sorting and pagination comes under spring data jpa

Yes — **100% correct** ✅

> **Sorting and pagination are features provided by Spring Data JPA**, not by JPA itself.

Let's explain **why**, **how**, and **where they sit in the flow**, with clear examples.

1️⃣ Who provides sorting & pagination?

Layer	Provides sorting & pagination?
JPA	❌ No
Hibernate	❌ No (directly)
Spring Data JPA	✅ YES

📌 JPA only defines `Query`, `EntityManager`
📌 Spring Data JPA adds `Page`, `Pageable`, `Sort` **abstractions**

2️⃣ Pagination in Spring Data JPA

Example

java Copy code

```java
Page<Employee> findByDepartment(String dept, Pageable pageable);
```

Usage

java Copy code

```java
Pageable pageable = PageRequest.of(0, 10);
Page<Employee> page = repository.findByDepartment("HR", pageable);
```

sorting and pagination comes under spring data jpa

Yes — **100% correct** ✅

> **Sorting and pagination are features provided by Spring Data JPA**, not by JPA itself.

Let's explain **why**, **how**, and **where they sit in the flow**, with clear examples.

1️⃣ Who provides sorting & pagination?

Layer	Provides sorting & pagination?
JPA	❌ No
Hibernate	❌ No (directly)
Spring Data JPA	✅ YES

📌 JPA only defines `Query`, `EntityManager`
📌 Spring Data JPA adds `Page`, `Pageable`, `Sort` **abstractions**

2️⃣ Pagination in Spring Data JPA

Example

java Copy code

```java
Page<Employee> findByDepartment(String dept, Pageable pageable);
```

Usage

java Copy code

```java
Pageable pageable = PageRequest.of(0, 10);
Page<Employee> page = repository.findByDepartment("HR", pageable);
```

sorting and pagination comes under spring data jpa

Yes — **100% correct** ✅

> **Sorting and pagination are features provided by Spring Data JPA**, not by JPA itself.

Let's explain **why**, **how**, and **where they sit in the flow**, with clear examples.

1️⃣ Who provides sorting & pagination?

Layer	Provides sorting & pagination?
JPA	❌ No
Hibernate	❌ No (directly)
Spring Data JPA	✅ YES

📌 JPA only defines `Query`, `EntityManager`
📌 Spring Data JPA adds `Page`, `Pageable`, `Sort` **abstractions**

2️⃣ Pagination in Spring Data JPA

Example

java Copy code

```java
Page<Employee> findByDepartment(String dept, Pageable pageable);
```

Usage

java Copy code

```java
Pageable pageable = PageRequest.of(0, 10);
Page<Employee> page = repository.findByDepartment("HR", pageable);
```

sorting and pagination comes under spring data jpa

Yes — **100% correct** ✅

> **Sorting and pagination are features provided by Spring Data JPA**, not by JPA itself.

Let's explain **why**, **how**, and **where they sit in the flow**, with clear examples.

1️⃣ Who provides sorting & pagination?

Layer	Provides sorting & pagination?
JPA	❌ No
Hibernate	❌ No (directly)
Spring Data JPA	✅ YES

📌 JPA only defines `Query`, `EntityManager`
📌 Spring Data JPA adds `Page`, `Pageable`, `Sort` **abstractions**

2️⃣ Pagination in Spring Data JPA

Example

java Copy code

```java
Page<Employee> findByDepartment(String dept, Pageable pageable);
```

Usage

java Copy code

```java
Pageable pageable = PageRequest.of(0, 10);
Page<Employee> page = repository.findByDepartment("HR", pageable);
```

What Spring Data JPA does

- Converts `Pageable` → JPQL with:
  - `LIMIT`
  - `OFFSET`
- Also creates a **COUNT query**

Generated JPQL (conceptually):

```jpql
SELECT e FROM Employee e WHERE e.department = :dept
```

Count query:

```jpql
SELECT COUNT(e) FROM Employee e WHERE e.department = :dept
```

3️⃣ Sorting in Spring Data JPA

Example

```java
List<Employee> findByDepartment(String dept, Sort sort);
```

Usage:

```java
Sort sort = Sort.by("salary").descending();
repository.findByDepartment("HR", sort);
```

Spring Data JPA adds:

```jpql
ORDER BY e.salary DESC
```

What Spring Data JPA does

- Converts `Pageable` → JPQL with:
  - `LIMIT`
  - `OFFSET`
- Also creates a **COUNT query**

Generated JPQL (conceptually):

```jpql
SELECT e FROM Employee e WHERE e.department = :dept
```

Count query:

```jpql
SELECT COUNT(e) FROM Employee e WHERE e.department = :dept
```

3️⃣ Sorting in Spring Data JPA

Example

```java
List<Employee> findByDepartment(String dept, Sort sort);
```

Usage:

```java
Sort sort = Sort.by("salary").descending();
repository.findByDepartment("HR", sort);
```

Spring Data JPA adds:

```jpql
ORDER BY e.salary DESC
```

4️⃣ Sorting + Pagination together

java

Copy code

```java
Page<Employee> findByDepartment(String dept, Pageable pageable);
```

java

Copy code

```java
Pageable pageable = PageRequest.of(
        0, 10,
        Sort.by("salary").descending()
);
```

Spring Data JPA handles:

- Sorting
- Pagination
- Count query

All automatically.

5️⃣ Where does JPQL come in?

- Sorting & pagination are **NOT JPQL features**
- They are **Spring Data JPA features**
- Spring Data JPA **modifies the JPQL** before execution

Flow:

pgsql

Copy code

```pgsql
Repository method
   ↓
Spring Data JPA
   ↓ (adds ORDER BY / LIMIT / OFFSET)
JPQL
   ↓
EntityManager
   ↓
Hibernate
   ↓
SQL
```

4️⃣ Sorting + Pagination together

java

Copy code

```java
Page<Employee> findByDepartment(String dept, Pageable pageable);
```

java

Copy code

```java
Pageable pageable = PageRequest.of(
        0, 10,
        Sort.by("salary").descending()
);
```

Spring Data JPA handles:

- Sorting
- Pagination
- Count query

All automatically.

5️⃣ Where does JPQL come in?

- Sorting & pagination are **NOT JPQL features**
- They are **Spring Data JPA features**
- Spring Data JPA **modifies the JPQL** before execution

Flow:

pgsql

Copy code

```pgsql
Repository method
   ↓
Spring Data JPA
   ↓ (adds ORDER BY / LIMIT / OFFSET)
JPQL
   ↓
EntityManager
   ↓
Hibernate
   ↓
SQL
```

6 Why JPA doesn't provide pagination abstraction?

JPA provides only:

```java
query.setFirstResult()
query.setMaxResults()
```

Spring Data JPA:

- Wraps this in `Pageable`
- Adds count queries
- Adds sorting abstraction

---

7 Important return types

Return Type	Meaning
`Page<T>`	Full pagination info
`Slice<T>`	No total count
`List<T>`	No pagination info

---

8 Best practices

- ✔ Use `Page<T>` for UI pagination
- ✔ Use `Slice<T>` for infinite scroll
- ✔ Always index sorted columns
- ✔ Avoid pagination on huge offsets (performance)

---

9 One-line interview answer (perfect)

> Sorting and pagination are provided by Spring Data JPA, which adds abstractions like `Pageable`, `Page`, and `Sort` on top of JPA and transparently applies them to generated JPQL queries.

