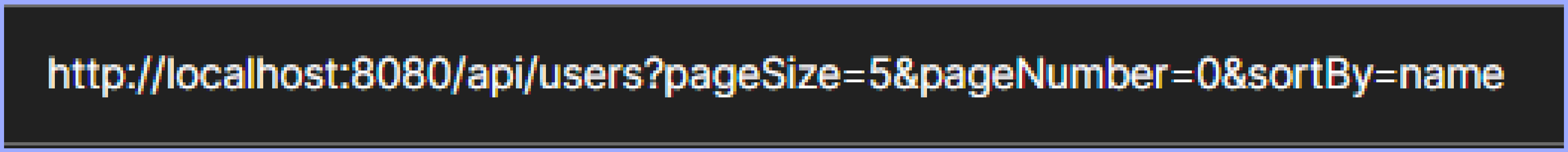
Pagination & Sorting

IN SPRING BOOT

ROHAN THAPA
thaparohan2019@gmail.com

What is Pagination?

Pagination is the process of **dividing** a **large dataset** into smaller, manageable chunks or pages.
This helps in efficiently retrieving and displaying a subset of data at a time, improving both **performance** and **user experience**.



```
http://localhost:8080/api/users?pageSize=5&pageNumber=0&sortBy=name
```

Why Use Pagination?

- **Performance:** Reduces the amount of data processed and **transferred at once**, which improves performance and reduces memory usage.

- **User Experience:** Presents data in manageable portions, making it easier for users to navigate and consume information.

- **Scalability:** Allows applications to handle large datasets more effectively by loading only a portion of data at a time.

Key Concepts

- **Page:** A subset of data that is displayed at one time.
- **Page Number:** The index of the page to retrieve **(usually zero-based)**.
- **Page Size:** The number of items to display per page.
- **Total Items:** The total number of items in the dataset.
- **Total Pages:** The total number of pages required to display all items.

Basic Pagination Structure

- **Request Parameters:**
    - **page:** The page number to retrieve (zero-based index).
    - **size:** The number of items per page.

- **Response Structure:**
    - **items:** The list of items on the current page.
    - **currentPage:** The current page number.
    - **totalItems:** The total number of items in the dataset.
    - **totalPages:** The total number of pages.
    - **pageSize:** The number of items per page.

Internal Mechanism

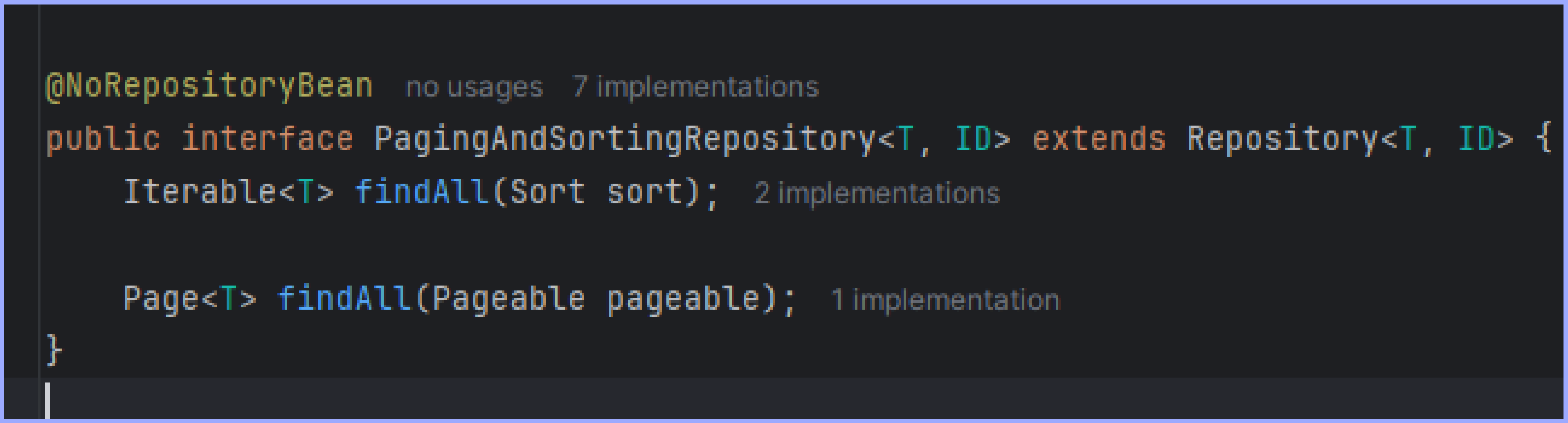
Spring Data JPA provides powerful abstractions for pagination and sorting through its **Pageable** and **PageRequest interfaces**.

Pageable is an interface that provides **pagination** information. It includes details like **page number, page size, and sorting criteria**.

PageRequest is a concrete implementation of the **Pageable interface** that provides a way to **specify pagination** and **sorting parameters**.

PaginationAndSortingRepository

The **PaginationAndSortingRepository** interface in **Spring Data JPA** is an extension of the **CrudRepository** interface that provides additional methods for **pagination and sorting**. This interface simplifies the implementation of paginated and sorted queries without needing to write custom repository methods.

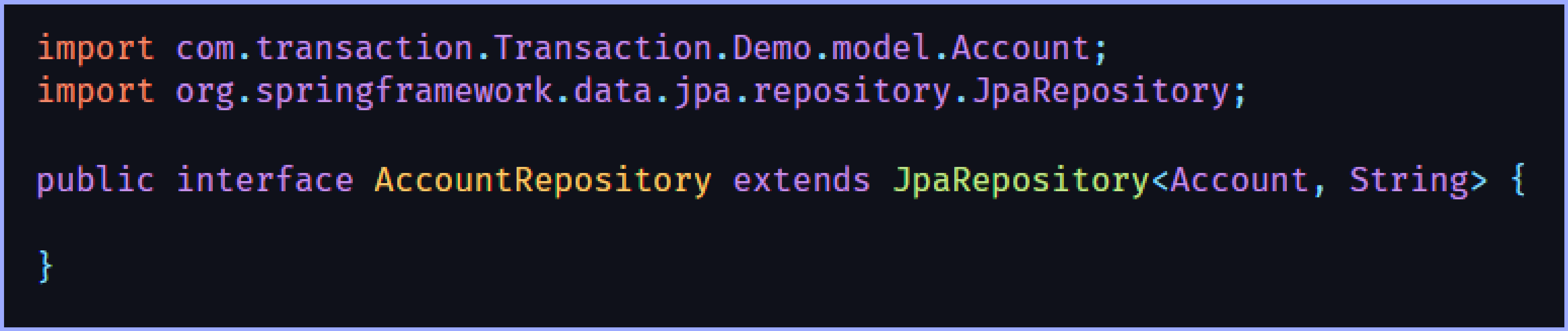


```
@NoRepositoryBean  no usages  7 implementations
public interface PagingAndSortingRepository<T, ID> extends Repository<T, ID> {
    Iterable<T> findAll(Sort sort);  2 implementations

    Page<T> findAll(Pageable pageable);  1 implementation
}
```

Implementation in Spring Boot

Dependencies: Ensure you have the necessary dependencies for **Spring Data JPA** in your **pom.xml** or **build.gradle**.

