# Pagination and Sorting using Spring Data JPA

### Description

## Introduction

In this post we are going to see the pagination and sorting feature provided by Spring data JPA and how it will avoid problems and boost application performance.

## Pagination in Spring Data JPA

Pagination is one of the important feature every developer should use wisely. Pagination is used to represent the large dataset into smaller chunks. Paginating the database results is one of the best ways to increase the performance of your JPA application. If Pagination is not used then it might slow down the performance of your application drastically if the dataset is very large.

Consider if your application is retrieving 1 million records, imagine sending the data to UI and display all the records at once. Application slowness would surely test your patience ðŸ˜‰. Lets see how to implement Pagination in Spring Data JPA.

### Pagination Example

#### BillionaireEntity class

This Entity class has all the attributes of the Billionaire's.

```
@Entity
@Table(name="billionaire")
@Data
public class BillionaireEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Integer id;
    @Column(name="first_name")
    String firstName;
    @Column(name="last_name")
    String lastName;
    @Column(name="company")
    String company;
    @Column(name="wealth")
    BigDecimal wealth;
}
```

### Controller class

Note that we are accepting the **pageSize** and the **pageNumber** as a RequestParam from the consumer of our service. The consumer is responsible to pass how many objects they want in response and from which page number. Lets say there are total 20 records and user wants 5 records at a time so there will be 4 pages and each page will contain 5 records each.

**Note:-** The pageNumber starts with a value of 0.

```
@RestController
@RequestMapping("/BillionaireApp/api")
public class BillionaireController {

    @Autowired
    BillionaireService billionaireService;

    @GetMapping("/billionaires")
    public List<BillionaireEntity> getAllBillionaire(
            @RequestParam("pageSize") Integer pageSize,
            @RequestParam("pageNumber") Integer pageNumber
    ) {
        return billionaireService.getAllBillionaire(pageNumber, pageSize);
    }
}
```

### Service Class

Here we are constructing a PageRequest object which takes in the **pageNumber** and **pageSize** sent by the consumer. This PageRequest object should be passed to the repository interface. i.e. **BillionaireRepository** in our case.

```
@Service
public class BillionaireServiceImplv1 implements BillionaireService {

    @Autowired
    BillionaireRepository repository;

    @Override
    public List<BillionaireEntity> getAllBillionaire(Integer pageNumber, Integ
        Pageable pageRequest = PageRequest.of(pageNumber, pageSize);
        Page<BillionaireEntity> billionaires = repository.findAll(pageRequest)
        return billionaires.stream().toList();
    }
}
```

**Repository**

The repository should extend **PagingAndSortingRepository** for pagination to work. But here we are extending **BillionaireRepository** with **JpaRepository**, so how will that work?

Well if you check the **JpaRepository** interface, it in turn extends **PagingAndSortingRepository**. This is the reason why Pagination will work even though if you extend JpaRepository.

```
@Repository
public interface BillionaireRepository extends JpaRepository<BillionaireEntity
}
```

# Sorting in Spring Data JPA

Sorting in Spring Data JPA can be implemented by passing an instance of **Sort** as a third parameter in **PageRequest** object. If we want to get the sorted results based on a specific column say firstName then we can use the sorting feature provided in Spring data JPA.

## Sorting Example

The Sort parameter in PageRequest object as shown below.

```
    @Override
    public List<BillionaireEntity> getAllBillionaire(Integer pageNumber, Integeg
        Pageable pageRequest = PageRequest.of(pageNumber, pageSize, Sort.by("f
        Page<BillionaireEntity> billionaires = repository.findAll(pageRequest)
        return billionaires.stream().toList();
    }
```

We can also get the results sorted in the Descending order in Spring Data JPA as shown below.

```
Pageable pageRequest = PageRequest.of(pageNumber, pageSize,
                          Sort.by("firstName").descending());
```

Sorting the results using two columns or attributes is also possible using Spring Data JPA as shown below.

```
Pageable pageRequest = PageRequest.of(pageNumber, pageSize,
                     Sort.by("firstName").descending().and(Sort.by("lastName"))
```

After creating your Pageable instance you can pass it to the repository’s method.

## Conclusion

I hope after reading the post you got the know the importance of Pagination and sorting the results and how it will help you to develop faster applications. I hope you enjoyed this article. Feel free to comment or ask questions ðŸ˜Š Happy Learning.

**Category**

1. Hands on
2. Spring Boot

**Date Created**
May 2, 2023
**Author**
kk-ravi144gmail-com