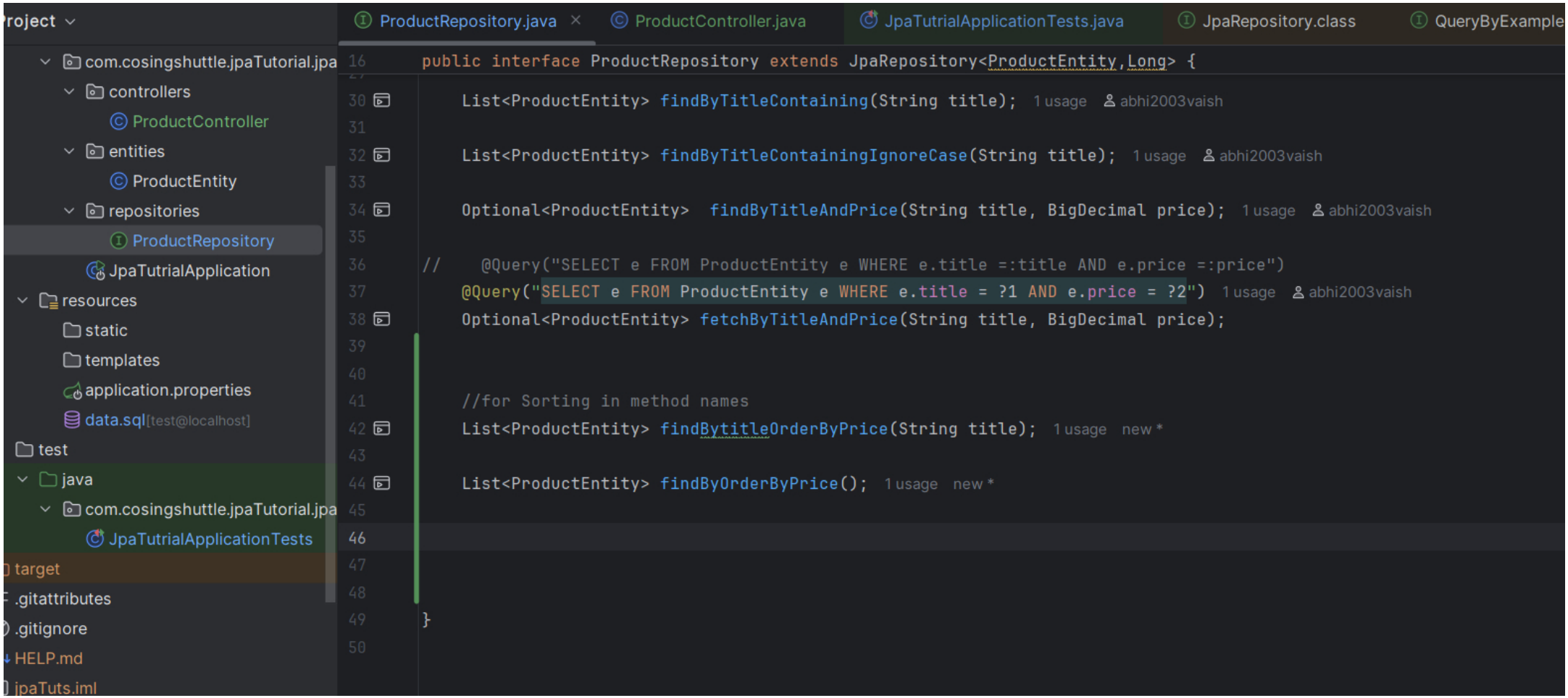
🧠 Final mental model

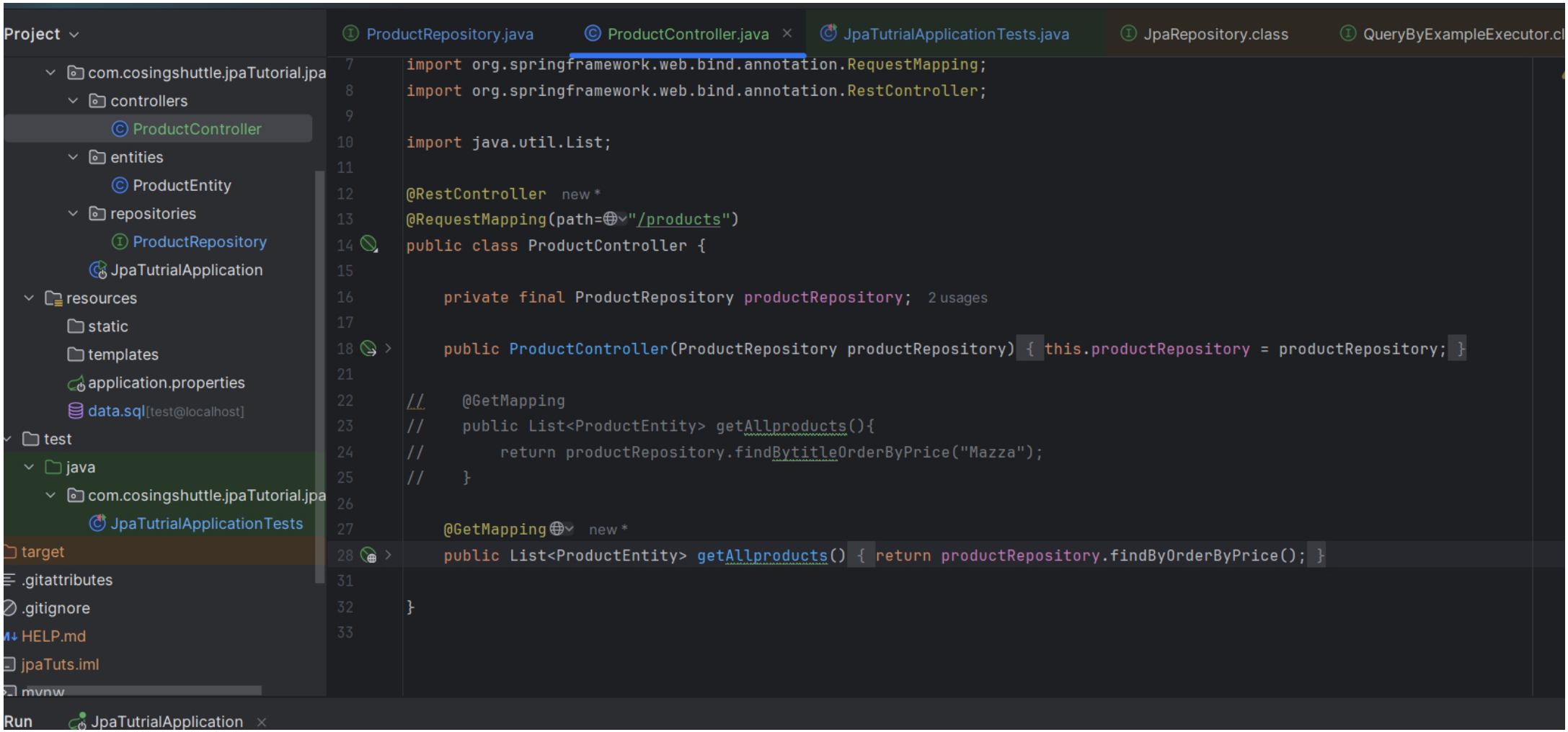
- JPA → persistence rules
- Hibernate → execution
- **Spring Data JPA → convenience (sorting, pagination, repositories)**