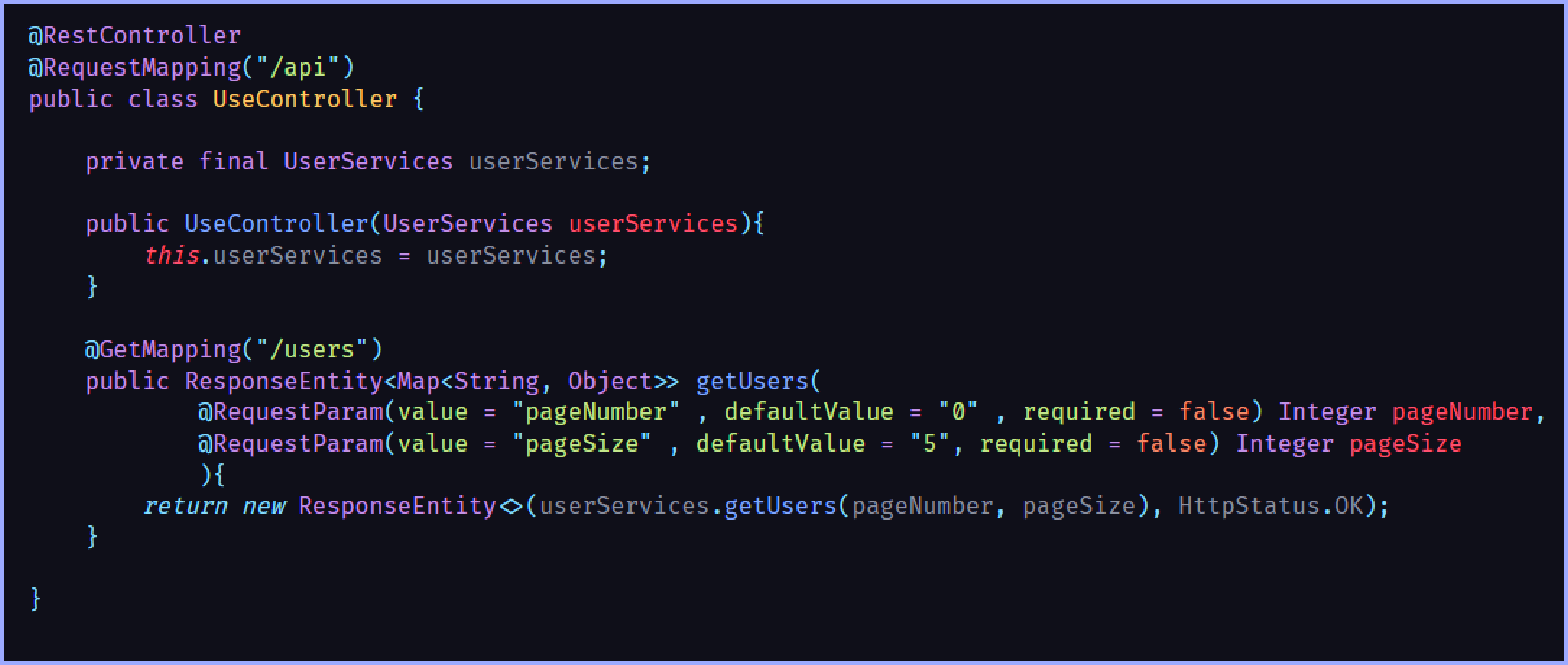
Repository Interface: Define a repository interface **extending JpaRepository** to handle data access.



```java
import com.transaction.Transaction.Demo.model.Account;
import org.springframework.data.jpa.repository.JpaRepository;

public interface AccountRepository extends JpaRepository<Account, String> {

}
```

