



Xideral

Java Academy

Week 4-Day 2

REST API with Spring Data JPA

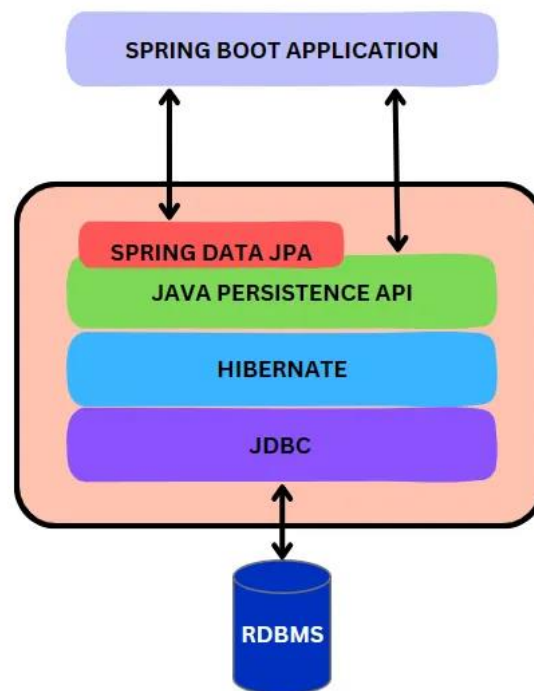
Presented by:

Edgar Itzak Sánchez Rogers

Monterrey Nuevo León México at 28 august 2024

Introduction:

Spring Data JPA, part of the larger Spring Data family, makes it easy to easily implement Java Persistence API repositories. It makes it easier to build Spring-powered applications that use data access technologies. Spring Data JPA aims to significantly improve the implementation of data access layers by reducing the effort to the amount that's actually needed.



JpaRepository:

JpaRepository is a generic interface that allows interact with a database in a abstract and efficient way. By implementing JpaRepository, you can take advantage of a number of features:

- **CRUD operations:** Methods like `save()`, `findById()`, `findAll()`, `deleteById()`, etc.
- **Custom Queries:** You can define specific queries using the method names method.
- **Pagination and sorting:** Methods to obtain paginated and ordered results.

Java example:

application.properties

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

Entity tree

```
1 package com.edgaritzak.nurserySystemDataJPA.entity;
2
3 import jakarta.persistence.Column;
4
14
15
16 @Data
17 @NoArgsConstructor @AllArgsConstructor
18 @Getter @Setter
19 @Entity
20 @Table(name="tbl_trees")
21 public class Tree {
22
23     @Id
24     @GeneratedValue(strategy=GenerationType.IDENTITY)
25     @Column(name="id")
26     private int id;
27
28     @Column(name="name")
29     private String name;
30
31     @Column(name="description")
32     private String description;
33
34     @Column(name="price")
35     private float price;
36
37 }
38
```

Tree repository extends JpaRepository

```
1 package com.edgaritzak.nurserySystemDataJPA.dao;
2
3 import java.util.List;
4
10
11 public interface TreeRepository extends JpaRepository<Tree,Integer>{
12
13     // CUSTOM METHOD SELECT LIKE();
14     @Query("Select t from Tree t where t.name like :name")
15     List<Tree> findNameLike(@Param("name")String name);
16 }
```

Service interface

```
1 package com.edgaritzak.nurserySystemDataJPA.service;
2
3 import java.util.List;
4
5
6
7 public interface TreeService {
8
9     //SELECT
10    List<Tree> findNameLike(String text);
11    List<Tree> findAll();
12    Tree findById(int id);
13    //INSERT
14    Tree save(Tree tree);
15    //UPDATE
16    Tree update(Tree tree);
17    //DELETE
18    Tree deleteById(int id);
19
20 }
```

