1. Create Account

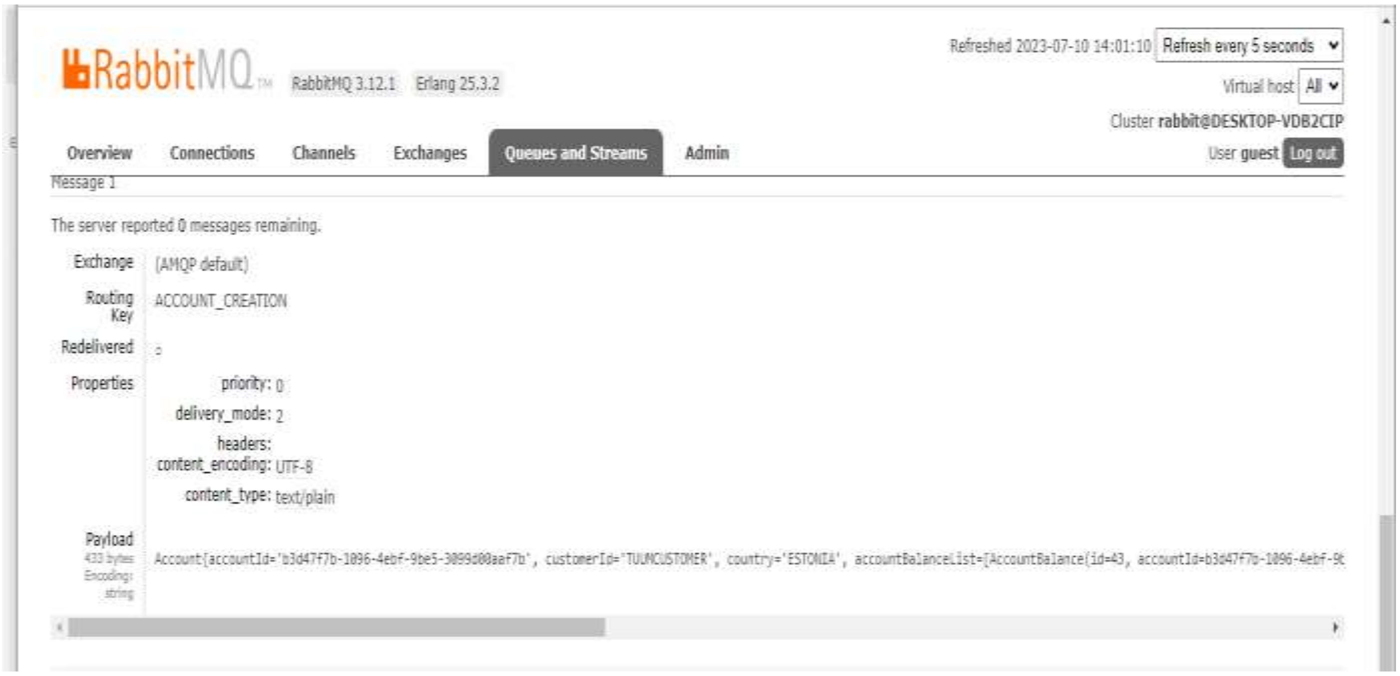
- API: Create account with account balances for given currency types. After creating the account, account creation notification with account details will be push to rabbitmq queue “ACCOUNT_CREATION”.
- Request Type : POST request
- Request URL : http://localhost:9093/tuum/v1/accountsHandler/accounts
- Request Body : all the attributes are mandatory.

```
{
        "customerId":"TUUMCUSTOMER",
        "currencyTypeList": ["EUR","SEK","GBP"],
        "country": "Estonia”
}
```

- Response :

```
{
    "accountId": "062895bb-0cbb-4efa-8baa-1ac074352516",
    "customerId": "TUUMCUSTOMER",
    "country": "ESTONIA",
    "accountBalanceList": [
        {
            "id": 34,
            "accountId": "062895bb-0cbb-4efa-8baa-1ac074352516",
            "currencyType": "EUR",
            "balance": 0.00
        },
        {
            "id": 35,
            "accountId": "062895bb-0cbb-4efa-8baa-1ac074352516",
            "currencyType": "SEK",
            "balance": 0.00
        },
        {
            "id": 36,
            "accountId": "062895bb-0cbb-4efa-8baa-1ac074352516",
            "currencyType": "GBP",
            "balance": 0.00
        }
    ]
}
```

- Publish account creation event : Publish Account Creation to RabbitMQ ACCOUNT_CREATION queue



Message 1

The server reported 0 messages remaining.