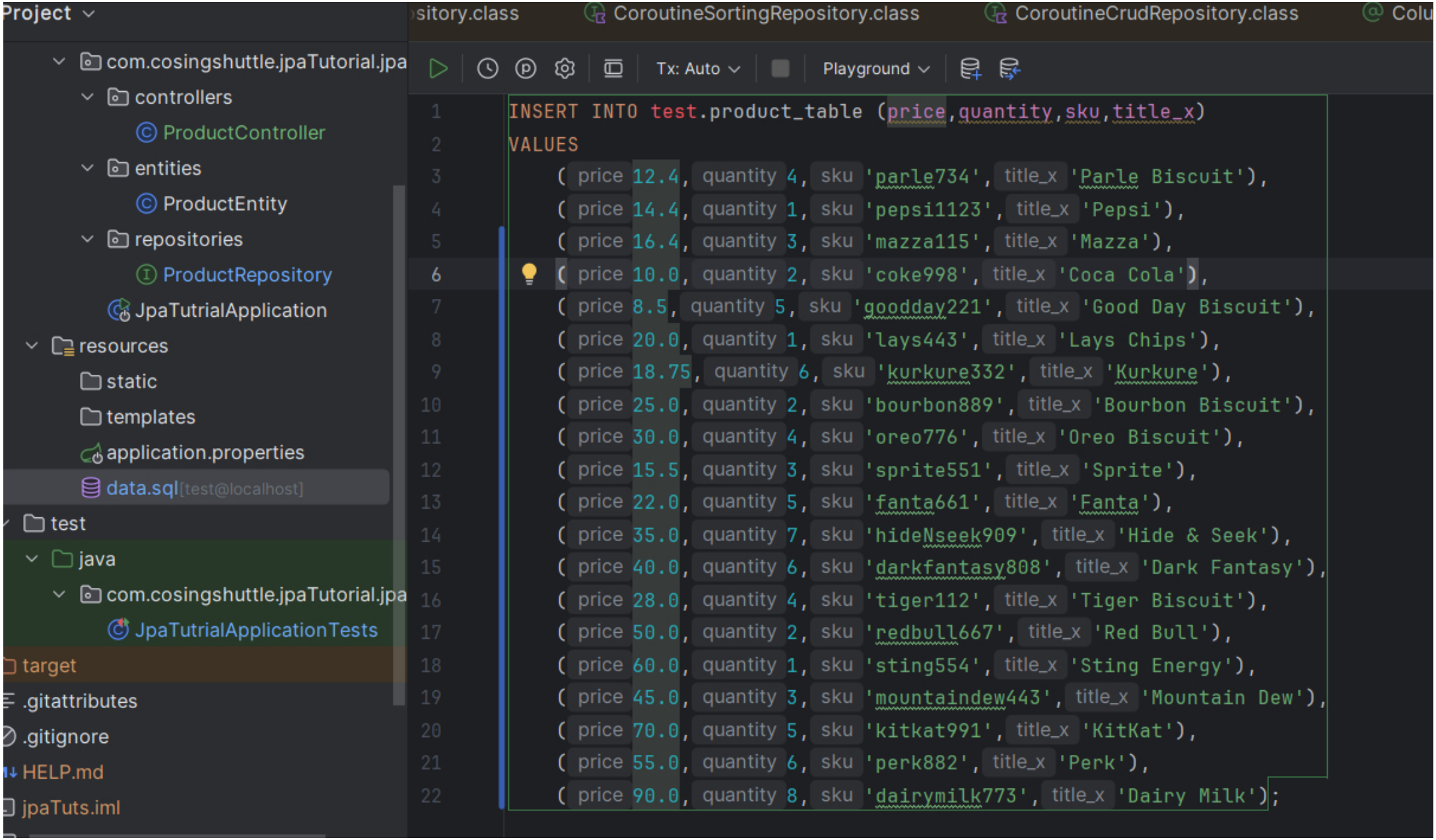
Sorting with method queries

OrderBy

```
@Repository
public interface EmployeeRepository extends JpaRepository<Employee,
Long> {
    List<Employee> findAllByOrderByNameAsc();
    List<Employee> findAllByOrderByNameDesc();
}
```


Project
com.cosingshuttle.jpaTutorial.jpa
controllers
ProductController
entities
ProductEntity
repositories
ProductRepository
JpaTutrialApplication
resources
static
templates
application.properties
data.sql[test@localhost]
test
java
com.cosingshuttle.jpaTutorial.jpa
JpaTutrialApplicationTests
target
.gitattributes
.gitignore
HELP.md
jpaTuts.iml
ProductRepository.java
ProductController.java
JpaTutrialApplicationTests.java
JpaRepository.class
QueryByExample
public interface ProductRepository extends JpaRepository<ProductEntity,Long> {
List<ProductEntity> findByTitleContaining(String title); 1 usage abhi2003vaish
List<ProductEntity> findByTitleContainingIgnoreCase(String title); 1 usage abhi2003vaish
Optional<ProductEntity> findByTitleAndPrice(String title, BigDecimal price); 1 usage abhi2003vaish
// @Query("SELECT e FROM ProductEntity e WHERE e.title =:title AND e.price =:price")
@Query("SELECT e FROM ProductEntity e WHERE e.title = ?1 AND e.price = ?2") 1 usage abhi2003vaish
Optional<ProductEntity> fetchByTitleAndPrice(String title, BigDecimal price);
//for Sorting in method names
List<ProductEntity> findBytitleOrderByPrice(String title); 1 usage new *
List<ProductEntity> findByOrderByPrice(); 1 usage new *
}


Project
com.cosingshuttle.jpaTutorial.jpa
controllers
ProductController
entities
ProductEntity
repositories
ProductRepository
JpaTutrialApplication
resources
static
templates
application.properties
data.sql[test@localhost]
test
java
com.cosingshuttle.jpaTutorial.jpa
JpaTutrialApplicationTests
target
.gitattributes
.gitignore
HELP.md
jpaTuts.iml
mvnw
ProductRepository.java
ProductController.java
JpaTutrialApplicationTests.java
JpaRepository.class
QueryByExampleExecutor.cl
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.List;
@RestController new *
@RequestMapping(path="/products")
public class ProductController {
private final ProductRepository productRepository; 2 usages
public ProductController(ProductRepository productRepository) { this.productRepository = productRepository; }
// @GetMapping
// public List<ProductEntity> getAllproducts(){
// return productRepository.findBytitleOrderByPrice("Mazza");
// }
@GetMapping new *
public List<ProductEntity> getAllproducts() { return productRepository.findByOrderByPrice(); }
}
Run
JpaTutrialApplication


Project
sitory.class
CoroutineSortingRepository.class
CoroutineCrudRepository.class
Colu
com.cosingshuttle.jpaTutorial.jpa
controllers
ProductController
entities
ProductEntity
repositories
ProductRepository
JpaTutrialApplication
resources
static
templates
application.properties
data.sql [test@localhost]
test
java
com.cosingshuttle.jpaTutorial.jpa
JpaTutrialApplicationTests
target
.gitattributes
.gitignore
HELP.md
jpaTuts.iml
Tx: Auto
Playground
INSERT INTO test.product_table (price,quantity,sku,title_x)
VALUES
(price 12.4, quantity 4, sku 'parle734', title_x 'Parle Biscuit'),
(price 14.4, quantity 1, sku 'pepsi1123', title_x 'Pepsi'),
(price 16.4, quantity 3, sku 'mazza115', title_x 'Mazza'),
(price 10.0, quantity 2, sku 'coke998', title_x 'Coca Cola'),
(price 8.5, quantity 5, sku 'goodday221', title_x 'Good Day Biscuit'),
(price 20.0, quantity 1, sku 'lays443', title_x 'Lays Chips'),
(price 18.75, quantity 6, sku 'kurkure332', title_x 'Kurkure'),
(price 25.0, quantity 2, sku 'bourbon889', title_x 'Bourbon Biscuit'),
(price 30.0, quantity 4, sku 'oreo776', title_x 'Oreo Biscuit'),
(price 15.5, quantity 3, sku 'sprite551', title_x 'Sprite'),
(price 22.0, quantity 5, sku 'fanta661', title_x 'Fanta'),
(price 35.0, quantity 7, sku 'hideNseek909', title_x 'Hide & Seek'),
(price 40.0, quantity 6, sku 'darkfantasy808', title_x 'Dark Fantasy'),
(price 28.0, quantity 4, sku 'tiger112', title_x 'Tiger Biscuit'),
(price 50.0, quantity 2, sku 'redbull667', title_x 'Red Bull'),
(price 60.0, quantity 1, sku 'sting554', title_x 'Sting Energy'),
(price 45.0, quantity 3, sku 'mountaindew443', title_x 'Mountain Dew'),
(price 70.0, quantity 5, sku 'kitkat991', title_x 'KitKat'),
(price 55.0, quantity 6, sku 'perk882', title_x 'Perk'),
(price 90.0, quantity 8, sku 'dairymilk773', title_x 'Dairy Milk');

Sorting with the Sort class

Sort Parameter In Query Methods

```
@Repository
public interface EmployeeRepository extends JpaRepository<Employee,
Long> {
    List<Employee> findByDepartment(String department, Sort sort);
}
```

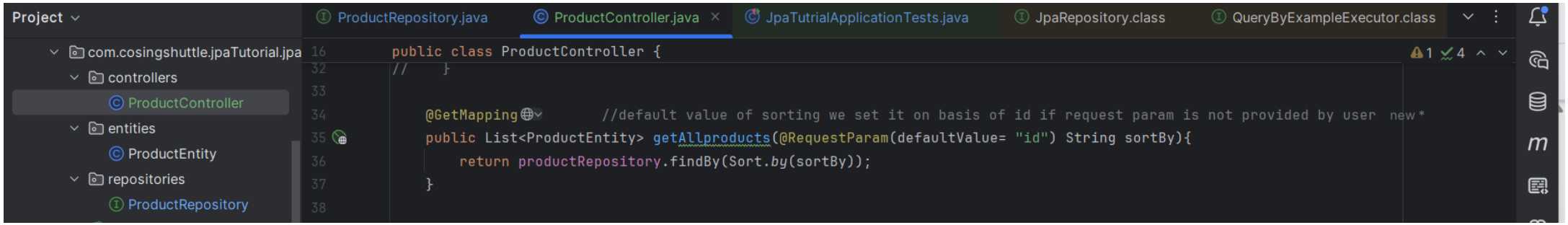
Sorting with the Sort class

Sort Parameter In Query Methods

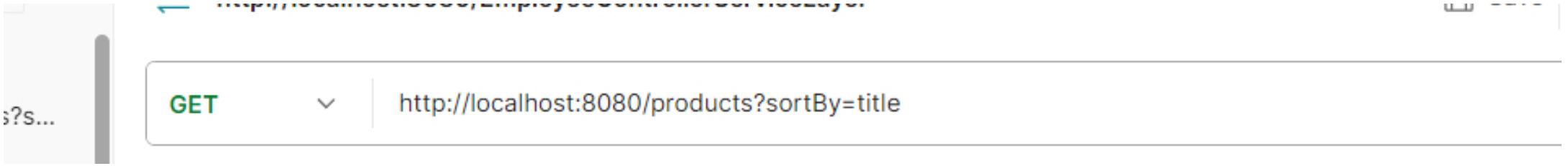
```
@Repository
public interface EmployeeRepository extends JpaRepository<Employee,
Long> {
    List<Employee> findByDepartment(String department, Sort sort);
}
```

