Xideral

Java Academy

Week 4-Day 1

REST API with JPA

Presented by:

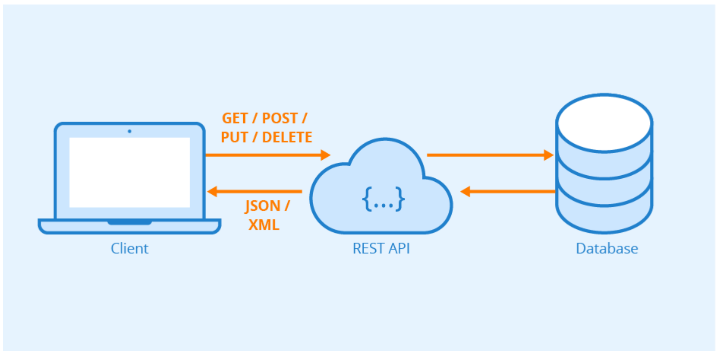
Edgar Itzak Sánchez Rogers

Monterrey Nuevo León México at 5 September 2024

Introduction:

At the most basic level, an API is a mechanism that allows an application or service to access a resource within another application or service. The application or service accessing the resource is the client, and the application or service containing the resource is the server. REST APIs communicate via HTTP requests to perform standard database functions, such as creating, reading, updating and deleting records within a resource.

A REST API would use a GET request to retrieve a record. A POST request creates a new record. A PUT request updates a record and a DELETE request deletes a record. All HTTP methods can be used in API calls.



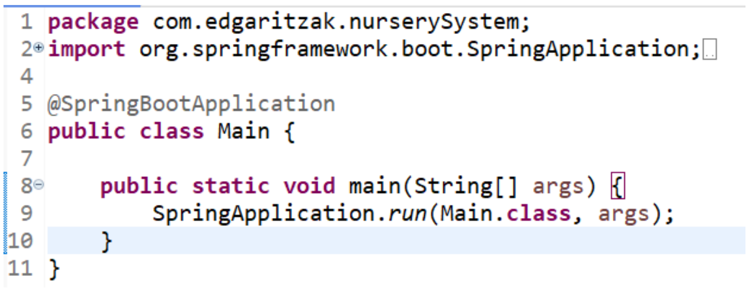
Java Persistence API

JPA represents a simplification of the persistence programming model. The JPA specification explicitly defines object-relational mapping, rather than relying on vendor-specific mapping implementations.

- The EntityManager API can update, read, delete or apply persistence to objects in a database.
- The EntityManager API and object-relational correlation metadata handle most database operations without requiring JDBC or SQL code to be written to maintain persistence.
- JPA provides a query language, which extends the standalone EJB query language, also known as JPQL, which you can use to retrieve objects without writing specific SQL queries to the database you are working with.

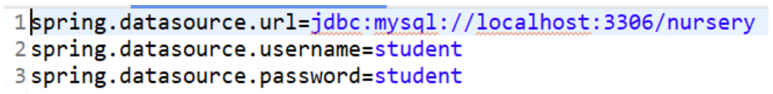
Java example:

Main class:



