Spring RESTful API
Pagination & Sorting

By

Pichet Limvajiranan

Spring Data REST: Pagination and Sorting

- The PagingAndSortingRepository is an extension of CrudRepository to provide additional methods to retrieve entities using the pagination and sorting abstraction. It implicitly provides two methods:
  - **Page<T> findAll(Pageable pageable)**
    returns a Page of entities meeting the paging restriction provided in the Pageable object.

    ```
    Pageable firstPageTwoElements = PageRequest.of(0, 2); Pageable
    secondPageFiveElements = PageRequest.of(1, 5);
    ```

  - **Iterable<T> findAll(Sort sort)**
    returns all entities sorted by the given options. No paging is applied here.

    ```
    Sort sortedByName = Sort.by("name");
    ```

  - Pagination & Sorting

    ```
    Pageable sortedByPriceDescNameAsc = PageRequest.of(0, 5,
    Sort.by("price").descending().and(Sort.by("name")));
    ```

Spring Data Sort and Order

- The Sort class provides sorting options for database queries with more flexibility in choosing single/multiple sort columns and directions (ascending/descending).
  - we use by(), descending(), and() methods to create Sort object and pass it to Repository.findAll()
- You can sort results by Sort and Order object with one or more specified variables.
- Sorting can be done in ascending or descending order.

```
@Service
 :
 :
public List<Customer> getAllCustomers(String sortBy) {
   return repository.findAll(Sort.Direction.DESC, Sort.by(sortBy));
}
```

Sort & Order object example

```
// order by 'published' column - ascending
List<Tutorial> tutorials = tutorialRepository.findAll(Sort.by("published"));

// order by 'published' column, descending
tutorialRepository.findAll(Sort.by("published").descending());

// order by 'published' column - descending, then order by 'title' - ascending
tutorialRepository.findAll(Sort.by("published").descending().and(Sort.by("title")));
```

```
List<Sort.Order> orders = new ArrayList();
Sort.Order order1 = new Sort.Order(Sort.Direction.DESC, "published");
orders.add(order1);
Sort.Order order2 = new Sort.Order(Sort.Direction.ASC, "title");
orders.add(order2);

List<Tutorial> tutorials = tutorialRepository.findAll(Sort.by(orders));
```

Exercise 1:

- Create REST API service for Products as end-points below

URI	HTTP verb	Description
/products	GET	Get all products filter by price between and product name contains sorting as request specify

```java
public interface ProductRepository extends JpaRepository<Product, String> {
   List<Product> getProductsByPriceBetweenAndProductNameContains(
         Double lower, Double upper, String partOfName, Sort sort);
   Product findFirstByOrderByPriceDesc();
}
```

```java
@Service
public class ProductService {
   @Autowired
   ProductRepository repository;
   public List<Product> getAllProducts(Double lower, Double upper,
                              String partOfName, String sortBy, String direction) {
      if (upper==0 && lower==0) { upper = repository.findFirstByOrderByPriceDesc().getPrice(); }
      if(sortBy.isEmpty()) { sortBy = "productCode" ; }
      Sort.Order sortOrder = new Sort.Order((direction.equalsIgnoreCase("asc")?
            Sort.Direction.ASC : Sort.Direction.DESC), sortBy);
      return repository.getProductsByPriceBetweenAndProductNameContains(
            lower, upper, partOfName, Sort.by(sortOrder));
   }
}
```

```java
@RestController
@RequestMapping("/products")
public class ProductController {
    @Autowired
    ProductService service;
    @GetMapping("")
    public List<Product> getAllProducts(
            @RequestParam(defaultValue = "") String partOfProductName,
            @RequestParam(defaultValue = "0") Double lower,
            @RequestParam(defaultValue = "0") Double upper,
            @RequestParam(defaultValue = "") String sortBy,
            @RequestParam(defaultValue = "ASC") String sortDirection
    ) {
        return service.getAllProducts(lower, upper, partOfProductName, sortBy, sortDirection);
    }
}
```