Using the Sort class

```
Sort sort = Sort.by(Sort.Direction.ASC, sortField);
Sort sort = Sort.by(Sort.Order.asc("name"), Sort.Order.desc("salary"));
```


Project
com.cosingshuttle.jpaTutorial.jpa
controllers
ProductController
entities
ProductEntity
repositories
ProductRepository
ProductRepository.java
ProductController.java
JpaTutrialApplicationTests.java
JpaRepository.class
QueryByExampleExecutor.class
public class ProductController {
//   }
@GetMapping
//default value of sorting we set it on basis of id if request param is not provided by user
new *
public List<ProductEntity> getAllproducts(@RequestParam(defaultValue= "id") String sortBy){
return productRepository.findBy(Sort.by(sortBy));
}


Project
com.cosingshuttle.jpaTutorial.jpa
controllers
ProductController
entities
ProductEntity
repositories
ProductRepository
JpaTutrialApplication
ProductRepository.java
ProductController.java
JpaTutrialApplicationTests.java
JpaRepository.class
QueryByExampleExecutor.cl
17 public interface ProductRepository extends JpaRepository<ProductEntity,Long> {
47 List<ProductEntity> findBy(Sort sort); 1 usage new *


s?s...
GET
http://localhost:8080/products?sortBy=title

Sorting with the sort class (Sort parameter in query methods)

Below is a **clear, practical guide to Sorting using the `Sort` class in Spring Data JPA**, focused on **how the `Sort` parameter works in query methods**, what Spring does internally, and best practices.

1️⃣ What is `Sort`?

> `Sort` **is a Spring Data JPA abstraction that lets you specify ORDER BY rules programmatically**, without writing JPQL/SQL.

- Package: `org.springframework.data.domain.Sort`
- Provided by **Spring Data JPA** (not JPA)

2️⃣ Using `Sort` as a repository method parameter

Repository

java Copy code

```java
List<Employee> findByDepartment(String department, Sort sort);
```

Service / Controller

java Copy code

```java
Sort sort = Sort.by("salary").descending();
List<Employee> list = repository.findByDepartment("HR", sort);
```

What Spring Data JPA generates (conceptually)

jpql Copy code

```jpql
SELECT e FROM Employee e
WHERE e.department = :department
ORDER BY e.salary DESC
```

📌 **You didn't write JPQL**

📌 **Spring Data JPA appends `ORDER BY`**

3 Multiple sort fields

java Copy code

```java
Sort sort = Sort.by(
    Sort.Order.desc("salary"),
    Sort.Order.asc("name")
);

repository.findByDepartment("HR", sort);
```

JPQL:

jpql Copy code

```jpql
ORDER BY e.salary DESC, e.name ASC
```

---

4 Ascending vs Descending

java Copy code

```java
Sort.by("name")                 // ASC by default
Sort.by("name").ascending()     // explicit ASC
Sort.by("name").descending()    // DESC
```

---

5 Case-insensitive sorting

java Copy code

```java
Sort sort = Sort.by(
    Sort.Order.by("name").ignoreCase()
);
```

JPQL (conceptually):

jpql Copy code

```jpql
ORDER BY LOWER(e.name)
```

6️⃣ Sorting with derived query methods

java Copy code

```java
List<Employee> findByStatus(String status, Sort sort);
```

Usage:

java Copy code

```java
repository.findByStatus("ACTIVE", Sort.by("createdAt").descending());
```

📌 Works with **any derived query**

7️⃣ Sorting with `@Query` methods

JPQL without ORDER BY

java Copy code

```java
@Query("SELECT e FROM Employee e WHERE e.department = :dept")
List<Employee> findEmployees(@Param("dept") String dept, Sort sort);
```

Spring Data JPA:

- **Injects ORDER BY automatically**

⚠️ If your `@Query` already has `ORDER BY`, the `Sort` parameter is **ignored**.

8 Sorting + Pagination together

java Copy code

```java
Page<Employee> findByDepartment(String dept, Pageable pageable);
```

java Copy code

```java
Pageable pageable = PageRequest.of(
    0, 10,
    Sort.by("salary").descending()
);
```

- `Sort` is inside `Pageable`
- Spring applies both `ORDER BY` and `LIMIT/OFFSET`

---

9 Rules & constraints (VERY IMPORTANT)

✅ Property names must match entity fields

java Copy code

```java
Sort.by("salary");   // OK if entity has salary
Sort.by("salary_x"); // ❌ column name, not field name
```

❌ You cannot sort by:

- Computed expressions (unless in `@Query`)
- Non-selected aliases

---

10 Performance best practices

- ✔ Index columns used in sorting
- ❌ Avoid sorting on large text columns
- ❌ Avoid high OFFSET pagination with sorting (slow)
- ✔ Prefer keyset pagination for large datasets

1️⃣1️⃣ Common mistakes ❌

- Using DB column name instead of entity field name
- Expecting `Sort` to work with native queries (it won't unless you handle it)
- Combining `Sort` with `@Query` that already has `ORDER BY`

1️⃣2️⃣ Quick comparison

Approach	Who writes ORDER BY
Method name (`OrderBy`)	You (in name)
`Sort` parameter	Spring Data JPA
`@Query` with ORDER BY	You
Native query	You

✅ One-line interview answer

> `Sort` is a Spring Data JPA abstraction that allows dynamic sorting by appending ORDER BY clauses to generated JPQL queries when passed as a parameter to repository methods

Mental model to remember

- `Sort` = dynamic ORDER BY
- Entity field names, not column names
- Applied by Spring Data JPA before execution

in Sort.by(direction,properties,properties,...) all the properties follow the direction specify in it

Yes — **that's exactly right** ✅

Let's confirm it clearly and remove any edge-case confusion.

1️⃣ What `Sort.by(Direction, properties...)` means

```java
Sort sort = Sort.by(
        Sort.Direction.DESC,
        "salary",
        "createdAt",
        "name"
);
```

👉 **ALL the properties listed will use the SAME direction** (`DESC` here).

Generated ORDER BY (conceptually)

```jpql
ORDER BY
  e.salary DESC,
  e.createdAt DESC,
  e.name DESC
```

2️⃣ Why this constructor exists

This form is for **simple, uniform sorting**, when:

- All fields should be `ASC`, or
- All fields should be `DESC`

It keeps the code short and readable.

3 When you need DIFFERENT directions per property

You must use `Sort.Order`.

java Copy code

```java
Sort sort = Sort.by(
        Sort.Order.desc("salary"),
        Sort.Order.asc("name"),
        Sort.Order.desc("createdAt")
);
```

ORDER BY

jpql Copy code

```jpql
ORDER BY
  e.salary DESC,
  e.name ASC,
  e.createdAt DESC
```

---

4 Quick reference (important)

API	Direction behavior
`Sort.by("name", "salary")`	All ASC
`Sort.by(Direction.DESC, "name", "salary")`	All DESC
`Sort.by(Order...)`	Each property controls its own direction

5️⃣ Common mistake ❌

```java
Sort.by(Direction.DESC, "salary", "name ASC");
```

❌ Invalid

Properties must be **entity field names only**, not SQL fragments.

6️⃣ One-line interview answer

> Yes, when using `Sort.by(Direction, properties...)`, the specified direction applies uniformly to all listed properties.

🧠 Mental rule to remember

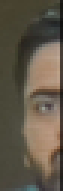
> Direction-first constructor = one direction for all fields.
> Order-based constructor = per-field control.

```java
    @GetMapping       //default value of sorting we set it on basis of id if request param is not provided by user  new*
    public List<ProductEntity> getAllproducts(@RequestParam(defaultValue= "id") String sortBy){
//        Sort.by(Direction,properties,....)
        return productRepository.findBy(Sort.by(Sort.Direction.ASC, ...properties: sortBy,"quantity","price"));
    }

//            Sort.by(Order,Order,....)
    //        return productRepository.findBy(Sort.by(Sort.Order.desc(sortBy),Sort.Order.desc("quantity")));
```

Key Concepts of Pagination

- Page: A single chunk of data that contains a subset of the total dataset. It is an interface representing a page of data, including information about the total number of pages, total number of elements, and the current page's data.
- Pageable: An interface that provides pagination information such as page number, page size, and sorting options.
- PageRequest: A concrete implementation of Pageable that provides methods to create pagination and sorting information.