Service Implementation

```
1 package com.edgaritzak.nurserySystemDataJPA.service;
2
3 import java.util.List;
4
5
6
7
8
9
10
11
12
13 @Service
14 public class TreeServiceImpl implements TreeService {
15
16     private TreeRepository treeDAO;
17
18     @Autowired
19     public TreeServiceImpl(TreeRepository treeDAO) {
20         this.treeDAO = treeDAO;
21     }
22
23     //SELECT ALL
24     @Override
25     public List<Tree> findAll() {
26         List<Tree> myList = treeDAO.findAll(); //SPRING DATA JPA METHOD
27         return myList;
28     }
29
30     //SELECT LIKE
31     @Override
32     public List<Tree> findNameLike(String text) {
33         return treeDAO.findNameLike(text); //SPRING DATA JPA METHOD (CUSTOP QUERY)
34     }
35 }
```

```

36 //SELECT BY ID
37 @Override
38 public Tree findById(int id) {
39     Tree tree;
40     Optional<Tree> container = (treeDAO.findById(id)); //SPRING DATA JPA METHOD
41
42     if(container.isPresent()) {
43         tree = container.get();
44     } else {
45         throw new RuntimeException("Did not find tree id - " + id);
46     }
47     return tree;
48 }
49
50 //INSERT
51 @Override
52 public Tree save(Tree tree) {
53     treeDAO.save(tree); //SPRING DATA JPA METHOD
54     return tree;
55 }
56
57 //UPDATE
58 @Override
59 public Tree update(Tree tree) {
60     Tree treeidToCompare = treeDAO.findById(tree.getId()) .orElseThrow(() -> new
61     return treeDAO.save(treeidToCompare); //SPRING DATA JPA METHOD
62 }
63
64 //DELETE
65 @Override
66 public Tree deleteById(int id) {
67     Optional<Tree> tempTree = treeDAO.findById(id); //SPRING DATA JPA METHOD
68     if(tempTree.isPresent()) {
69         treeDAO.deleteById(id); //SPRING DATA JPA METHOD
70         return tempTree.get();
71     } else {
72         throw new IllegalStateException("Error - ID not found");
73     }
74 }
75

```

REST Controller

```
1 package com.edgaritzak.nurserySystemDataJPA.controller;
2
3 import java.util.List;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18 @RestController
19 @RequestMapping("/api")
20 public class TreeRestController {
21
22     TreeServiceImpl treeService;
23
24     @Autowired
25     public TreeRestController(TreeServiceImpl treeService) {
26         super();
27         this.treeService = treeService;
28     }
29
30
31     //SELECT
32     @GetMapping("/treelist")
33     public List<Tree> selectAll(){
34         List<Tree> list = treeService.findAll();
35         return list;
36     }
37     //SELECT
38     @GetMapping("/treelist/{treename}")
39     public List<Tree> selectLike(@PathVariable("treename") String text){
40         List<Tree> list = treeService.findNameLike(text);
41         return list;
42     }
43     //SELECT
44     @GetMapping("/tree/{id}")
45     public Tree selectById(@PathVariable("id") int id) {
46         return treeService.findById(id);
47     }
48
49     //INSERT
50     @PostMapping("/addtree")
51     public Tree insert(@RequestBody Tree tree) {
52         treeService.save(tree);
53         return tree;
54     }
55
56     //UPDATE
57     @PutMapping("/update")
58     public Tree update(@RequestBody Tree tree) {
59         Tree updatedTree = treeService.update(tree);
60         return updatedTree;
61     }
62
63     //DELETE
64     @DeleteMapping("/delete/{id}")
65     public Tree delete(@PathVariable("id") int id) {
66         return treeService.deleteById(id);
67     }
68 }
```

Main class:

```
1 package com.edgaritzak.nurserySystemDataJPA;
2 import org.springframework.boot.SpringApplication;
3
4
5 @SpringBootApplication
6 public class Main {
7
8     public static void main(String[] args) {
9         SpringApplication.run(Main.class, args);
10    }
11 }
```

Explanation:

To implement the query methods simply add

‘extends JpaRepository<ENTITY,PRIMARY_KEY>’ to the interface

```
10
11 public interface TreeRepository extends JpaRepository<Tree,Integer>{
12
```

JpaRepository methods are now accessible when creating TreeRepository instance, methods such as:

- save()
- SaveAll ()
- FindById ()
- FindAll ()
- FindAllById ()
- deleteById ()

The service interface and the service implementation are now created. The service implementation uses the JpaRepository query methods and these services are used by the REST Controller.

The results of using the REST service are shown below.

Results:

Table:

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

Testing REST service

Seach list

localhost:8080/api/treelist

Save

GET

localhost:8080/api/treelist

Send

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

Key	Value
-----	-------

Bulk Edit

Body

Cookies

Headers (5)

Test Results

200 OK

283 ms

1.36 KB

Save Response

Pretty

Raw

Preview

Visualize

JSON

```
1 {
2   {
3     "id": 1,
4     "name": "Maple",
5     "description": "A deciduous tree known for its vibrant fall foliage and sweet sap.",
6     "price": 75.0
7   },
8   {
9     "id": 2,
10    "name": "Oak",
11    "description": "A large and sturdy tree, providing ample shade and producing acorns.",
12    "price": 100.0
13  },
14 }
```


Search by name

localhost:8080/api/treelist

Save

GET

localhost:8080/api/treelist/Oak

Send

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

	Key	Value	Bulk Edit
--	-----	-------	-----------

Body

Cookies

Headers (5)

Test Results

200 OK

6 ms

286 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

6

7

8

[

{

"id": 2,

"name": "Oak",

"description": "A large and sturdy tree, providing ample shade and producing acorns.",

"price": 100.0

}

]

Insert:

Home

Workspaces

Explore

Search Postman

Sign In

Create Account

You are using the Lightweight API Client, sign in or create an account to work with collections, environments and unlock all free features in Postman.

