Banking API Documentation

Berat Asrın CAFEROĞLU

Contents

1. Technologies Used in Development

In this project, **Java Programming Language** is utilized with **Spring Boot Framework** in order to design the API. On the other hand, as you know the data should be stored using some database, so that, **MongoDB** is preferred in terms of ease of use, and **Spring Data** is used to read/create/update/delete the data. Also, encapsulation is an essential topic in object-oriented programming, yet developers struggle while coding constructors, getters, and setters. In that sense, **Lombok** is used to create them using annotations.

2. User Side

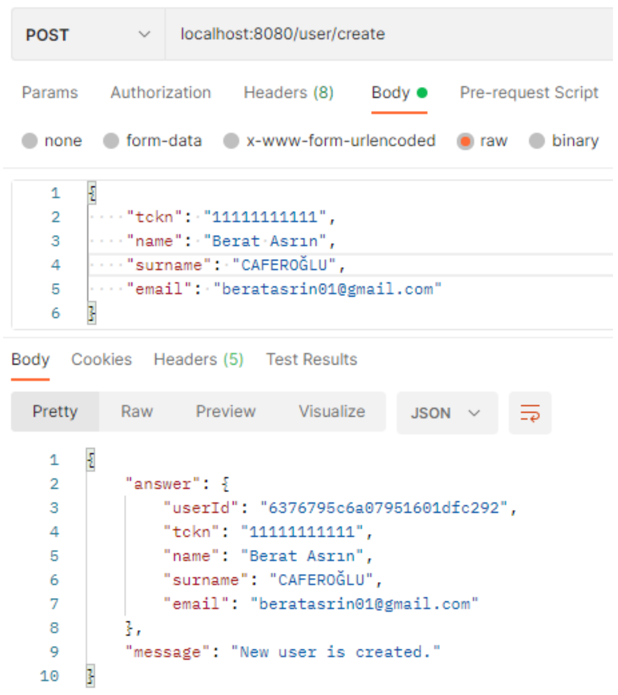
2.1. Create User

Each user has their own properties as follows: name, surname, Turkish identification number (TCKN), and email. So, in process of creating a new user, we need to get these parameters in the request. In this function, there are important points that I would like to mention. First, before creating a new user we are searching for TCKN to find out that if any user has the same identification number. In case of conflict, API throws a message and does not create the user. Otherwise, I mean if the TCKN does not exist in the MongoDB database, the user is created successfully, and a unique primary key is assigned to the user.

URL: localhost:8080/user/create

HTTP Method: POST

Required Body Parameters: tckn (String), name (String), surname (String), email (String)


POST localhost:8080/user/create

Params Authorization Headers (8) Body Pre-request Script

none form-data x-www-form-urlencoded raw binary

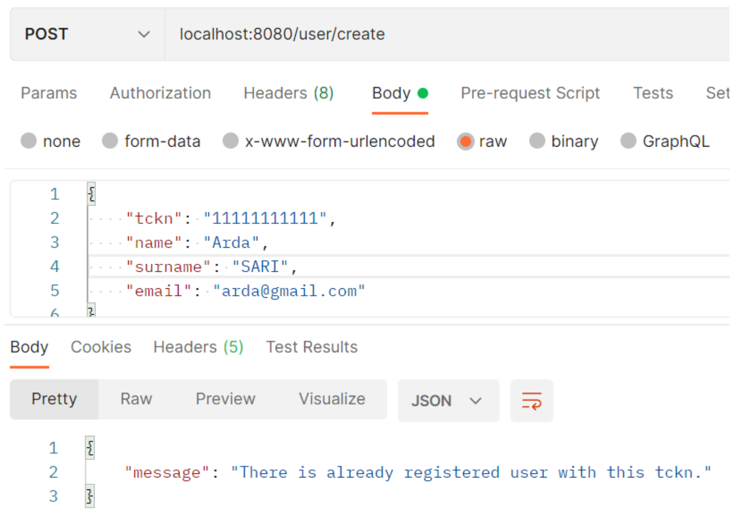
```
{
    "tckn": "11111111111",
    "name": "Berat Asrın",
    "surname": "CAFEROĞLU",
    "email": "beratasrin01@gmail.com"
}
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "answer": {
        "userId": "6376795c6a07951601dfc292",
        "tckn": "11111111111",
        "name": "Berat Asrın",
        "surname": "CAFEROĞLU",
        "email": "beratasrin01@gmail.com"
    },
    "message": "New user is created."
}
```

Figure 1: Create user request and response body


POST localhost:8080/user/create

Params Authorization Headers (8) Body Pre-request Script Tests Set

none form-data x-www-form-urlencoded raw binary GraphQL

```
{
    "tckn": "11111111111",
    "name": "Arda",
    "surname": "SARI",
    "email": "arda@gmail.com"
}
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "message": "There is already registered user with this tckn."
}
```

