# Banking Services Platform Test Guideline By Sanders

This document entails the interaction of 3 separate spring cloud micro-services that were developed to come up with a demo banking services platform as part of the Equals Test Requirements.

The following technologies were used to come up with the solution:

```
 1.Java 17
 2.Spring Boot 3
 3.Spring Cloud (Eureka service Discovery, Spring Cloud Gateway, Resillience4j circuit
breaker and Feign client)
 4.RabbitMQ
 5.Redis Server (cache)
 6.Postgres
 7.Mongo DB
 8.Docker
 9.Docker Compose
10. J-Unit & Mockito (for tests)
```

## The 3 Main services are:

```
Account Management Service
```

```
Transaction Processing Service
```

```
Customer Support Service
```

## Other Important Details To Note

1. All the request have a common base url even though there are separate services. This is because every request is coming through the spring cloud gateway which then maps request to appropriate service behind the scenes.
2. The base URL for all the request is http://localhost:2003/**
3. All monetary values are stored in cents for accuracy reasons, thus when making a request which involves monetary figures for example (deposit request) make sure the amount value is submitted in cents.
4. All dates are expected in the format (yyyy-MM-dd)
5. We are using redis caching technology on balance enquiry and findAll methods, thus sometimes

you may not get up-to-date data just after making a request, however the cache has a living period of 60s.

6. Most of the requests were tested effectively using J-Unit and you can find the tests under the test folder of each and every service.
7. All the http and curl request and responses attached in this document were generated automatically by the JUnit tests that were carried out.
8. The Eureka service discovery dashboard is available at http://localhost:2004. We can access this portal to view all the services that are up running within our cluster.
9. Instructions on how to boostrap this application using are provided inside the README file located in the github repository of this project.

## Account Management Service

Test 1: Create User Account

Inorder to create a customer account, the following test was done successfully

*request*

```
$ curl 'http://localhost:2003/account/create' -i -X POST \
    -H 'Content-Type: application/json;charset=UTF-8' \
    -d '{
  "accountNumber" : "4145121212201",
  "accountName" : "Manuel Akanji",
  "accountDetails" : "This is a test checking account for Akanji",
  "accountType" : "CHECKING_ACCOUNT"
}'
```

```
POST /account/create HTTP/1.1
Content-Type: application/json;charset=UTF-8
Content-Length: 202
Host: localhost:2003

{
  "accountNumber" : "4145121212201",
  "accountName" : "Manuel Akanji",
  "accountDetails" : "This is a test checking account for Akanji",
  "accountType" : "CHECKING_ACCOUNT"
}
```

*response*

```
HTTP/1.1 201 Created
Content-Type: text/plain;charset=UTF-8
Content-Length: 13
```

```
4145121212201
```

Test 2: Find An existing Customer Account

To find an already existing customer account inside the database, the following test was carried out successfully:

*request*

```
$ curl 'http://localhost:2003/account/find/41445062354200' -i -X GET
```

```
GET /account/find/41445062354200 HTTP/1.1
Host: localhost:2003
```

*response*

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 175

{"accountNumber":"41445062354200","accountName":"Pep Joseph
Guardiola","accountBal":0,"accountDetails":"This is a test Saving account for
Pep","accountType":"SAVINGS_ACCOUNT"}
```

Test 3: Find Non-Existing Account

The following test was carried out to observe what happens when one tries to find a non-existing account:

*request*

```
$ curl 'http://localhost:2003/account/find/41325435476200' -i -X GET
```

```
GET /account/find/41325435476200 HTTP/1.1
Host: localhost:2003
```

*response*

```
HTTP/1.1 404 Not Found
Content-Type: application/json
Content-Length: 70

{"errorMessage":"User Account with account 41325435476200 not found!"}
```

# Transaction Processing Service

Test 1: Deposit Funds into an existing account

The following test was done to observe the deposit funds functionality:

*request*

```
$ curl 'http://localhost:2003/transaction/deposit/4145121212201/400000' -i -X POST
```

```
POST /transaction/deposit/4145121212201/400000 HTTP/1.1
Host: localhost:2003
Content-Type: application/x-www-form-urlencoded
```

*response*

```
HTTP/1.1 200 OK
Content-Type: text/plain;charset=UTF-8
Content-Length: 50

Deposit successful. New account balance is $4000.0
```

Test 2: Check New Balance After deposit

The following test was carried out to observe the check balance functionality:

*request*

```
$ curl 'http://localhost:2003/transaction/balance?account=4145121212201' -i -X GET
```