Implementation in Spring Boot

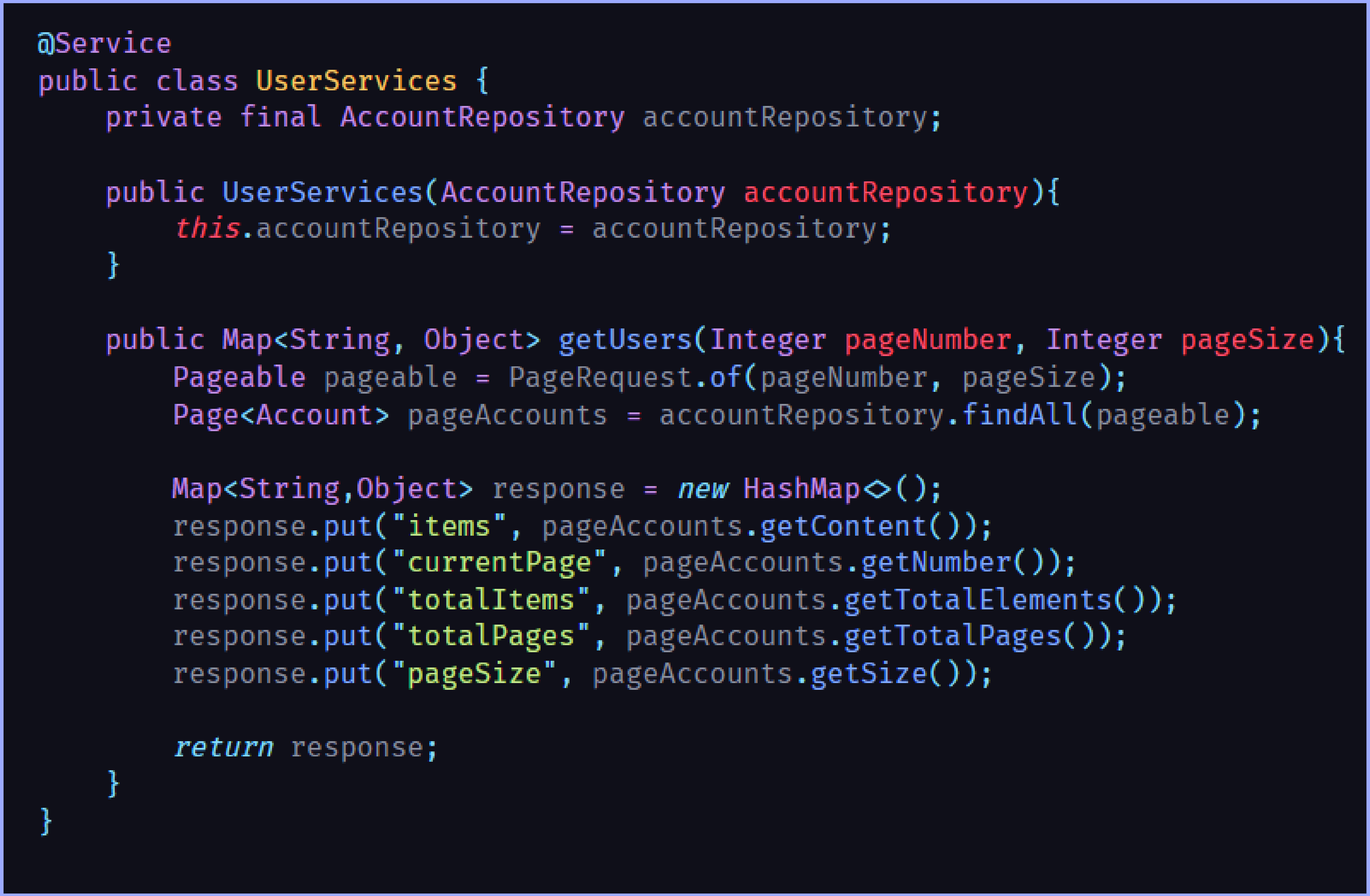
Controller: Create a controller to handle requests and provide paginated data.



```java
@RestController
@RequestMapping("/api")
public class UseController {

    private final UserServices userServices;

    public UseController(UserServices userServices){
        this.userServices = userServices;
    }

    @GetMapping("/users")
    public ResponseEntity<Map<String, Object>> getUsers(
            @RequestParam(value = "pageNumber" , defaultValue = "0" , required = false) Integer pageNumber,
            @RequestParam(value = "pageSize" , defaultValue = "5", required = false) Integer pageSize
            ){
        return new ResponseEntity<>(userServices.getUsers(pageNumber, pageSize), HttpStatus.OK);
    }

}
```

Implementation in Spring Boot

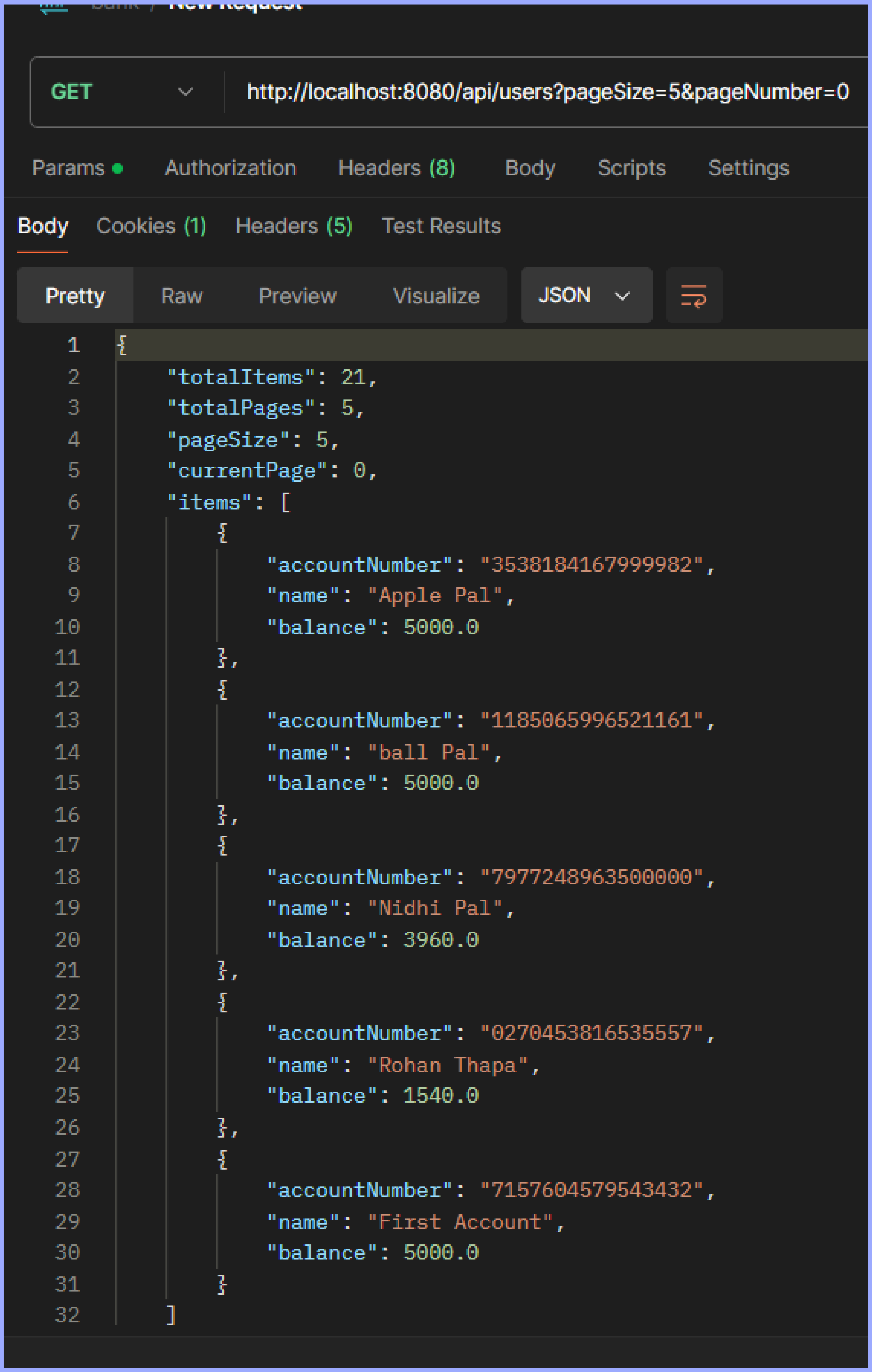
Service: Create a service to fetch the page result and map with other pagination data.



```java
@Service
public class UserServices {
    private final AccountRepository accountRepository;

    public UserServices(AccountRepository accountRepository){
        this.accountRepository = accountRepository;
    }

    public Map<String, Object> getUsers(Integer pageNumber, Integer pageSize){
        Pageable pageable = PageRequest.of(pageNumber, pageSize);
        Page<Account> pageAccounts = accountRepository.findAll(pageable);

        Map<String,Object> response = new HashMap<>();
        response.put("items", pageAccounts.getContent());
        response.put("currentPage", pageAccounts.getNumber());
        response.put("totalItems", pageAccounts.getTotalElements());
        response.put("totalPages", pageAccounts.getTotalPages());
        response.put("pageSize", pageAccounts.getSize());

        return response;
    }
}
```