Figure 2: Error message when the same tckn is entered

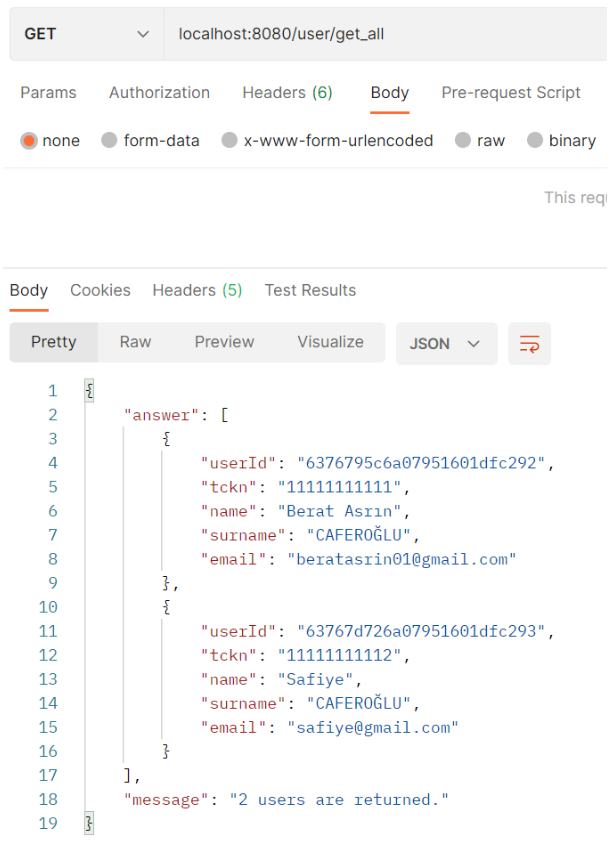
2.2. Get All Users

Using this function, we are able to see all registered users without giving any request body.

URL: localhost:8080/user/get_all

HTTP Method: GET

Required Body Parameters: No need



```
{
    "answer": [
        {
            "userId": "6376795c6a07951601dfc292",
            "tckn": "11111111111",
            "name": "Berat Asrın",
            "surname": "CAFEROĞLU",
            "email": "beratasrin01@gmail.com"
        },
        {
            "userId": "63767d726a07951601dfc293",
            "tckn": "11111111112",
            "name": "Safiye",
            "surname": "CAFEROĞLU",
            "email": "safiye@gmail.com"
        }
    ],
    "message": "2 users are returned."
}
```

Figure 3: All users

2.3. Update User

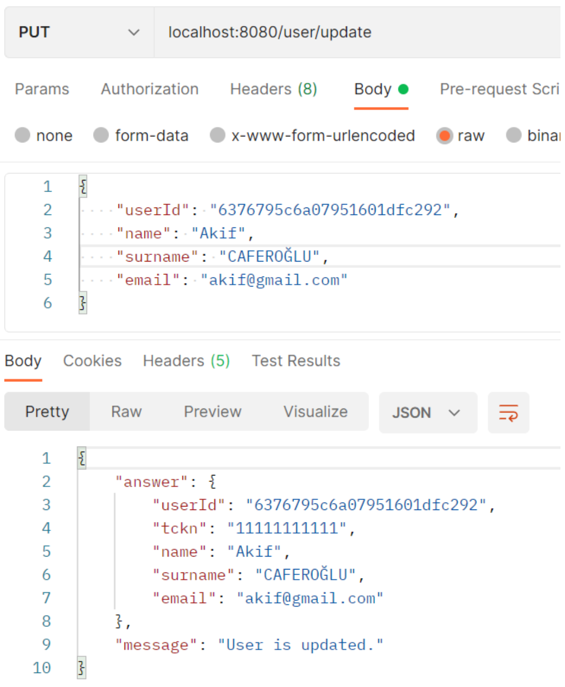
This function is designed to update the information in the database related to the user. It allows users to update names, surnames, and emails. But does not allow updating TCKN. Here in this function, the user that will be updated is found by their “userId” which was defined as the primary key. But, the most important thing about this function is if the request body contains “tckn” the function returns a special message.

URL: localhost:8080/user/update

HTTP Method: PUT

Required Body Parameters: userId (String), name (String), surname (String), email (String)

Please remember that “6376795c6a07951601dfc292” was userId of “Berat Asrın CAFEROĞLU”. Now we will update this user via the given URL.