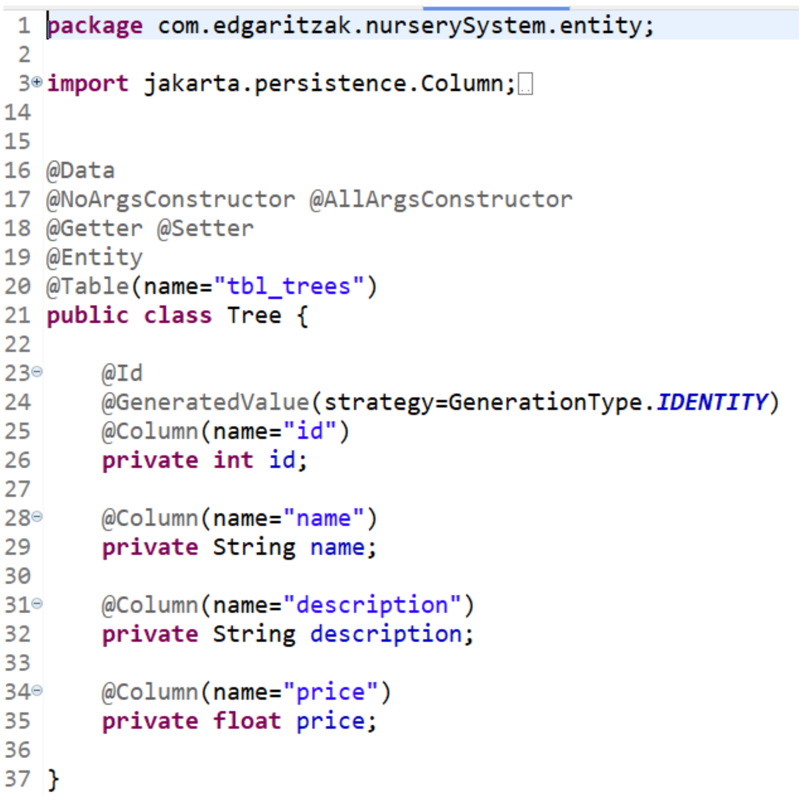
```java
package com.edgaritzak.nurserySystem;
import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class Main {

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

Application.properties:



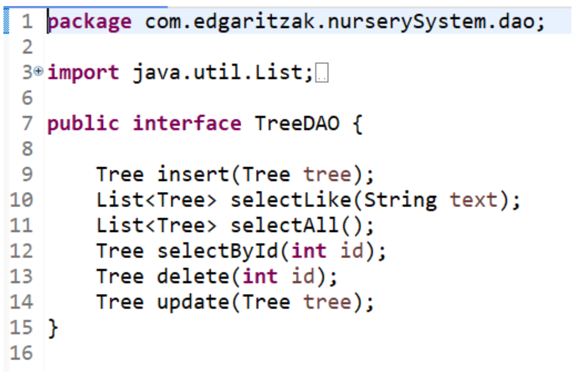
```
spring.datasource.url=jdbc:mysql://localhost:3306/nursery
spring.datasource.username=student
spring.datasource.password=student
```

Entity:



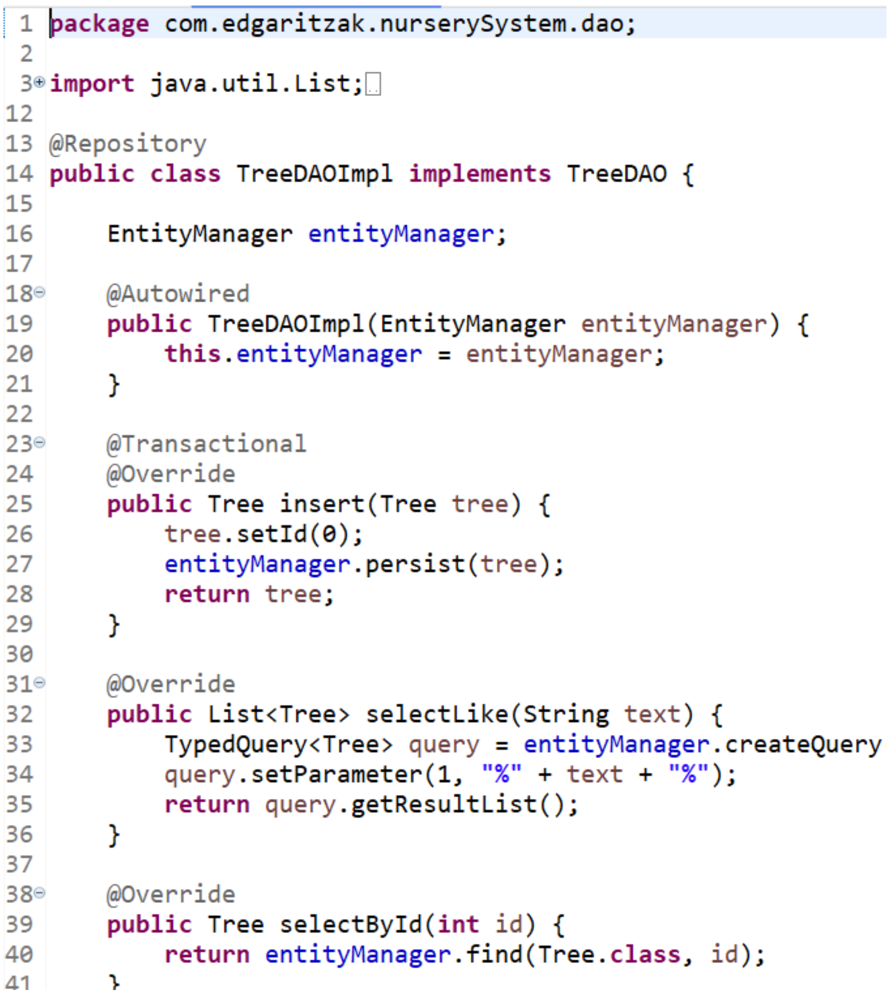
```java
package com.edgaritzak.nurserySystem.entity;

import jakarta.persistence.Column;


@Data
@NoArgsConstructor @AllArgsConstructor
@Getter @Setter
@Entity
@Table(name="tbl_trees")
public class Tree {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    @Column(name="name")
    private String name;

    @Column(name="description")
    private String description;

    @Column(name="price")
    private float price;

}
```

DAO interface:

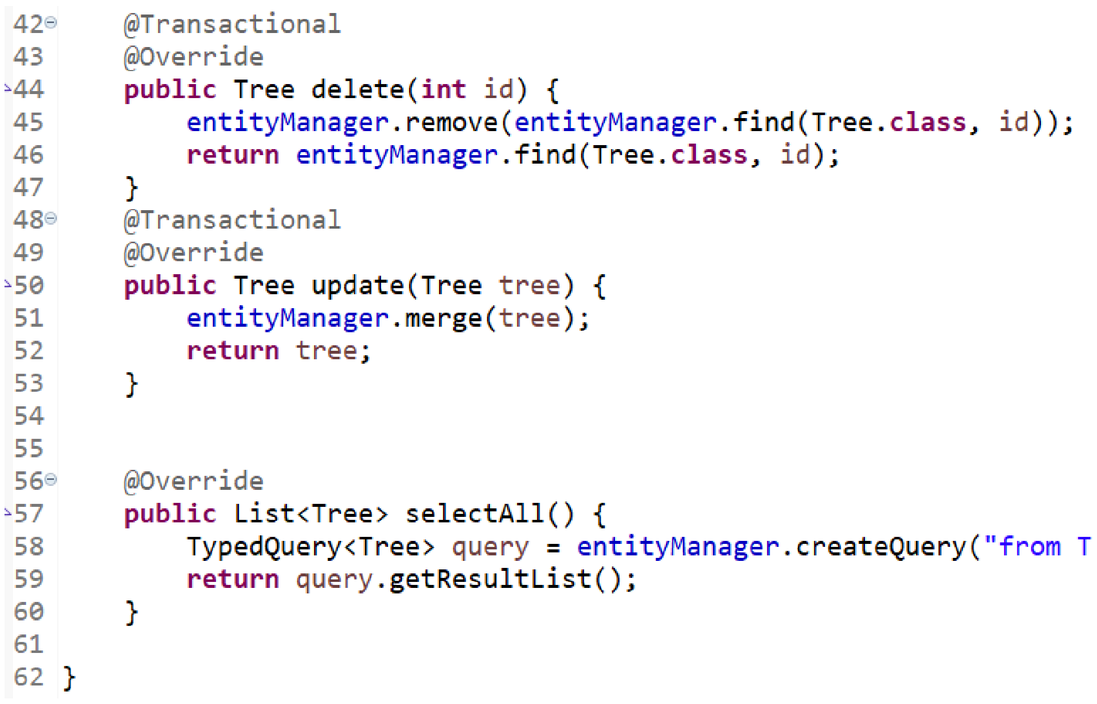


```java
package com.edgaritzak.nurserySystem.dao;

