# Spring Boot | Pagination and Sorting with Spring Data JPA

Sort and Pagination both are comes from Spring framework data domain.

**import org.springframework.data.domain.Sort;**

We will learn How to Implement Sorting and Pagination mechanism in Spring Boot using Spring Data JPA. Basically, Pagination is helpful when we have a large dataset, and we want to present it to the user in a smaller chunk.

For example, we have a one backend api which is giving response as 100 records when we have an api call. So, if I will try to map the 100 objects to a particular table then user will end up with the scroll down to see all the records. I think minimum 5 to 6 times will scroll, which is not recommended.

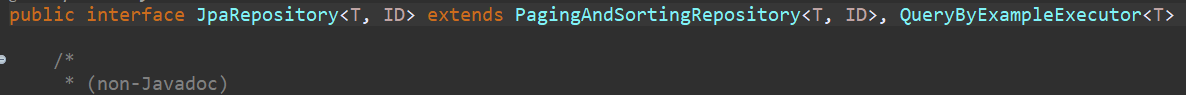
Why we will not give the flexibility to the end user, they can set their limit, or they can set their page size the number of row they wants to view. So for example user only want to view 10 records per page so we can set the limit to view only that much records into a single page. So you can customize based on ur requirements.

Let’s say User want to view 25 records in a single API call. This is how we can give flexibility to end-user. They can paginate the response the way they want.

**Application**- **spring-boot-pagination-sorting**

**Dependencies**- **Lombok** (avoid manually writing getter/setter), **Spring Web** (To expose REST API), **MySQL Driver** (MySQL Connector), **Spring Data JPA**

So, we have created one Entity called **Product** and table name **PRODUCT\_TBL.** And fields like an id, name quantity and price. Similarly, I have created a **ProductRepository** which extends from **PagingAndSortingRepository<Product, Integer**>**.**



As we can see if we use JpaRepository the By default JpaRepository extends **PagingAndSortingRepository<Product, Integer**>**.** by extending JpaRepository as well we can enhance all Pagination and sorting logic.

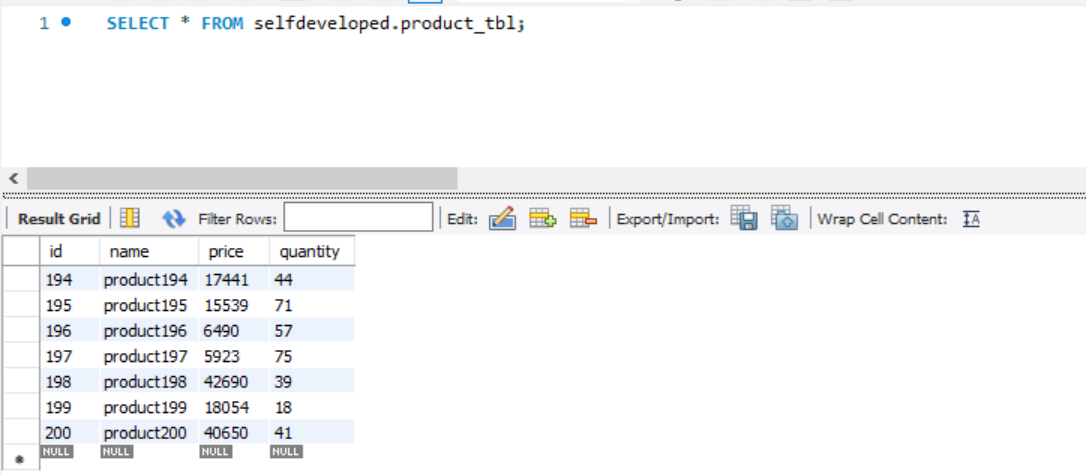
Similarly let’s create a service.

Let’s write 2 methods…

So, 1st method will execute at the time of my application startup, bcz it annotates with **@PostConstruct.**

So, I have created 200 Product object and saved in our database.

So, I just need to comment this because already I executed this so if u observed in our db 200 data has been stored.



So, this is like **init** () method so the alternate of **init** () method is **@PostConstruct.**

So, at the time of application startup if u want to perform some pre-processing logic, then u can go for **@PostConstruct.**

And the code is simple here, I just iterate a loop from 1 to 200, then I Randomly created 200 Product object and I just collect them as a list and then save them in our repo. So I have commented out bcz already we have a records in db.

***@PostConstruct***

**public void initDB() {**

**List<Product> products = IntStream.*rangeClosed*(1, 200)**

**.mapToObj(i -> new Product("product" + i, new Random().nextInt(100), new Random().nextInt(50000)))**

**.collect(Collectors.*toList*());**

**repository.saveAll(products);**

**}**

**To get all data from database**

**public Iterable<Product> findAllProducts () {**

**return repository. findAll ();**

**}**

**We have created controller …the main class I am using as a controller**

So here I wrote one method to get all the products presents in my db. If u observer also I created one **DTO** class **APIResponse,** here I also want to count the records getting from the database.