PUT localhost:8080/user/update

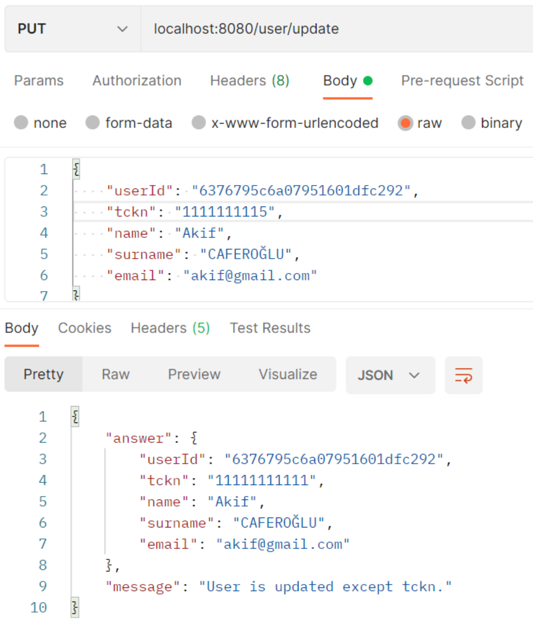
Body (raw):

```
{
    "userId": "6376795c6a07951601dfc292",
    "name": "Akif",
    "surname": "CAFEROĞLU",
    "email": "akif@gmail.com"
}
```

Response (Pretty, JSON):

```
{
    "answer": {
        "userId": "6376795c6a07951601dfc292",
        "tckn": "11111111111",
        "name": "Akif",
        "surname": "CAFEROĞLU",
        "email": "akif@gmail.com"
    },
    "message": "User is updated."
}
```

Figure 4: User is updated


PUT localhost:8080/user/update

Body (raw):

```
{
    "userId": "6376795c6a07951601dfc292",
    "tckn": "1111111115",
    "name": "Akif",
    "surname": "CAFEROĞLU",
    "email": "akif@gmail.com"
}
```

Response (Pretty, JSON):

```
{
    "answer": {
        "userId": "6376795c6a07951601dfc292",
        "tckn": "11111111111",
        "name": "Akif",
        "surname": "CAFEROĞLU",
        "email": "akif@gmail.com"
    },
    "message": "User is updated except tckn."
}
```

Figure 5: User is updated except TCKN

3. Account Side

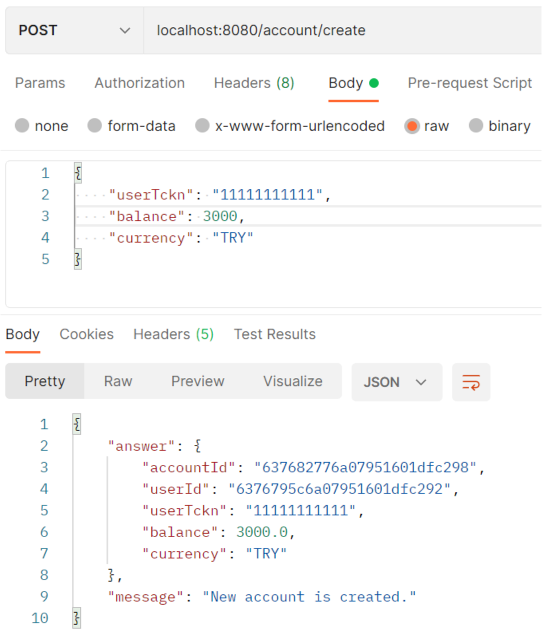
3.1. Create Account

This function is designed to open new accounts for users in the bank. While creating an account currency type and balance need to be given in the request body. Yet, according to my design, there are two ways to find the user who will be the owner of the account. The first way is by entering “userId”, and the second is by entering “userTckn”. Either way, the account will be created, but one of these parameters must be in request body.

URL: localhost:8080/account/create

HTTP Method: POST

Required Body Parameters: userTckn (String) or userId(String) (One of them must be given), balance (float), currency (String)

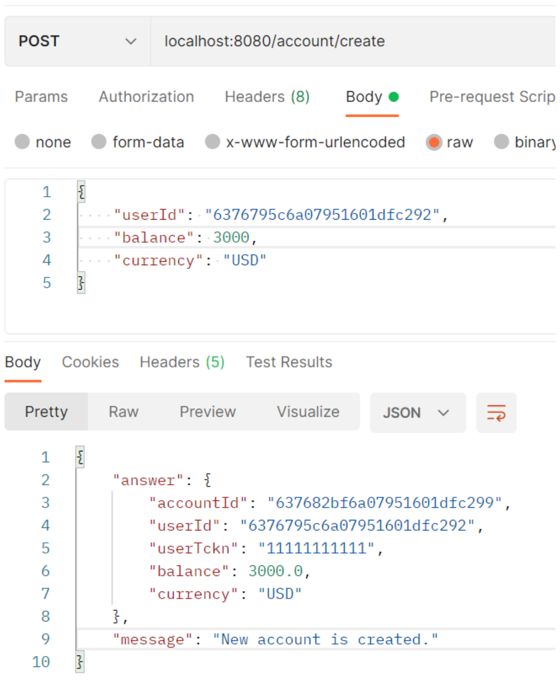


localhost:8080/account/create

```
{
    "userTckn": "11111111111",
    "balance": 3000,
    "currency": "TRY"
}
```

```
{
    "answer": {
        "accountId": "637682776a07951601dfc298",
        "userId": "6376795c6a07951601dfc292",
        "userTckn": "11111111111",
        "balance": 3000.0,
        "currency": "TRY"
    },
    "message": "New account is created."
}
```