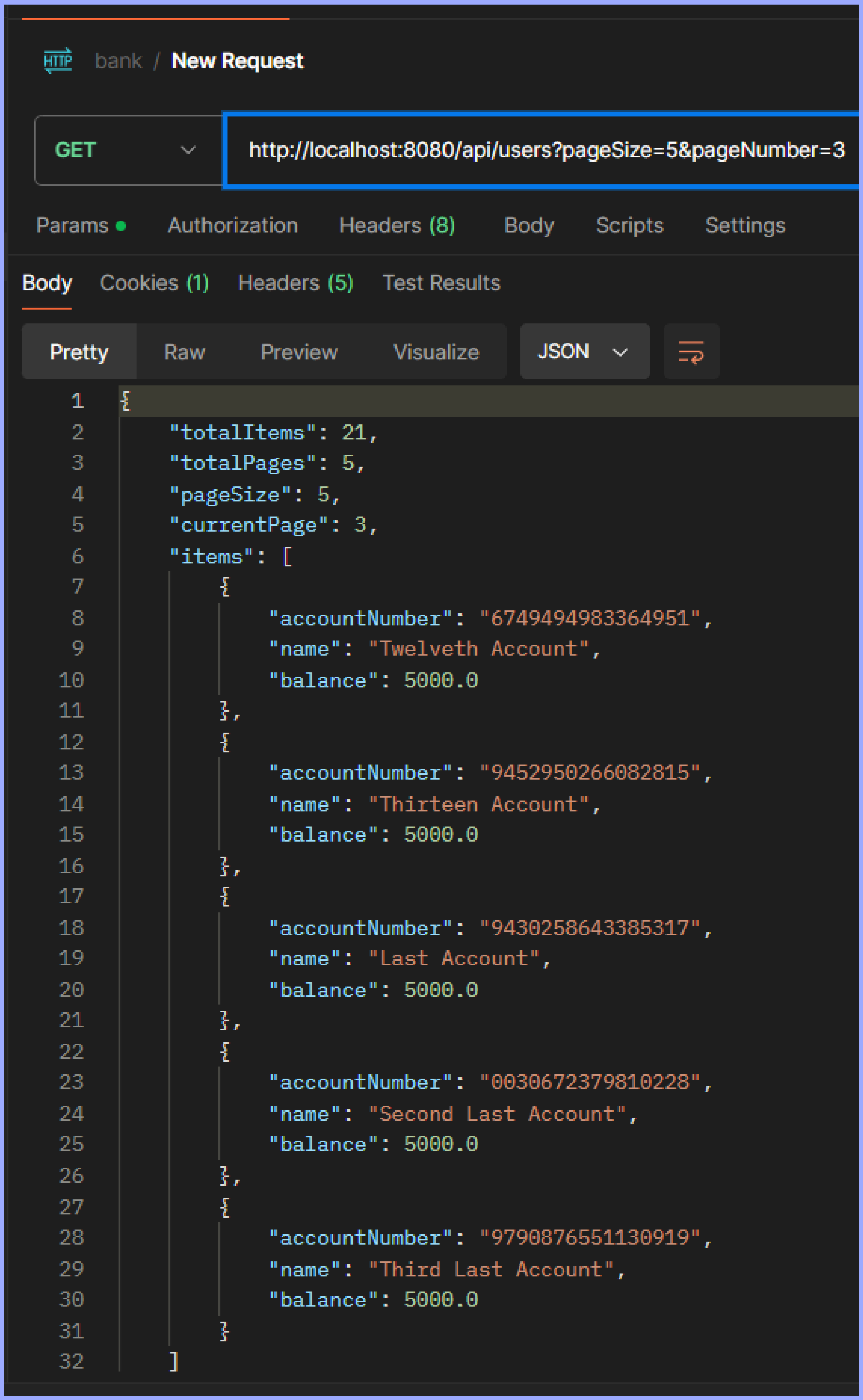
Implementation in Spring Boot

Result:



```
http://localhost:8080/api/users?pageSize=5&pageNumber=0
```

```
{
    "totalItems": 21,
    "totalPages": 5,
    "pageSize": 5,
    "currentPage": 0,
    "items": [
        {
            "accountNumber": "3538184167999982",
            "name": "Apple Pal",
            "balance": 5000.0
        },
        {
            "accountNumber": "1185065996521161",
            "name": "ball Pal",
            "balance": 5000.0
        },
        {
            "accountNumber": "7977248963500000",
            "name": "Nidhi Pal",
            "balance": 3960.0
        },
        {
            "accountNumber": "0270453816535557",
            "name": "Rohan Thapa",
            "balance": 1540.0
        },
        {
            "accountNumber": "7157604579543432",
            "name": "First Account",
            "balance": 5000.0
        }
    ]
```



```
http://localhost:8080/api/users?pageSize=5&pageNumber=3
```

```
{
    "totalItems": 21,
    "totalPages": 5,
    "pageSize": 5,
    "currentPage": 3,
    "items": [
        {
            "accountNumber": "6749494983364951",
            "name": "Twelveth Account",
            "balance": 5000.0
        },
        {
            "accountNumber": "9452950266082815",
            "name": "Thirteen Account",
            "balance": 5000.0
        },
        {
            "accountNumber": "9430258643385317",
            "name": "Last Account",
            "balance": 5000.0
        },
        {
            "accountNumber": "0030672379810228",
            "name": "Second Last Account",
            "balance": 5000.0
        },
        {
            "accountNumber": "9790876551130919",
            "name": "Third Last Account",
            "balance": 5000.0
        }
    ]
```

Handling Pagination Details

Total Pages:

- Use **getTotalPages()** method from the Page object to get the total number of pages.