Using Pageable

```
public interface UserRepository extends JpaRepository<User, Long> {
    Page<User> findAll(Pageable pageable);
    Page<User> findByLastName(String lastName, Pageable pageable);
}
```

Using Pageable

```
public interface UserRepository extends JpaRepository<User, Long> {
    Page<User> findAll(Pageable pageable);
    Page<User> findByLastName(String lastName, Pageable pageable);
}
```

Creating Pageable instance:

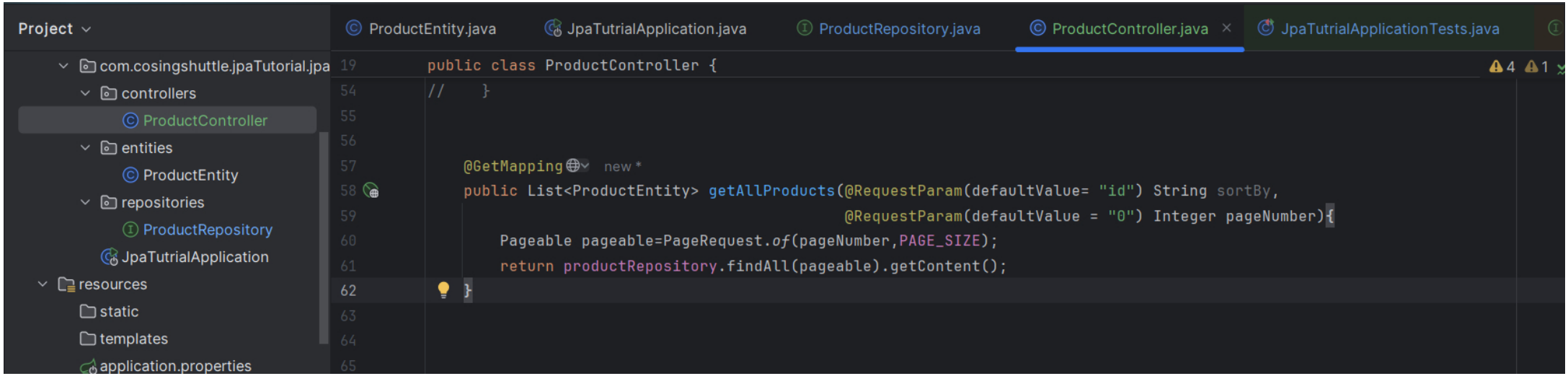
```
Pageable pageable = PageRequest.of(pageNumber, size,
                                   Sort.by("lastName").ascending());
```

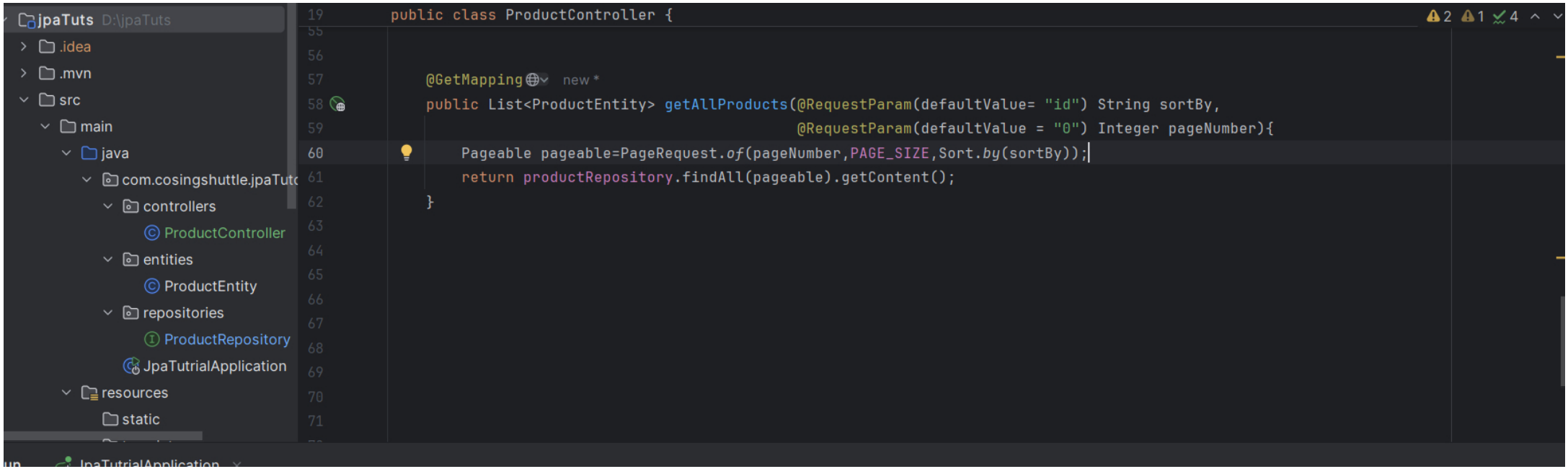

com.cosingshuttle.jpaTutorial.jpa
controllers
ProductController
entities
ProductEntity
repositories
ProductRepository
JpaTutrialApplication
resources
static
templates
19 public class ProductController {
46
47 private final Integer PAGE_SIZE=5; 1 usage
48
49 @GetMapping new *
50 public Page<ProductEntity> getAllProducts(@RequestParam(defaultValue= "id") String sortBy,
51 @RequestParam(defaultValue = "0") Integer pageNumber){
52 Pageable pageable=PageRequest.of(pageNumber,PAGE_SIZE);
53 return productRepository.findAll(pageable);
54 }
55
56


JDBC Hibernate JPA comparisor
localhost:8080/products
localhost:8080/products
Pretty print
{
"content": [
{
"id": 1,
"sku": "parle734",
"title": "Parle Biscuit",
"price": 12.4,
"quantity": 4,
"createdAt": null,
"updatedAt": null
},
{
"id": 2,
"sku": "pepsi1123",
"title": "Pepsi",
"price": 14.4,
"quantity": 1,
"createdAt": null,
"updatedAt": null
},
{
"id": 3,
"sku": "mazza115",
"title": "Mazza",
"price": 16.4,
"quantity": 3,
"createdAt": null,
"updatedAt": null
},
{
"id": 4,
"sku": "coke998",
"title": "Coca Cola",
"price": 10,
"quantity": 2,
"createdAt": null,
"updatedAt": null
},
{
"id": 5,
"sku": "goodday221",
"title": "Good Day Biscuit",
"price": 8.5,
"quantity": 5,
"createdAt": null,
"updatedAt": null
9
11°C

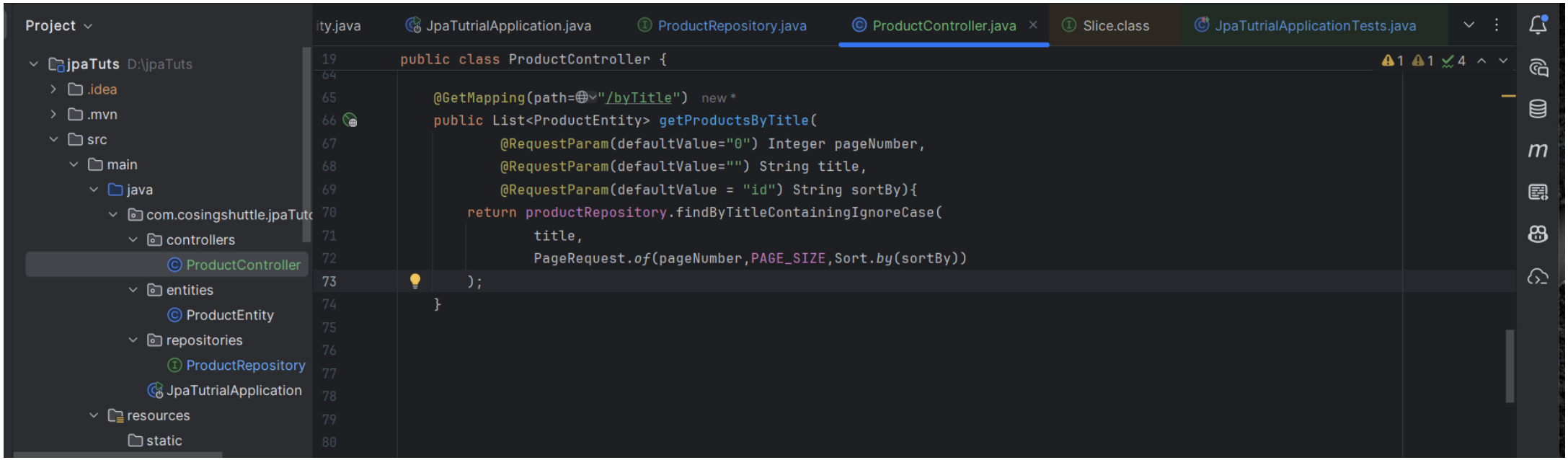
Pretty print


```
    },
    {
      "id": 4,
      "sku": "coke998",
      "title": "Coca Cola",
      "price": 10,
      "quantity": 2,
      "createdAt": null,
      "updatedAt": null
    },
    {
      "id": 5,
      "sku": "goodday221",
      "title": "Good Day Biscuit",
      "price": 8.5,
      "quantity": 5,
      "createdAt": null,
      "updatedAt": null
    }
  ],
  "empty": false,
  "first": true,
  "last": false,
  "number": 0,
  "numberOfElements": 5,
  "pageable": {
    "offset": 0,
    "pageNumber": 0,
    "pageSize": 5,
    "paged": true,
    "sort": {
      "empty": true,
      "sorted": false,
      "unsorted": true
    },
    "unpaged": false
  },
  "size": 5,
  "sort": {
    "empty": true,
    "sorted": false,
    "unsorted": true
  },
  "totalElements": 22,
  "totalPages": 5
}
```