Figure 6: Create new account using userTckn



localhost:8080/account/create

```
{
    "userId": "6376795c6a07951601dfc292",
    "balance": 3000,
    "currency": "USD"
}
```

```
{
    "answer": {
        "accountId": "637682bf6a07951601dfc299",
        "userId": "6376795c6a07951601dfc292",
        "userTckn": "11111111111",
        "balance": 3000.0,
        "currency": "USD"
    },
    "message": "New account is created."
}
```

Figure 7: Create new account using userId

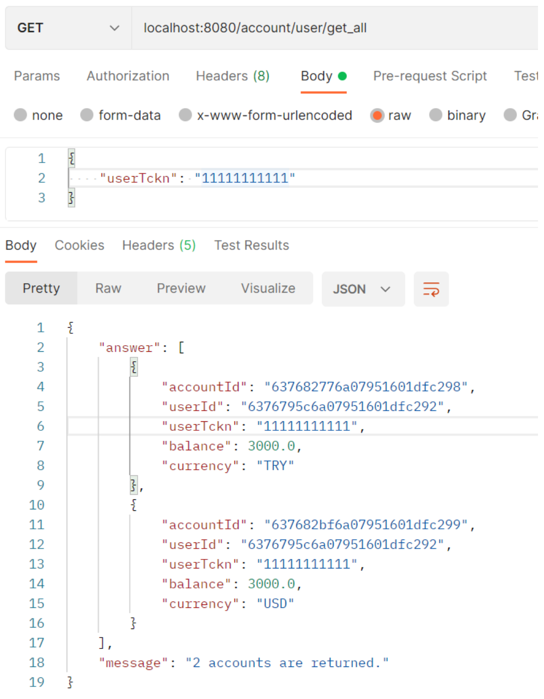
3.2. Get All Accounts of User

This function allows us to get the accounts of the specified user.

URL: localhost:8080/account/user/get_all

HTTP Method: GET

Required Body Parameters: userTckn (String) or userId(String) (One of them should be given)



GET localhost:8080/account/user/get_all

Params Authorization Headers (8) Body Pre-request Script Test

none form-data x-www-form-urlencoded [✓ raw] binary Gr

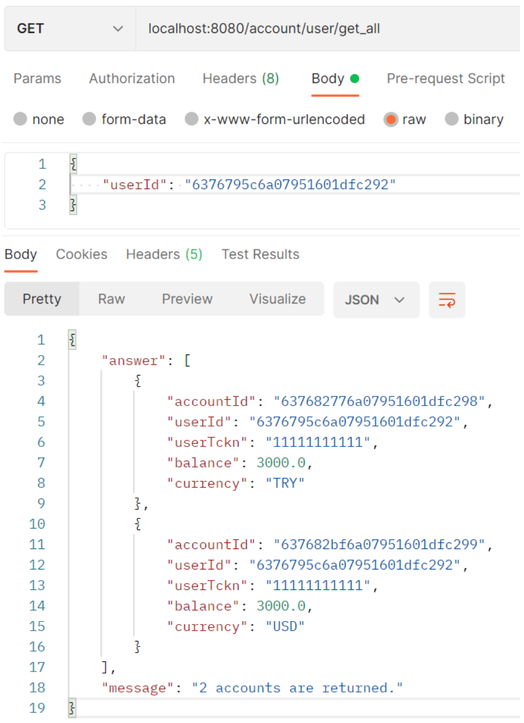
```
{
    "userTckn": "11111111111"
}
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "answer": [
        {
            "accountId": "637682776a07951601dfc298",
            "userId": "6376795c6a07951601dfc292",
            "userTckn": "11111111111",
            "balance": 3000.0,
            "currency": "TRY"
        },
        {
            "accountId": "637682bf6a07951601dfc299",
            "userId": "6376795c6a07951601dfc292",
            "userTckn": "11111111111",
            "balance": 3000.0,
            "currency": "USD"
        }
    ],
    "message": "2 accounts are returned."
}
```

Figure 8: Accounts of user



GET localhost:8080/account/user/get_all

Params Authorization Headers (8) Body Pre-request Script

none form-data x-www-form-urlencoded [✓ raw] binary

```
{
    "userId": "6376795c6a07951601dfc292"
}
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
{
    "answer": [
        {
            "accountId": "637682776a07951601dfc298",
            "userId": "6376795c6a07951601dfc292",
            "userTckn": "11111111111",
            "balance": 3000.0,
            "currency": "TRY"
        },
        {
            "accountId": "637682bf6a07951601dfc299",
            "userId": "6376795c6a07951601dfc292",
            "userTckn": "11111111111",
            "balance": 3000.0,
            "currency": "USD"
        }
    ],
    "message": "2 accounts are returned."
}
```

