Spring Boot Pagination & Filter example | Spring JPA, Pageable

bezkoder.com/spring-boot-pagination-filter-jpa-pageable

bezkoder Last modified: February 15, 2022

In previous post, we've known how to build Spring Boot Rest CRUD Apis with Spring Data JPA. In this tutorial, I will continue to make Server side Pagination and Filter with Spring Data JPA and Pageable.

Related Post:
– Spring Boot, Spring Data JPA – Rest CRUD API example
– Spring Boot Sort/Order by multiple Columns | Spring Data JPA
– Spring Boot @ControllerAdvice & @ExceptionHandler example
– Spring Boot Pagination and Sorting example

More Practice:
– Spring Boot Token based Authentication with Spring Security & JWT
– With MongoDB: Spring Boot MongoDB Pagination & Filter example with Spring Data

Clients for this Server:
– React with Material-UI / React with react-table v7
– Angular 8 / Angular 10 / Angular 11 / Angular 12
– Vue with Bootstrap / Vuetify

Spring Boot Pagination & Filter example overview

One of the most important things to make a website friendly is the response time, and pagination comes for this reason. For example, this bezkoder.com website has hundreds of tutorials, and we don't want to see all of them at once. Paging means displaying a small number of all, by a page.

Assume that we have **tutorials** table in database like this:

id	description	published	title
1	Tut#1 Description	0	Spring Boot Tut#1
2	Tut#2 Description	1	Spring Data Tut#2
3	Tut#3 Description	1	MySQL Database Tut#3
4	Tut#4 Description	0	Hibernate Tut#4
5	Tut#5 Description	1	Spring Cloud Tut#5
6	Tut#6 Description	0	Microservices Tut#6
7	Tut#7 Description	1	MongoDB Database Tut#7
8	Tut#8 Description	1	Spring Data JPA Tut#8

Here are some url samples for pagination (with/without filter):

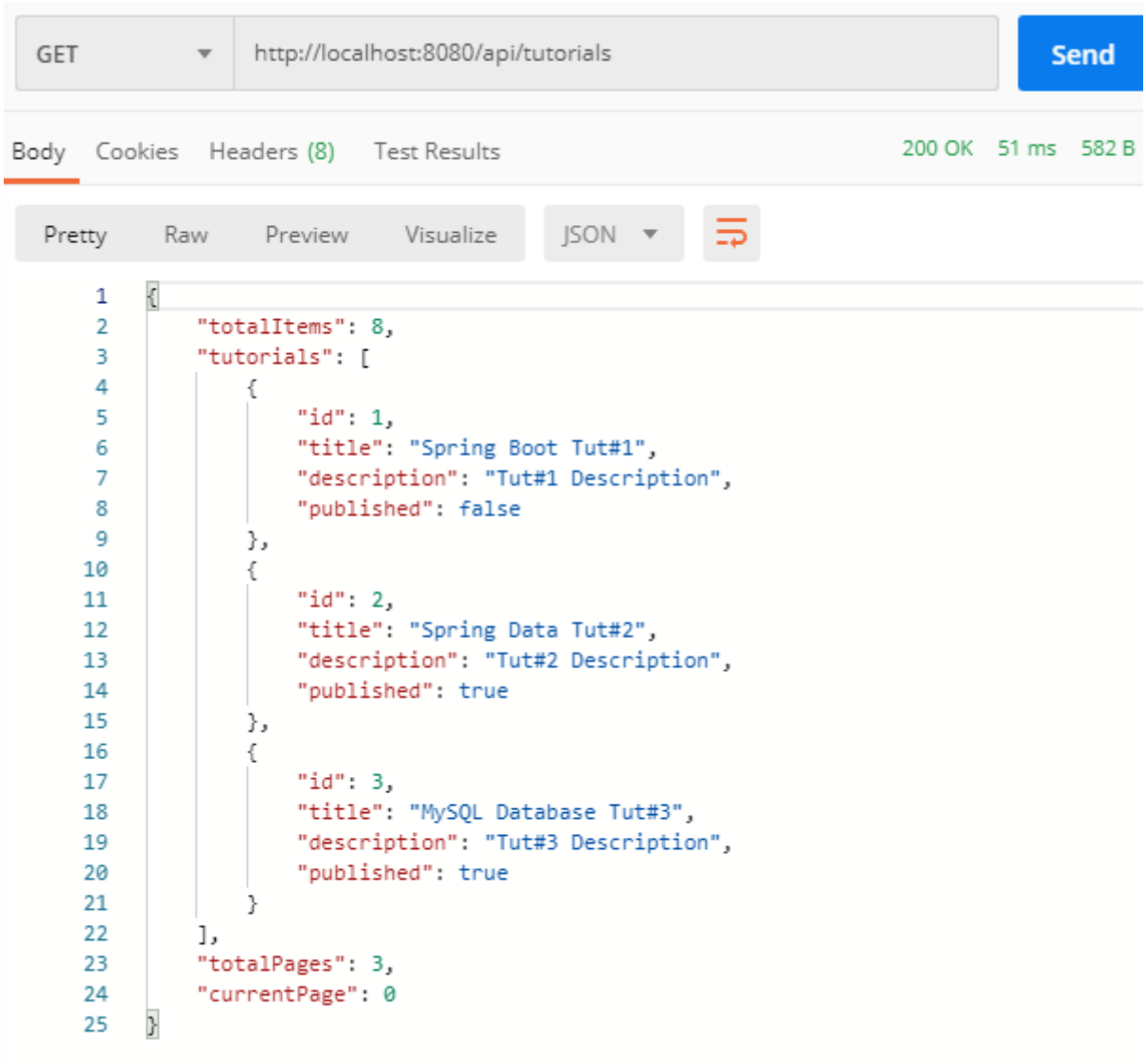
- `/api/tutorials?page=1&size=5`
- `/api/tutorials?size=5` : using default value for page
- `/api/tutorials?title=data&page=1&size=3` : pagination & filter by title containing 'data'

