Pagination using Spring Data

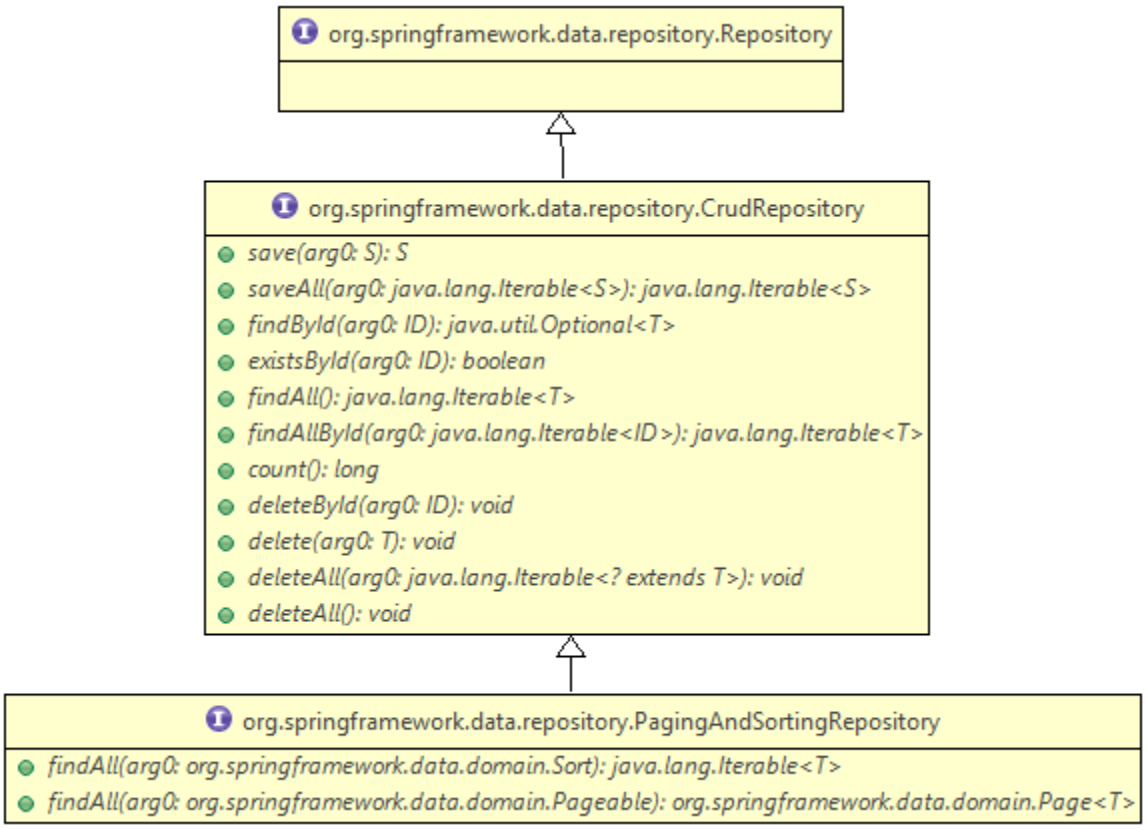
Let us consider one more scenario. Suppose, in a banking application a customer wants to see the details of all transactions performed by him/her. A customer can have hundreds of transactions and displaying all of them in a single page is not advisable. Also, sometimes fetching data of all the transactions at once is time consuming.

It would be much more better if we display the transaction details to the customer in smaller chunks, for example, 10 records per page. This can be implemented by using **pagination**.

In pagination, all the records are not fetched at same time, rather a subset of records is fetched first and then the next subset of records in fetched if required and so on. This subset of records is called as Page. Every page has two fields – the **page number** and **page size**. The page number represents the individual subset of records and page size represents the number of records in a page.

Also, the customers might want to view the transactions in a specific order. For example, the customer might want to view the transactions in increasing order of transaction date. This can be implemented by **sorting** the records.

