Sorting using Spring Data

You have learnt how to implement pagination using PagingAndSortingRepository. Now, you will learn how to implement sorting using this repository.

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 the 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

For sorting the transaction details, you need to create the following TransactionRepository interface by extending PagingAndSortingReository interface as follows:

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


3.  }
```

Since this repository is extending PagingAndSortingReository repository, so findAll(Sort sort) is available to this interface for sorting.

The findAll(Sort sort) method accepts object of Sort class as parameter. This class provides different ways to sort data using single and multiple columns and direction of sorting, i.e., ascending or descending using different methods. Some important methods of the Sort class are as follows:

Method	Method description
public static Sort by(String... properties)	Creates a Sort object for given properties.
public Sort descending()	Returns a Sort object with current data in descending order.
public Sort ascending()	Returns a Sort object with current data in ascending order.

To sort the transaction details using transaction date, you need to first create an object of Sort class using by() method with transactionDate as parameter value and then pass the Sort object to findAll() method as follows:

```
1.  // Sorted by 'transactionDate' attribute in ascending order
2.  Sort sort = Sort.by("transactionDate");
```

```
3. Iterable<Transaction> transactions = transactionRepository.findAll(sort);
```

To sort the details in descending order of transaction date, you need to use the descending () method as follows:

```
1. // ordered by 'transactionDate' attribute in descending order
2. Sort sort = Sort.by("transactionDate").descending();
3. Iterable<Transaction> transactions = transactionRepository.findAll(sort);
```

You can also sort by transaction date and then by transaction amount as follows:

```
1. // sorted by 'transactionDate' attribute in descending order and
   'transactionAmount' in ascending order
2. Sort sort =
   Sort.by("transactionDate").descending().and(Sort.by("transactionAmount"));
3. Iterable<Transaction> transactions = transactionRepository.findAll(sort);
```

If you want to find transaction details after a specific transaction date and sorted based on transaction date along with pagination, then you have to add query method to TransactionRepository as follows:

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

3. }
```

If you want to fetch the details of transactions which are performed after 29-Jan-1996 and sorted based on transaction date with page size = 2, then, you need to create an object of Sort using by() method with transactionDate as parameter value. Then, you need to create Pageable object with page number = 0, page size = 2 and object of Sort class and pass this Pageable object to findByTransactionDateAfter() method along with transaction date as follows:

```
1. LocalDate transactionDate = LocalDate.of(1996, 1, 29);
2. Sort sort = Sort.by("transactionDate");
3. Pageable pageable = PageRequest.of(0,2,sort);
4. List<Transaction> transactionList =
   transactionRepository.findByTransactionDateAfter(transactionDate, pageable);
```

Sorting using Spring Data - Demo

Objective :

In this demo, you will learn how to implement sorting using Spring Data PagingAndSortingRepository .

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_Sorting
- Name: Demo_SpringData_Sorting
- 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 commands:

```
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);

commit;


select * from transaction;
```

Step 3 : Configure the data source

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

```
#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;

```

```
6.    public TransactionDTO(Integer transactionId, LocalDate transactionDate,
   Float transactionAmount) {
7.            super();
8.            this.transactionId = transactionId;
9.            this.transactionDate = transactionDate;
10.           this.transactionAmount = transactionAmount;
11.   }
12.
13.   public Integer getTransactionId() {
14.           return transactionId;
15.   }
16.
17.   public void setTransactionId(Integer transactionId) {
18.           this.transactionId = transactionId;
19.   }
20.
21.   public LocalDate getTransactionDate() {
22.           return transactionDate;
23.   }
24.
25.   public void setTransactionDate(LocalDate transactionDate) {
26.           this.transactionDate = transactionDate;
27.   }
28.
29.   public Float getTransactionAmount() {
30.           return transactionAmount;
31.   }
32.
33.   public void setTransactionAmount(Float transactionAmount) {
34.           this.transactionAmount = transactionAmount;
35.   }
36.
37.   @Override
38.   public String toString() {
39.           return "Transaction [transactionId=" + transactionId + ",
   transactionDate=" + transactionDate
40.                           + ", transactionAmount=" + transactionAmount +
   "]";
41.   }


42. }
```

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

```
1.  @Entity
2.  public class Transaction {
3.
4.    @Id
5.    private Integer transactionId;
6.    private LocalDate transactionDate;
7.    private Float transactionAmount;
8.
9.    public Integer getTransactionId() {
10.           return transactionId;
11.   }
12.
13.   public void setTransactionId(Integer transactionId) {
14.           this.transactionId = transactionId;
15.   }
16.
17.   public LocalDate getTransactionDate() {
18.           return transactionDate;
19.   }
20.
21.   public void setTransactionDate(LocalDate transactionDate) {
22.           this.transactionDate = transactionDate;
23.   }
```

```
24.
25.   public Float getTransactionAmount() {
26.           return transactionAmount;
27.   }
28.
29.   public void setTransactionAmount(Float transactionAmount) {
30.           this.transactionAmount = transactionAmount;
31.   }
32.
33.   @Override
34.   public int hashCode() {
35.           final int prime = 31;
36.           int result = 1;
37.           result = prime * result + ((this.getTransactionId() == null) ? 0 :
   this.getTransactionId().hashCode());
38.           return result;
39.   }
40.   @Override
41.   public boolean equals(Object obj) {
42.           if (this == obj)
43.                   return true;
44.           if (obj == null)
45.                   return false;
46.           if (getClass() != obj.getClass())
47.                   return false;
48.           Transaction other = (Transaction) obj;
49.           if (this.getTransactionId() == null) {
50.                   if (other.getTransactionId() != null)
51.                           return false;
52.           }
53.           else if
   (!this.getTransactionId().equals(other.getTransactionId()))
54.                   return false;
55.           return true;
56.   }
57.
58.   @Override
59.   public String toString() {
60.           return "Transaction [transactionId=" + transactionId + ",
   transactionDate=" + transactionDate
61.                           + ", transactionAmount=" + transactionAmount +
   "]";
62.   }
63. }


64.
```