Exchange	(AMQP default)
Routing Key	ACCOUNT_CREATION
Redelivered	○
Properties	priority: 0 delivery_mode: 2 headers: content_encoding: UTF-8 content_type: text/plain
Payload 433 bytes Encoding: string	Account{accountId='b3d47f7b-1096-4ebf-9be5-3099d00aaf7b', customerId='TUUMCUSTOMER', country='ESTONIA', accountBalanceList=[AccountBalance{id=43, accountId=b3d47f7b-1096-4ebf-9t

2. Get Account

- Api: Get the account with account balance list by account id
- Request Type : GET request
- Request URL: http://localhost:9093/tuum/v1/accountsHandler/e9678c4c-2971-4880-ac56-b884a2e95b93
- Request Parameter : Need to send account id as path variable
- Response:

```
{
    "accountId": "062895bb-0cbb-4efa-8baa-1ac074352516",
    "customerId": "TUUMCUSTOMER",
    "country": "ESTONIA",
    "accountBalanceList": [
        {
            "id": 34,
            "accountId": "062895bb-0cbb-4efa-8baa-1ac074352516",
            "currencyType": "EUR",
            "balance": 0.00
        },
        {
            "id": 35,
            "accountId": "062895bb-0cbb-4efa-8baa-1ac074352516",
            "currencyType": "SEK",
            "balance": 0.00
        },
        {
            "id": 36,
            "accountId": "062895bb-0cbb-4efa-8baa-1ac074352516",
            "currencyType": "GBP",
            "balance": 0.00
        }
    ]
}
```

3. Create transaction

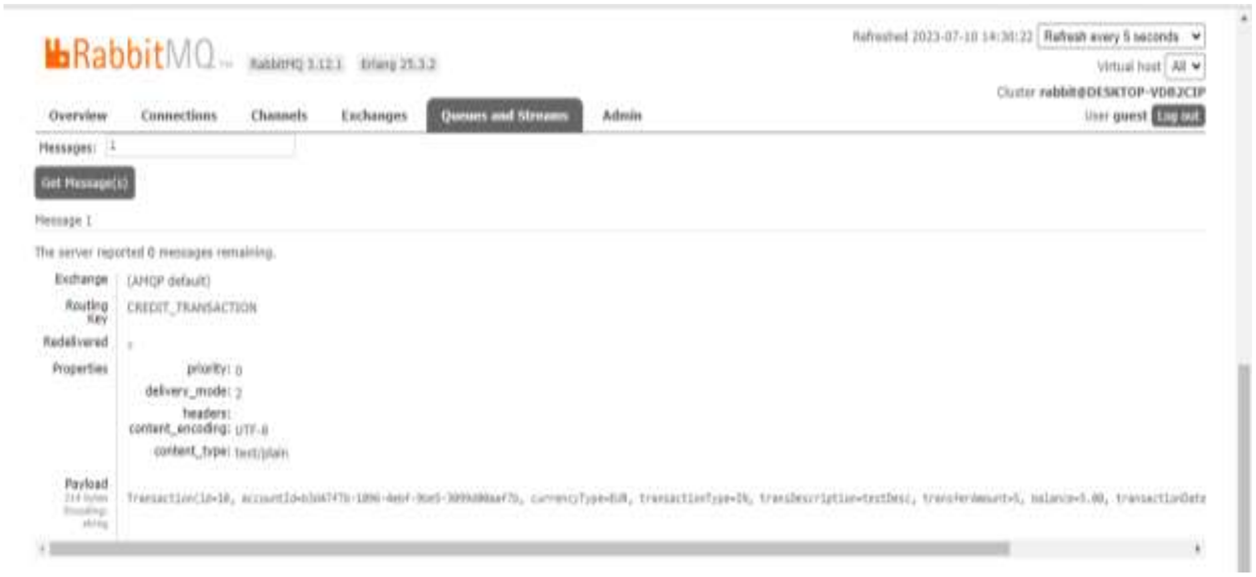
- Api: Create transaction and update the account balance depend on transaction type is IN or OUT. After creating the transaction, transaction creation notification with transaction details will be push to rabbitmq queue “CREDIT_TRANSACTION” or “DEBIT_TRANSACTION” depend on the transaction type.
- Request Type : POST request
- Request URL: http://localhost:9093/tuum/v1/transactionHandler/transactions
- Request Body : all the attributes are mandatory.

```
{
"accountId":"98ca4038-d485-4108-b2f9-f9d80ce0cc14",
"currencyType": "EUR",
"transactionType": "IN",
"transDescription": "testDesc",
"amount" : 5
}
```