Figure 9: Accounts of user

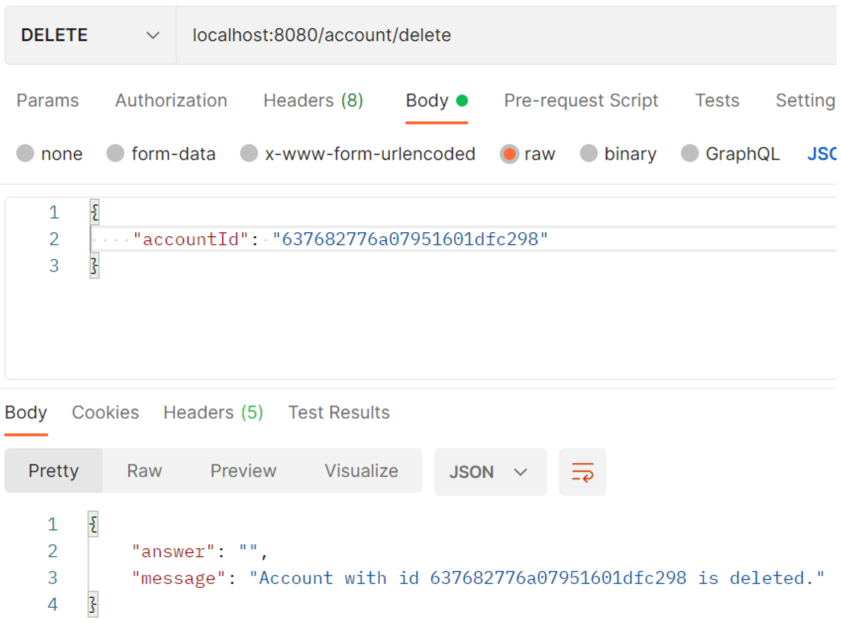
3.3. Remove Account

An account can be deleted via this function by giving its accountId.

URL: localhost:8080/account/delete

HTTP Method: DEL

Required Body Parameters: accountId (String)



```
{
    "accountId": "637682776a07951601dfc298"
}
```

```
{
    "answer": "",
    "message": "Account with id 637682776a07951601dfc298 is deleted."
}
```

Figure 10: Account with given accountId is deleted

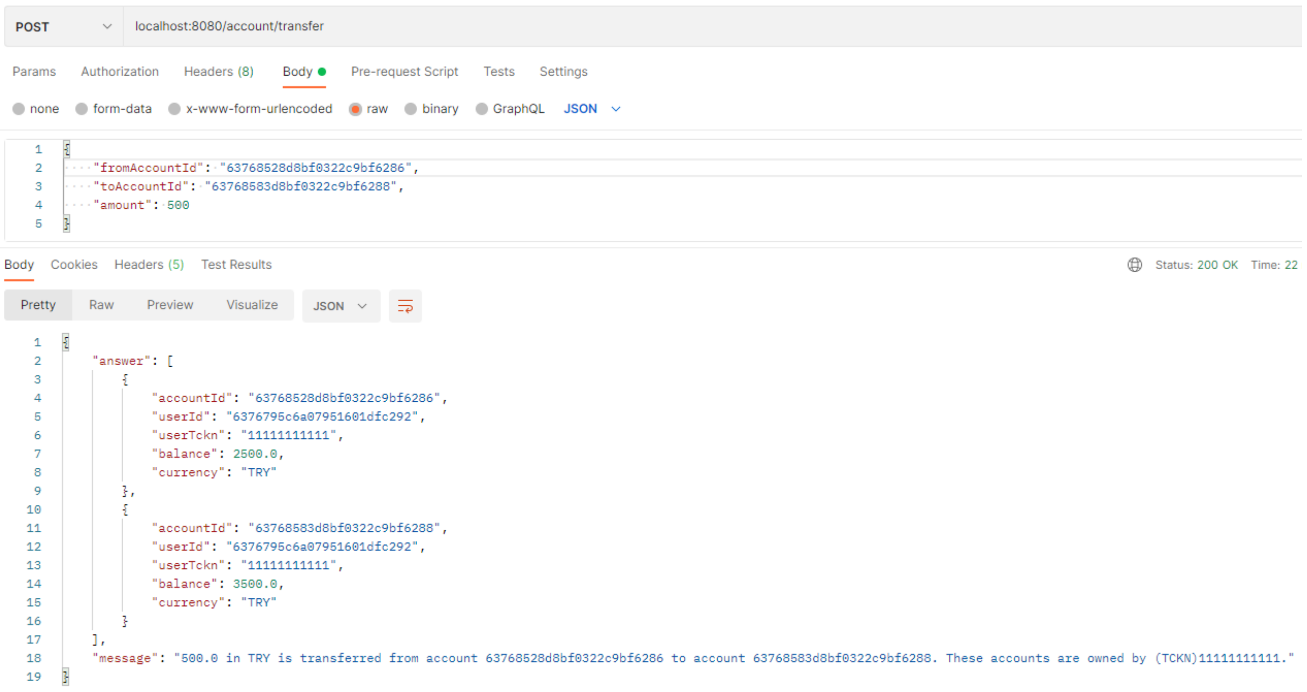
3.4. Transfer Balance

This function is used to transfer balance between accounts if they have same type of currency. If the currencies are different, then, the transaction is not done. On the other hand, if these two accounts have the same currency but the account which sends the money does not have enough balance, then, again transaction is not done. Beyond these, if the transaction is between different users, their TCKNs are given in the message.

