Problem Specification

The MiniBank online system (MBS) is a small bank for personal and small business banking. The banking system enables customer to access their bank account and perform their everyday banking needs. MBS has many customers, and each customer has one or more accounts. The MBS use the Euro as currency. The smallest unit of the currency is the single Euro. There are no cents and all transactions resolve evenly.

When customer open an account, they provide an information pack which contain unique email id, personal code, and password information for login into the banking system. The bank certifies the information pack and approve the customer's account. After that, Customers can view their account information, perform banking operations online, such as displaying the balance of an account or transferring money. The customers can also interact with the MBS to perform common transactions such as performing withdrawals, deposit and transferring money.

The MBS has requested a complete robust online banking system to enable customers to bank over the Internet and to automate the operations. The system architecture must be RESTful and scalable enough to grow as the number of customers increases and add more modules. Customers must be able to access the online banking system by using any secure browser, mobile apps and the system must be platform independent.

The Big picture of the problem Specification

After analyzing the problem description, using use case method we showed the required functionalities for the MiniBank system (MBS).

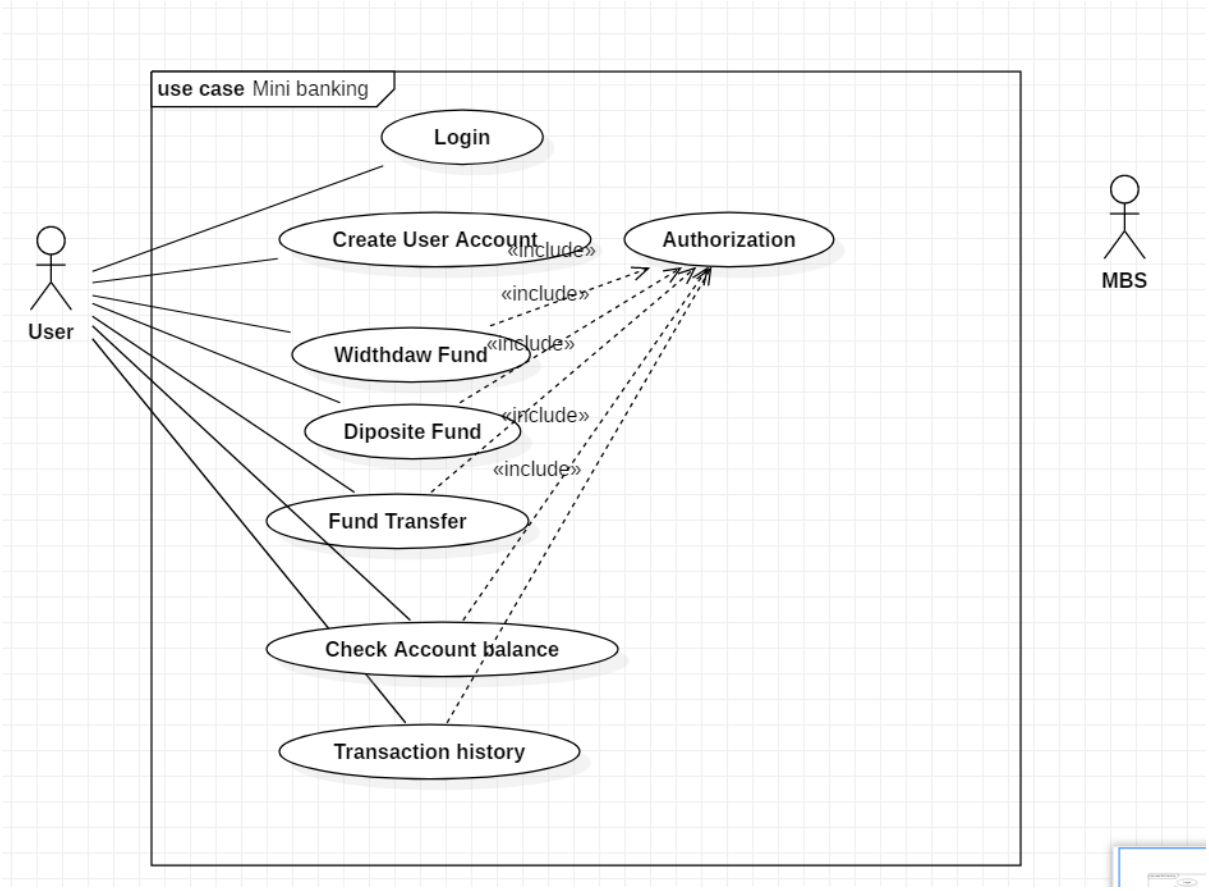

Figure 1: Use case Diagram of MBS

Requirements Details

User Stories:

Register:

As a user, I want to register, so that I can have a personalized banking experience with the service.

Deposit:
As a user, I want to deposit the fund into my account, so that I can experience the banking service

Create login:
As a User, I want to login account into the service, so that I can see my account information

Fund Transfer:
As a User I want transfer fund from my account, so that I can experience the banking service

Check Account balance:
As a User, I want to check account balance into the system, so that I can see my account balance

Transaction History:

As a User, I want to check the transaction history, so that I can observe my transaction at least for 30 days.