History

New

Import

localhost:8080/api/treelist

localhost:8080/api/treelist/

localhost:8080/api/treelist/asd

localhost:8080/api/treelist/Pi

localhost:8080/api/treelist/a

localhost:8080/api/treelist/Oa

localhost:8080/api/treelist/Oa

localhost:8080/api/treelist/Oa

localhost:8080/api/treelist/Oa

localhost:8080/api/treelist/a

localhost:8080/api/treelist/a

localhost:8080/api/treelist

localhost:8080/api/tree/10

localhost:8080/api/tree/6

localhost:8080/api/tree/1

localhost:8080/api/tree/2

localhost:8080/api/tree/7id-1

localhost:8080/api/treelist

Save

</>

POST

localhost:8080/api/addtree

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Cookies

Beautiful

none

form-data

x-www-form-urlencoded

raw

binary

JSON

1

2

3

4

5

6

7

[

{

"id": 0,

"name": "Kiri Tree",

"description": "Has large, heart-shaped leaves and fragrant purple flowers.",

"price": 110.0

}

]

Body

Cookies

Headers (5)

Test Results

200 OK

31 ms

282 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

6

[

{

"id": 11,

"name": "Kiri Tree",

"description": "Has large, heart-shaped leaves and fragrant purple flowers.",

"price": 110.0

}

]

Console

Not connected to a Postman account

Buscar

07:47 p. m.

	id	name	description	price
	1	Maple	A deciduous tree known for its vibrant fall foliage and sweet sap.	75.00
	2	Kiri Tree	Has large, heart-shaped leaves and fragrant purple flowers.	110.00
	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
	11	Kiri Tree	Has large, heart-shaped leaves and fragrant purple flowers.	110.00

Update:

▶ 8MagnoliaA tree with large, fragrant flowers and shiny, dark green leaves.110.00

PUTlocalhost:8080/api/updateSend

ParamsAuthorizationHeaders (8)BodyPre-request ScriptTestsSettingsCookies

noneform-datax-www-form-urlencodedrawbinaryJSONBeautify

```
1
2
3      "id": 8,
4      "name": "Magnolia",
5      "description": "A tree with large, fragrant flowers and shiny, dark green leaves.",
6      "price": 155.0
7
```

BodyCookiesHeaders (5)Test Results200 OK14 ms286 BSave Response

PrettyRawPreviewVisualizeJSON

```
1
2      "id": 8,
3      "name": "Magnolia",
4      "description": "A tree with large, fragrant flowers and shiny, dark green leaves.",
5      "price": 110.0
6
```

▶ 8MagnoliaA tree with large, fragrant flowers and shiny, dark green leaves.155.00

Delete:

DELETElocalhost:8080/api/delete/2Send

ParamsAuthorizationHeaders (8)BodyPre-request ScriptTestsSettingsCookies

Query Params

	Key	Value	Bulk Edit
	Key	Value	

BodyCookiesHeaders (5)Test Results200 OK40 ms281 BSave Response

PrettyRawPreviewVisualizeJSON

```
1
2      "id": 2,
3      "name": "Kiri Tree",
4      "description": "Has large, heart-shaped leaves and fragrant purple flowers.",
5      "price": 110.0
6
```

▶ 1MapleA deciduous tree known for its vibrant fall foliage and sweet sap.75.00

▶ 3PineAn evergreen tree with needle-like leaves and a conical shape.50.00

Conclusion:

Spring Data JPA simplifies application development by providing an abstraction layer on top of JPA (Java Persistence API). Spring Data JPA eliminates the need to write much of the repetitive code associated with data management, thanks to its pre-defined repository interfaces and automatic query generation. It includes integrated functionalities for pagination and sorting of results, enabling efficient management of large volumes of data.

References:

- [1] [Spring Data JPA](#)
- [2] [What is Spring Data JPA? - GeeksforGeeks](#)
- [3] [Getting Started | Accessing Data with JPA \(spring.io\)](#)