URL: localhost:8080/account/transfer

HTTP Method: POST

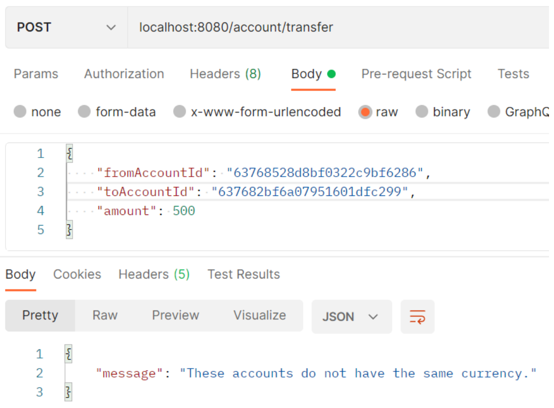
Required Body Parameters: fromAccountId (String), toAccountId (String), amount (Float)



```
{
    "fromAccountId": "63768528d8bf0322c9bf6286",
    "toAccountId": "63768583d8bf0322c9bf6288",
    "amount": 500
}
```

```
{
    "answer": [
        {
            "accountId": "63768528d8bf0322c9bf6286",
            "userId": "6376795c6a07951601dfc292",
            "userTckn": "11111111111",
            "balance": 2500.0,
            "currency": "TRY"
        },
        {
            "accountId": "63768583d8bf0322c9bf6288",
            "userId": "6376795c6a07951601dfc292",
            "userTckn": "11111111111",
            "balance": 3500.0,
            "currency": "TRY"
        }
    ],
    "message": "500.0 in TRY is transferred from account 63768528d8bf0322c9bf6286 to account 63768583d8bf0322c9bf6288. These accounts are owned by (TCKN)11111111111."
}
```