- `/api/tutorials/published?page=2` : pagination & filter by 'published' status

This is structure of the Server side pagination result that we want to get from the APIs:

```
{
    "totalItems": 8,
    "tutorials": [...],
    "totalPages": 3,
    "currentPage": 1
}
```

Read Tutorials with default page index (0) and page size (3):



Indicate page index = 2 but not specify size (default: 3) for total 8 items:

- page_0: 3 items
- page_1: 3 items
- page_2: 2 items

GET http://localhost:8080/api/tutorials?page=2 Send

Body Cookies Headers (8) Test Results 200 OK 23 ms 499 B

Pretty Raw Preview Visualize JSON


```
{
    "totalItems": 8,
    "tutorials": [
        {
            "id": 7,
            "title": "MongoDB Database Tut#7",
            "description": "Tut#7 Description",
            "published": true
        },
        {
            "id": 8,
            "title": "Spring Data JPA Tut#8",
            "description": "Tut#8 Description",
            "published": true
        }
    ],
    "totalPages": 3,
    "currentPage": 2
}
```


Indicate size = 5 but not specify page index (default: 0):

GET http://localhost:8080/api/tutorials?size=5 Send

Body Cookies Headers (8) Test Results 200 OK 30 ms 758 B

Pretty Raw Preview Visualize JSON


```
{
    "totalItems": 8,
    "tutorials": [
        {
            "id": 1,
            "title": "Spring Boot Tut#1",
            "description": "Tut#1 Description",
            "published": false
        },
        {
            "id": 2,
            "title": "Spring Data Tut#2",
            "description": "Tut#2 Description",
            "published": true
        },
        {
            "id": 3,
            "title": "MySQL Database Tut#3",
            "description": "Tut#3 Description",
            "published": true
        },
        {
            "id": 4,
            "title": "Hibernate Tut#4",
            "description": "Tut#4 Description",
            "published": false
        },
        {
            "id": 5,
            "title": "Spring Cloud Tut#5",
            "description": "Tut#5 Description",
            "published": true
        }
    ],
    "totalPages": 2,
    "currentPage": 0
}
```


For page index = 1 and page size = 5 (in total 8 items):