Spring Data provides support for pagination and sorting through the **PagingAndSortingRepository** repository interface.



This interface extends CrudRepository interface and provides the following methods for supporting pagination and sorting:

- **Page<T> findAll(Pageable pageable)**
  - This method returns a Page containing entities and accepts a Pageable object as parameter.
  - Page interface contains information about the content of the page and some other information such as total number of pages, current page number and whether the current is first page or last page. Some important methods are as follows:

Method	Method description
List<T> getContent()	Returns the content of a page as List.
boolean hasContent()	Returns true if page has content else returns false.

- Pageable interface contains information about pagination such as size and page number. It is implemented by PageRequest class. You can create its objects using the following methods of PageRequest class:

Method	Method description
static PageRequest of(int page, int size)	Creates PageRequest object without any
static PageRequest of(int page, int size, Sort sort)	Creates PageRequest object along with

- **Iterable<T> findAll(Sort sort)**
  - This method returns all the entities in sorted order.
  - It accepts object of Sort class as parameter. This object contains information about the fields on which sorting is done and direction of sorting, i.e., ascending or descending.

You will now see how to use PagingAndSortingRepository interface for pagination.

Pagination

Consider the following entity class:

```
1.  @Entity
2.  public class Transaction{
3.    @Id
4.    private Integer transactionId;
5.    private LocalDate transactionDate;
6.    private Float transactionAmount;
7.    //getter and setter
8.  }
```

This entity class is mapped to following table:

transaction_id	transaction_date	transaction_amount
1	1995-06-06	1000
2	1998-03-02	1569
3	1996-07-10	4567
4	1995-09-14	2345
5	1996-10-18	6745
6	1995-12-23	2123
7	1996-04-01	6789
8	1995-06-09	3434
9	1995-09-24	2000
10	1996-01-29	6000
11	1995-05-07	1500
12	1995-10-04	1200
13	1996-01-12	2500
14	1996-01-29	6500
15	1996-01-29	7000

For pagination of transaction details, you need to create the following TransactionRepository interface by extending PagingAndSortingReository interface.

```
1. public interface TransactionRepository extends
   PagingAndSortingRepository<Transaction, Integer>{
2.


3. }
```

Since this repository is extending PagingAndSortingReository repository, so findAll(Pageable pageable) is available to this interface.

Suppose you want to fetch details of 5 transactions first, then next 5 and so on. To implement this requirement, you need to create an object of Pageable with page number = 0 and page size = 5, pass it to findAll() method and then invoke getContent() method as follows:

```
1. Pageable firstPagewithFiveRecords = PageRequest.of(0,5);
2. Page<Transaction> transactions =
   transactionRepository.findAll(firstPageWithFiveRecords);


3. List<Transaction> entityList = transactions.getContent();
```

After executing the above code snippet, entityList will contain the details of following transactions:

transaction_id	transaction_date	transaction_amount
1	1995-06-06	1000
2	1998-03-02	1569
3	1996-07-10	4567
4	1995-09-14	2345
5	1996-10-18	6745

Now for accessing the next page, create an object of Pageable with page number = 1 and page size = 5 , pass it to findAll() method and then invoke getContent() method:

```
1. Pageable secondPagewithFiveRecords = PageRequest.of(1,5);
2. Page<Transaction> transactions =
   transactionRepository.findAll(secondPageWithTwoElements);

3. List<Transaction> entityList = transactions.getContent();
```

After executing the above code snippet, entityList will contain the details of following transactions:

transaction_id	transaction_date	transaction_amount
6	1995-12-23	2123
7	1996-04-01	6789
8	1995-06-09	3434
9	1995-09-24	2000
10	1996-01-29	6000

Similarly for getting subsequent pages, you can keep on incrementing page number by 1.