- Response

```
{
    "id": 11,
    "accountId": "b3d47f7b-1096-4ebf-9be5-3099d00aaf7b",
    "currencyType": "EUR",
    "transactionType": "IN",
    "transDescription": "testDesc",
    "transferAmount": 5,
    "balance": 5.00,
    "transactionDatetime": "2023-07-10T14:31:05.6225104"
}
```

- Publish transaction creation event : Publish Transaction Creation to RabbitMQ CREDIT_TRANSACTION queue



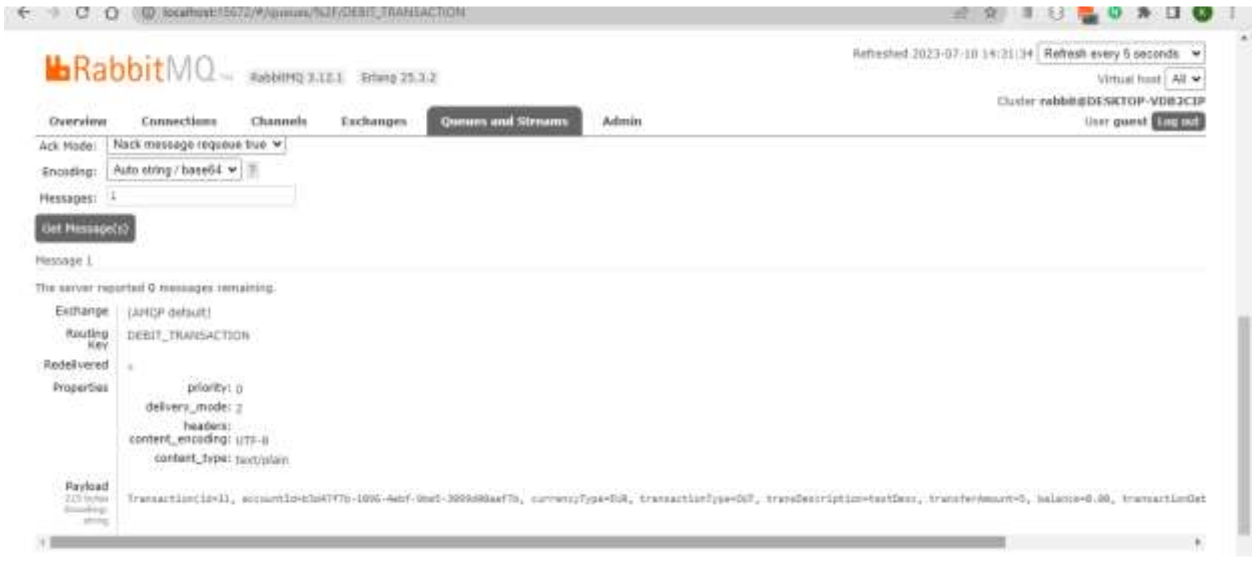
Messages: 1

Message 1

The server reported 0 messages remaining.

Exchange	(AMQP default)
Routing Key	CREDIT_TRANSACTION
Redelivered	○
Properties	priority: 0 delivery_mode: 2 headers: content_encoding: UTF-8 content_type: text/plain
Payload 214 bytes Encoding: string	Transaction(id=10, accountId=b3d47f7b-1096-4ebf-9be5-3099d00aaf7b, currencyType=EUR, transactionType=IN, transDescription=testDesc, transferAmount=5, balance=5.00, transactionDate

- Publish transaction creation event : Publish Transaction Creation to RabbitMQ DEBIT_TRANSACTION queue



Refreshed 2023-07-10 14:21:34 Refresh every 5 seconds

Virtual host All

Cluster rabbit@DESKTOP-VDB2CIP

User guest

Ack Mode: Nack message requeue true

Encoding: Auto string / base64

Messages: 1

Message 1

The server reported 0 messages remaining.

