JAVA MINI Assignment 1

Problem Statement:

To implement the "Get Transactions" endpoint for the given account number and status value:

http://localhost:8090/transactions/{accountNumber}?status={ALL,success,pending,failure}

let's consider two scenarios:

1. **Scenario A: When the status value is "ALL":**
   In this case, we need to fetch all transactions from their respective backend servers concurrently.
   - By implementing this approach, we can effectively develop three independent backend servers that will return hard-coded responses from their respective JPA Repositories. (Refer PART 1 in Approach Section).
   - To achieve this solution, we can leverage the concepts of Java 8 or above, such as Thread Executor and Completable Future for asynchronous processing. For making REST API calls in Spring Boot, we can use Feign client or Web Client instead of Rest Template, as Rest Template is deprecated. (Refer PART 2 in Approach Section).

To achieve the desired functionality, we can implement the following approach:

PART 1:

1. Create three separate components in Spring Boot, each representing a different backend server.

2. Utilize JPA Repositories within each component to retrieve hard-coded responses.

Component 1 (Act as Backend Server 1 for returning success transactions):
For Success Transactions: http://localhost:8081/backendserver1/success/{accountNumber}

```
{
 AccountNumber:"12233300011001",
 success:[{
 transactionid:"123456789",
 status:"success",
 amount:"500",
 date:"30-05-2023"
 },{
 transactionid:"789566233",
 status:"success",
 amount:"100",
 date:"31-05-2023"
 },{
 transactionid:"556666678",
 status:"success",
 amount:"700",
 date:"20-05-2023"
 }]
 }
```

Component 2 (Act as Backend Server 2 for returning failure transactions):

For Failure Transactions: http://localhost:8082/backendserver2/failure/{accountNumber}

```
{
 AccountNumber:"12233300011001",
 failure:[
 {
 transactionid:"345577721",
 status:"fail",
 amount:"500",
 date:"30-04-2023"
 },
 {
 transactionid:"245900023",
 status:"fail",
 amount:"50",
 date:"29-05-2023"
 }
 ]
 }
```

Component 3 (Act as Backend Server 3 for returning pending transactions):

For Pending Transactions: http://localhost:8083/backendserver3/pending/{accountNumber}

```
{
AccountNumber:"12233300011001",
pending:[
 {
 transactionid:"88797721",
 status:"pending",
 amount:"500",
 date:"30-04-2023"
 }
 ]
}
```

JAVA MINI Assignment 1

PART 2:

- Initiate concurrent requests to fetch transactions from each backend server using Completable Future.
- Collect the responses and combine them into a single list using Stream APIs.
- If any response is missing or not received from a particular backend server, return an empty array or appropriate error response for that specific server.

Final Expected Response:

The response from the "Get Transaction" endpoint should be a JSON array containing the consolidated transactions from all the backend servers.

```
{
 success: [{
 transactionid:"123456789",
 status:"sucess",
 amount:"500",
 date:"30-05-2023"
 },{
 transactionid:"789566233",
 status:"sucess",
 amount:"100",
 date:"31-05-2023"
 },{
 transactionid:"556666678",
 status:"sucess",
 amount:"700",
 date:"20-05-2023"
 }],
 failure:[
 {
 transactionid:"345577721",
 status:"fail",
 amount:"500",
 date:"30-04-2023"
 },
 {
 transactionid:"245900023",
 status:"fail",
 amount:"50",
 date:"29-05-2023"
 }
 ], pending:[]
}
```

2. **Scenario B: When the status value is "Success," "Failure," or "Pending,"**
    - we need to dynamically choose the corresponding backend bean to fetch the transactions. We can implement this using the Factory Design Pattern in Spring.
    - This approach allows us to dynamically select the backend implementation at runtime based on the status value provided.

By using the Factory Design Pattern, we achieve loose coupling between the controller/service and the backend implementation, making the code more maintainable and scalable.

Final Expected Response for second Scenario (Suppose query param value is Success):

```
{
 success:[{
 transactionid:"123456789",
 status:"success",
 amount:"500",
 date:"30-05-2023"
 },{
 transactionid:"789566233",
 status:"success",
 amount:"100",
 date:"31-05-2023"
 },{
 transactionid:"556666678",
 status:"success",
 amount:"700",
 date:"20-05-2023"
 }]
}
```

NOTE: Write Junit Test Cases for above Endpoint, covering all the scenarios. Usage of Mockito for mocking the response is mandatory.