Use Cases

Registering

For opening an account, user must perform the following steps. As we know one user can have multiple account.

Actors	Customer
Preconditions	The Customer has interest in using MBS
Goal	Gain access to MBS Account and offering
Step 1	The Customer inserted their personal Information
Step2	The Customer enters their identification document

Exception	The User has an incomplete information
Exception	The User did not provide sufficient identification documents
Step 4	The bank accepted identification proof documents
Exception	The bank did not accept the Identity proof document
Postconditions	The Customer's account is created and personalized

Table 1: Use case for Register a User

View Payments

For Viewing a payment made by the user earlier, the user must perform the following steps.

Actors	Customer
Preconditions	The Customer has interest in viewing the payment history
Goal	Viewing the transactions
Step 1	The Customer inserted their authentication Information
Exception	user provided info is not valid
Step 2	The customer logged in to the system dashboard
Step 3	The customer chooses to see transaction history
step 4	the customer wants to see history between specific date difference
Exception	date value is not correct
step 5	showed a list of transaction made from customer account
post condition	The payment made from the account has been shown

Table 2: Use case for View a Payment

Create Transfer between Two Account

As MBS is a completely online banking system, so each transfer must have one source account and one destination account, the transaction will perform one credit and one debit transaction.

Actors	Customer
Preconditions	The Customer has interest in using Transfer service
Goal	Transfer Money between two account
Step 1	The Customer inserted identification Id and password
exception	The inserted identity is not valid
Step2	The Customer enters the dashboard and show account information
Step 3	The customer selects money transfer menu
Step 4	The customer sees a Transaction form to fill up
Step 5	The customer fills up the form and submit
exception	The recipient account is not valid
exception	The balance is not enough
Step 6	The customers showed a confirmation page with edit options
exception	The customer doesn't like any of the options and decides not to proceed the transaction any further.
Step 7	Display transaction success notification message
Postconditions	The Transaction between two account has been made

Table 3: Use case for Create Transaction

Functional Requirements

ID	Description
OF01	Create account and necessary pack of user information
OF02	The balance amount of the account should be shown
OF03	Transfer the Money between two account
OF04	Withdrawal and deposit money to the account
OF05	Checking the transaction

Table 4: Functional Requirements

Nonfunctional Requirements

ID	Description
NF01	Usability: Newbies can register in less than 90s
NF02	Usability: Newbies can find transaction history at least for 30 days in less than 30s
NF03	Usability: Newbies can filter the transaction in less than 20s with a smartphone or any other device
NF04	Performance: The service can fulfil at least 10k queries per hour
NF05	Performance: No system action should take more than 10s
NF06	Extensibility: Third parties can extend the service via a public API
NF07	Security: Users information and statistics are protected by default

Table 5: Nonfunctional Requirements

Domain Modelling

Concept Description

Concept	Description
bank	a financial establishment that uses money
online system	Computer or device connected to a network and provide service over it
Customer	Users who are using the system
bank Account	An Account which on bank for money transaction
Euro	single European currency,
Transaction	an instance of buying or selling something.
Account Open	Customer
information	Data
information pack	A bunch on specific data
unique ID	a unique identifier is any identifier which is guaranteed to be unique among all identifiers
Password	a secret word or phrase
Login	
Account Information	Specific Data
Banking Operation	The legal transactions executed by a bank in its daily business
Display Balance	Show Balance on Currency
Transfer Money	Transfer Money from one account to another
Automate Operation	
Scalable	It can be Intendable
Secure Browser	protect from breaches of privacy or malware.
Mobile App	An Application Runs of mobile devices
Platform Independent	An application Runs on different Platform

Domain Model

By using Use case diagrams info and collecting key concepts here a domain model has been developed. the domain model reflects a deeper analysis of the problem domain instead of simply stating requirements like use cases. All the concepts in the final domain model are also found in the use case diagrams.

According to use case diagram, the functionalities to be implemented: Login, create account, withdrawal, deposit, check account balance, fund transfer to another account, and see transaction history at least for 30 days. MBS is completely online, so all the transaction will be online and will be limited to transferring money between account to account only. SMB does not have any interaction with other resources like: physical resources(ATM), or other online third parties, that's why all the basic transactions like withdrawal, and deposit will be performed with simple monetary transaction between account to account.

Domain modeling made to think through the roles of different users, but for the simplicity of the application requirements we will strict to the end user role only. Each user has one or more account and each monetary transaction occurs between two accounts have two mandatory steps, withdrawal from source and deposit to destination account. For an example Mr. X, send 100 euro to Mr. Y's account. The transaction will be made by following

2) debit the amount from the Mr. X's bank account (withdrawal).
3) credit the amount to the Mr. Y's bank account (deposit)

In domain model, some new relationships have been made between the concepts that were not clearly visible when doing the requirements engineering work.