GET http://localhost:8080/api/tutorials?page=1&size=5 Send

Body Cookies Headers (8) Test Results 200 OK 45 ms 590 B

Pretty Raw Preview Visualize JSON


```
{
    "totalItems": 8,
    "tutorials": [
        {
            "id": 6,
            "title": "Microservices Tut#6",
            "description": "Tut#6 Description",
            "published": false
        },
        {
            "id": 7,
            "title": "MongoDB Database Tut#7",
            "description": "Tut#7 Description",
            "published": true
        },
        {
            "id": 8,
            "title": "Spring Data JPA Tut#8",
            "description": "Tut#8 Description",
            "published": true
        }
    ],
    "totalPages": 2,
    "currentPage": 1
}
```


Pagination and filter by title that contains a string:

GET http://localhost:8080/api/tutorials?title=data&size=3 Send

Body Cookies Headers (8) Test Results 200 OK 150 ms 586 B

Pretty Raw Preview Visualize JSON


```
{
    "totalItems": 4,
    "tutorials": [
        {
            "id": 2,
            "title": "Spring Data Tut#2",
            "description": "Tut#2 Description",
            "published": true
        },
        {
            "id": 3,
            "title": "MySQL Database Tut#3",
            "description": "Tut#3 Description",
            "published": true
        },
        {
            "id": 7,
            "title": "MongoDB Database Tut#7",
            "description": "Tut#7 Description",
            "published": true
        }
    ],
    "totalPages": 2,
    "currentPage": 0
}
```


Pagination and filter by published status:

GET http://localhost:8080/api/tutorials/published?page=1&size=2 Send

Body Cookies Headers (8) Test Results 200 OK 26 ms 496 B

Pretty Raw Preview Visualize JSON


```
{
    "totalItems": 5,
    "tutorials": [
        {
            "id": 5,
            "title": "Spring Cloud Tut#5",
            "description": "Tut#5 Description",
            "published": true
        },
        {
            "id": 7,
            "title": "MongoDB Database Tut#7",
            "description": "Tut#7 Description",
            "published": true
        }
    ],
    "totalPages": 3,
    "currentPage": 1
}
```


Pagination and Filter with Spring Data JPA

To help us deal with this situation, Spring Data JPA provides way to implement pagination with PagingAndSortingRepository.

`PagingAndSortingRepository` extends CrudRepository to provide additional methods to retrieve entities using the pagination abstraction.

```
public interface PagingAndSortingRepository<T, ID> extends CrudRepository<T, ID> {
  Page<T> findAll(Pageable pageable);
}
```

`findAll(Pageable pageable)` : returns a `Page` of entities meeting the paging condition provided by `Pageable` object.

Spring Data also supports many useful Query Creation from method names that we're gonna use to filter result in this example such as:

```
Page<Tutorial> findByPublished(boolean published, Pageable pageable);
Page<Tutorial> findByTitleContaining(String title, Pageable pageable);
```

You can find more supported keywords inside method names here.

To sort multiple fields with paging, please visit the tutorial:
Spring Data JPA Sort/Order by multiple Columns | Spring Boot

Spring Data Page

Let's look at the Page object.

`Page` is a sub-interface of `Slice` with a couple of additional methods. It contains total amount of elements and total pages of the entire list.

```
public interface Page<T> extends Slice<T> {
  static <T> Page<T> empty();
  static <T> Page<T> empty(Pageable pageable);
  long getTotalElements();
  int getTotalPages();
  <U> Page<U> map(Function<? super T,? extends U> converter);
}
```

If the number of items increases, the performance could be affected, it's the time you should think about Slice.

A `Slice` object knows less information than a `Page`, for example, whether the next one or previous one is available or not, or this slice is the first/last one. You can use it when you don't need the total number of items and total pages.

```
public interface Slice<T> extends Streamable<T> {
  int getNumber();
  int getSize();
  int getNumberOfElements();
  List<T> getContent();
  boolean hasContent();
  Sort getSort();
  boolean isFirst();
  boolean isLast();
  boolean hasNext();
  boolean hasPrevious();
  ...
}
```

Spring Data Pageable

Now we're gonna see the Pageable parameter in Repository methods above. Spring Data infrastructure will recognizes this parameter automatically to apply pagination and sorting to database.