Project
com.cosingshuttle.jpaTutorial.jpa
controllers
ProductController
entities
ProductEntity
repositories
ProductRepository
JpaTutrialApplication
resources
static
templates
application.properties
ProductEntity.java
JpaTutrialApplication.java
ProductRepository.java
ProductController.java
JpaTutrialApplicationTests.java
public class ProductController {
//   }
@GetMapping new *
public List<ProductEntity> getAllProducts(@RequestParam(defaultValue= "id") String sortBy,
@RequestParam(defaultValue = "0") Integer pageNumber){
Pageable pageable=PageRequest.of(pageNumber,PAGE_SIZE);
return productRepository.findAll(pageable).getContent();
}


jpaTuts D:\jpaTuts
.idea
.mvn
src
main
java
com.cosingshuttle.jpaTut
controllers
ProductController
entities
ProductEntity
repositories
ProductRepository
JpaTutrialApplication
resources
static
public class ProductController {
@GetMapping new *
public List<ProductEntity> getAllProducts(@RequestParam(defaultValue= "id") String sortBy,
@RequestParam(defaultValue = "0") Integer pageNumber){
Pageable pageable=PageRequest.of(pageNumber,PAGE_SIZE,Sort.by(sortBy));
return productRepository.findAll(pageable).getContent();
}


Project
jpaTuts D:\jpaTuts
.idea
.mvn
src
main
java
com.cosingshuttle.jpaTut
controllers
ProductController
entities
ProductEntity
repositories
ProductRepository
JpaTutrialApplication
resources
ity.java
JpaTutrialApplication.java
ProductRepository.java
ProductController.java
Slice.class
JpaTutrialApplicationTests.java
public interface ProductRepository extends JpaRepository<ProductEntity,Long> {
List<ProductEntity> findBy(Sort sort); no usages new *
List<ProductEntity> findByTitleContainingIgnoreCase(String title,Pageable pageable); no usages new *


Project
jpaTuts D:\jpaTuts
.idea
.mvn
src
main
java
com.cosingshuttle.jpaTut
controllers
ProductController
entities
ProductEntity
repositories
ProductRepository
JpaTutrialApplication
resources
static
ity.java
JpaTutrialApplication.java
ProductRepository.java
ProductController.java
Slice.class
JpaTutrialApplicationTests.java
public class ProductController {
@GetMapping(path="/byTitle") new *
public List<ProductEntity> getProductsByTitle(
@RequestParam(defaultValue="0") Integer pageNumber,
@RequestParam(defaultValue="") String title,
@RequestParam(defaultValue = "id") String sortBy){
return productRepository.findByTitleContainingIgnoreCase(
title,
PageRequest.of(pageNumber,PAGE_SIZE,Sort.by(sortBy))
);
}

Search history

Today

GET http://localhost:8080/products/b...
GET http://localhost:8080/products/b...
GET http://localhost:8080/products/b...
GET http://localhost:8080/products/b...
GET http://localhost:8080/products?s...
GET http://localhost:8080/products
GET http://localhost:8080/products

Yesterday

GET http://localhost:8080/products
GET http://localhost:8080/products
GET http://localhost:8080/products
GET http://localhost:8080/products?s...
GET http://localhost:8080/products?s...
GET http://localhost:8080/products?s...
GET http://localhost:8080/products?s...
GET http://localhost:8080/products?s...
GET http://localhost:8080/products?s...
GET http://localhost:8080/products?s...
GET http://localhost:8080/products?s...
GET http://localhost:8080/products?s...
GET http://localhost:8080/products?s...
GET http://localhost:8080/products

http://localhost:8080/EmployeeControllerServiceLayer

Save

GET http://localhost:8080/products/byTitle?sortBy=quantity&title=kurkure

Docs Params Authorization Headers (8) Body Scripts Settings

Query Params

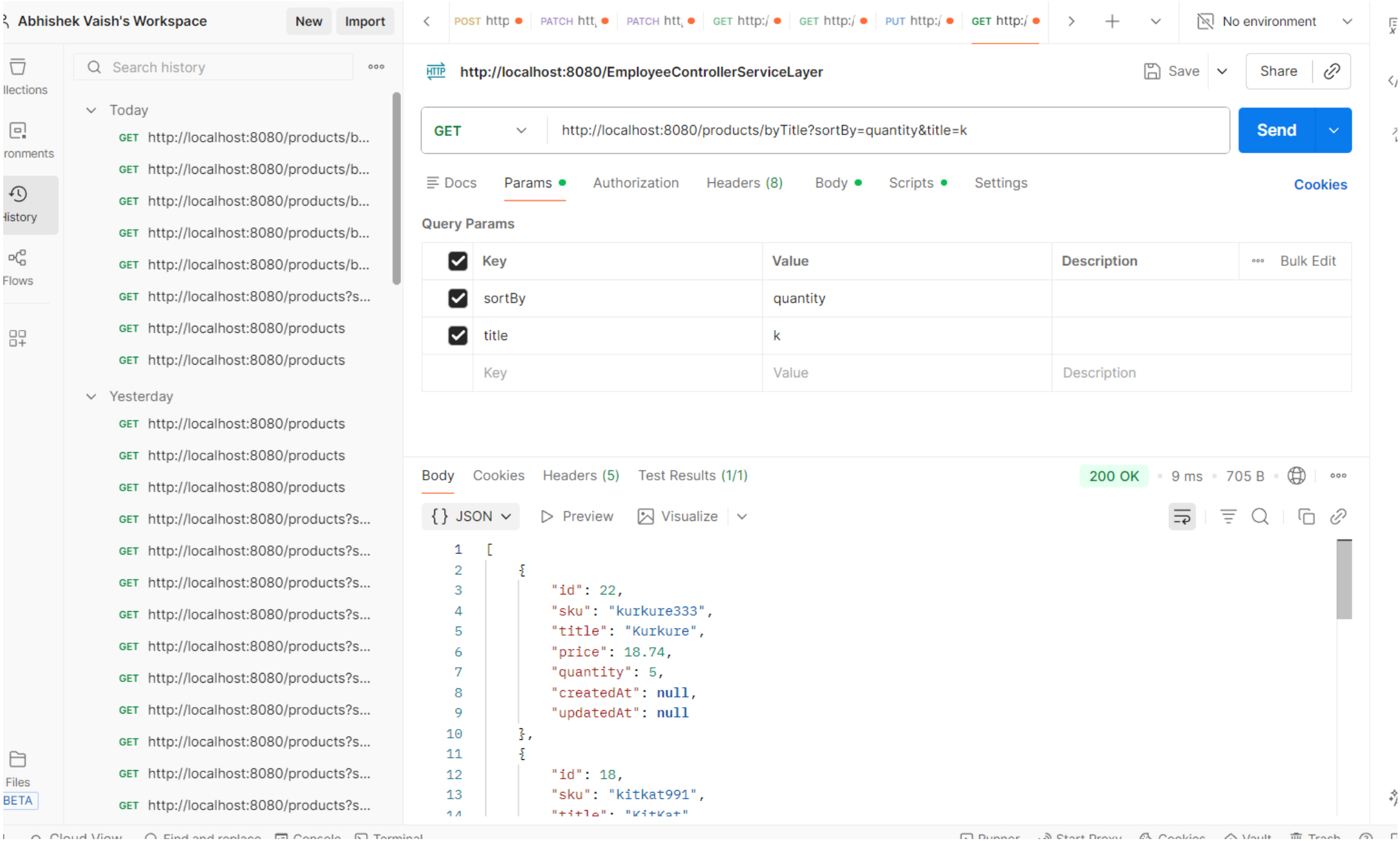
Key	Value	Description
sortBy	quantity	
title	kurkure	
Key	Value	Description