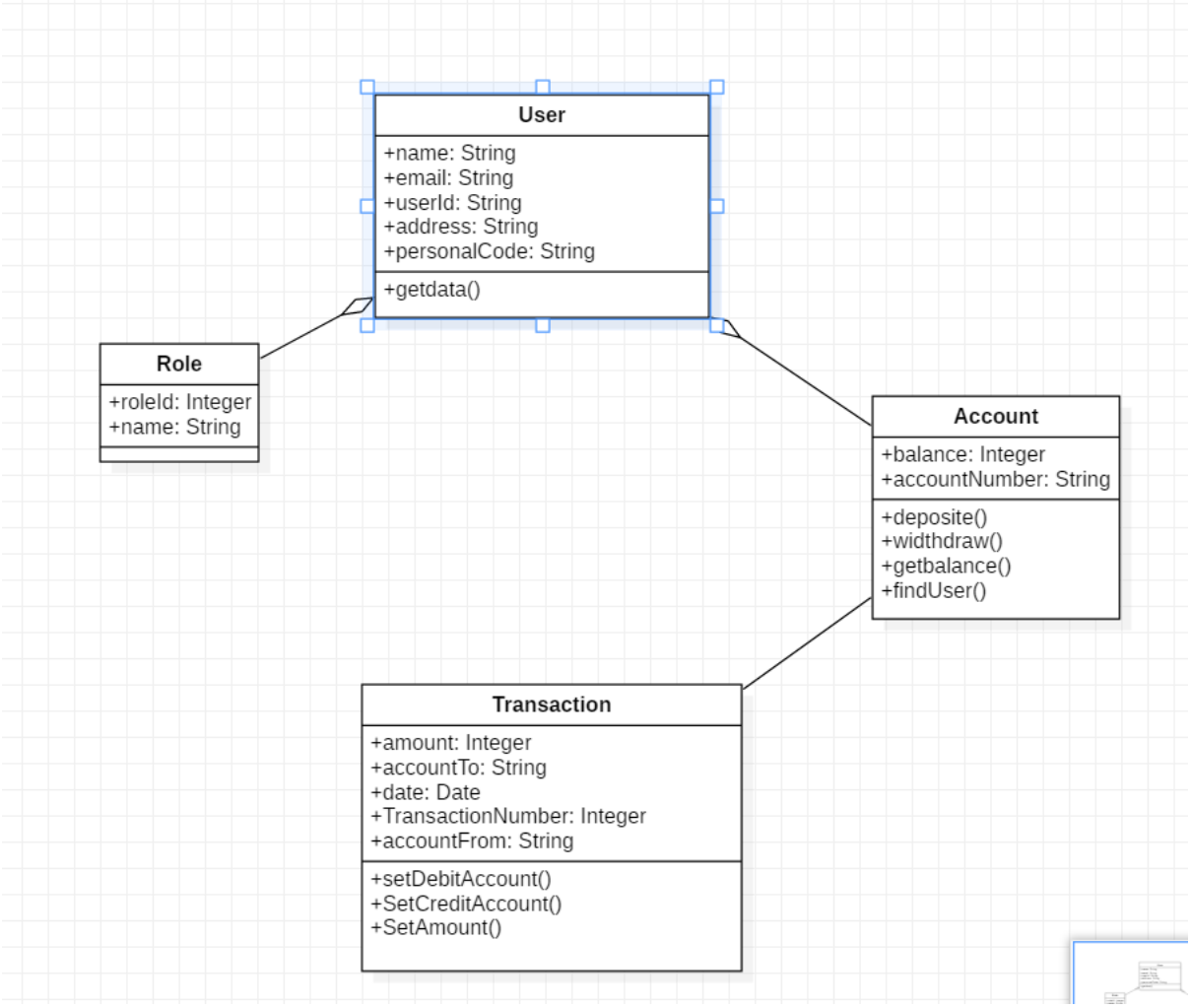

Figure 2: Domain Model of MBS

ER Diagram

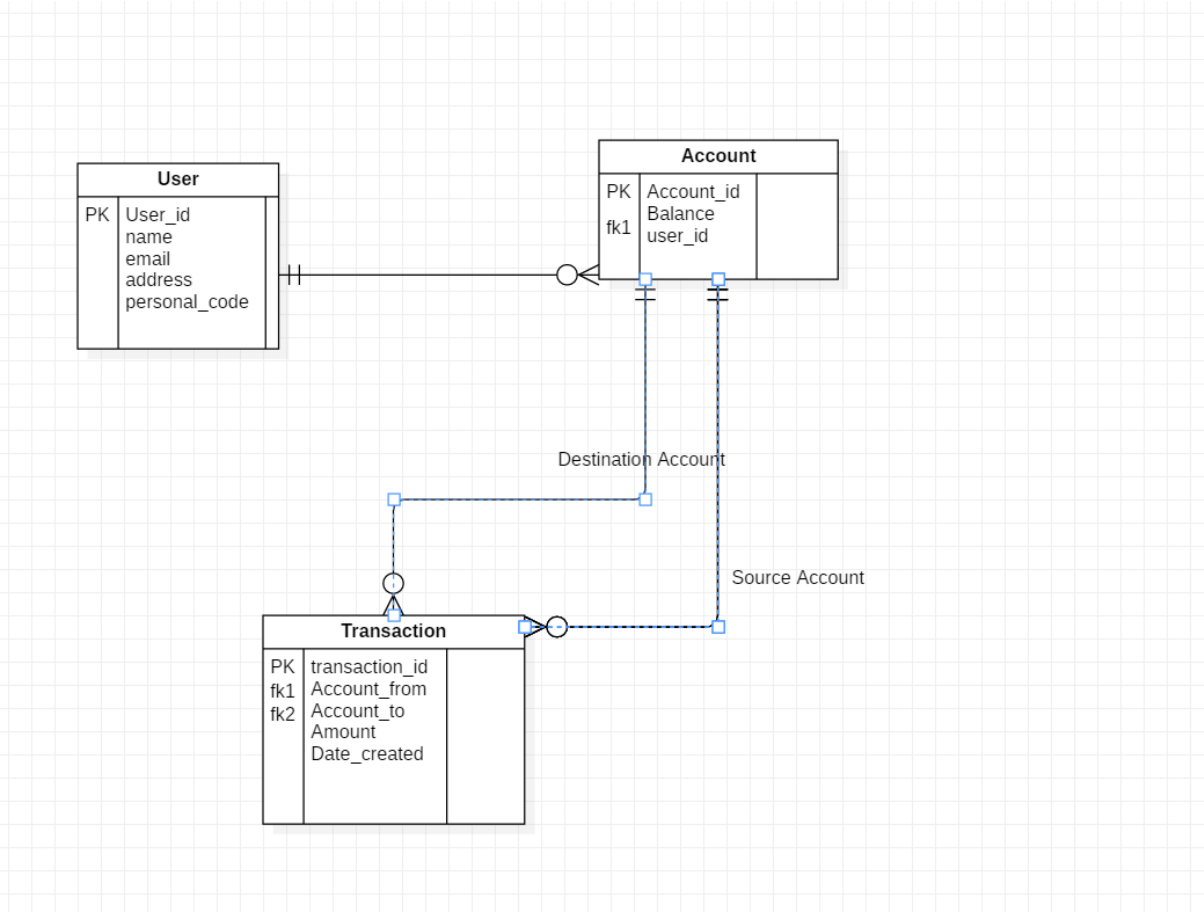

Figure 3: ERD of MBS

Account entity is uniquely identified by the account_id primary key attribute. It has a required balance attribute (because an account must have a balance) and an optional overdraft_limit (because not all accounts have an overdraft facility). Th Account has a user_id foreign key attribute that identifies the account user via the relationship to the User entity.

Two Account entities can participate in a funds transfer represented by the Transaction entity. A Funds Transfer must link to two accounts which take the roles of the source account and destination account, but any given account can participate in any number of funds transfers.

Transaction entity is uniquely identified by the transaction_id primary key attribute. It has a required amount attribute (because every funds transfer must be of an amount) and a required datetime attribute (because each transfer must be timestamped). The account_to and account_from foreign key attributes identify the two Bank Accounts of the two relationships.

A User is uniquely identified by the user_id primary key attribute, and each customer has a required name. A customer has few pairs of attributes labeled as name, addresses, phone, email for the customer as indicated by the relationships.

Architecturally Significant Requirements

scalability, usability and security as our most important architecturally significant requirements here with the SMB system. It is also considered that from an architectural viewpoint, we must consider that third parties can also use this API on their application.

Scalability is necessary for SMB as aspire to be; a service that can be used by thousands of users daily at the same time. This is something that must be considered with regards to the architecture from the very beginning of the design and implementation process. Appropriate modularity - keeping different components small and separate enough - can help make the software more easily scalable.