Figure 11: Balance transfer between accounts

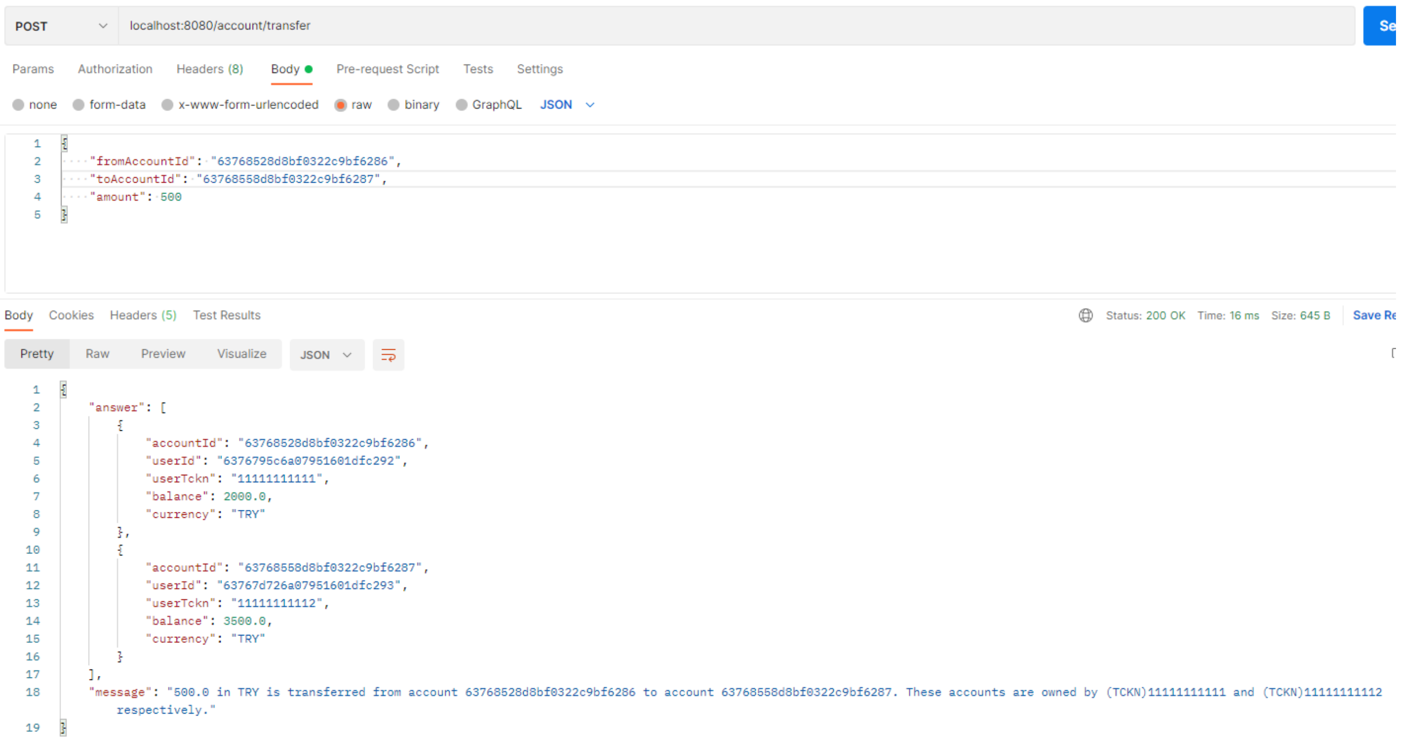


POST localhost:8080/account/transfer

```
{
    "fromAccountId": "63768528d8bf0322c9bf6286",
    "toAccountId": "637682bf6a07951601dfc299",
    "amount": 500
}
```

```
{
    "message": "These accounts do not have the same currency."
}
```

Figure 12: Balance transfer error due to different currencies

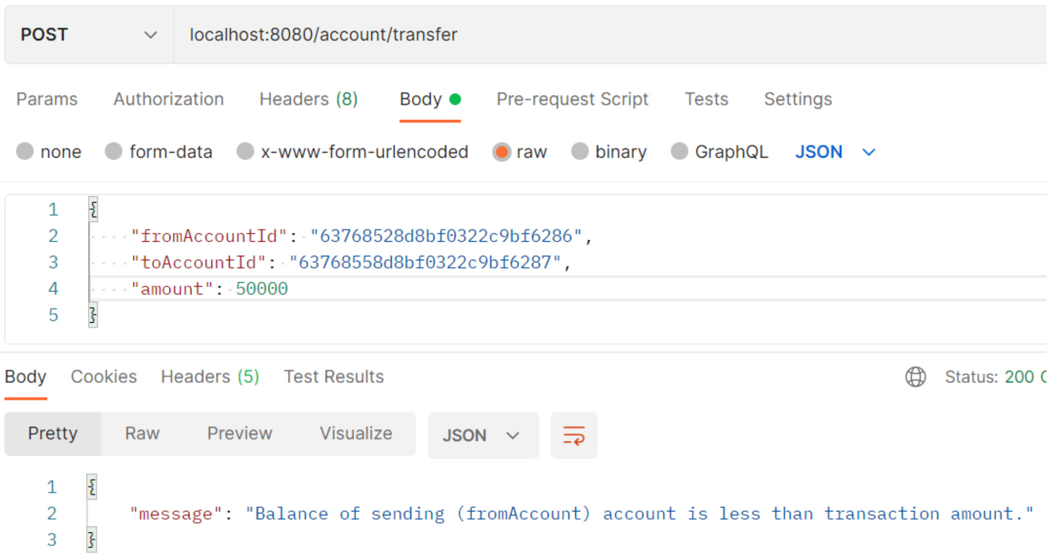


POST localhost:8080/account/transfer

```
{
    "fromAccountId": "63768528d8bf0322c9bf6286",
    "toAccountId": "63768558d8bf0322c9bf6287",
    "amount": 500
}
```

```
{
    "answer": [
        {
            "accountId": "63768528d8bf0322c9bf6286",
            "userId": "6376795c6a07951601dfc292",
            "userTckn": "11111111111",
            "balance": 2000.0,
            "currency": "TRY"
        },
        {
            "accountId": "63768558d8bf0322c9bf6287",
            "userId": "63767d726a07951601dfc293",
            "userTckn": "11111111112",
            "balance": 3500.0,
            "currency": "TRY"
        }
    ],
    "message": "500.0 in TRY is transferred from account 63768528d8bf0322c9bf6286 to account 63768558d8bf0322c9bf6287. These accounts are owned by (TCKN)11111111111 and (TCKN)11111111112
        respectively."
}
```

Figure 13: Balance transfer between different users



POST localhost:8080/account/transfer

```
{
    "fromAccountId": "63768528d8bf0322c9bf6286",
    "toAccountId": "63768558d8bf0322c9bf6287",
    "amount": 50000
}
```

```
{
    "message": "Balance of sending (fromAccount) account is less than transaction amount."
}
```