```
GET /transaction/balance?account=4145121212201 HTTP/1.1
Host: localhost:2003
```

*response*

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 6

4000.0
```

Test 3: Deposit Funds To A Non-Existing Account

The following test was carried out to see how the service handles enquiries on non-existing accounts:

*request*

```
$ curl 'http://localhost:2003/transaction/deposit/4145121212201111/2000' -i -X POST
```

```
POST /transaction/deposit/4145121212201111/2000 HTTP/1.1
Host: localhost:2003
Content-Type: application/x-www-form-urlencoded
```

*response*

```
HTTP/1.1 404 Not Found
Content-Type: text/plain;charset=UTF-8
Content-Length: 72

{"errorMessage":"User Account with account 4145121212201111 not found!"}
```

Test 4: Deposit Funds With Invalid Amount

The following test was carried out to see how the service handles invalid amount deposits:

*request*

```
$ curl 'http://localhost:2003/transaction/deposit/4145121212201/-3000' -i -X POST
```

```
POST /transaction/deposit/4145121212201/-3000 HTTP/1.1
Host: localhost:2003
Content-Type: application/x-www-form-urlencoded
```

*response*

```
HTTP/1.1 400 Bad Request
Content-Type: text/plain;charset=UTF-8
Content-Length: 44

Direct Deposit amount should be at-least $10
```

Test 5: Withdraw Funds

The following test was carried out to observe the withdraw funds functionality. Here Charges are also expected to apply depending on amount withdrawn:

*request*

```
$ curl 'http://localhost:2003/transaction/withdraw/4145121212201/4500' -i -X POST
```

```
POST /transaction/withdraw/4145121212201/4500 HTTP/1.1
Host: localhost:2003
Content-Type: application/x-www-form-urlencoded
```

*response*

```
HTTP/1.1 200 OK
Content-Type: text/plain;charset=UTF-8
Content-Length: 9

CF-714113
```

Test 6: Withdraw Funds With Insufficient Credit

The following test was carried out to observe how the service handles insufficient credit withdraws:

*request*

```
$ curl 'http://localhost:2003/transaction/withdraw/4145121212201/500000' -i -X POST
```

```
POST /transaction/withdraw/4145121212201/500000 HTTP/1.1
Host: localhost:2003
Content-Type: application/x-www-form-urlencoded
```

*response*

```
HTTP/1.1 400 Bad Request
Content-Type: text/plain;charset=UTF-8
Content-Length: 43

Insufficient credit to perform transaction!
```

Test 7: Transfer Funds

The following test was carried out to observe the funs transfer functionality (ZIPIT, INTERNAL TRANSFERS etc).

Ps. Charges also apply here depending on the amount to be transferred.

*request*

```
$ curl 'http://localhost:2003/transaction/transfer' -i -X POST \
    -H 'Content-Type: application/json;charset=UTF-8' \
    -d '{
  "fromAccount" : "4145121212201",
  "toAccount" : "41445062354200",
  "amount" : 10000,
  "transactionType" : "INTERNAL_TRANSFER"
}'
```

```
POST /transaction/transfer HTTP/1.1
Content-Type: application/json;charset=UTF-8
Content-Length: 134
Host: localhost:2003

{
  "fromAccount" : "4145121212201",
  "toAccount" : "41445062354200",
  "amount" : 10000,
  "transactionType" : "INTERNAL_TRANSFER"
}
```

*response*

```
HTTP/1.1 200 OK
Content-Type: text/plain;charset=UTF-8
Content-Length: 9

TF-380021
```

Test 8: Generate Mini-Statement Functionality (Recent transactions - last 5)

The following test was carried out to observe the generate mini-statement functionality:

*request*

```
$ curl 'http://localhost:2003/transaction/statement/4145121212201' -i -X GET
```

```
GET /transaction/statement/4145121212201 HTTP/1.1
Host: localhost:2003
```

*response*

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 627

[{"id":3,"fromAccount":"4145121212201","toAccount":"41445062354200","amount":10000,"tr
ansactionDate":[2023,11,24],"transactionType":"INTERNAL_TRANSFER","transactionRef":"TF
-380021","description":"Funds
Transfer"},{"id":2,"fromAccount":"4145121212201","toAccount":null,"amount":4500,"trans
actionDate":[2023,11,24],"transactionType":"WITHDRAWAL","transactionRef":"CF-
714113","description":"Account Withdrawal
performed"},{"id":1,"fromAccount":null,"toAccount":"4145121212201","amount":400000,"tr
ansactionDate":[2023,11,24],"transactionType":"DIRECT_DEPOSIT","transactionRef":"DF-
575649","description":"Direct Deposit performed"}]
```

Test 9: Generate Account Statement From A Given Date

The following test was carried out to observe the generate account statement functionality from a given date up to present:

*request*