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

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


4.
```

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

```
1.  public interface TransactionService {
2.    public List<TransactionDTO> getAllTransaction(Sort sort) throws
    TrainingBankException;
3.    public List<TransactionDTO>
    getAllTransactionByTransactionDateAfter(LocalDate transactionDate, Pageable
    pageable) throws TrainingBankException;
4.  }
```

```
5.
```

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

```
1.  @Service(value = "transactionService")
2.  public class TransactionServiceImpl implements TransactionService {
3.
4.    @Autowired
5.    private TransactionRepository transactionRepository;
6.
7.    @Override
8.    public List<TransactionDTO> getAllTransaction(Sort sort) throws
    TrainingBankException {
9.            Iterable<Transaction> transactions =
    transactionRepository.findAll(sort);
10.           List<TransactionDTO> transactionDTOs = new ArrayList<>();
11.           transactions.forEach(transaction -> {
12.                   TransactionDTO t = new
    TransactionDTO(transaction.getTransactionId(),
    transaction.getTransactionDate(),
13.                                   transaction.getTransactionAmount());
14.                   transactionDTOs.add(t);
15.           });
16.
17.           if (transactionDTOs.isEmpty()) {
18.                   throw new
    TrainingBankException("Service.NO_CUSTOMERS_IN_THIS_PAGE");
19.           }
20.           return transactionDTOs;
21.   }
22.
23.   @Override
24.   public List<TransactionDTO>
    getAllTransactionByTransactionDateAfter(LocalDate transactionDate, Pageable
    pageable) throws TrainingBankException {
25.           List<Transaction> transactions =
    transactionRepository.findByTransactionDateAfter(transactionDate, pageable);
26.           if (transactions.isEmpty()) {
27.                   throw new
    TrainingBankException("Service.NO_CUSTOMERS_IN_THIS_PAGE");
28.           }
29.
30.           return transactions.stream()
31.                           .map(p -> new
    TransactionDTO(p.getTransactionId(), p.getTransactionDate(),
    p.getTransactionAmount()))
32.                           .collect(Collectors.toList());
33.
34.   }
35.


36. }
```

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

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

Step 8 : Modify the application class

Modify the application class as follows:

```
1.  @SpringBootApplication
```

```
2. public class DemoSpringDataSortingApplication implements CommandLineRunner {
3.
4.   private static final Log LOGGER =
   LogFactory.getLog(DemoSpringDataSortingApplication.class);
5.
6.   @Autowired
7.   TransactionService transactionService;
8.
9.   @Autowired
10.  Environment environment;
11.
12.  public static void main(String[] args) {
13.          SpringApplication.run(DemoSpringDataSortingApplication.class,
   args);
14.  }
15.
16.  @Override
17.  public void run(String... args) throws Exception {
18.          getAllTransactions();
19.          getAllTransactionsAfterTransactionDate();
20.  }
21.
22.  public void getAllTransactions() {
23.          try {
24.                  Sort sort = Sort.by("transactionDate");
25.                  List<TransactionDTO> transactionDTOs =
   transactionService.getAllTransaction(sort);
26.                  transactionDTOs.forEach(LOGGER::info);
27.          } catch (Exception e) {
28.                  String message = environment.getProperty(e.getMessage(),
29.                                  "Some exception occured. Please check log
   file for more details!!");
30.                  LOGGER.info(message);
31.          }
32.
33.  }
34.
35.  public void getAllTransactionsAfterTransactionDate() {
36.          try {
37.                  LocalDate transactionDate = LocalDate.of(1996, 1, 29);
38.                  Sort sort = Sort.by("transactionDate");
39.                  Pageable pageable = PageRequest.of(0, 2,sort);
40.
41.                  List<TransactionDTO> transactionDTOs =
   transactionService.getAllTransactionByTransactionDateAfter(transactionDate,
   pageable);
42.                  transactionDTOs.forEach(LOGGER::info);
43.
44.          } catch (Exception e) {
45.                  String message = environment.getProperty(e.getMessage(),
46.                                  "Some exception occured. Please check log
   file for more details!!");
47.                  LOGGER.info(message);
48.          }
49.
50.  }
51.


52. }
```

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

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

- Invoking getAllTransactionAfterTransactionDate() will give the following output:

```
Transaction [transactionId=3, transactionDate=1996-07-12, transactionAmount=4567.0]
Transaction [transactionId=5, transactionDate=1996-10-20, transactionAmount=6745.0]
```