Confidential and personal data, such as transaction history, email addresses and personal account info, must be stored and used securely. in order to reach an effective solution, security must be considered from the very beginning of the system architecture design.

Significant Design Decisions and RESTful Architecture

According to MBS requirements, we chose a RESTful architecture because it allows loosely coupled components that are easy to understand, highly scalable and are separately testable. The architecture may also be driven by current trends and best practices. REST is a popular architectural style at the moment, and thus, can be easily adopted by any third-party client.

From the domain model and use case diagram we understood the user requirements clearly. We understood each user wishes to create account, register themselves into the system and can have one or more accounts. Though the account the user wishes to use the services those are provided by the MBS system. User wishes to transfer the money from one account to another account. User wishes to watch the transaction history that they made over the time on the system.

As user wishes to create account and login into the MBS system for using the services, so user authentication is very important here. We will use the stateless authentication methods for this purpose. Each request from the client system to server must contain all of the information necessary to understand the request. For that purposes, we will use token-based authentication, Token Authentication have a "sign in" URL that accepts a POST request with username and password and responds with a token. A token is simply a string, usually randomly or cryptographically generated and store into the database. From the token itself, servers can extract the user info, permissions, etc. With this capability, other services can operate normally when the user authentication is needed.

Transfer money will go for two transaction process, one it will debit the amount from the source account and credit to the destination account. For the purpose of showing transaction history for an account, the system will show the transaction made from the account on the basis of date difference. By default, it will show 30 days transaction history once the user chooses to see the transaction history.