import java.util.List;

public interface TreeDAO {

    Tree insert(Tree tree);
    List<Tree> selectLike(String text);
    List<Tree> selectAll();
    Tree selectById(int id);
    Tree delete(int id);
    Tree update(Tree tree);
}

```

DAO Implementation:

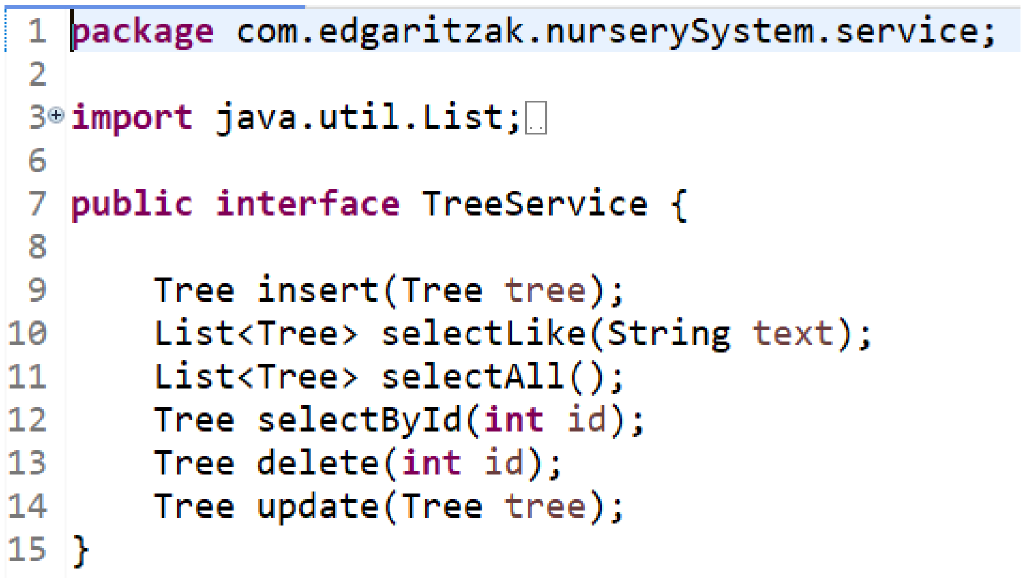


```java
package com.edgaritzak.nurserySystem.dao;

import java.util.List;

@Repository
public class TreeDAOImpl implements TreeDAO {

    EntityManager entityManager;

    @Autowired
    public TreeDAOImpl(EntityManager entityManager) {
        this.entityManager = entityManager;
    }

    @Transactional
    @Override
    public Tree insert(Tree tree) {
        tree.setId(0);
        entityManager.persist(tree);
        return tree;
    }

    @Override
    public List<Tree> selectLike(String text) {
        TypedQuery<Tree> query = entityManager.createQuery
        query.setParameter(1, "%" + text + "%");
        return query.getResultList();
    }

    @Override
    public Tree selectById(int id) {
        return entityManager.find(Tree.class, id);
    }
```



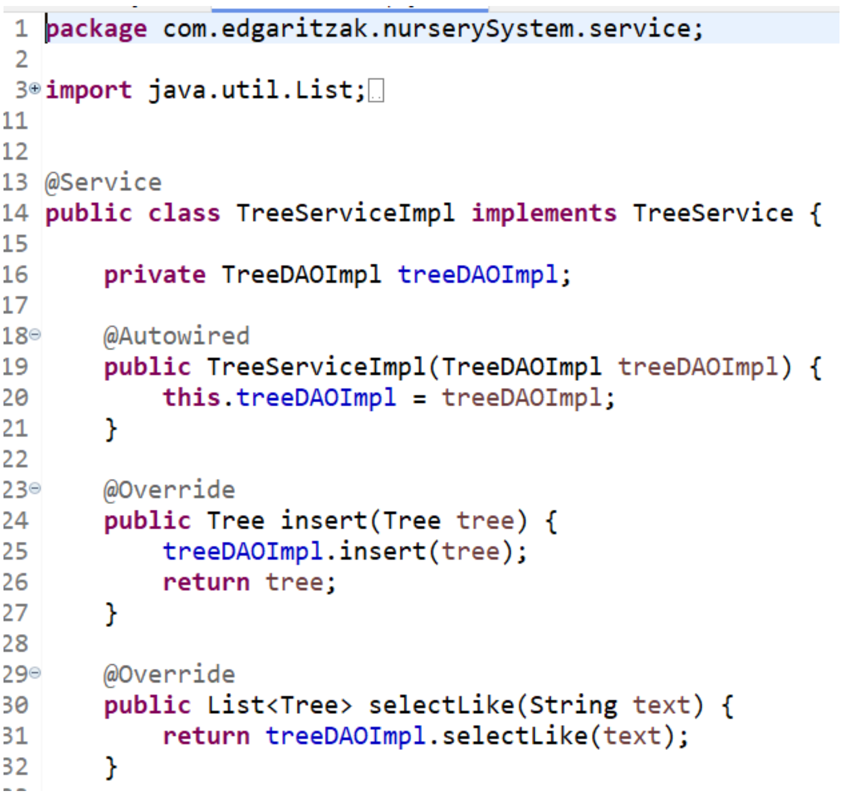
```java
    @Transactional
    @Override
    public Tree delete(int id) {
        entityManager.remove(entityManager.find(Tree.class, id));
        return entityManager.find(Tree.class, id);
    }
    @Transactional
    @Override
    public Tree update(Tree tree) {
        entityManager.merge(tree);
        return tree;
    }