Exchange: (AMQP default)

Routing Key: DEBIT_TRANSACTION

Redelivered: ○

Properties:
priority: 0
delivery_mode: 2
headers:
content_encoding: UTF-8
content_type: text/plain

Payload (215 bytes, Encoding: string):

```
Transaction(id=11, accountId=b3d47f7b-1096-4ebf-9be5-3099d00aaf7b, currencyType=EUR, transactionType=OUT, transDescription=testDesc, transferAmount=5, balance=0.00, transactionDat
```

4. **Get transaction**
   - Api: Get the transaction list by account id
   - Request Type : GET request
   - Request URL: http://localhost:9093/tuum/v1/transactionHandler/transactions/52b6338a-3f13-45ea-b735-4fa88a9b7cdb
   - Request Parameter : Need to send account id as path variable
   - Response:

```
[
    {
        "id": 10,
        "accountId": "b3d47f7b-1096-4ebf-9be5-3099d00aaf7b",
        "currencyType": "EUR",
        "transactionType": "IN",
        "transDescription": "testDesc",
        "transferAmount": 5.00,
        "balance": 5.00,
        "transactionDatetime": null
    },
    {
        "id": 11,
        "accountId": "b3d47f7b-1096-4ebf-9be5-3099d00aaf7b",
        "currencyType": "EUR",
        "transactionType": "OUT",
        "transDescription": "testDesc",
        "transferAmount": 5.00,
        "balance": 0.00,
        "transactionDatetime": null
    }
]
```

5. **Important Choices taken**

➢ Design Architecture

Used layered architecture with controller, service and repository modules. Bank application is built as monolithic application.

➢ Data Model

Create separate entity Account, AccountBalance and Transaction to hold application data.
If we save the account and the account balance in the same table it will be difficult to maintain the account balance and currency. On the other hand it will be difficult to update account detail when new currency type is introduced.

Create messaging module as separate module to minimize the dependency with other module.

➢ Validation and Error Handling:

Implement input validation for account and transaction data to ensure data integrity at the entity level.
Utilize spring's validation framework, custom validators, and exception handling mechanisms. Implement ControllerAdvice to Handle and report errors gracefully to users by providing meaningful error messages and appropriate HTTP status codes.

➢ Use spring's declarative transaction management to avoid data inconsistency when adding, updating account and transactions.

➢ API Design:
Design a clear and consistent API for account and transaction operations.
Follow RESTful principles and naming conventions for endpoints and HTTP methods.
Use appropriate request/response structures (e.g., JSON) and adhere to RESTful API best practices.
Valid all the inputs at the controller level before processing the request.

➢ Write unit test and integration testing for controller, service and repository levels.
➢ Implement logging to capture important events and errors for troubleshooting and auditing purposes.
➢ RabbitMQ message

Exchange: Used default exchange since I want to send the message to only specific queue based on the queue name
Queues: Used separate ques for Account creation, Debit transactions and Credit transactions

6. How to scale application horizontally

- Having multiple application nodes with the Load balancer

  Having multiple instances of the application will reduce the incoming traffic to one node. By implementing load balancer, incoming traffic can be distribute among nodes.

- Database replication

  Having more than one database instance will reduce the load on one database node and also helps to avoid single point of failure. But this solution comes with the cost of synchronizing data between the database instances. Using Active passive database instance will reduce the overhead of data synchronization but it does not reduce the load on one database server.

- Database Sharding

  Since there will be lot of transaction happened through different Accounts, it will be wise to use database Sharding to save transaction depend on its created data. By doing this it will reduce the overhead when querying the recent transactions.

- Stateless application

  If the application is not maintaining any session it will be easy to scale application horizontally since it is not needed to synchronize session details among nodes. Implementing JWT based authentication and authorization mechanisms helps to maintain stateless application.