```
$ curl 'http://localhost:2003/transaction/statement/4145121212201?startDate=2023-11-
23' -i -X GET
```

```
GET /transaction/statement/4145121212201?startDate=2023-11-23 HTTP/1.1
Host: localhost:2003
```

*response*

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 627

[{"id":1,"fromAccount":null,"toAccount":"4145121212201","amount":400000,"transactionDa
te":[2023,11,24],"transactionType":"DIRECT_DEPOSIT","transactionRef":"DF-
575649","description":"Direct Deposit
performed"},{"id":2,"fromAccount":"4145121212201","toAccount":null,"amount":4500,"tran
sactionDate":[2023,11,24],"transactionType":"WITHDRAWAL","transactionRef":"CF-
714113","description":"Account Withdrawal
performed"},{"id":3,"fromAccount":"4145121212201","toAccount":"41445062354200","amount
":10000,"transactionDate":[2023,11,24],"transactionType":"INTERNAL_TRANSFER","transact
ionRef":"TF-380021","description":"Funds Transfer"}]
```

Test 10: Generate Account Statement Up To A Given Date

The following test was carried out to observe the generate account statement functionality up to a given date:

*request*

```
$ curl 'http://localhost:2003/transaction/statement/4145121212201?endDate=2023-11-20'
-i -X GET
```

```
GET /transaction/statement/4145121212201?endDate=2023-11-20 HTTP/1.1
Host: localhost:2003
```

*response*

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 2

[]
```

Test 11: Generate Account Statement Between Two Givem Dates (Start Date & End Date)

The following test was carried out to observe the generate account statement functionality between 2 given dates:

*request*

```
$ curl 'http://localhost:2003/transaction/statement/4145121212201?startDate=2023-11-
23&endDate=2023-11-24' -i -X GET
```

```
GET /transaction/statement/4145121212201?startDate=2023-11-23&endDate=2023-11-24
HTTP/1.1
Host: localhost:2003
```

*response*

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 627

[{"id":1,"fromAccount":null,"toAccount":"4145121212201","amount":400000,"transactionDa
te":[2023,11,24],"transactionType":"DIRECT_DEPOSIT","transactionRef":"DF-
575649","description":"Direct Deposit
performed"},{"id":2,"fromAccount":"4145121212201","toAccount":null,"amount":4500,"tran
sactionDate":[2023,11,24],"transactionType":"WITHDRAWAL","transactionRef":"CF-
714113","description":"Account Withdrawal
performed"},{"id":3,"fromAccount":"4145121212201","toAccount":"41445062354200","amount
":10000,"transactionDate":[2023,11,24],"transactionType":"INTERNAL_TRANSFER","transact
ionRef":"TF-380021","description":"Funds Transfer"}]
```

## Customer Support  Service

Test 1: Log Customer Issue

The following test was carried out to verify the functionality of the log issue: The logged issue will also be published to RabbitMQ Queue for services such as email & sms to consume.

*request*

```
$ curl 'http://localhost:2003/support/log' -i -X POST \
    -H 'Content-Type: application/json;charset=UTF-8' \
    -d '{
  "issueType" : "TRANSACTIONAL",
  "description" : "This is just a test issue being logged by me",
  "fromAccount" : "4145121212201"
}'
```

```
POST /support/log HTTP/1.1
Content-Type: application/json;charset=UTF-8
Content-Length: 136
Host: localhost:2003

{
  "issueType" : "TRANSACTIONAL",
  "description" : "This is just a test issue being logged by me",
  "fromAccount" : "4145121212201"
}
```

*response*

```
HTTP/1.1 201 Created
Content-Type: text/plain;charset=UTF-8
Content-Length: 51

Issue logged successfully. Ticket track number => 1
```

Test 2: Track Issue with Ticket Id

The following test was carried out to see if the service is able to track logged issue effectively:

*request*

```
$ curl 'http://localhost:2003/support/track/1' -i -X GET
```

```
GET /support/track/1 HTTP/1.1
Host: localhost:2003
```

*response*

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 303

{"issueId":1,"ticketId":1,"issueStatus":"PENDING","description":"AWAITING
RESPONSE","dateLogged":"2023-11-
24","issueLog":{"issueId":1,"issueType":"TRANSACTIONAL","description":"This is just a
test issue being logged by
me","fromAccount":"4145121212201","issueStatus":"PENDING","issueDate":"2023-11-24"}}
```