    @Override
    public List<Tree> selectAll() {
        TypedQuery<Tree> query = entityManager.createQuery("from T
        return query.getResultList();
    }

}
```

TreeService Interface:

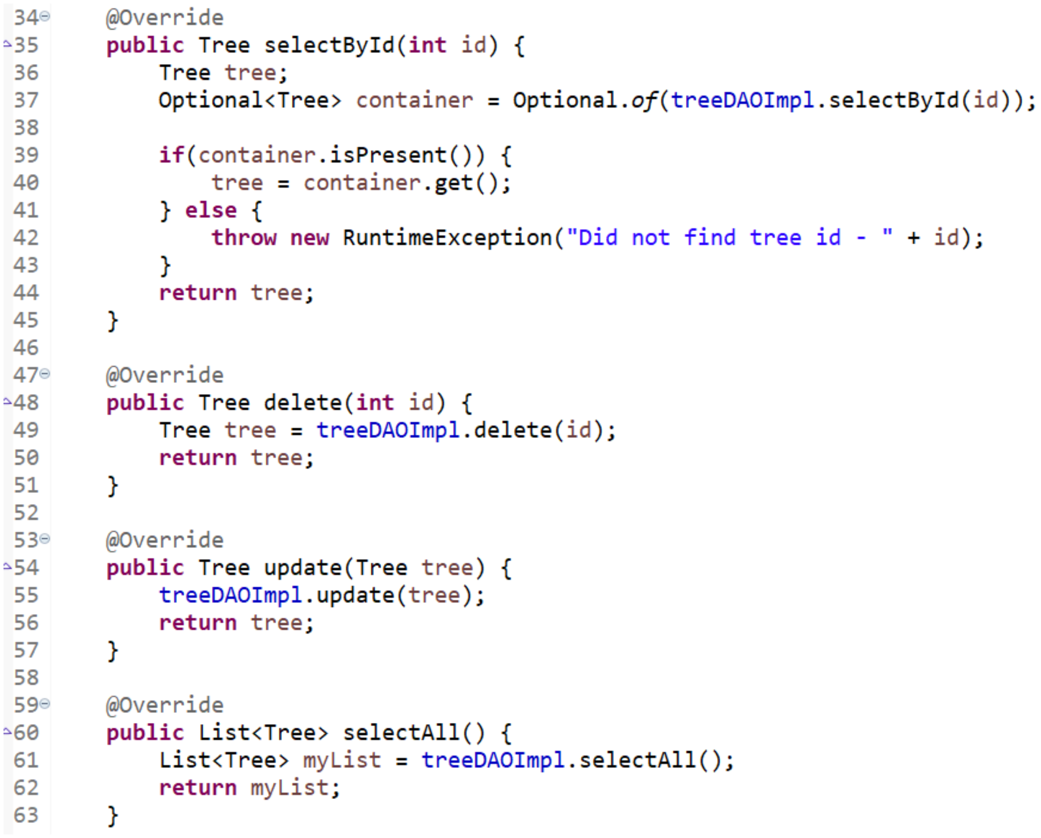


```java
package com.edgaritzak.nurserySystem.service;

import java.util.List;

public interface TreeService {

    Tree insert(Tree tree);
    List<Tree> selectLike(String text);
    List<Tree> selectAll();
    Tree selectById(int id);
    Tree delete(int id);
    Tree update(Tree tree);
}
```

TreeService Implementation:

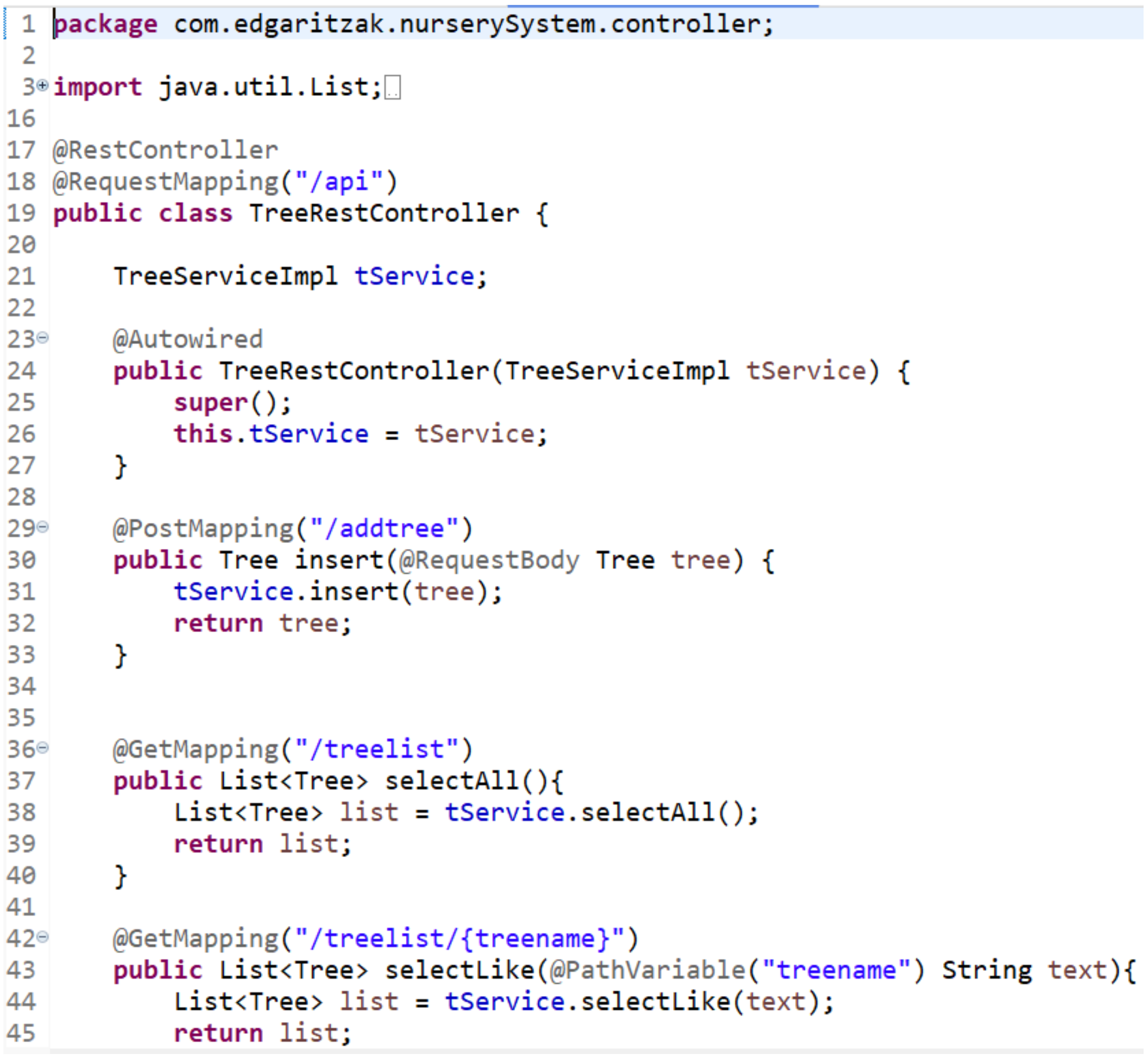


