith(suffix);

}

}

**PersonController.java (Controller)**

package com.example.personjpql.controller;

import com.example.personjpql.model.Person;

import com.example.personjpql.service.PersonService;

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

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/persons")

public class PersonController {

@Autowired

private PersonService service;

@GetMapping("/firstname")

public List<Person> getPersonsByFirstNamePrefix(@RequestParam String prefix) {

return service.getPersonsByFirstNamePrefix(prefix);

}

@GetMapping("/lastname")

public List<Person> getPersonsByLastNameSuffix(@RequestParam String suffix) {

return service.getPersonsByLastNameSuffix(suffix);

}

}

**Result:**

Thus The program to retrieve Person Details Using JPQL with Conditions.

**4. Build a Web Application for Managing Person Details Using Custom JPQL Queries via RESTful APIs**

**Aim:**

To develop a web application for managing person details using custom JPQL queries via RESTful APIs, supporting POST and GET operations.

**Algorithm:**

1. Set up a Spring Boot project with JPA and MySQL dependencies.

2. Create a `Person` entity class and repository interface.

3. Define custom JPQL queries using the `@Query` annotation to handle complex queries.

4. Develop service and controller layers to handle POST and GET requests.

5. Test the application using Postman or another REST client.

**Project Structure:**

src/main/java/com/example/personcustomjpql/

├── controller

│ └── PersonController.java

├── model

│ └── Person.java

├── repository

│ └── PersonRepository.java

├── service

│ └── PersonService.java

└── PersonCustomJPQLApplication.java

**Code:**

**Person.java (Model)**

package com.example.personcustomjpql.model;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

@Entity

public class Person {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String firstName;

private String lastName;

private String email;

private int age;

public Person() {}

public Person(String firstName, String lastName, String email, int age) {

this.firstName = firstName;

this.lastName = lastName;

this.email = email;

this.age = age;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

}

**PersonRepository.java (Repository)**

package com.example.personcustomjpql.repository;

import com.example.personcustomjpql.model.Person;

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

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

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

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface PersonRepository extends JpaRepository<Person, Long> {

@Query("SELECT p FROM Person p WHERE p.firstName LIKE :prefix%")

List<Person> findByFirstNameStartingWith(@Param("prefix") String prefix);

@Query("SELECT p FROM Person p WHERE p.age >= :age")

List<Person> findByAgeGreaterThanEqual(@Param("age") int age);

@Query("SELECT p FROM Person p WHERE p.email = :email")

Person findByEmail(@Param("email") String email);

}

**PersonService.java (Service Layer)**

package com.example.personcustomjpql.service;

import com.example.personcustomjpql.model.Person;

import com.example.personcustomjpql.repository.PersonRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class PersonService {

@Autowired

private PersonRepository repository;

public Person savePerson(Person person) {

return repository.save(person);

}

public List<Person> getPersonsByFirstNamePrefix(String prefix) {

return repository.findByFirstNameStartingWith(prefix);

}

public List<Person> getPersonsByAge(int age) {

return repository.findByAgeGreaterThanEqual(age);

}

public Person getPersonByEmail(String email) {

return repository.findByEmail(email);

}

}

**PersonController.java (Controller Layer)**

package com.example.personcustomjpql.controller;

import com.example.personcustomjpql.model.Person;

import com.example.personcustomjpql.service.PersonService;

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

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/persons")

public class PersonController {

@Autowired

private PersonService service;

@PostMapping

public Person addPerson(@RequestBody Person person) {

return service.savePerson(person);

}

@GetMapping("/firstname")

public List<Person> getPersonsByFirstNamePrefix(@RequestParam String prefix) {

return service.getPersonsByFirstNamePrefix(prefix);

}

@GetMapping("/age")

public List<Person> getPersonsByAge(@RequestParam int age) {

return service.getPersonsByAge(age);

}

@GetMapping("/email")

public Person getPersonByEmail(@RequestParam String email) {

return service.getPersonByEmail(email);

}

}

**Result:**

Thus The program to build a Web Application for Managing Person Details Using Custom JPQL Queries via RESTful APIs.