The `Pageable` interface contains the information about the requested page such as the size and the number of the page.

```
public interface Pageable {
  int getPageNumber();
  int getPageSize();
  long getOffset();
  Sort getSort();
  Pageable next();
  Pageable previousOrFirst();
  Pageable first();
  boolean hasPrevious();
  ...
}
```

So when we want to get pagination (with or without filter) in the results, we just add `Pageable` to the definition of the method as a parameter.

```
Page<Tutorial> findAll(Pageable pageable);
Page<Tutorial> findByPublished(boolean published, Pageable pageable);
Page<Tutorial> findByTitleContaining(String title, Pageable pageable);
```

This is how we create `Pageable` objects using PageRequest class which implements `Pageable` interface:

```
Pageable paging = PageRequest.of(page, size);
```

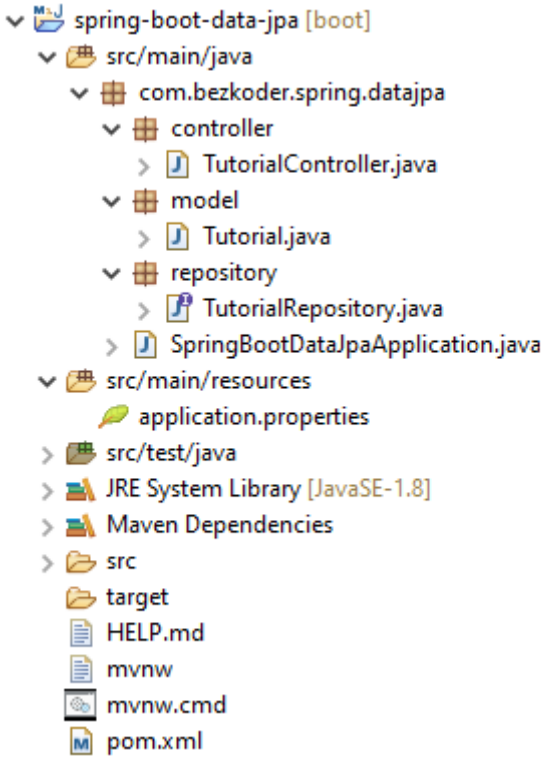
- `page` : zero-based page index, must NOT be negative.
- `size` : number of items in a page to be returned, must be greater than 0.

Create Spring Boot Application

You can follow step by step, or get source code in this post:
Spring Boot, Spring Data JPA – Rest CRUD API example

The Spring Project contains structure that we only need to add some changes to make the pagination work well.

Or you can get the new Github source code (including paging and sorting) at the end of this tutorial.



Data Model

We have Tutorial entity like this:

```
package com.bezkoder.spring.data.jpa.pagingsorting.model;
import javax.persistence.*;
@Entity
@Table(name = "tutorials")
public class Tutorial {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private long id;
  @Column(name = "title")
  private String title;
  @Column(name = "description")
  private String description;
  @Column(name = "published")
  private boolean published;
  public Tutorial() {
  }
  public Tutorial(String title, String description, boolean published) {
    this.title = title;
    this.description = description;
    this.published = published;
  }
  public long getId() {
    return id;
  }
  public String getTitle() {
    return title;
  }
  public void setTitle(String title) {
    this.title = title;
  }
  public String getDescription() {
    return description;
  }
  public void setDescription(String description) {
    this.description = description;
  }
  public boolean isPublished() {
    return published;
  }
  public void setPublished(boolean isPublished) {
    this.published = isPublished;
  }
  @Override
  public String toString() {
    return "Tutorial [id=" + id + ", title=" + title + ", desc=" + description +
", published=" + published + "]";
  }
}
```

Repository that supports Pagination and Filter

Early in this tutorial, we know `PagingAndSortingRepository`, but in this example, for keeping the continuity and taking advantage Spring Data JPA, we continue to use JpaRepository which extends `PagingAndSortingRepository` interface.

```
package com.bezkoder.spring.data.jpa.pagingsorting.repository;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import com.bezkoder.spring.data.jpa.pagingsorting.model.Tutorial;
public interface TutorialRepository extends JpaRepository<Tutorial, Long> {
  Page<Tutorial> findByPublished(boolean published, Pageable pageable);
  Page<Tutorial> findByTitleContaining(String title, Pageable pageable);
}
```

In the code above, we use add `pageable` parameter with Spring Query Creation to find all Tutorials which title containing input string.