```java
package com.edgaritzak.nurserySystem.service;

import java.util.List;


@Service
public class TreeServiceImpl implements TreeService {

    private TreeDAOImpl treeDAOImpl;

    @Autowired
    public TreeServiceImpl(TreeDAOImpl treeDAOImpl) {
        this.treeDAOImpl = treeDAOImpl;
    }

    @Override
    public Tree insert(Tree tree) {
        treeDAOImpl.insert(tree);
        return tree;
    }

    @Override
    public List<Tree> selectLike(String text) {
        return treeDAOImpl.selectLike(text);
    }
```

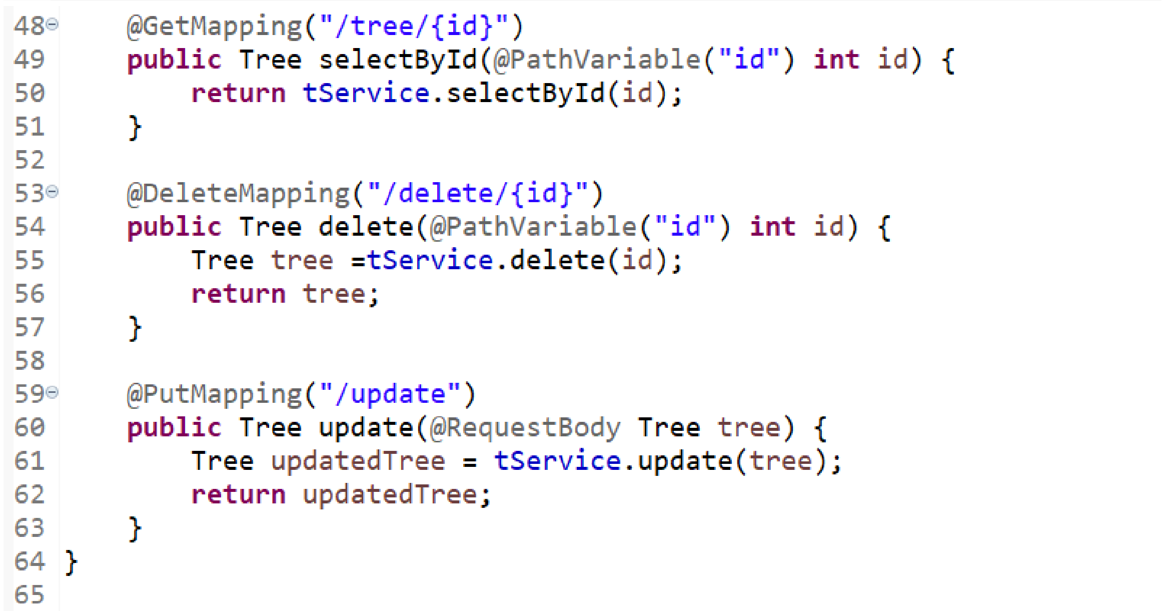


```java
    @Override
    public Tree selectById(int id) {
        Tree tree;
        Optional<Tree> container = Optional.of(treeDAOImpl.selectById(id));

        if(container.isPresent()) {
            tree = container.get();
        } else {
            throw new RuntimeException("Did not find tree id - " + id);
        }
        return tree;
    }

    @Override
    public Tree delete(int id) {
        Tree tree = treeDAOImpl.delete(id);
        return tree;
    }

    @Override
    public Tree update(Tree tree) {
        treeDAOImpl.update(tree);
        return tree;
    }