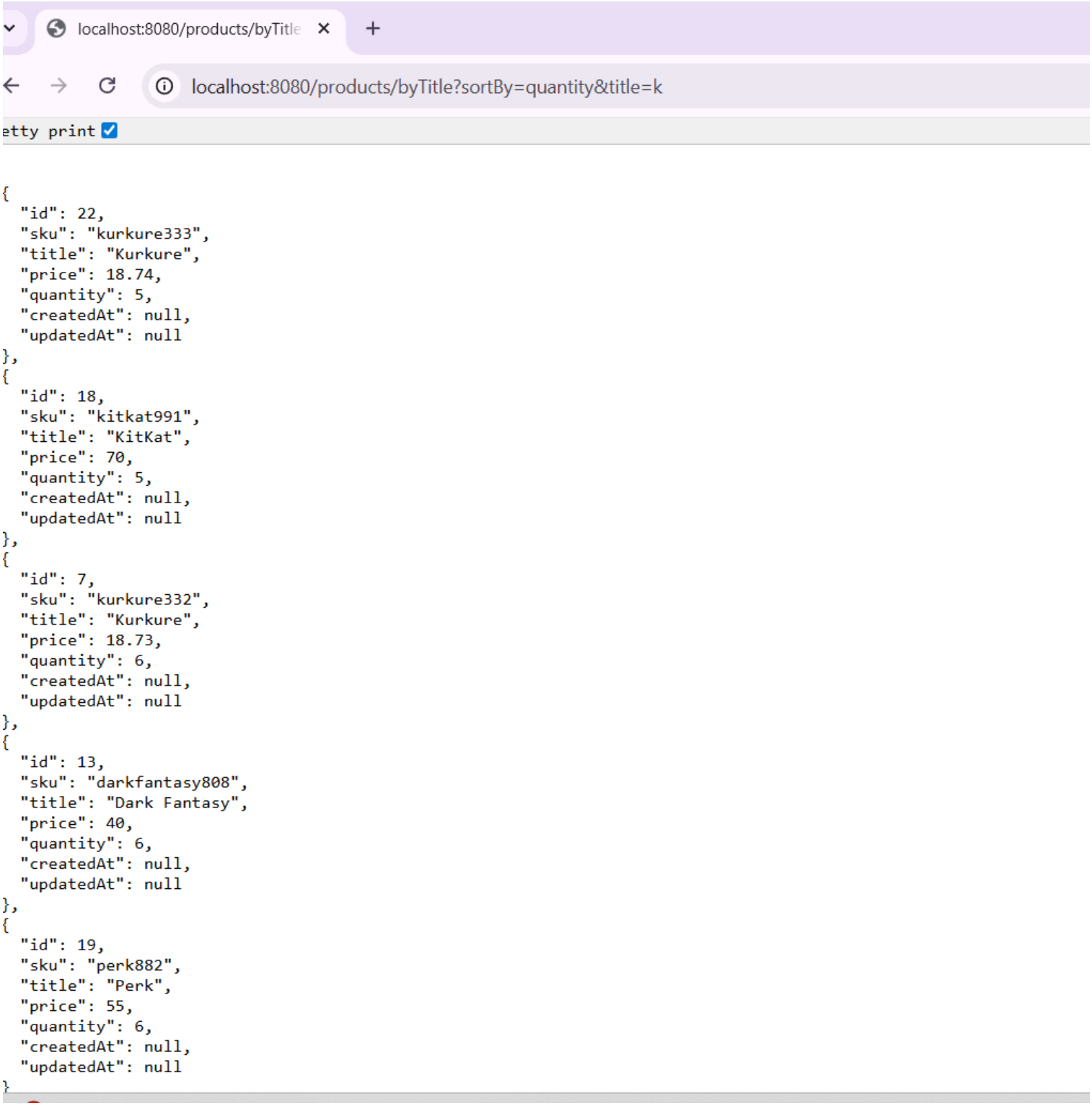
Body Cookies Headers (5) Test Results (1/1)

200 OK 11 ms 488 B

JSON Preview Visualize


```
[
    {
        "id": 22,
        "sku": "kurkure333",
        "title": "Kurkure",
        "price": 18.74,
        "quantity": 5,
        "createdAt": null,
        "updatedAt": null
    },
    {
        "id": 7,
        "sku": "kurkure332",
        "title": "Kurkure"
```


Abhishek Vaish's Workspace
New
Import
POST http
PATCH htt
PATCH htt
GET http:/
GET http:/
PUT http:/
GET http:/
No environment
Search history
Today
GET http://localhost:8080/products/b...
GET http://localhost:8080/products/b...
GET http://localhost:8080/products/b...
GET http://localhost:8080/products/b...
GET http://localhost:8080/products/b...
GET http://localhost:8080/products?s...
GET http://localhost:8080/products
GET http://localhost:8080/products
Yesterday
GET http://localhost:8080/products
GET http://localhost:8080/products
GET http://localhost:8080/products
GET http://localhost:8080/products?s...
GET http://localhost:8080/products?s...
GET http://localhost:8080/products?s...
GET http://localhost:8080/products?s...
GET http://localhost:8080/products?s...
GET http://localhost:8080/products?s...
GET http://localhost:8080/products?s...
GET http://localhost:8080/products?s...
GET http://localhost:8080/products?s...
Collections
Environments
History
Flows
Files
BETA
http://localhost:8080/EmployeeControllerServiceLayer
Save
Share
GET
http://localhost:8080/products/byTitle?sortBy=quantity&title=k
Send
Docs
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
sortBy
quantity
title
k
Key
Value
Description
Body
Cookies
Headers (5)
Test Results (1/1)
200 OK
9 ms
705 B
JSON
Preview
Visualize
1 [
2 {
3 "id": 22,
4 "sku": "kurkure333",
5 "title": "Kurkure",
6 "price": 18.74,
7 "quantity": 5,
8 "createdAt": null,
9 "updatedAt": null
10 },
11 {
12 "id": 18,
13 "sku": "kitkat991",
Cloud View
Find and replace
Console
Terminal
Runner
Start Proxy
Cookies
Vault
Trash




```
{
  "id": 22,
  "sku": "kurkure333",
  "title": "Kurkure",
  "price": 18.74,
  "quantity": 5,
  "createdAt": null,
  "updatedAt": null
},
{
  "id": 18,
  "sku": "kitkat991",
  "title": "KitKat",
  "price": 70,
  "quantity": 5,
  "createdAt": null,
  "updatedAt": null
},
{
  "id": 7,
  "sku": "kurkure332",
  "title": "Kurkure",
  "price": 18.73,
  "quantity": 6,
  "createdAt": null,
  "updatedAt": null
},
{
  "id": 13,
  "sku": "darkfantasy808",
  "title": "Dark Fantasy",
  "price": 40,
  "quantity": 6,
  "createdAt": null,
  "updatedAt": null
},
{
  "id": 19,
  "sku": "perk882",
  "title": "Perk",
  "price": 55,
  "quantity": 6,
  "createdAt": null,
  "updatedAt": null
}
```

Pretty print


```
[
  {
    "id": 22,
    "sku": "kurkure333",
    "title": "Kurkure",
    "price": 18.74,
    "quantity": 5,
    "createdAt": null,
    "updatedAt": null
  },
  {
    "id": 7,
    "sku": "kurkure332",
    "title": "Kurkure",
    "price": 18.73,
    "quantity": 6,
    "createdAt": null,
    "updatedAt": null
  },
  {
    "id": 21,
    "sku": "kurkure331",
    "title": "Kurkure",
    "price": 18.75,
    "quantity": 6,
    "createdAt": null,
    "updatedAt": null
  }
]
```

1️⃣ What is Pagination (in simple words)

> **Pagination splits a large result set into smaller pages** so you fetch only what you need (e.g., 10 records at a time).

Why it matters:

- Faster responses
- Lower memory usage
- Better UX (page 1, page 2, next/prev)

2️⃣ Who provides Pagination?

Layer	Pagination
JPA	❌ (only low-level setters)
Hibernate	❌ (no high-level abstraction)
Spring Data JPA	✅ **YES**

Spring Data JPA adds:

- `Pageable`
- `Page`
- `Slice`
- `PageRequest`

3️⃣ Core classes (must know)

`Pageable`

> Describes **which page**, **how many records**, and **sorting**.

`Page<T>`

> A page of data **plus metadata** (total pages, total elements, etc.).

`Slice<T>`

> A lightweight page **without total count** (faster).

`PageRequest`

> Factory to create `Pageable`.