Controller with Pagination and Filter

Generally, in the HTTP request URLs, paging parameters are optional. So if our Rest API supports server side pagination, we should provide default values to make paging work even when Client does not specify these parameters.

```
package com.bezkoder.spring.data.jpa.pagingsorting.controller;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
...
@RestController
@RequestMapping("/api")
public class TutorialController {
  @Autowired
  TutorialRepository tutorialRepository;
  @GetMapping("/tutorials")
  public ResponseEntity<Map<String, Object>> getAllTutorials(
        @RequestParam(required = false) String title,
        @RequestParam(defaultValue = "0") int page,
        @RequestParam(defaultValue = "3") int size
      ) {
    try {
      List<Tutorial> tutorials = new ArrayList<Tutorial>();
      Pageable paging = PageRequest.of(page, size);

      Page<Tutorial> pageTuts;
      if (title == null)
        pageTuts = tutorialRepository.findAll(paging);
      else
        pageTuts = tutorialRepository.findByTitleContaining(title, paging);
      tutorials = pageTuts.getContent();
      Map<String, Object> response = new HashMap<>();
      response.put("tutorials", tutorials);
      response.put("currentPage", pageTuts.getNumber());
      response.put("totalItems", pageTuts.getTotalElements());
      response.put("totalPages", pageTuts.getTotalPages());
      return new ResponseEntity<>(response, HttpStatus.OK);
    } catch (Exception e) {
      return new ResponseEntity<>(null, HttpStatus.INTERNAL_SERVER_ERROR);
    }
  }

  @GetMapping("/tutorials/published")
  public ResponseEntity<Map<String, Object>> findByPublished(
        @RequestParam(defaultValue = "0") int page,
        @RequestParam(defaultValue = "3") int size
      ) {
    try {
      List<Tutorial> tutorials = new ArrayList<Tutorial>();
      Pageable paging = PageRequest.of(page, size);

      Page<Tutorial> pageTuts = tutorialRepository.findByPublished(true, paging);
      tutorials = pageTuts.getContent();

      Map<String, Object> response = new HashMap<>();
      response.put("tutorials", tutorials);
      response.put("currentPage", pageTuts.getNumber());
      response.put("totalItems", pageTuts.getTotalElements());
      response.put("totalPages", pageTuts.getTotalPages());

      return new ResponseEntity<>(response, HttpStatus.OK);
    } catch (Exception e) {
      return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
    }
```

```
  }
  ...
}
```

In the code above, we accept paging parameters using `@RequestParam` annotation for `page`, `size`. By default, `3` Tutorials will be fetched from database in page index `0`.

Next, we create a `Pageable` object with `page` & `size`.
Then check if the `title` parameter exists or not.

- If it is null, we call Repository `findAll(paging)` with paging is the `Pageable` object above.
- If Client sends request with `title`, use `findByTitleContaining(title, paging)`.

Both methods return a `Page` object. We call:

- `getContent()` to retrieve the List of items in the page.
- `getNumber()` for current Page.
- `getTotalElements()` for total items stored in database.
- `getTotalPages()` for number of total pages.

Conclusion

In this post, we have learned how to create pagination and filter for result in Spring Boot application using Spring Data JPA, Page and Pageable interface.