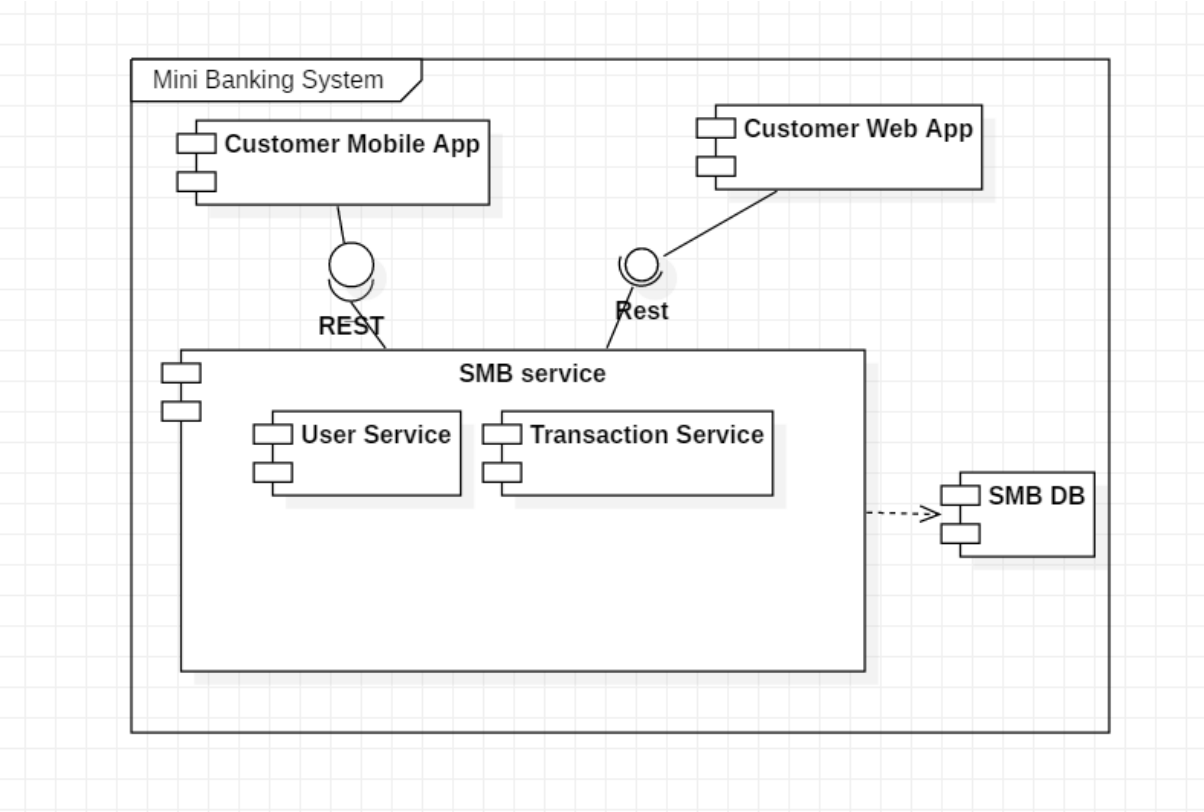

Figure 4: Component diagram of MBS

RESTful API is the restricted for set of verbs that HTTP provides. We can only use the documented HTTP methods: GET, POST, PUT, PATCH, DELETE, etc. As we can see, we don't have a "create User" verb, even though at first glance our application requires it. According to domain model and requirement analysis, we will use few verbs for interacting with the service from the frontend. for an example:

CreateUser (customer open their account with required information)
getUser(single user account info)
getUsers(return all the user account)
getBalance (return Balance of the specific account)
getTransaction (specific single transaction return)
transaction (accept single transaction)
getTransactions (return all the transaction and also accept date difference parameter)

Creating User:
Request: POST /createuser/
Request body:

```
{
   "name": "Ashiq Ahmed",
   "email": "ashiqahmed005@gmail.com",
   "personal_code": 10000000001
}
```

Response:
HTTP/1.1 201 Created
Content-Type: application/json

```
[{
   "name": "Ashiq Ahmed",
   "email": "ashiqahmed005@gmail.com",
   "personal_code": 10000000001,
  “ token”:  "5dac4af44f7da18f802cfae8685c00a3de73755b"
}
]
```


Getting User Info:

Request: GET /getUser/00003
Request body:

```
{}
```

Response:
HTTP/1.1 200 OK
Content-Type: application/json

```
[{
   "name": "Ashiq Ahmed",
   "email": "ashiqahmed005@gmail.com",
   "personal_code": 10000000001
}
]
```


Transfer example

Single transaction

Request: GET /transaction/00001
Request body: (empty)
Response body:
HTTP/1.1 200 OK
Content-Type: application/json

[{
"id": 00001,
"account_from": 00000001,
"account_to": 00000002,
"amount": 100,
"create_date": "2017-06-08T19:30:39+00:00",

}]

List of Transactions:

Request: GET /transactions/
Request body: (empty)
Response body:
HTTP/1.1 200 OK
Content-Type: application/json

[{
"id": 00001,
"account_from": 00000001,
"account_to": 00000002,
"amount": 100,
"create_date": "2017-06-08T19:30:39+00:00"

},
{
"id": 00002,
"account_from": 00000001,
"account_to": 00000002,
"amount": 100,
"create_date": "2017-06-08T19:30:39+00:00"

},
{
"id": 00001,
"account_from": 00000001,
"account_to": 00000002,
"amount": 100,
"create_date": "2017-06-08T19:30:39+00:00”

}
]

