Pagination and Sorting using Spring Data JPA

Last modified: December 16, 2018

by [Shubhra Srivastava](https://www.baeldung.com/author/shubhra-srivastava/)

* [**Spring Data**](https://www.baeldung.com/category/persistence/spring-persistence/spring-data/)

**I just announced the new *Learn Spring*course, focused on the fundamentals of Spring 5 and Spring Boot 2:**

[**>> CHECK OUT THE COURSE**](https://www.baeldung.com/ls-course-start)

**1. Overview**

Pagination is often helpful when we have a large dataset and we want to present it to the user in smaller chunks.

Also, we often need to sort that data by some criteria while paging.

In this tutorial, **we’ll learn how to easily paginate and sort using Spring Data JPA.**

**2. Initial Setup**

First, let’s say we have a *Product* entity:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | @Entity  public class Product {        @Id      private long id;      private String name;      private double price;        // constructors, getters and setters    } |

as our domain class. Each of our *Product* instances has a unique identifier – *id*, its *name* and its *price*associated with it.

**3. Creating a Repository**

To access our *Product*s, we’ll need a *ProductRepository*:

|  |  |
| --- | --- |
| 1  2  3  4 | public interface ProductRepository extends PagingAndSortingRepository<Product, Integer> {        List<Product> findAllByPrice(double price, Pageable pageable);  } |

By having it extend[***PagingAndSortingRepository***](https://docs.spring.io/spring-data/data-commons/docs/current/api/org/springframework/data/repository/PagingAndSortingRepository.html)**, we get *findAll(Pageable pageable)*and*findAll(Sort sort)*methods for paging and sorting.**

Or, we could have chosen to extend [*JpaRepository*](https://www.baeldung.com/spring-data-repositories) instead, as it extends *PagingAndSortingRepository*, too.

Once we extend *PagingAndSortingRepository*, **we can add our own methods that take *Pageable* and *Sort*as parameters**, as we did here with *findAllByPrice*.

Let’s take a look at how to paginate our *Product*s using our new method.

**4. Pagination**

Once we have our repository extending from *PagingAndSortingRepository*, we just need to:

1. Create or obtain a *PageRequest* object, which is an implementation of the *Pageable* interface
2. Pass the *PageRequest* object as an argument to the repository method we intend to use

We can create a *PageRequest* object by passing in the requested page number and the page size.

Here,**the page counts starts at zero:**

|  |  |
| --- | --- |
| 1  2  3 | Pageable firstPageWithTwoElements = PageRequest.of(0, 2);    Pageable secondPageWithFiveElements = PageRequest.of(1, 5); |

In Spring MVC, we can also choose to obtain the *Pageable* instance in our controller using [Spring Data Web Support](https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#core.web).

Once we have our *PageRequest* object\, we can pass it in while invoking our repository’s method:

|  |  |
| --- | --- |
| 1  2  3  4 | Page<Product> allProducts = productRepository.findAll(firstPageWithTwoElements);    List<Product> allTenDollarProducts =    productRepository.findAllByPrice(10, secondPageWithFiveElements); |

The *findAll(Pageable pageable)* method by default returns a *Page<T>* object.

However, **we can choose to return either a *Page<T>,*a*Slice<T>* or a *List<T>* from any of our custom methods returning a paginated data**.

A *Page<T>* instance, in addition to having the list of *Product*s, also knows about the total number of available pages. **It triggers an additional count query to achieve it. To avoid such an overhead cost, we can instead return a *Slice<T>* or a *List<T>*.**

A *Slice* only knows about whether the next slice is available or not.

**5. Pagination and Sorting**

Similarly, to just have our query results sorted, we can simply [pass an instance of *Sort*](https://www.baeldung.com/spring-data-sorting) to the method:

|  |  |
| --- | --- |
| 1 | Page<Product> allProductsSortedByName = productRepository.findAll(Sort.by("name")); |

However, what if we want to**both sort and page our data?**

We can do that by passing the sorting details into our *PageRequest* object itself:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | Pageable sortedByName =    PageRequest.of(0, 3, Sort.by("name"));    Pageable sortedByPriceDesc =    PageRequest.of(0, 3, Sort.by("price").descending());    Pageable sortedByPriceDescNameAsc =    PageRequest.of(0, 5, Sort.by("price").descending().and(Sort.by("name"))); |

Based on our sorting requirements, **we can specify the sort fields and the sort direction** while creating our *PageRequest* instance.

As usual, we can then pass this *Pageable* type instance to the repository’s method.