JpaRepository with Pagination

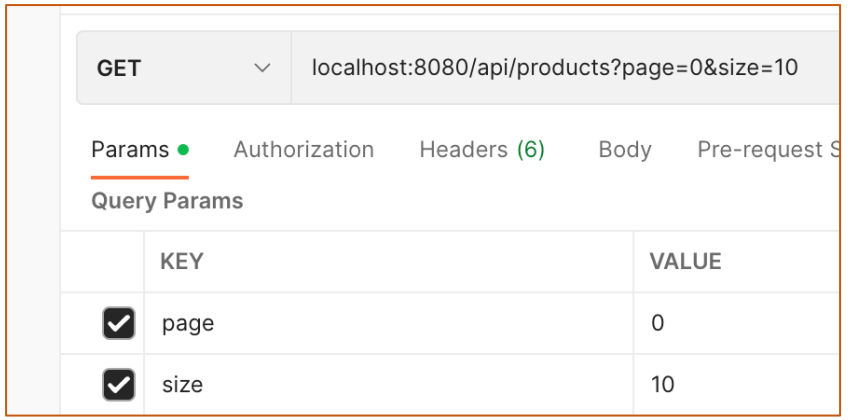
- findAll(Pageable pageable): returns a Page of entities meeting the paging condition provided by Pageable object.
- Pagination can be added by creation of PageRequest object which is implementation of Pageable interface.
- Similar to sorting adding pagination depends from type of Repository extended by our interface.

```
@Service
  :
public Page<Customer> getAllCustomers(int page, int pageSize) {
        Pageable pageable = PageRequest.of(page, pageSize);
        return repository.findAll(pageable);
}
```

Accepting Page and Sort Parameters

- Generally, paging and sorting parameters are optional and thus part of the request URL as query parameters. If any API supports paging and sorting, ALWAYS provide default values to these parameters – to be used when the client does not choose to specify any paging or sorting preferences.
- Example:

```
@GetMapping("")
  public List<Customer> getAllCustomers(
        @RequestParam(defaultValue = "id") String sortBy,
        @RequestParam(defaultValue = "0") Integer page,
        @RequestParam(defaultValue = "10") Integer pageSize) {
         Page<Customer> customers = service.findAll(sortBy, page, pageSize);
        return customers.getContent();
  }
```



Page<T> Object


```
{
  "content": [
    {
      "id": 323,
      "customerName": "Down Under Souveniers, Inc",
      "contactLastName": "Graham",
      "contactFirstName": "Mike",
      "phone": "+64 9 312 5555",
      "addressLine1": "162-164 Grafton Road",
      "addressLine2": "Level 2",
      :
  ]
  "pageable": {
    "sort": {
      "empty": false,
      "sorted": true,
      "unsorted": false
    },
    "offset": 5,
    "pageSize": 5,
    "pageNumber": 1,
    "unpaged": false,
    "paged": true
  },
  "last": false,
  "totalPages": 25,
  "totalElements": 122,
  "size": 5,
  "number": 1,
  "sort": {
    "empty": false,
    "sorted": true,
    "unsorted": false
  },
  "numberOfElements": 5,
  "first": false,
  "empty": false
}
```

Service - Paging & Sorting

```
@Service
public class CustomerService {
   @Autowired
   private CustomerRepository repository;

    public Page<Customer> getAllCustomers(
            String sortBy, int page, int pageSize) {
      Pageable pageble = PageRequest.of(page, pageSize);
      Page<Customer> customers = repository.findAll(pageble);
      return customers;
   }
}
```