***@Data***

***@AllArgsConstructor***

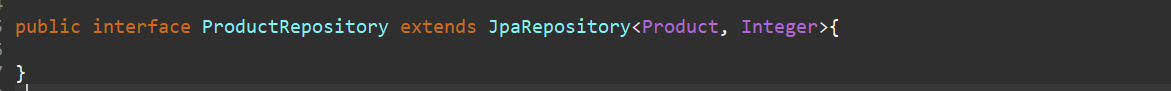
***@NoArgsConstructor***

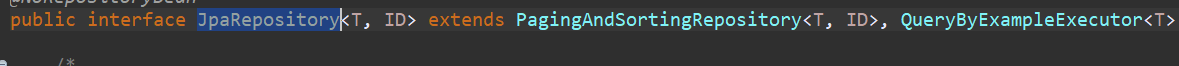
**public class APIResponse<T> {**

**int recordCount;**

**T response;**

**}**





***@GetMapping***

**private APIResponse<List<Product>> getProducts() {**

**List<Product> allProducts = service.findAllProducts();**

**return new APIResponse<>(allProducts.size(), allProducts);**

**}**

Let’s start application and go to postman and fetch all 200 records.

**GET** [**http://localhost:9090/products**](http://localhost:9090/products)

Response:

**{**

**"recordCount": 200,**

**"response": [**

**{**

**"id": 1,**

**"name": "product1",**

**"quantity": 65,**

**"price": 32609**

**},**

**{**

**"id": 2,**

**"name": "product2",**

**"quantity": 31,**

**"price": 1306**

**},**

**{**

**"id": 3,**

**"name": "product3",**

**"quantity": 35,**

**"price": 1959**

**},**

**.......................**

**{**

**"id": 199,**

**"name": "product199",**

**"quantity": 18,**

**"price": 18054**

**},**

**{**

**"id": 200,**

**"name": "product200",**

**"quantity": 41,**

**"price": 40650**

**}**

**]**

**}**

Now the record count is 200. and we have all the response with us. **Now let’s implement the Sorting based on the any field which would be dynamic and pagination.**

Let’s go to service class we will write one method which maintain the sorting dynamically based on to the fields.

So, there is one method

**Sort.by(field)**

**public List<Product> findProductsWithSorting (String field){**

**return repository.findAll(Sort.*by*(field));**

**}**

>> if u want to sort based on Ascending or Descending order then

**Sort.by (Sort.Direction.ASC, field)**

**public List<Product> findProductsWithSorting (String field) {**

**return repository. findAll (Sort.*by*(Sort.*Direction*.*ASC*, field));**

**}**

Now I will go to the controller and I ll call this method.

***@GetMapping*("/{field}")**

**private APIResponse<List<Product>> getProductsWithSorting(*@PathVariable* String field) {**

**List<Product> allProducts = service.findProductsWithSorting(field);**

**return new APIResponse<>(allProducts.size(), allProducts);**

**}**

Now let me start my application …

So, I just want to sort based on quantity…

**Request Url:** [**http://localhost:9090/products/quantity**](http://localhost:9090/products/quantity)

**Response---**

{

    "recordCount": 200,

    "response": [

        {

            "id": 189,

            "name": "product189",

            "quantity": 2,

            "price": 35646

        },

        {

            "id": 15,

            "name": "product15",

            "quantity": 4,

            "price": 20501

        },

        {

            "id": 58,

            "name": "product58",

            "quantity": 4,

            "price": 33615

        },

        {

            "id": 176,

            "name": "product176",

            "quantity": 4,

            "price": 9597

        },

        {

            "id": 83,

            "name": "product83",

            "quantity": 6,

            "price": 39847

        },

        {

            "id": 150,

            "name": "product150",

            "quantity": 6,

            "price": 38187

        },

        {

            "id": 167,

            "name": "product167",

            "quantity": 6,

            "price": 24404

        },

        {

            "id": 30,

            "name": "product30",

            "quantity": 8,

            "price": 3524

        },

        {

            "id": 98,

            "name": "product98",

            "quantity": 10,

            "price": 17122

        },

        {

            "id": 64,

            "name": "product64",

            "quantity": 11,

            "price": 40015

        },

        {

            "id": 153,

            "name": "product153",

            "quantity": 11,

            "price": 4070

        },

        {

            "id": 175,

            "name": "product175",

            "quantity": 11,

            "price": 17717

        }

……

]

}

Similarly, you can sort based on any fields….