- Having Microservice based architecture

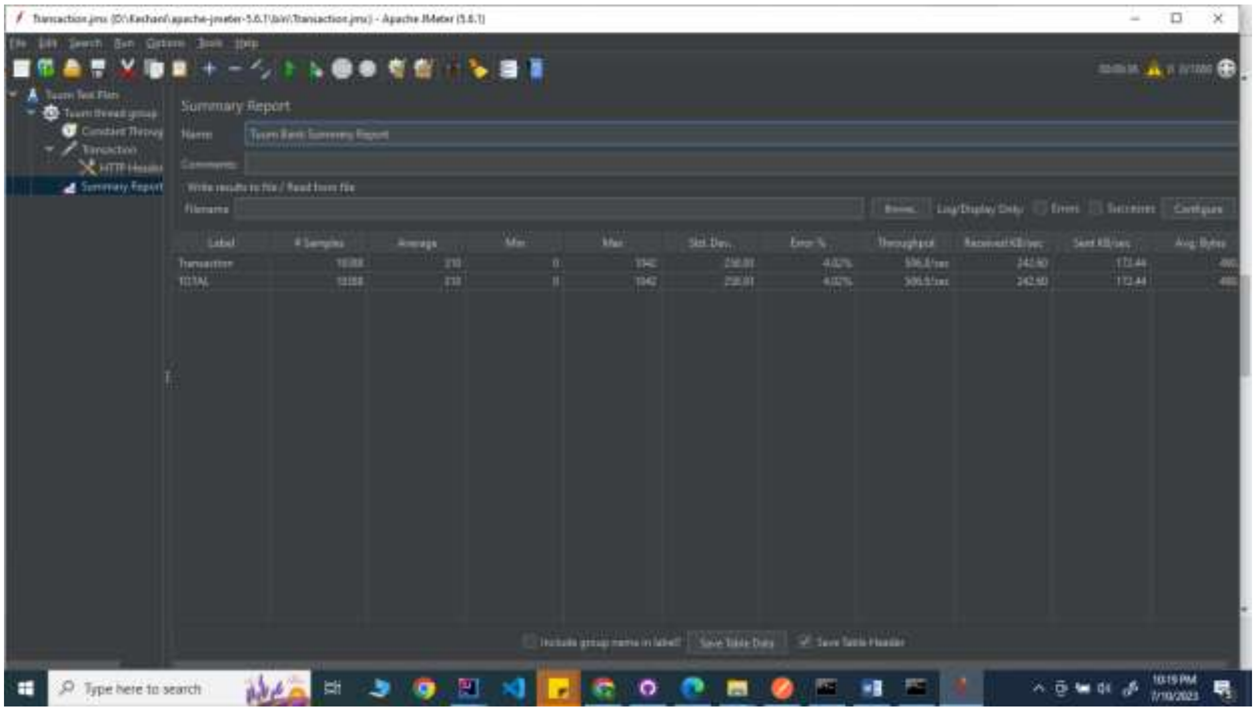
  It will be easy to horizontally scale application if it is microservice based application. Based on the traffic for each microsrvice, it can be decided how many instance each microservice should have. As example if we have separate microservice to handle account and transaction, we can easily horizontally scale each microservice separately.

- Implementing Caching

  It is better to have separate Caching mechanism not bind to the application like spring cache. Having separate cache like Redis will give us feasibility to make cache cluster outside the application so that it will be easy to synchronize among that cluster. Since the Caching cluster module and application nodes are separate modules, scaling up or down one module does not impact on the other module.

7. **Number of Transactions Per Second**

Number of transactions that system can handle (throughput) = 506.8/sec



Transaction.jmx (D:\Keshani\apache-jmeter-5.6.1\bin\Transaction.jmx) - Apache JMeter (5.6.1)

Summary Report

Name: Tuum Bank Summary Report

Comments:

Write results to file / Read from file

Filename

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Transaction	10388	210	0	1042	230.01	4.02%	506.8/sec	242.60	172.44	[illegible]
TOTAL	10388	210	0	1042	230.01	4.02%	506.8/sec	242.60	172.44	[illegible]

8. **Failed To Implement**

- Create RabbitMQ queus when it start through docker compose :

  __Approach Taken:__
  Create queue by defining the queue names, exchange and bindings in rabbitmq_def.json file
  Create rabbitmq.conf file referring newly created rabbitmq_def.json.
  Synch above files to rabbitmq docker container through volumes.

  __Result:__
  Since this approached did not create queue in the rabbitmq, had to create it manually through the rabbitmq management.