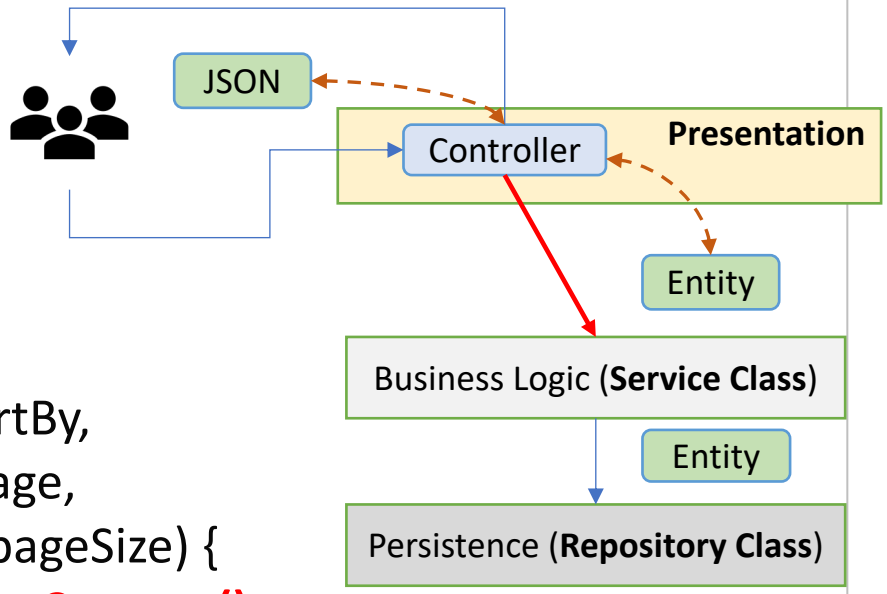

```
{
 "content " : {
      { },
      { },
      { }
  },
  "page": {
    "size": 20,
    "totalElements": 11,
    "totalPages": 1,
    "number": 0
  }
}
```

Controller - Paging & Sorting

```
@RestController
@RequestMapping("/api/customers")
public class CustomerController {
   @Autowired
   private CustomerService service;

   @GetMapping("")
   public List<Customer> getAllCustomers(
         @RequestParam(defaultValue = "id") String sortBy,
         @RequestParam(defaultValue = "0") Integer page,
         @RequestParam(defaultValue = "10") Integer pageSize) {
         return service.findAll(sortBy, page, pageSize).getContent();
   }
}
```

localhost:port/api/customers?sortBy=id&page=0&pageSize=10



Exercise 2:

- Create REST API service for Products as end-points below

URI	HTTP verb	Description
/products	GET	Get all products filter by price between and product name contains sorting as request specify with pagination

Add New Query Method to support pagination

```
public interface ProductRepository extends JpaRepository<Product, String> {
   List<Product> getProductsByPriceBetweenAndProductNameContains(
         Double lower, Double upper, String partOfName, Sort sort);
   Product findFirstByOrderByPriceDesc();

   Page<Product> getProductsByPriceBetweenAndProductNameContains(
         Double lower, Double upper, String partOfName, Pageable pageable);
}
```

Overload service method to support pagination

```
public Page<Product> getAllProducts(Double lower, Double upper,
                                    String partOfName, String sortBy, String direction,
                                    int pageNo, int pageSize) {
    if (upper == 0 && lower == 0) {
        upper = repository.findFirstByOrderByPriceDesc().getPrice();
    }
    if (sortBy.isEmpty()) {
        sortBy = "productCode";
    }
    Sort.Order sortOrder = new Sort.Order(
            (direction.equalsIgnoreCase("asc") ? Sort.Direction.ASC : Sort.Direction.DESC), sortBy);
    Pageable pageable = PageRequest.of(pageNo, pageSize, Sort.by(sortOrder));
    return repository.getProductsByPriceBetweenAndProductNameContains(lower, upper, partOfName, pageable);
}
```

Controller: Create new end-point/Modify method to support pagination

```
@RestController
@RequestMapping("/products")
public class ProductController {
   @Autowired
   ProductService service;
   @GetMapping("")
   public Page<Product> getAllProducts(
         @RequestParam(defaultValue = "") String partOfProductName,
         @RequestParam(defaultValue = "0") Double lower,
         @RequestParam(defaultValue = "0") Double upper,
         @RequestParam(defaultValue = "") String sortBy,
         @RequestParam(defaultValue = "ASC") String sortDirection,
         @RequestParam(defaultValue = ”0") int pageNo, @RequestParam(defaultValue = "10") int pageSize
   ) {
      return service.getAllProducts(lower, upper, partOfProductName, sortBy, sortDirection, pageNo, pageSize);
   }
}
```