4️⃣ Repository method signatures (rules)

java Copy code

```java
Page<Employee> findByDepartment(String dept, Pageable pageable);
Slice<Employee> findByStatus(String status, Pageable pageable);
List<Employee> findByRole(String role, Pageable pageable); // allowed, but no metadata
```

Rules

- `Pageable` must be the **last parameter**
- Works with **derived queries** and **@Query (JPQL)**

5 Creating a `Pageable`

java Copy code

```java
Pageable pageable = PageRequest.of(0, 10); // page 0, size 10
```

- Page index is **0-based**
- `size` = records per page

With sorting

java Copy code

```java
Pageable pageable = PageRequest.of(
    0, 10,
    Sort.by("salary").descending()
);
```

---

6 Calling the repository

java Copy code

```java
Page<Employee> page =
    repository.findByDepartment("HR", pageable);
```

7️⃣ What Spring Data JPA generates (important)

Spring Data JPA generates **TWO queries** for `Page<T>`:

A) Data query (with LIMIT/OFFSET)

```jpql
SELECT e FROM Employee e
WHERE e.department = :dept
ORDER BY e.salary DESC
```

→ Hibernate converts to SQL:

```sql
SELECT *
FROM employee
WHERE department = ?
ORDER BY salary DESC
LIMIT 10 OFFSET 0;
```

B) Count query

```jpql
SELECT COUNT(e)
FROM Employee e
WHERE e.department = :dept
```

This count query is why `Page<T>` can be slower on big tables.

8 `Page<T>` vs `Slice<T>` (VERY IMPORTANT)

Feature	Page	Slice
Content	✅	✅
Total elements	✅	❌
Total pages	✅	❌
Count query	❌ No	❌ No
Performance	Slower	Faster

Use `Slice` when:

- Infinite scroll
- You only need "is there a next page?"

---

9 Reading `Page<T>` metadata

java Copy code

```java
Page<Employee> page = ...;

page.getContent();       // List<Employee>
page.getNumber();        // current page (0-based)
page.getSize();          // page size
page.getTotalElements(); // total rows
page.getTotalPages();    // total pages
page.hasNext();          // boolean
page.isFirst();          // boolean
```

🔟 Pagination + `@Query`

JPQL without ORDER BY (recommended)

java Copy code

```java
@Query("SELECT e FROM Employee e WHERE e.department = :dept")
Page<Employee> find(@Param("dept") String dept, Pageable pageable);
```

Spring Data JPA:

- Appends `ORDER BY`
- Applies `LIMIT/OFFSET`
- Generates count query automatically

⚠️ If your `@Query` already has `ORDER BY`, `Sort` in `Pageable` is ignored.

1️⃣1️⃣ Pagination with native queries

java Copy code

```java
@Query(
  value = "SELECT * FROM employee WHERE department = :dept",
  countQuery = "SELECT COUNT(*) FROM employee WHERE department = :dept",
  nativeQuery = true
)
Page<Employee> findNative(@Param("dept") String dept, Pageable pageable);
```

Rule: For native queries, **you MUST provide** `countQuery`.

1️⃣2️⃣ Common pitfalls ❌

- ❌ Forgetting countQuery in native SQL
- ❌ Sorting by **column name** instead of **entity field**
- ❌ Large `OFFSET` (slow for big pages)
- ❌ Using `Page<T>` when `Slice<T>` would be enough

1️⃣2️⃣ Common pitfalls ❌

- ❌ Forgetting countQuery in native SQL
- ❌ Sorting by **column name** instead of **entity field**
- ❌ Large `OFFSET` (slow for big pages)
- ❌ Using `Page<T>` when `Slice<T>` would be enough

1️⃣3️⃣ Performance tips (real-world)

- ✔️ Index columns used in sorting/filtering
- ✔️ Prefer `Slice` for large datasets
- ✔️ Avoid high page numbers with OFFSET
- ✔️ Consider **keyset pagination** for very large tables

1️⃣4️⃣ Keyset pagination (when OFFSET is too slow)

Instead of:

```sql
LIMIT 10 OFFSET 100000
```

Use:

```sql
WHERE id > :lastSeenId
ORDER BY id
LIMIT 10
```

(Spring Data JPA supports this via custom queries.)

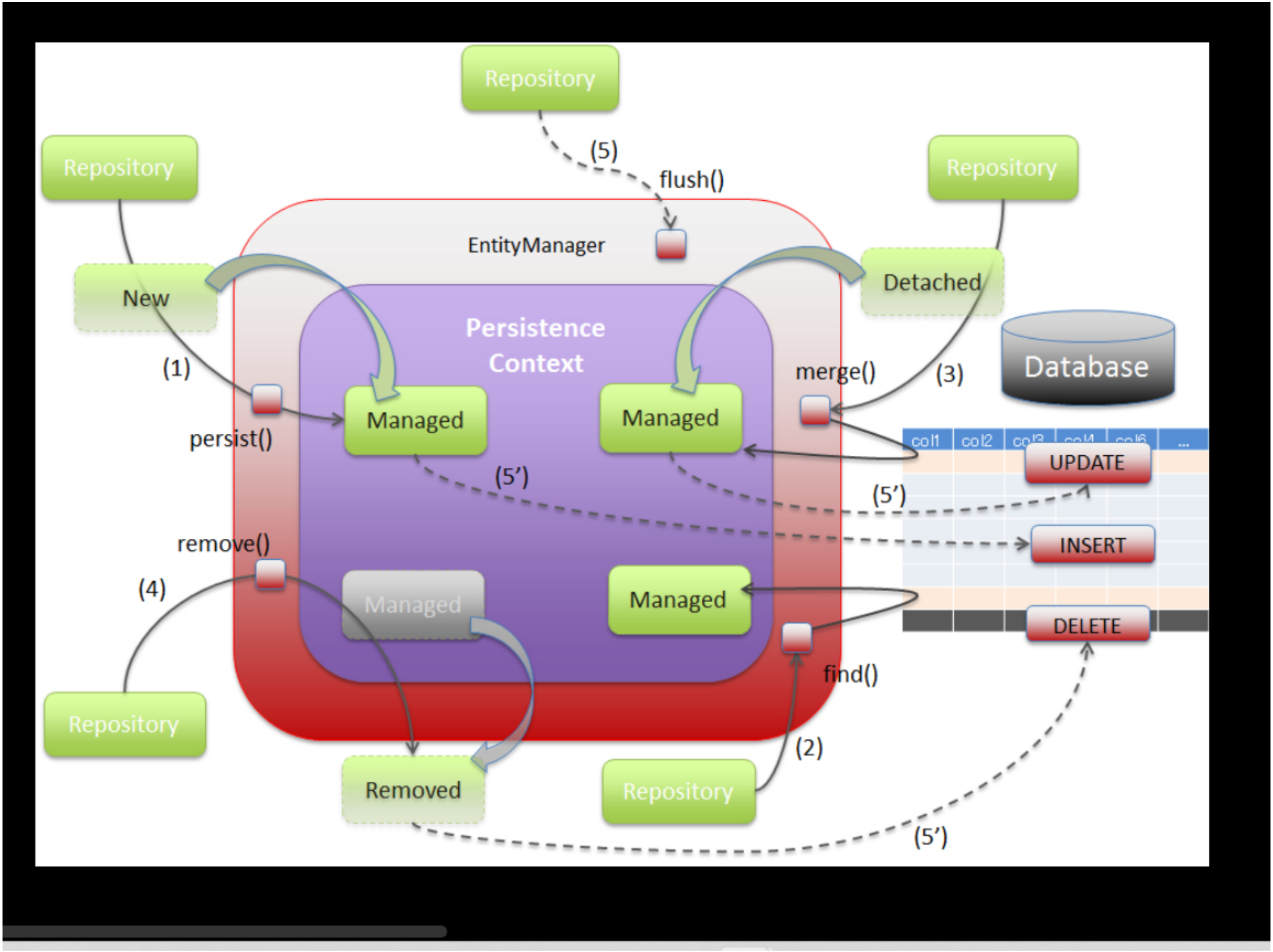
```pgsql
Controller
   ↓
Repository (Pageable)
   ↓
Spring Data JPA
   ↓  (adds ORDER BY, LIMIT, OFFSET)
EntityManager
   ↓
Hibernate
   ↓
SQL (LIMIT/OFFSET)
   ↓
Database
```

✅ One-line interview answer

> Pagination in Spring Data JPA is implemented using `Pageable`, where Spring Data generates paged JPQL/SQL queries with LIMIT/OFFSET and optionally a count query, returning results as `Page` or `Slice`.

🧠 Final memory hook

- `Pageable` = request (page + size + sort)
- `Page` = data + counts
- `Slice` = data + next/prev only
- **Spring Data JPA** does all the heavy lifting


Repository
(5)
flush()
Repository
Repository
EntityManager
New
Detached
Persistence
Context
Database
(1)
merge()
(3)
persist()
Managed
Managed
col1
col2
col3
col4
col6
...
UPDATE
(5')
(5')
INSERT
remove()
(4)
Managed
Managed
DELETE
find()
Repository
(2)
Removed
Repository
(5')