    @Override
    public List<Tree> selectAll() {
        List<Tree> myList = treeDAOImpl.selectAll();
        return myList;
    }
```

REST Controller:

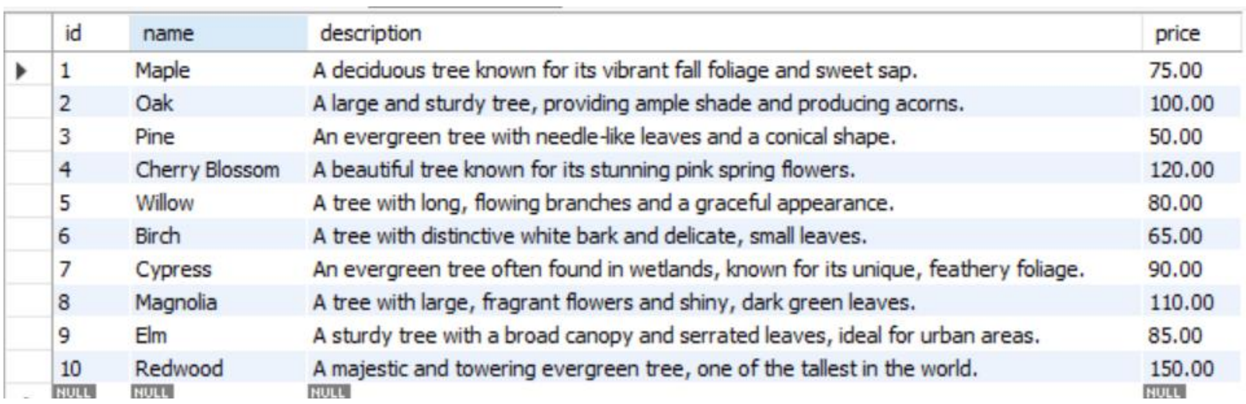


```java
package com.edgaritzak.nurserySystem.controller;

import java.util.List;

@RestController
@RequestMapping("/api")
public class TreeRestController {

    TreeServiceImpl tService;

    @Autowired
    public TreeRestController(TreeServiceImpl tService) {
        super();
        this.tService = tService;
    }

    @PostMapping("/addtree")
    public Tree insert(@RequestBody Tree tree) {
        tService.insert(tree);
        return tree;
    }


    @GetMapping("/treelist")
    public List<Tree> selectAll(){
        List<Tree> list = tService.selectAll();
        return list;
    }

    @GetMapping("/treelist/{treename}")
    public List<Tree> selectLike(@PathVariable("treename") String text){
        List<Tree> list = tService.selectLike(text);
        return list;
```



```java
    @GetMapping("/tree/{id}")
    public Tree selectById(@PathVariable("id") int id) {
        return tService.selectById(id);
    }

    @DeleteMapping("/delete/{id}")
    public Tree delete(@PathVariable("id") int id) {
        Tree tree =tService.delete(id);
        return tree;
    }

    @PutMapping("/update")
    public Tree update(@RequestBody Tree tree) {
        Tree updatedTree = tService.update(tree);
        return updatedTree;
    }
}

```

Table:



id	name	description	price
1	Maple	A deciduous tree known for its vibrant fall foliage and sweet sap.	75.00
2	Oak	A large and sturdy tree, providing ample shade and producing acorns.	100.00
3	Pine	An evergreen tree with needle-like leaves and a conical shape.	50.00
4	Cherry Blossom	A beautiful tree known for its stunning pink spring flowers.	120.00
5	Willow	A tree with long, flowing branches and a graceful appearance.	80.00
6	Birch	A tree with distinctive white bark and delicate, small leaves.	65.00
7	Cypress	An evergreen tree often found in wetlands, known for its unique, feathery foliage.	90.00
8	Magnolia	A tree with large, fragrant flowers and shiny, dark green leaves.	110.00
9	Elm	A sturdy tree with a broad canopy and serrated leaves, ideal for urban areas.	85.00
10	Redwood	A majestic and towering evergreen tree, one of the tallest in the world.	150.00
NULL	NULL	NULL	NULL

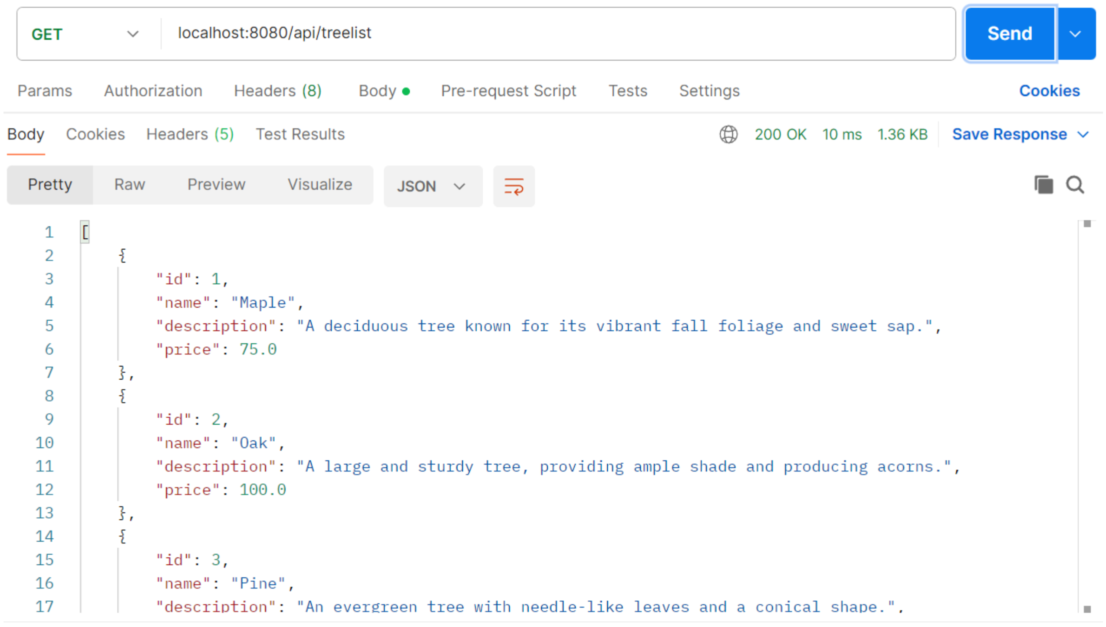
Explanation:

In order to generate this REST service, the following steps were taken:

- Create Java classes that represent the database tables using JPA annotations @Entity, @Table, @Id
- Configure custom methods if you need specific queries or additional operations.
- Implement a service class that contains the business logic and uses the repositories to interact with the database.
- Implement service methods for the operations that your REST API needs to expose.
- Implement a class that handles HTTP requests and uses the service layer to process those requests using annotations: @GetMapping, @PostMapping, @PutMapping, and @DeleteMapping to expose methods via HTTP routes.

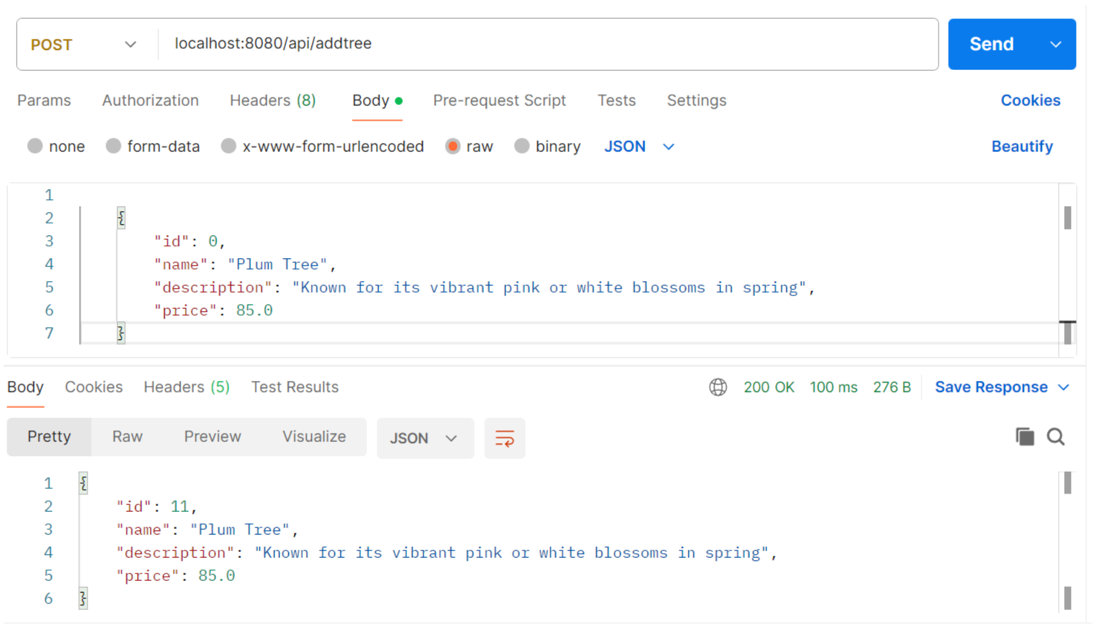
Testing REST service:

Select:



```
[
    {
        "id": 1,
        "name": "Maple",
        "description": "A deciduous tree known for its vibrant fall foliage and sweet sap.",
        "price": 75.0
    },
    {
        "id": 2,
        "name": "Oak",
        "description": "A large and sturdy tree, providing ample shade and producing acorns.",
        "price": 100.0
    },
    {
        "id": 3,
        "name": "Pine",
        "description": "An evergreen tree with needle-like leaves and a conical shape.",
```

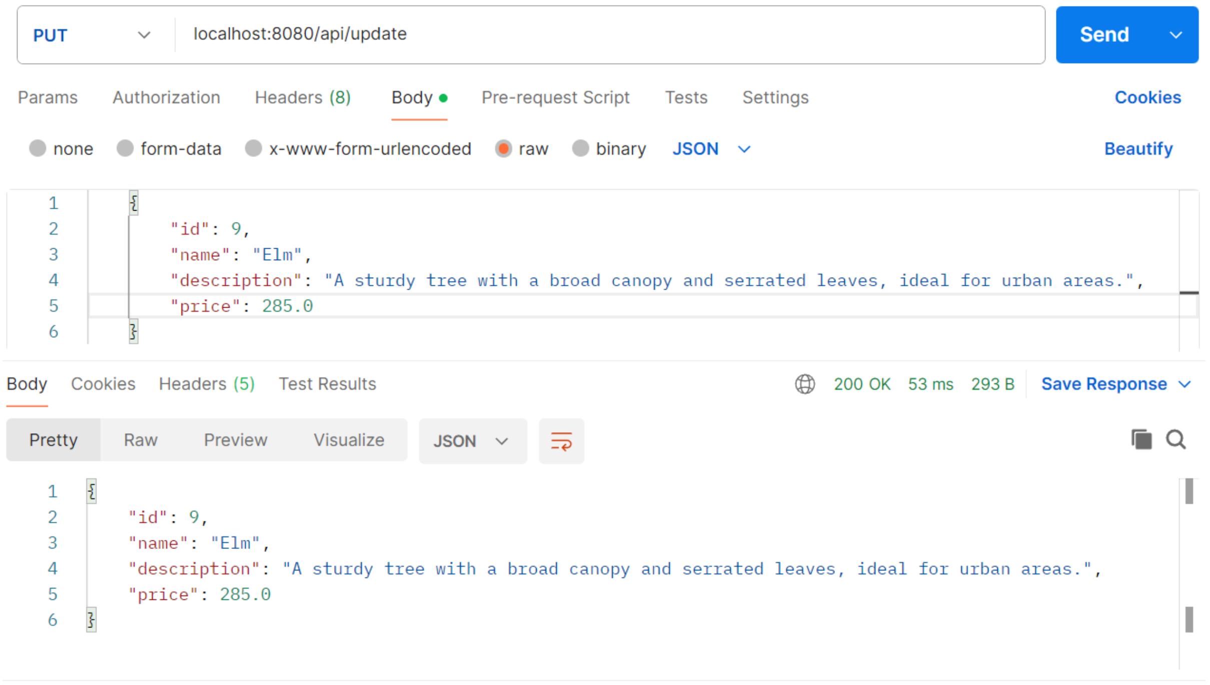
Insert:



```

    {
        "id": 0,
        "name": "Plum Tree",
        "description": "Known for its vibrant pink or white blossoms in spring",
        "price": 85.0
    }
```

```
{
    "id": 11,
    "name": "Plum Tree",
    "description": "Known for its vibrant pink or white blossoms in spring",
    "price": 85.0
}
```

Update:



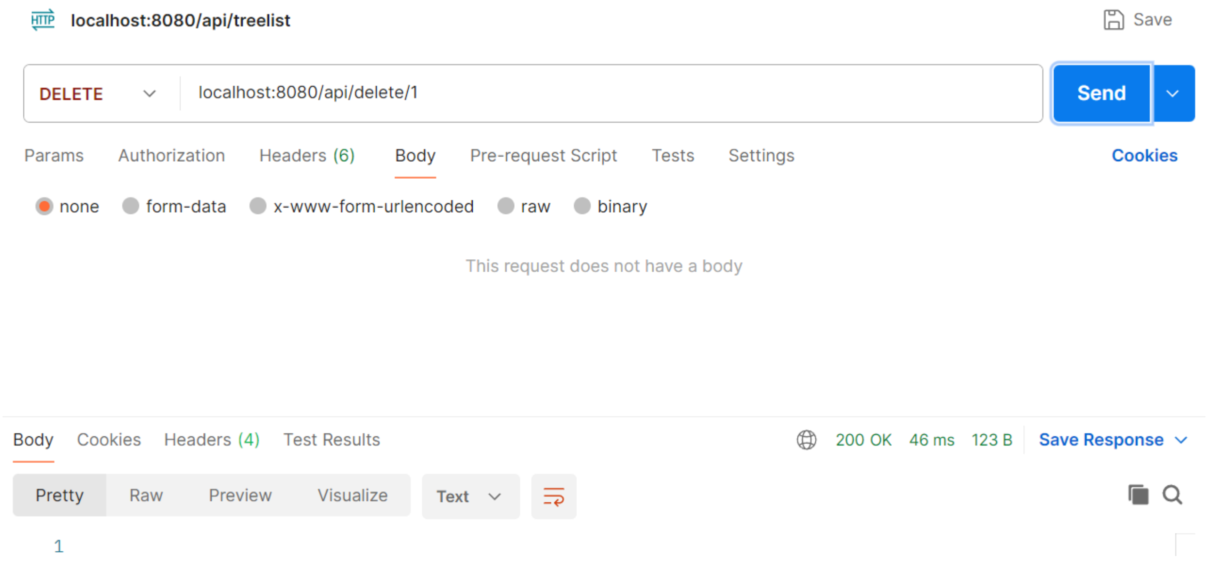
localhost:8080/api/update

```
{
    "id": 9,
    "name": "Elm",
    "description": "A sturdy tree with a broad canopy and serrated leaves, ideal for urban areas.",
    "price": 285.0
}
```

200 OK 53 ms 293 B

```
{
    "id": 9,
    "name": "Elm",
    "description": "A sturdy tree with a broad canopy and serrated leaves, ideal for urban areas.",
    "price": 285.0
}
```

Delete:



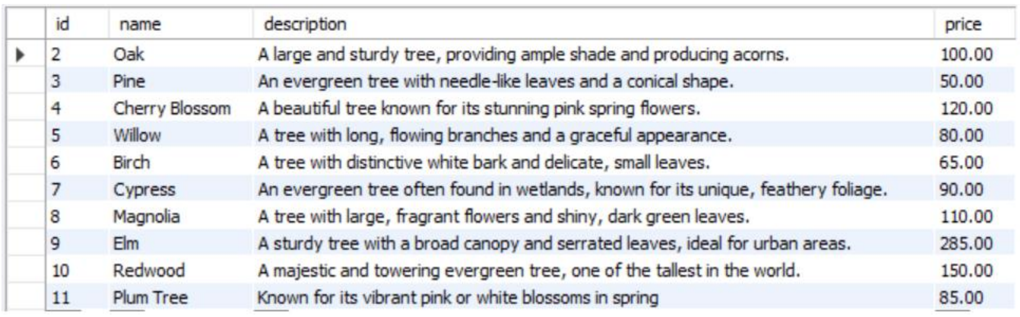
localhost:8080/api/treelist

localhost:8080/api/delete/1

This request does not have a body

200 OK 46 ms 123 B

Changes reflected in the database:



id	name	description	price
2	Oak	A large and sturdy tree, providing ample shade and producing acorns.	100.00
3	Pine	An evergreen tree with needle-like leaves and a conical shape.	50.00
4	Cherry Blossom	A beautiful tree known for its stunning pink spring flowers.	120.00
5	Willow	A tree with long, flowing branches and a graceful appearance.	80.00
6	Birch	A tree with distinctive white bark and delicate, small leaves.	65.00
7	Cypress	An evergreen tree often found in wetlands, known for its unique, feathery foliage.	90.00
8	Magnolia	A tree with large, fragrant flowers and shiny, dark green leaves.	110.00
9	Elm	A sturdy tree with a broad canopy and serrated leaves, ideal for urban areas.	285.00
10	Redwood	A majestic and towering evergreen tree, one of the tallest in the world.	150.00
11	Plum Tree	Known for its vibrant pink or white blossoms in spring	85.00

<-- Deleted id=1

<--Price updated

<-- Inserted data

Conclusion:

REST APIs combined with JPA (Java Persistence API) play a crucial role in modern web application development by providing an efficient and flexible way to handle client-server communication and data persistence management. The integration of REST APIs and JPA makes it possible to create applications that are not only scalable and maintainable, but also easy to develop and integrate with other systems. REST API provides a standardized and accessible interface for interacting with web services.

References:

[1] ¿Qué es una API REST? | IBM

[2] Java Persistence API (JPA) - Documentación de IBM