Workflow of Transferring Money

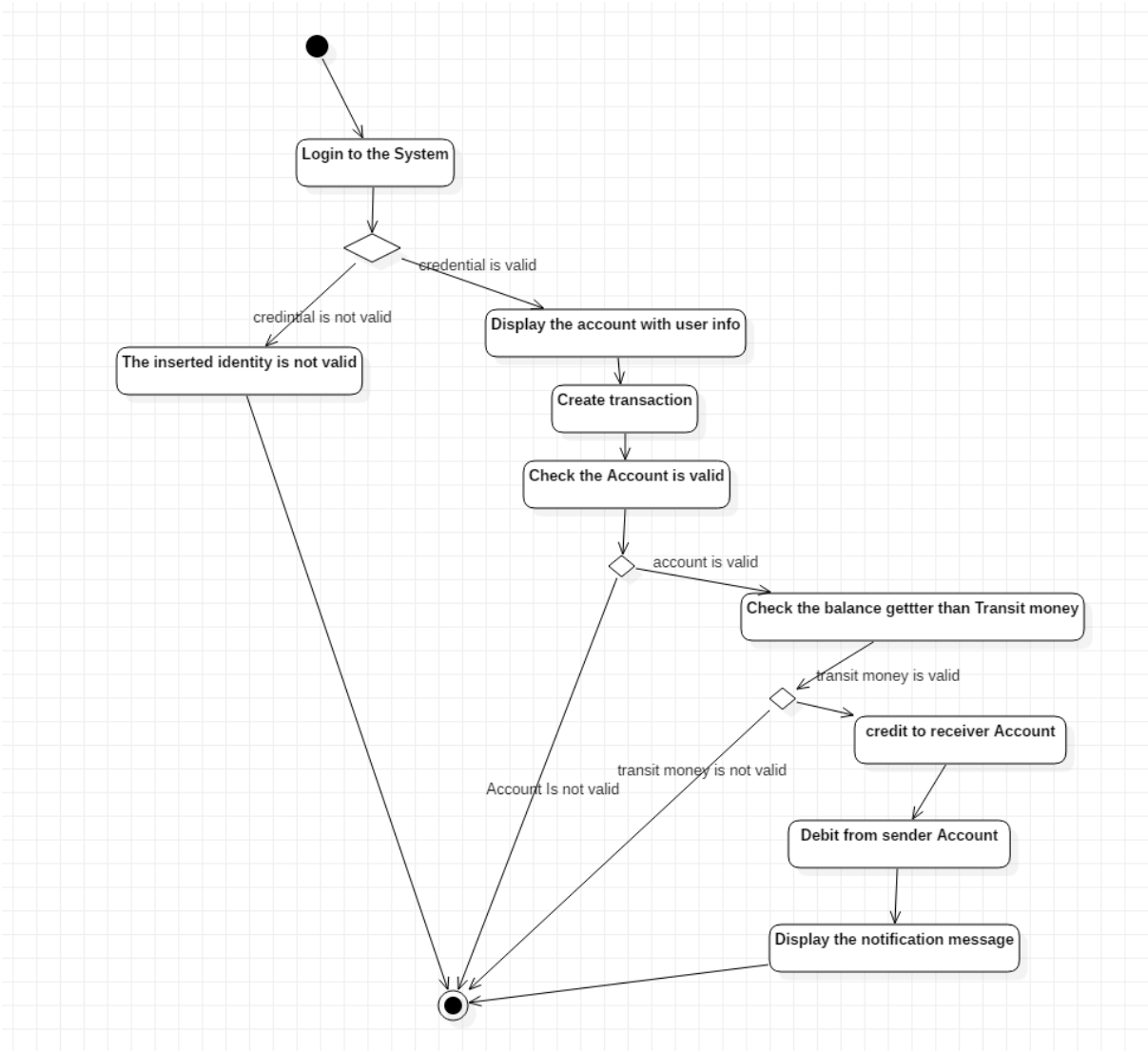

Figure 5: Activity Diagram of Transfer money between Two Account

The activity diagram shows the workflow of money transfer between accounts. The user login into the system with username and password. The bearer auth token verifies the user authentication and after being verified the user get into the dashboard of the MBS system. Then user will choose create transaction menu for creating a transaction. The menu will take the user to a fillable form and user will fill it and submit. After being submitted, the recipient account validity will be checked, and transit amount will be checked in respect to account balance. If it is not valid the process will end with a 404 error, otherwise it will proceed further, and the sender account will be debited, and receiver account will be credited with the transmitted amount. Finally, the user will get a successful notification.