Now suppose, you want to fetch the details of all transactions performed after a specified date and also want to do paging, then you can add query method to CustomerRepository as follows:

```
1. public interface TransactionRepository extends
   PagingAndSortingRepository<Transaction, Integer>{
2.   public List<Transaction> findByTransactionDateAfter(LocalDate
   transactionDate, Pageable pageable);
3. }

4.
```

In above repository, findByTransactionDateAfter() query method also accepts Pageable object as parameter for paging. If you want to fetch the details of transactions which are performed after 29-Jan-1996 and also implement paging with page size = 2, then, you need to create a Pageable object with page number = 0 and page size = 2, pass it to findByTransactionDateAfter() method along with the transaction date as follows:

```
1. LocalDate transactionDate = LocalDate.of(1996, 1, 29);
2. Pageable firstPagewithTwoRecords = PageRequest.of(0,2);
3. Pageable pageable = PageRequest.of(pageNo, pageSize);
4. List<Transaction> entityList =
   transactionRepository.findByTransactionDateAfter(transactionDate,pageable);
```

After executing the above code snippet, entityList will contain the details of following transactions:

transaction_id	transaction_date	transaction_amount
2	1998-03-02	1569
3	1996-07-10	4567

Pagination is also considered to be a best practice as it helps to avoid fetching more data than required.

Pagination using Spring Data - Demo

Objective :

In this demo, you will implement pagination using PagingAndSortingRepository by following the steps given below.

Steps :

Step 1 : Create a Spring Boot Maven project

Using Spring Initializr, create a Spring Boot project with following specifications:

- Spring Boot Version: 2.3.1 (The version keeps on changing, always choose the latest release)
- Group: com.training
- Artifact: Demo_SpringData_Pagination
- Name: Demo_SpringData_Pagination
- Package name: com.training
- Java Version: 11
- Dependencies: Spring Data JPA and MySQL Driver

Now import this project in Eclipse.

Step 2 : Create the database and table

Open MySQL terminal and execute the following command:

```
drop database if exists transaction_db;
create database transaction_db;
use  transaction_db;

create table transaction(
   transaction_id int auto_increment,
   transaction_date date,
   transaction_amount float,
   constraint ps_transaction_id_pk primary key (transaction_id)
);

insert into transaction values (1, sysdate()- interval 9136 day, 1000);
insert into transaction values (2, sysdate()- interval 8136 day, 1569);
insert into transaction values (3, sysdate()- interval 8736 day, 4567);
insert into transaction values (4, sysdate()- interval 9036 day, 2345);
insert into transaction values (5, sysdate()- interval 8636 day, 6745);
insert into transaction values (6, sysdate()- interval 8936 day, 2123);
insert into transaction values(7, sysdate()- interval 8836 day, 6789);
insert into transaction values (8, sysdate()- interval 9133 day, 3434);
insert into transaction values (9, sysdate()- interval 9026 day, 2000);
insert into transaction values (10, sysdate()- interval 8899 day, 6000);
insert into transaction values (11, sysdate()- interval 9166 day, 1500);
insert into transaction values (12, sysdate()- interval 9016 day, 1200);
insert into transaction values (13, sysdate()- interval 8916 day, 2500);
insert into transaction values (14, sysdate()- interval 8899 day, 6500);
insert into transaction values (15, sysdate()- interval 8899 day, 7000);


commit;
```

Step 3 : Configure the data source

Open application.properties in src/main/resources folder and add following properties for MySQL:

```
# MySQL settings
#Change these settings according to database you are using
spring.datasource.url=jdbc:mysql://localhost:3306/transaction_db
spring.datasource.username=root

#If MySQL installation is password proctored,then use below property to set password
spring.datasource.password=root

#JPA settings
spring.jpa.show-sql=true


spring.jpa.properties.hibernate.format_sql=true
```