Current Page:

- Use **getNumber()** method from the Page object to get the current page number.

Total Items:

- Use **getTotalElements()** method from the Page object to get the total number of items.

Page Size:

- Use **getSize()** method from the Page object to get the number of items per page.

Items:

- Use **getContent()** method from the Page object to get the list of items on the current page.

Sorting

Sorting is the process of **arranging data** in a **specific order**. In the context of databases and web applications, sorting helps present data in a meaningful and **organized way**. Sorting can be performed in **two primary orders**:

1. **Ascending Order:**
    - **Definition:** Data is arranged from the smallest to the largest value.
    - **Example:** Sorting names alphabetically (A to Z) or dates from oldest to newest.
2. **Descending Order:**
    - **Definition:** Data is arranged from the largest to the smallest value.
    - **Example:** Sorting names in reverse alphabetical order (Z to A) or dates from newest to oldest.

Sorting by Multiple Fields

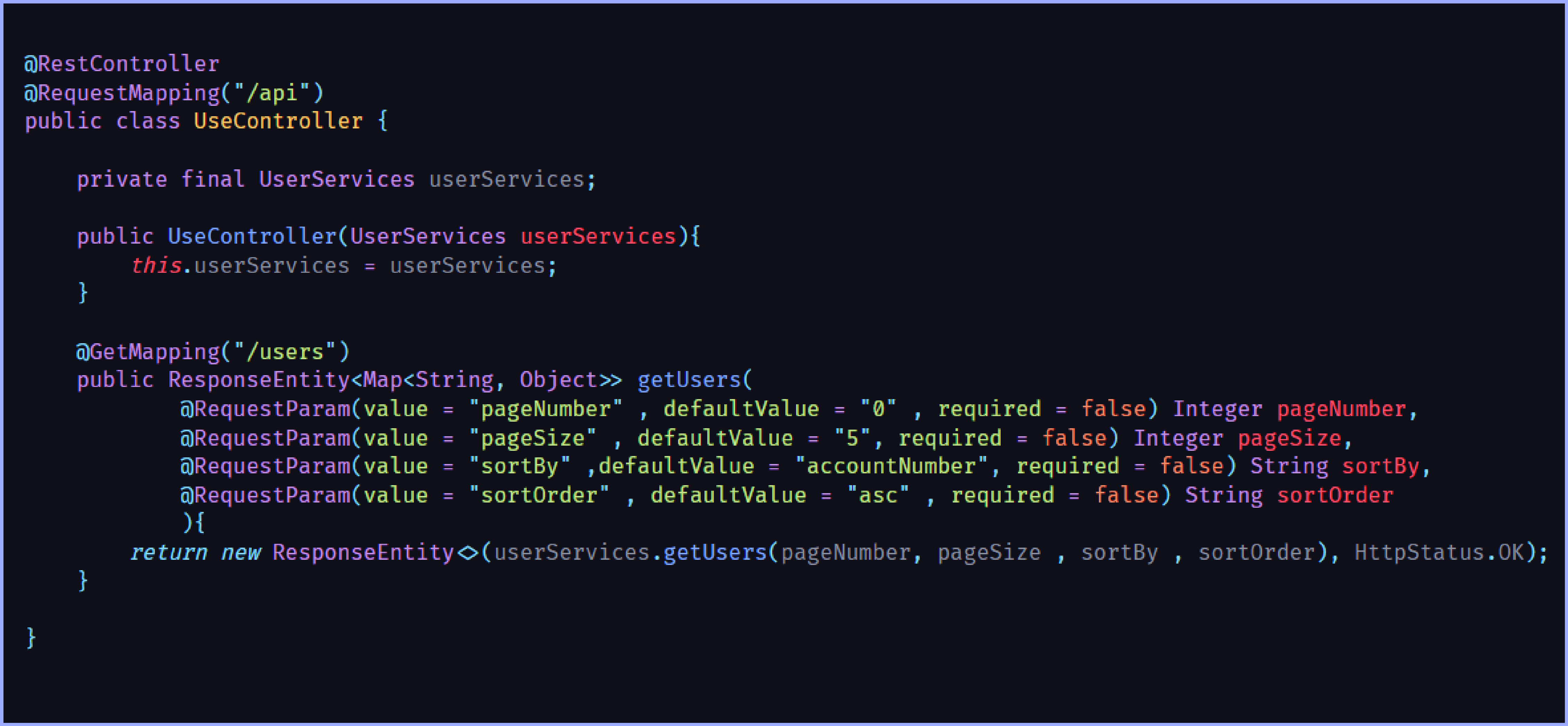
- **Definition**: Data can be sorted by **multiple fields**. For example, **first by name and then by price**.

- **Use Case:** Useful when you need to order data hierarchically, such as sorting by a primary criterion and then by a secondary criterion.

Pagination With Sorting

Controller

The controller processes sorting parameters along with pagination. It passes these parameters to the service layer to fetch the sorted and paginated data.