Now let’s move on Pagination Implementation…

So, if you observed the record count is 200, so I can’t return 200 records to the User Interface. Even if I return, they can’t map entire 200 records in angular or any front-end technology. They need to go for client-side pagination. But I just want to give the flexi from backend itself based on their input I will return number of elements to them. So, what I can do I can directly implement into that lets directly go to the service class and try to do that…

For pagination I need to pass 2 arguments one is **offset,** and another is **page size**. So here the page size will be item per page. And offset will be a Page or your next element. And it will return Page of object.

**public Page<Product> findProductsWithPagination (int offset, int pageSize) {**

**Page<Product> products= repository.findAll(PageRequest.*of*(offset, pageSize));**

**return products;**

**}**

Spring data will take care this Pagination based on the **PageRequest** class.

***@GetMapping*("/pagination/{offset}/{pageSize}")**

**private APIResponse<Page<Product>> getProductsWithPagination(*@PathVariable* int offset, *@PathVariable* int pageSize) {**

**Page<Product> productsWithPagination = service.findProductsWithPagination(offset, pageSize);**

**return new APIResponse<>(productsWithPagination.getSize(), productsWithPagination);**

**}**

Now we added the Pagination API and provided the offset and Page Size. Spring data will take care this Pagination based on the **PageRequest.**

Let’s re-start my application to verify pagination scenario. So, if u want to see next elements or next page elements u just need to change the offset value. Offset will start from 0.

**Request -** [**http://localhost:9090/products/pagination/0/5**](http://localhost:9090/products/pagination/0/5) **(**offset 0 means 1st page and 5 is per page records**)**

**Response-**

**{**

**"recordCount": 5,**

**"response": {**

**"content": [**

**{**

**"id": 1,**

**"name": "product1",**

**"quantity": 65,**

**"price": 32609**

**},**

**{**

**"id": 2,**

**"name": "product2",**

**"quantity": 31,**

**"price": 1306**

**},**

**{**

**"id": 3,**

**"name": "product3",**

**"quantity": 35,**

**"price": 1959**

**},**

**{**

**"id": 4,**

**"name": "product4",**

**"quantity": 14,**

**"price": 20466**

**},**

**{**

**"id": 5,**

**"name": "product5",**

**"quantity": 21,**

**"price": 12914**

**}**

**],**

**"pageable": {**

**"sort": {**

**"empty": true,**

**"sorted": false,**

**"unsorted": true**

**},**

**"offset": 0,**

**"pageNumber": 0,**

**"pageSize": 5,**

**"paged": true,**

**"unpaged": false**

**},**

**"last": false,**

**"totalPages": 40,**

**"totalElements": 200,**

**"size": 5,**

**"number": 0,**

**"sort": {**

**"empty": true,**

**"sorted": false,**

**"unsorted": true**

**},**

**"numberOfElements": 5,**

**"first": true,**

**"empty": false**

**}**

**}**

**Request -** [**http://localhost:9090/products/pagination/4/7**](http://localhost:9090/products/pagination/4/7) **(**offset 4 means 5th page and 7 is per page records**)**

**Response-**

**{**

**"recordCount": 7,**

**"response": {**

**"content": [**

**{**

**"id": 29,**

**"name": "product29",**

**"quantity": 65,**

**"price": 32952**

**},**

**{**

**"id": 30,**

**"name": "product30",**

**"quantity": 8,**

**"price": 3524**

**},**

**{**

**"id": 31,**

**"name": "product31",**

**"quantity": 48,**

**"price": 17432**

**},**

**{**

**"id": 32,**

**"name": "product32",**

**"quantity": 37,**

**"price": 43690**

**},**

**{**

**"id": 33,**

**"name": "product33",**

**"quantity": 46,**

**"price": 1696**

**},**

**{**

**"id": 34,**

**"name": "product34",**

**"quantity": 34,**

**"price": 33510**

**},**

**{**

**"id": 35,**

**"name": "product35",**

**"quantity": 99,**

**"price": 48143**

**}**

**],**

**"pageable": {**

**"sort": {**

**"empty": true,**

**"sorted": false,**

**"unsorted": true**

**},**

**"offset": 28,**

**"pageNumber": 4,**

**"pageSize": 7,**

**"paged": true,**

**"unpaged": false**

**},**

**"last": false,**

**"totalPages": 29,**

**"totalElements": 200,**

**"size": 7,**

**"number": 4,**

**"sort": {**

**"empty": true,**

**"sorted": false,**

**"unsorted": true**

**},**

**"numberOfElements": 7,**

**"first": false,**

**"empty": false**

**}**

**}**

**If u want to implement the pagination as well as sorting in same API that also you can do…**

So, this would be the combination of both Pagination as well as Sorting.