Step 4 : Create the following TransactionDTO class in com.training.dto package:

```
public class TransactionDTO {
  private Integer transactionId;
  private LocalDate transactionDate;
  private Float transactionAmount;

  public TransactionDTO(Integer transactionId, LocalDate transactionDate, Float transactionAmount) {
          super();
          this.transactionId = transactionId;
          this.transactionDate = transactionDate;
          this.transactionAmount = transactionAmount;
  }

  public Integer getTransactionId() {
          return transactionId;
  }

  public void setTransactionId(Integer transactionId) {
          this.transactionId = transactionId;
  }

  public LocalDate getTransactionDate() {
          return transactionDate;
  }

  public void setTransactionDate(LocalDate transactionDate) {
          this.transactionDate = transactionDate;
  }

  public Float getTransactionAmount() {
          return transactionAmount;
  }

  public void setTransactionAmount(Float transactionAmount) {
          this.transactionAmount = transactionAmount;
  }

  @Override
  public String toString() {
          return "Transaction [transactionId=" + transactionId + ", transactionDate=" + transactionDate
                          + ", transactionAmount=" + transactionAmount + "]";
  }


}
```

Step 5 : Create the Transaction entity class in com.training.entity package:

```java
@Entity
public class Transaction {
  @Id
  private Integer transactionId;
  private LocalDate transactionDate;
  private Float transactionAmount;

  public Integer getTransactionId() {
          return transactionId;
  }

  public void setTransactionId(Integer transactionId) {
          this.transactionId = transactionId;
  }

  public LocalDate getTransactionDate() {
          return transactionDate;
  }

  public void setTransactionDate(LocalDate transactionDate) {
          this.transactionDate = transactionDate;
  }

  public Float getTransactionAmount() {
          return transactionAmount;
  }

  public void setTransactionAmount(Float transactionAmount) {
          this.transactionAmount = transactionAmount;
  }

  @Override
  public int hashCode() {
          final int prime = 31;
          int result = 1;
          result = prime * result + ((this.getTransactionId() == null) ? 0 : this.getTransactionId().hashCode());
          return result;
  }

  @Override
  public boolean equals(Object obj) {
          if (this == obj)
                  return true;
          if (obj == null)
                  return false;
          if (getClass() != obj.getClass())
                  return false;
          Transaction other = (Transaction) obj;
          if (this.getTransactionId() == null) {
                  if (other.getTransactionId() != null)
                          return false;
          }
        else if (!this.getTransactionId().equals(other.getTransactionId()))
                  return false;
          return true;
  }

  @Override
  public String toString() {
          return "Transaction [transactionId=" + transactionId + ", transactionDate=" + transactionDate
                          + ", transactionAmount=" + transactionAmount + "]";
  }

```

```
}
```

Step 6 : Create the following repository in com.training.repository package:

```
public interface TransactionRepository extends PagingAndSortingRepository<Transaction, Integer> {
  public List<Transaction> findByTransactionDateAfter(LocalDate transactionDate, Pageable pageable);


}
```

Step 7 : Create the following TransactionService interface in com.training.service package:

```
public interface TransactionService {
  public List<TransactionDTO> getAllTransaction(Integer pageNo, Integer pageSize) throws TrainingBankException;

  public List<TransactionDTO> getAllTransactionByTransactionDateAfter(LocalDate transactionDate, Integer pageNo,
                  Integer pageSize) throws TrainingBankException;


}
```

Step 8 : Create the TransactionServiceImpl class in com.training.service package:

```
@Service(value = "transactionService")
@Transactional
public class TransactionServiceImpl implements TransactionService {
  @Autowired
  private TransactionRepository transactionRepository;
  @Override
  public List<TransactionDTO> getAllTransaction(Integer pageNo, Integer pageSize) throws TrainingBankException {
          Pageable pageable = PageRequest.of(pageNo, pageSize);
          Page<Transaction> page = transactionRepository.findAll(pageable);
          if (page.isEmpty()) {
                  throw new TrainingBankException("Service.NO_CUSTOMERS_IN_THIS_PAGE");
          }
          List<Transaction> entityList = page.getContent();
          List<TransactionDTO> transactionDTOs;
          transactionDTOs = entityList.stream()
                          .map(p -> new TransactionDTO(p.getTransactionId(), p.getTransactionDate(), p.getTransactionAmount()))
                          .collect(Collectors.toList());
          return transactionDTOs;
  }
  @Override
  public List<TransactionDTO> getAllTransactionByTransactionDateAfter(LocalDate transactionDate, Integer pageNo,
                  Integer pageSize) throws TrainingBankException {
          Pageable pageable = PageRequest.of(pageNo, pageSize);
          List<Transaction> transactions = transactionRepository.findByTransactionDateAfter(transactionDate, pageable);
          if (transactions.isEmpty()) {
                  throw new TrainingBankException("Service.NO_CUSTOMERS_IN_THIS_PAGE");
          }
          List<TransactionDTO> transactionDTOs;