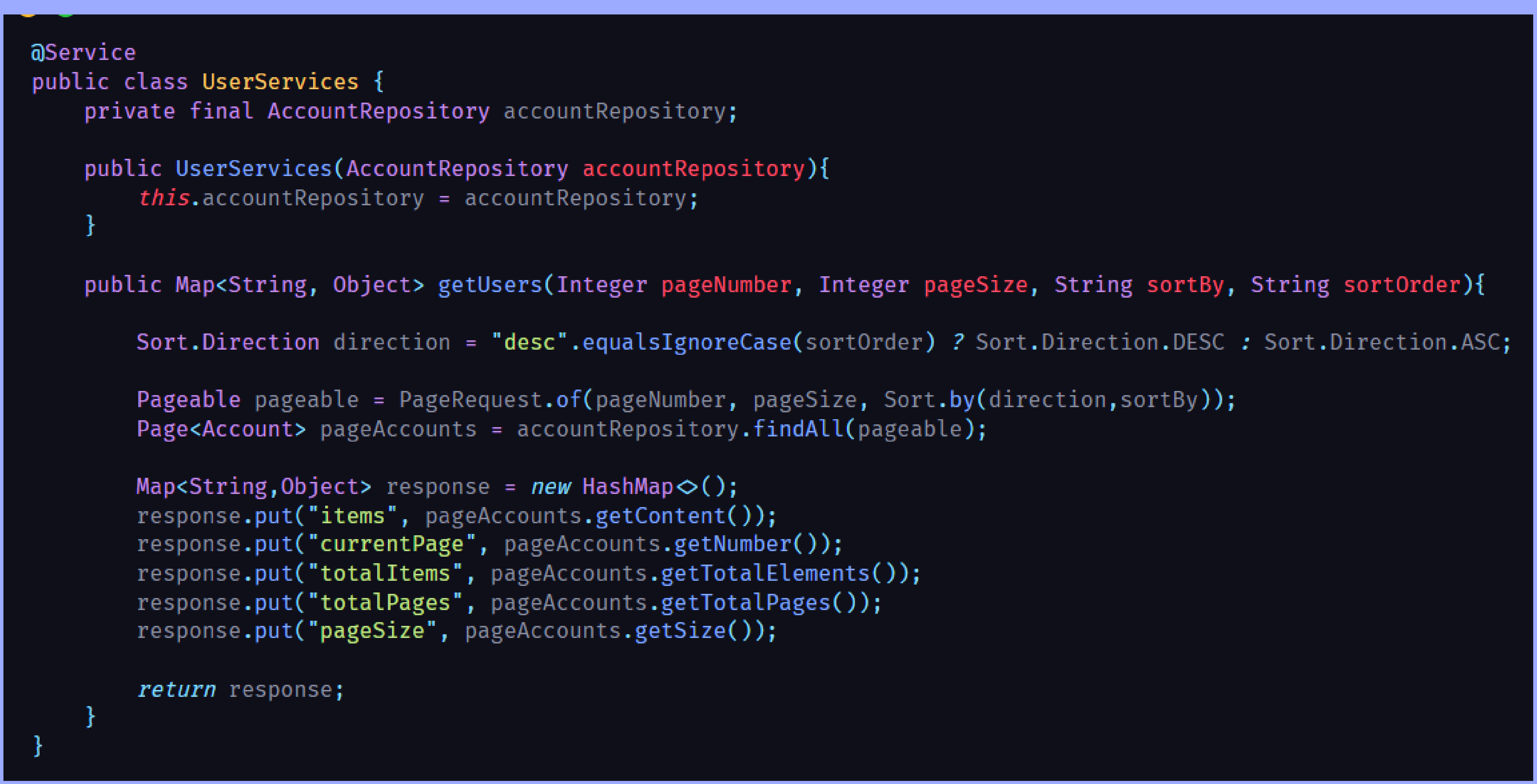


```java
@RestController
@RequestMapping("/api")
public class UseController {

    private final UserServices userServices;

    public UseController(UserServices userServices){
        this.userServices = userServices;
    }

    @GetMapping("/users")
    public ResponseEntity<Map<String, Object>> getUsers(
            @RequestParam(value = "pageNumber" , defaultValue = "0" , required = false) Integer pageNumber,
            @RequestParam(value = "pageSize" , defaultValue = "5", required = false) Integer pageSize,
            @RequestParam(value = "sortBy" ,defaultValue = "accountNumber", required = false) String sortBy,
            @RequestParam(value = "sortOrder" , defaultValue = "asc" , required = false) String sortOrder
            ){
        return new ResponseEntity<>(userServices.getUsers(pageNumber, pageSize , sortBy , sortOrder), HttpStatus.OK);
    }

}
```

Pagination With Sorting

Service

The service handles sorting by converting the sort direction and field into a Sort object, which is then used to create a PageRequest.

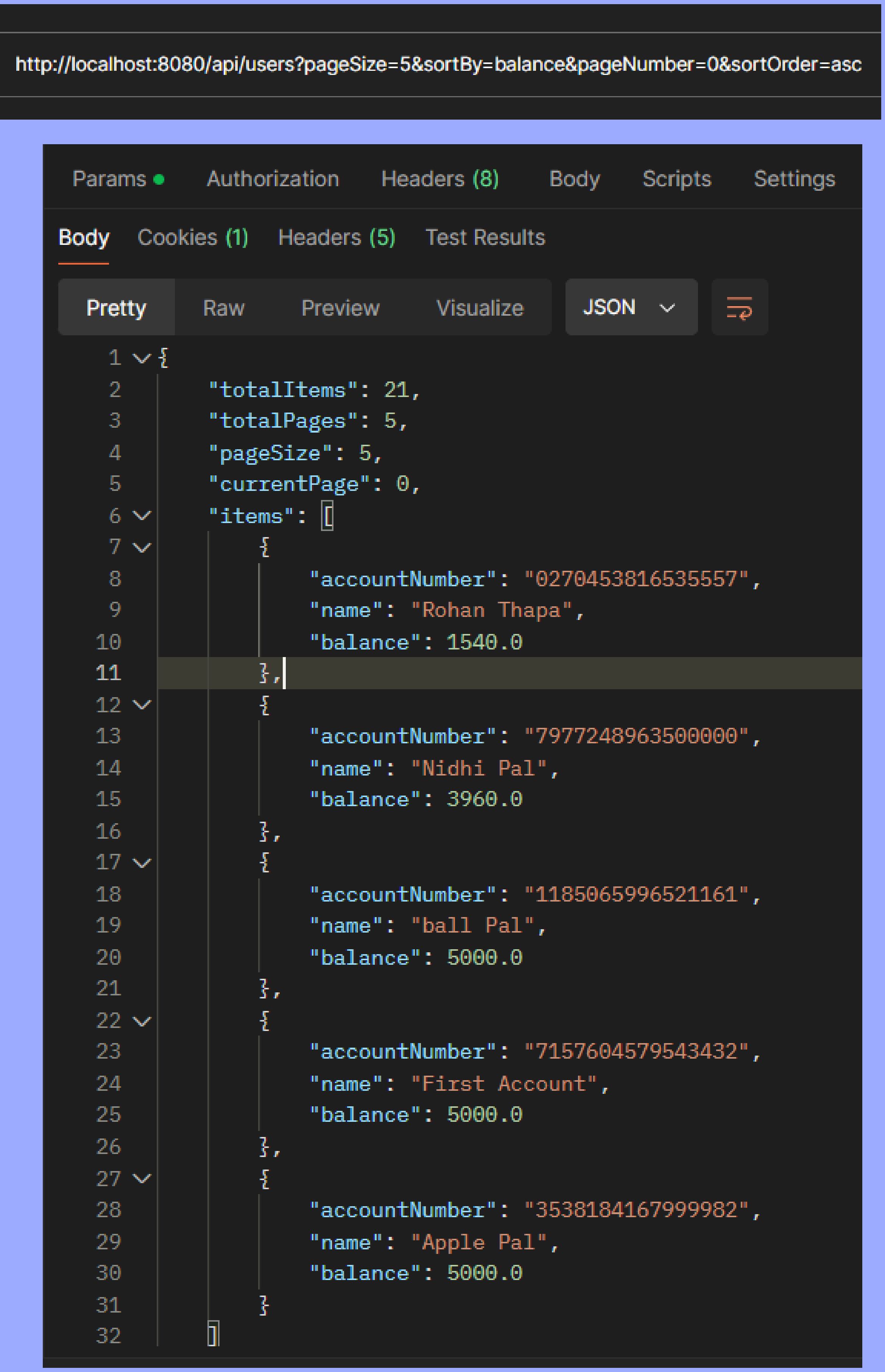


```java
@Service
public class UserServices {
    private final AccountRepository accountRepository;

    public UserServices(AccountRepository accountRepository){
        this.accountRepository = accountRepository;
    }

    public Map<String, Object> getUsers(Integer pageNumber, Integer pageSize, String sortBy, String sortOrder){

        Sort.Direction direction = "desc".equalsIgnoreCase(sortOrder) ? Sort.Direction.DESC : Sort.Direction.ASC;

        Pageable pageable = PageRequest.of(pageNumber, pageSize, Sort.by(direction,sortBy));
        Page<Account> pageAccounts = accountRepository.findAll(pageable);

        Map<String,Object> response = new HashMap<>();
        response.put("items", pageAccounts.getContent());
        response.put("currentPage", pageAccounts.getNumber());
        response.put("totalItems", pageAccounts.getTotalElements());
        response.put("totalPages", pageAccounts.getTotalPages());
        response.put("pageSize", pageAccounts.getSize());

        return response;
    }
}
```