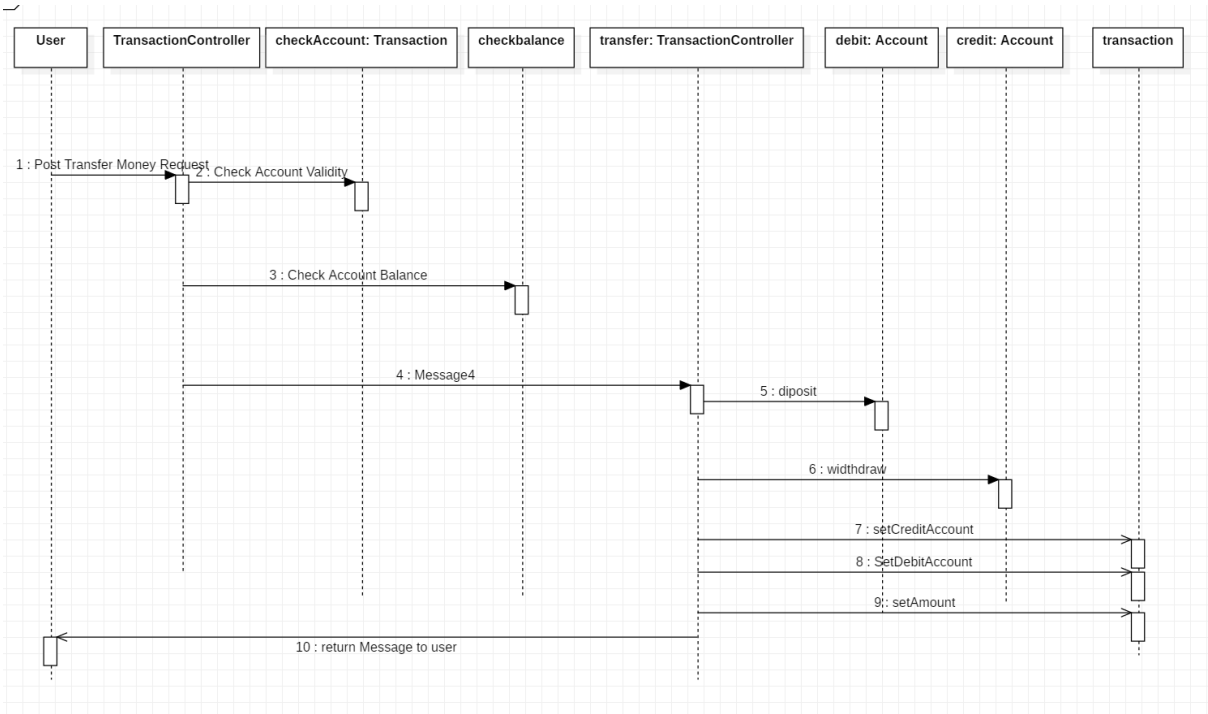

Figure 6: sequence diagram of Transfer money between Two Account

The diagram shows the sequence of steps that occurs when the customer clicks Transfer Money on the main menu of the online user interface. When the customer clicks Transfer Money, a form is displayed on the screen. The customer then completes the form. The form data is sent to the TransactionController class, which sends an async message to account class for finding the recipient user account, and balance of the sender account. The TransferMoney class then sends the deposit transaction information to the credit:Account class. After that it send similar async information with the withdrawal transaction information to the widthdraw:Account class. The TransferMoneyController class creates the transfer object and passes it the widthdraw, diposit, and amount information so that it can perform the transaction. A message is then displayed to the user. In any case, any of the operation failed the whole process starts from the beginning and database will rollback.

The Transfer Money sequence diagram completes the Transfer Money use-case realization which states static and dynamic information of the Transfer Money functional task. In Transfer Money Participants diagram, the diagram contains the operations that needed in the sequence diagram. The Transfer Money Participants diagram should look similar to the following figure:

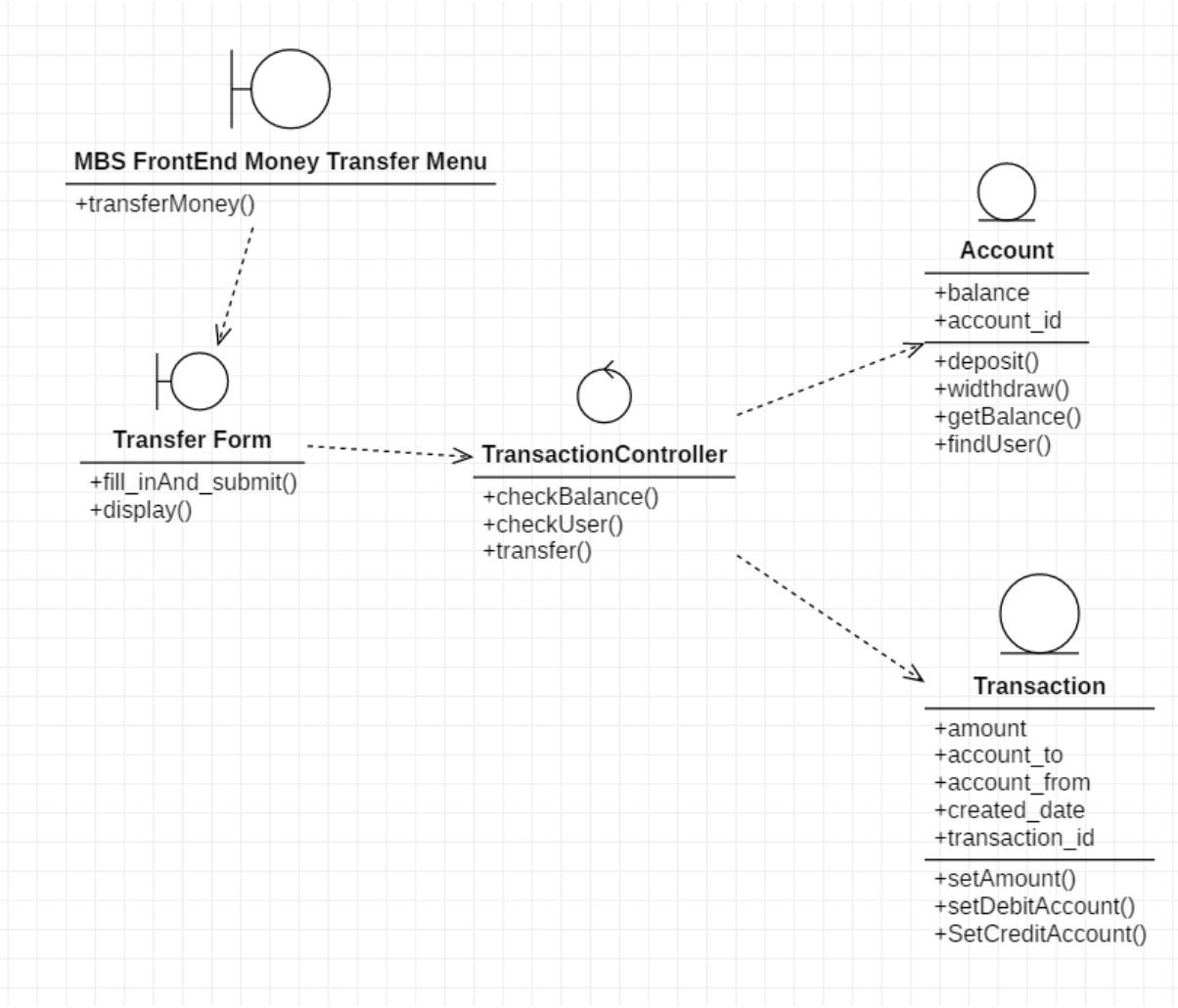

Figure 7: Transfer Money Participants diagram

Testing

Testing the user stories

All these user stories also need to be tested to ensure that they are correct, and they apply to MBS service. User stories state the value, that will be achieved with the action described. In the tests, the action will be tested with different variables to check, if the outcome is realized in every situation.

All the user stories have a story card, that describes the function. For the first user story, the

Register:
As a user, I want to register, so that I can have a personalized banking experience with the service.

description is 'Users need to register to the service to submit personal information'. This will be tested with three different variables:

1. Try to submit personal information without registering
2. Try to register with incorrect email address
3. Try to register with email address that is already registered

Deposit:
As a user, I want to deposit the fund into my account, so that I can experience the banking service

description is 'Users need to deposit money to the account'. This will
be tested with few different steps, and in online system transaction need to make between two account.

1. Try to submit without registering
2. Try to deposit without account number
3. Try to deposit without amount
4. Try to deposit with same account

Create login:
As a User, I want to login account into the service, so that I can see my account information

description is 'Users need to login into the system'. This will
be tested with few different steps, and in online system transaction need to make between two account.

1. Try to submit login info without registering
2. Try to submit incorrect password
3. Try to submit incorrect email id
4. Try to submit without complete info

Fund Transfer:
As a User I want transfer fund from my account, so that I can experience the banking service

description is ‘Users need to transfer money to another account’. This will
be tested with few different steps, and in online system transaction need to make between two account.

1. Try to submit without registering
2. Try to submit without account number
3. Try to submit without destination account number
4. Try to submit with same account number
5. Try to submit without amount
6. Try to submit unregistered account number for both source and destination account
7. Try to submit string in amount

Check Account balance:

As a User, I want to check account balance into the system, so that I can see my account balance

description is ‘Users need to check his account balance. This will
be tested with few different steps, and in online system transaction need to make between two account.

1. Try to check without registering
2. Try to check without logged in
3. Try to check previously stored auth token
4. Try to check directly through URL

Transaction History:

As a User, I want to check the transaction history, so that I can observe my transaction at least for 30 days.

description is ‘Users need to check his transaction history. This will

be tested with few different steps, and in online system transaction need to make between two account.

1. Try to check without registering
2. Try to check without logged in
3. Try to check with other account info
4. Try to check previously stored auth token
5. Try to check directly through url

Testing use case

Functional testing

Functionalities to test:

- Create a User
- Create a money transfer
- Show all the deposited amount by a specific user
- Show all User list.
- Show all transactions sorted by date.
- Show single transaction.

For each function, the correct information is either delivered to the user or not. Identifying whether the MBS system can deliver information at all is easy, but determining whether the delivered or shown information is correct is a much more challenging task. We propose simply hard coding or working out the expected information before testing each test case such that invalid functioning is easy to notice.

Stress testing

Stress testing is about identifying components, subsystems, resources and data inputs in order to design tests for them.

In this use case such items are:

- Subsystems: HTTP server, database.
- Constraints: poor network connection.
- Data: user, account, transaction related data; JS libraries.
- Resources: network speed, client device resources, server resources.

Based on analysis we propose the following stress test cases:

- Lots of simultaneous user requests.
- Lots of data in database.
- Data being changed at the same time data is retrieved.

- Normal use in bad network.
- No data available in database.
- Low resources on the client device.
- Low resources on the server hardware.