**public Page<Product> findProductsWithPaginationandSorting(int offset, int pageSize,String field) {**

**Page<Product> products= repository.findAll(PageRequest.*of*(offset, pageSize).withSort(Sort.*by*(field)));**

**return products;**

}

Let’s write an API-

***@GetMapping*("/pagination/{offset}/{pageSize}/{field}")**

**private APIResponse<Page<Product>> getProductsWithPaginationandSort(*@PathVariable* int offset, *@PathVariable* int pageSize, *@PathVariable* String field) {**

**Page<Product> productsWithPagination = service.findProductsWithPaginationandSorting(offset, pageSize,field);**

**return new APIResponse<>(productsWithPagination.getSize(), productsWithPagination);**

**}**

Let’s re-start our application-

I just change the url pagination and sort. And I just want to see the 10 elements from 0. And I just want to sort based on the id.

**GET** [**http://localhost:9090/products/paginationAndSort/0/10/id**](http://localhost:9090/products/paginationAndSort/0/10/id)

**Response -**

**{**

**"recordCount": 10,**

**"response": {**

**"content": [**

**{**

**"id": 1,**

**"name": "product1",**

**"quantity": 65,**

**"price": 32609**

**},**

**{**

**"id": 2,**

**"name": "product2",**

**"quantity": 31,**

**"price": 1306**

**},**

**{**

**"id": 3,**

**"name": "product3",**

**"quantity": 35,**

**"price": 1959**

**},**

**{**

**"id": 4,**

**"name": "product4",**

**"quantity": 14,**

**"price": 20466**

**},**

**{**

**"id": 5,**

**"name": "product5",**

**"quantity": 21,**

**"price": 12914**

**},**

**{**

**"id": 6,**

**"name": "product6",**

**"quantity": 18,**

**"price": 37206**

**},**

**{**

**"id": 7,**

**"name": "product7",**

**"quantity": 42,**

**"price": 17164**

**},**

**{**

**"id": 8,**

**"name": "product8",**

**"quantity": 20,**

**"price": 44750**

**},**

**{**

**"id": 9,**

**"name": "product9",**

**"quantity": 15,**

**"price": 7020**

**},**

**{**

**"id": 10,**

**"name": "product10",**

**"quantity": 65,**

**"price": 10947**

**}**

**],**

**"pageable": {**

**"sort": {**

**"empty": false,**

**"sorted": true,**

**"unsorted": false**

**},**

**"offset": 0,**

**"pageNumber": 0,**

**"pageSize": 10,**

**"paged": true,**

**"unpaged": false**

**},**

**"totalPages": 20,**

**"last": false,**

**"totalElements": 200,**

**"size": 10,**

**"number": 0,**

**"sort": {**

**"empty": false,**

**"sorted": true,**

**"unsorted": false**

**},**

**"numberOfElements": 10,**

**"first": true,**

**"empty": false**

**}**

**}**

Now, I just want to see the 6 elements from offset 1(**page:2**) and need to sort based on the **quantity**

**GET -** [**http://localhost:9090/products/paginationAndSort/1/6/quantity**](http://localhost:9090/products/paginationAndSort/1/6/quantity)

**{**

**"recordCount": 6,**

**"response": {**

**"content": [**

**{**

**"id": 167,**

**"name": "product167",**

**"quantity": 6,**

**"price": 24404**

**},**

**{**

**"id": 30,**

**"name": "product30",**

**"quantity": 8,**

**"price": 3524**

**},**

**{**

**"id": 98,**

**"name": "product98",**

**"quantity": 10,**

**"price": 17122**

**},**

**{**

**"id": 175,**

**"name": "product175",**

**"quantity": 11,**

**"price": 17717**

**},**

**{**

**"id": 64,**

**"name": "product64",**

**"quantity": 11,**

**"price": 40015**

**},**

**{**

**"id": 153,**

**"name": "product153",**

**"quantity": 11,**

**"price": 4070**

**}**

**],**

**"pageable": {**

**"sort": {**

**"empty": false,**

**"sorted": true,**

**"unsorted": false**

**},**

**"offset": 6,**

**"pageNumber": 1,**

**"pageSize": 6,**

**"paged": true,**

**"unpaged": false**

**},**

**"totalPages": 34,**

**"last": false,**

**"totalElements": 200,**

**"size": 6,**

**"number": 1,**

**"sort": {**

**"empty": false,**

**"sorted": true,**

**"unsorted": false**

**},**

**"numberOfElements": 6,**

**"first": false,**

**"empty": false**

**}**

**}**