Sort.Direction: Determines whether sorting should be ascending or descending based on the sortDirection parameter.

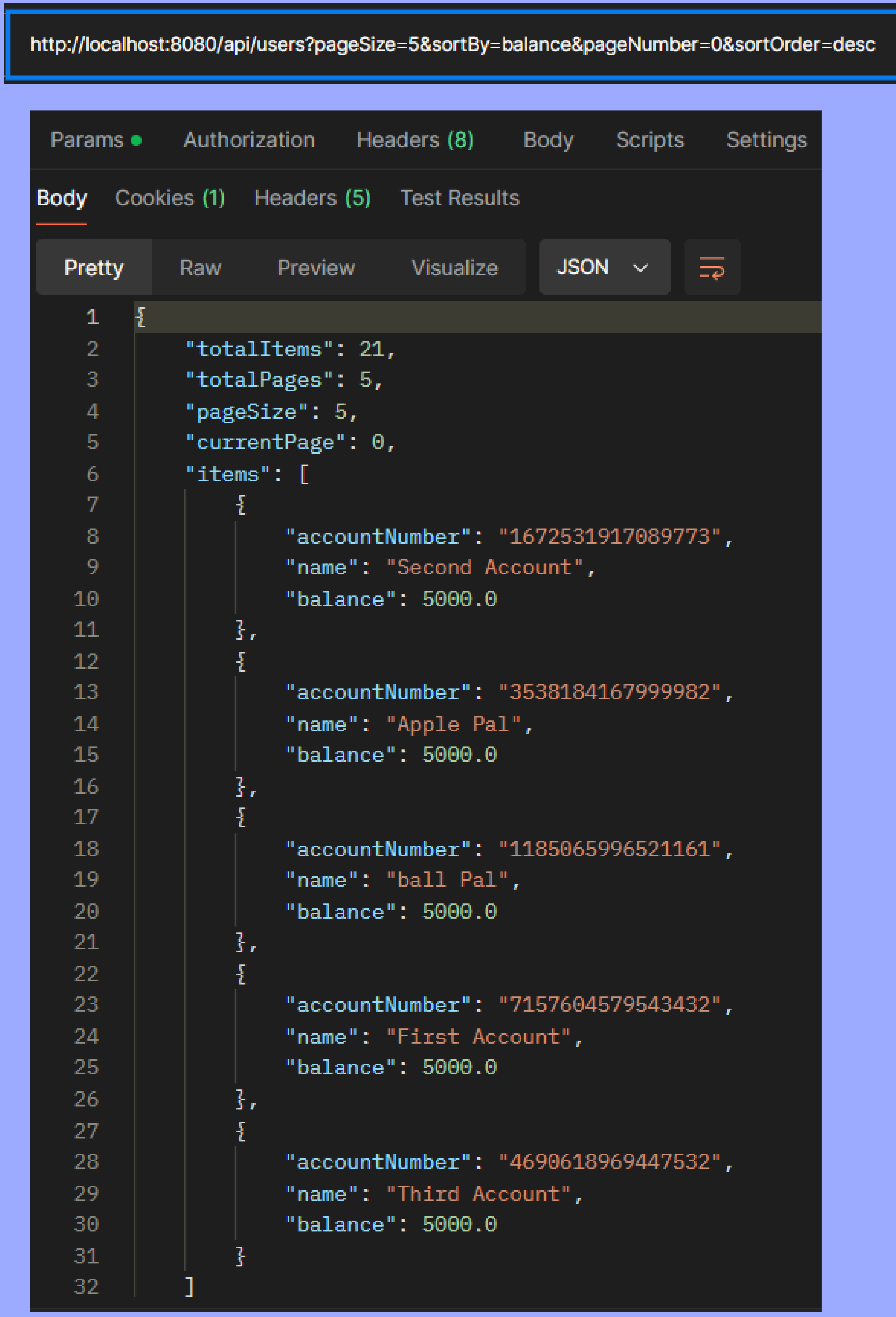
Pagination With Sorting

Result



http://localhost:8080/api/users?pageSize=5&sortBy=balance&pageNumber=0&sortOrder=asc

```json
{
    "totalItems": 21,
    "totalPages": 5,
    "pageSize": 5,
    "currentPage": 0,
    "items": [
        {
            "accountNumber": "0270453816535557",
            "name": "Rohan Thapa",
            "balance": 1540.0
        },
        {
            "accountNumber": "7977248963500000",
            "name": "Nidhi Pal",
            "balance": 3960.0
        },
        {
            "accountNumber": "1185065996521161",
            "name": "ball Pal",
            "balance": 5000.0
        },
        {
            "accountNumber": "7157604579543432",
            "name": "First Account",
            "balance": 5000.0
        },
        {
            "accountNumber": "3538184167999982",
            "name": "Apple Pal",
            "balance": 5000.0
        }
    ]
```