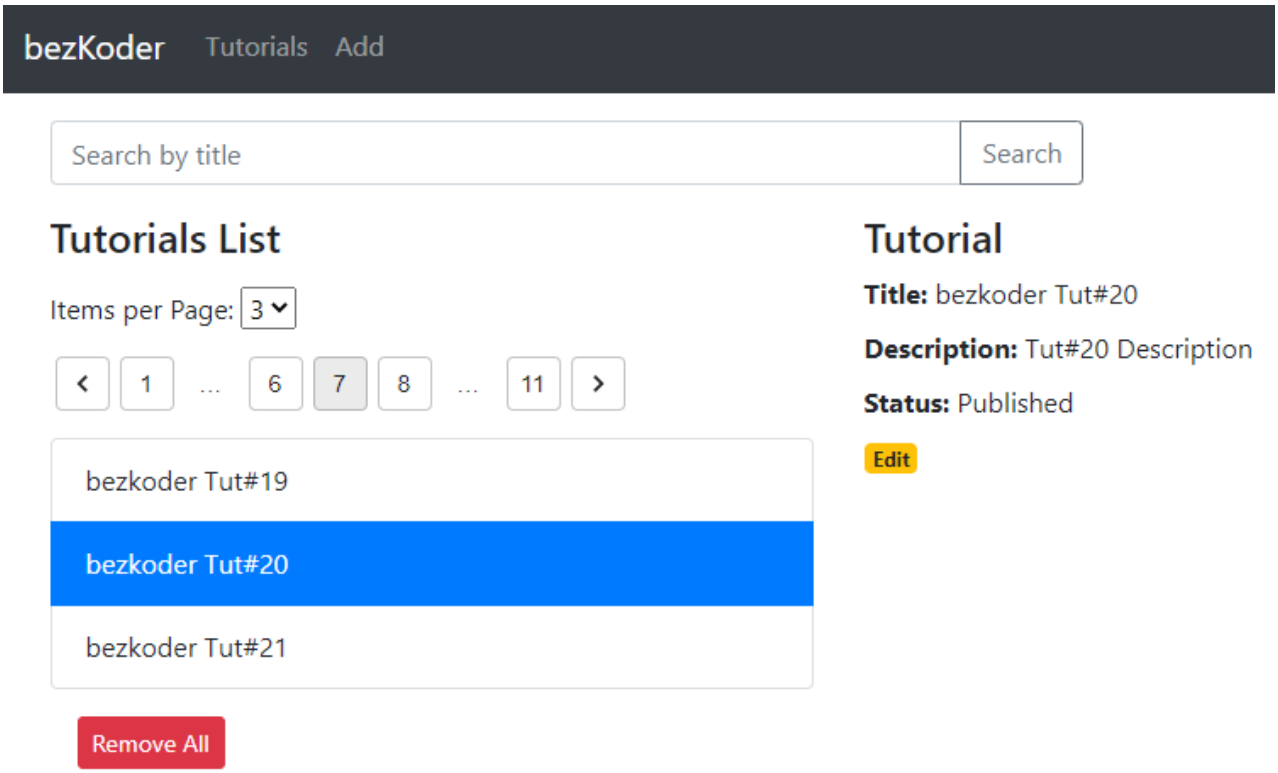
We also see that `JpaRepository` supports a great way to make server side pagination and filter methods without need of boilerplate code.

To bring pagination and sorting together, please visit:
Spring Boot Pagination and Sorting example

Handle Exception for this Rest APIs is necessary:
Spring Boot @ControllerAdvice & @ExceptionHandler example

You can also know how to deploy this Spring Boot App on AWS (for free) with this tutorial.

React Pagination Client that works with this Server:
– React Table Pagination using react-table v7
– React Pagination with API using Material-UI



Angular Client working with this server:
– Angular 8 Pagination example | ngx-pagination
– Angular 10 Pagination example | ngx-pagination
– Angular 11 Pagination example | ngx-pagination
– Angular 12 Pagination example | ngx-pagination

Or Vue Client:
– Vue Pagination example (Bootstrap)
– Vuetify Pagination example

Happy learning! See you again.

Further Reading

Deployment:
– Deploy Spring Boot App on AWS – Elastic Beanstalk
– Docker Compose: Spring Boot and MySQL example

Associations:
– JPA One To Many example with Hibernate and Spring Boot
– JPA Many to Many example with Hibernate in Spring Boot

Source Code

You can find the complete source code for this tutorial on Github.