```

```
30.           transactionDTOs = transactions.stream()
31.                           .map(transaction -> new
   TransactionDTO(transaction.getTransactionId(),
   transaction.getTransactionDate(), transaction.getTransactionAmount()))
32.                           .collect(Collectors.toList());
33.           return transactionDTOs;
34.   }

35. }
```

Step 9 : Modify the application.properties and add the below property:

```
1.  Service.NO_TRANSACTION_IN_THIS_PAGE=No transactions available.
```

Step 10 : Modify the application class as follows:

```
1.  @SpringBootApplication
2.  public class DemoSpringDataPaginationApplication implements
   CommandLineRunner{
3.
4.    private static final Log LOGGER =
   LogFactory.getLog(DemoSpringDataPaginationApplication.class);
5.
6.    @Autowired
7.    TransactionService transactionService;
8.
9.    @Autowired
10.   Environment environment;
11.
12.   public static void main(String[] args) {
13.           SpringApplication.run(DemoSpringDataPaginationApplication.class,
   args);
14.   }
15.   @Override
16.   public void run(String... args) throws Exception {
17.           getAllTransactions();
18.           getAllTransactionsByTransactionDate();
19.   }
20.   public void getAllTransactions() {
21.           try {
22.                   List<TransactionDTO> transactionList =
   transactionService.getAllTransaction(0, 5);
23.                   transactionList.forEach(LOGGER::info);
24.           } catch (Exception e) {
25.                   String message = environment.getProperty(e.getMessage(),
26.                                   "Some exception occurred. Please check
   log file for more details!!");
27.                   LOGGER.info(message);
28.           }
29.   }
30.   public void getAllTransactionsByTransactionDate() {
31.           try {
32.                   LocalDate transactionDate = LocalDate.of(1996, 1, 29);
33.                   List<TransactionDTO> transactionList =
   transactionService.getAllTransactionByTransactionDateAfter(transactionDate,
   0,2);
34.                   transactionList.forEach(LOGGER::info);
35.           } catch (Exception e) {
36.                   String message = environment.getProperty(e.getMessage(),
37.                                   "Some exception occurred. Please check
   log file for more details!!");
38.                   LOGGER.info(message);
39.           }
40.   }
```

```
41.


42. }
```

- Invoking getAllTransactions() method which will give the following output:

```
Transaction [transactionId=1, transactionDate=1995-06-08, transactionAmount=1000.0]
Transaction [transactionId=2, transactionDate=1998-03-04, transactionAmount=1569.0]
Transaction [transactionId=3, transactionDate=1996-07-12, transactionAmount=4567.0]
Transaction [transactionId=4, transactionDate=1995-09-16, transactionAmount=2345.0]
Transaction [transactionId=5, transactionDate=1996-10-20, transactionAmount=6745.0]
```

Modify the page number in getAllTransactions() method to get other pages.

- Invoking getAllTransactionsByTransactionDate() will give the following output:

```
Transaction [transactionId=2, transactionDate=1998-03-04, transactionAmount=1569.0]
Transaction [transactionId=3, transactionDate=1996-07-12, transactionAmount=4567.0]
```

Modify the page number in getAllTransactionsByTransactionDate() method to get other pages.