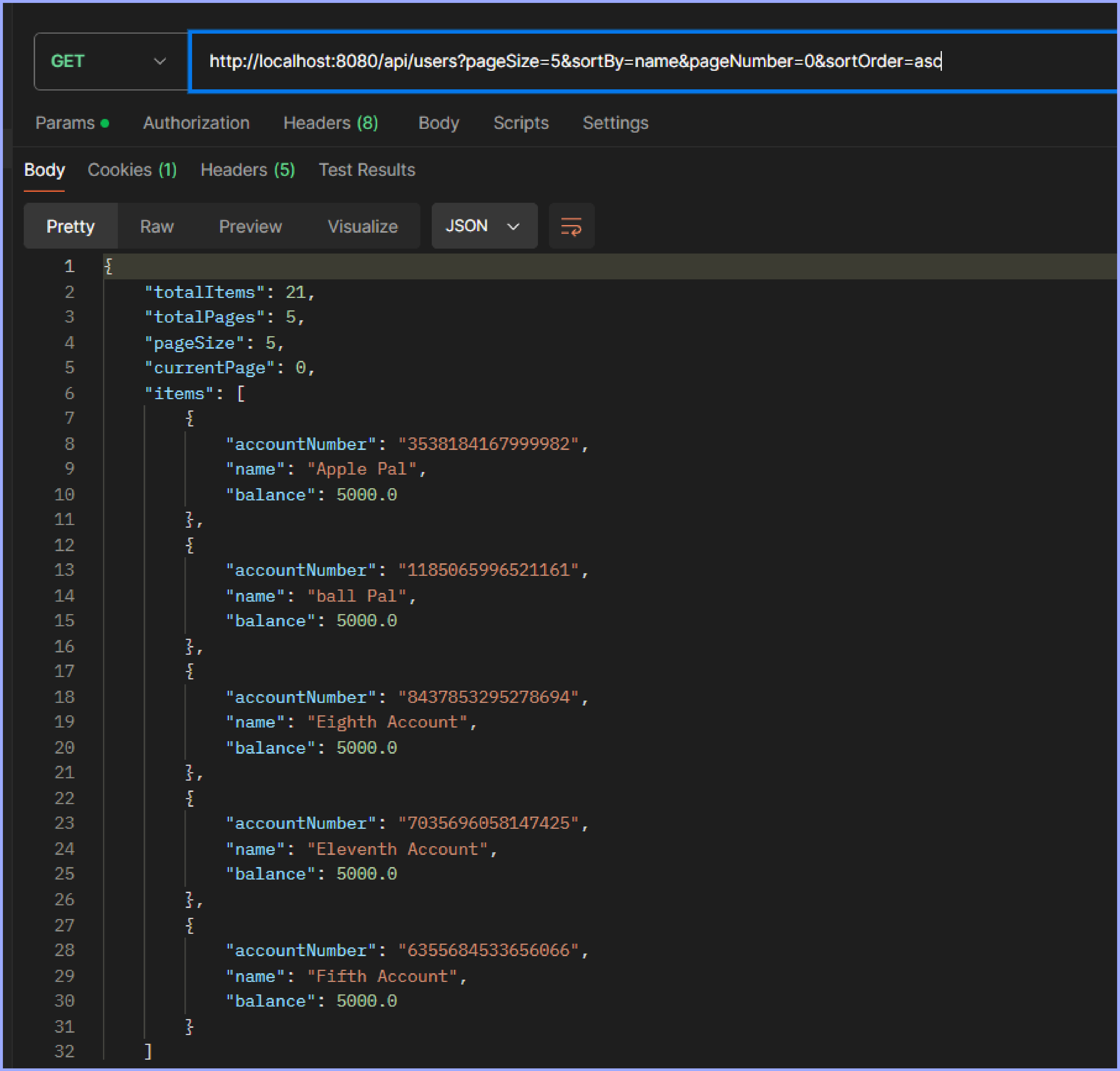


http://localhost:8080/api/users?pageSize=5&sortBy=balance&pageNumber=0&sortOrder=desc

```json
{
    "totalItems": 21,
    "totalPages": 5,
    "pageSize": 5,
    "currentPage": 0,
    "items": [
        {
            "accountNumber": "1672531917089773",
            "name": "Second Account",
            "balance": 5000.0
        },
        {
            "accountNumber": "3538184167999982",
            "name": "Apple Pal",
            "balance": 5000.0
        },
        {
            "accountNumber": "1185065996521161",
            "name": "ball Pal",
            "balance": 5000.0
        },
        {
            "accountNumber": "7157604579543432",
            "name": "First Account",
            "balance": 5000.0
        },
        {
            "accountNumber": "4690618969447532",
            "name": "Third Account",
            "balance": 5000.0
        }
    ]
```

Pagination With Sorting

Result



GET http://localhost:8080/api/users?pageSize=5&sortBy=name&pageNumber=0&sortOrder=asc

```
{
    "totalItems": 21,
    "totalPages": 5,
    "pageSize": 5,
    "currentPage": 0,
    "items": [
        {
            "accountNumber": "3538184167999982",
            "name": "Apple Pal",
            "balance": 5000.0
        },
        {
            "accountNumber": "1185065996521161",
            "name": "ball Pal",
            "balance": 5000.0
        },
        {
            "accountNumber": "8437853295278694",
            "name": "Eighth Account",
            "balance": 5000.0
        },
        {
            "accountNumber": "7035696058147425",
            "name": "Eleventh Account",
            "balance": 5000.0
        },
        {
            "accountNumber": "6355684533656066",
            "name": "Fifth Account",
            "balance": 5000.0
        }
    ]
```

Conclusion

Pagination and **sorting** are essential features for handling and presenting large datasets efficiently.

Pagination allows users to view data in manageable chunks, while sorting helps to organize data based on specific criteria.

By integrating both features, you can provide a robust and user-friendly data management experience in your Spring Boot applications.

Thank You

ROHAN THAPA
thaparohan2019@gmail.com