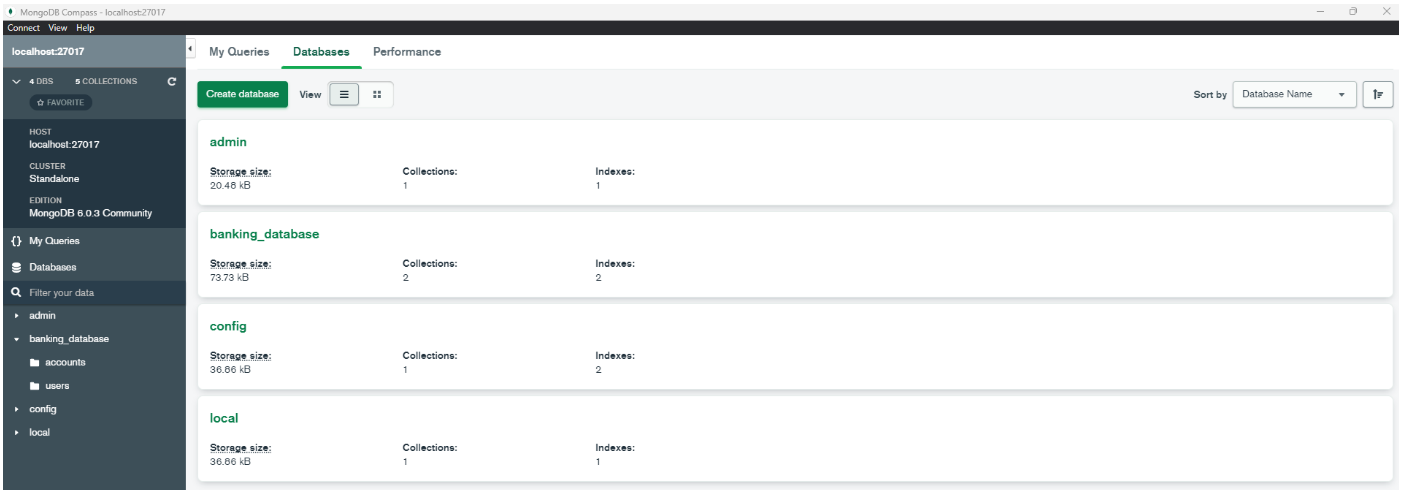
Figure 14: Not enough balance

4. Database (MongoDB) Structure Explanation

In this section, I would like to explain database structure that I have designed shortly. The most common part is creating a database name, so, the database that is used in this project is named “banking_database” as shown in Figure 15.

After that, in this project, we have mainly 2 entities (users, and accounts) that should be stored. Therefore, collections named “users”, and “accounts” are created as shown in Figure 16.

Finally stored data in them are shown in Figure 17, and Figure 18.



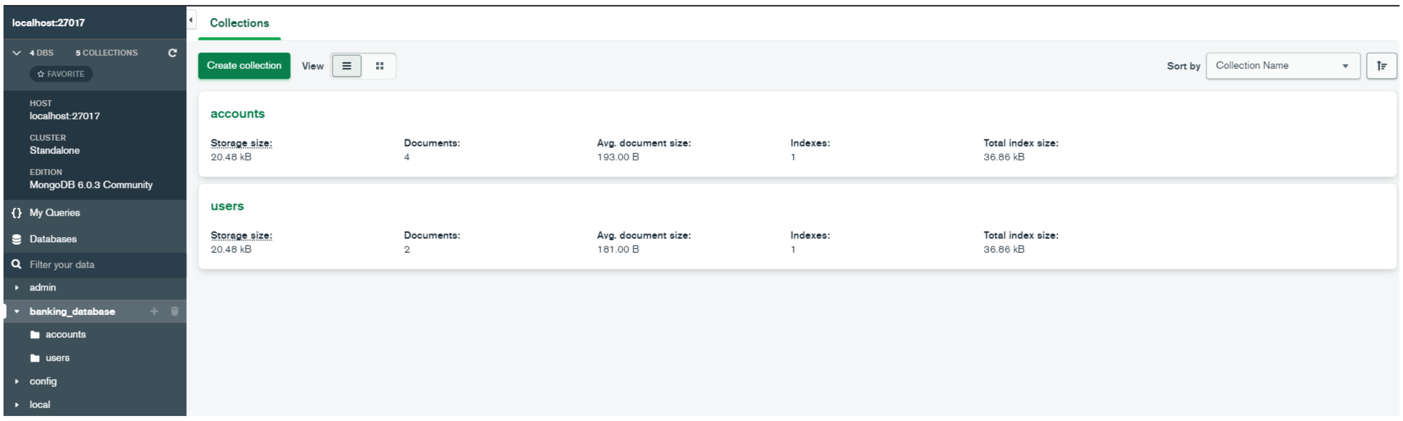
admin
Storage size: 20.48 kB Collections: 1 Indexes: 1

banking_database
Storage size: 73.73 kB Collections: 2 Indexes: 2

config
Storage size: 36.86 kB Collections: 1 Indexes: 2

local
Storage size: 36.86 kB Collections: 1 Indexes: 1

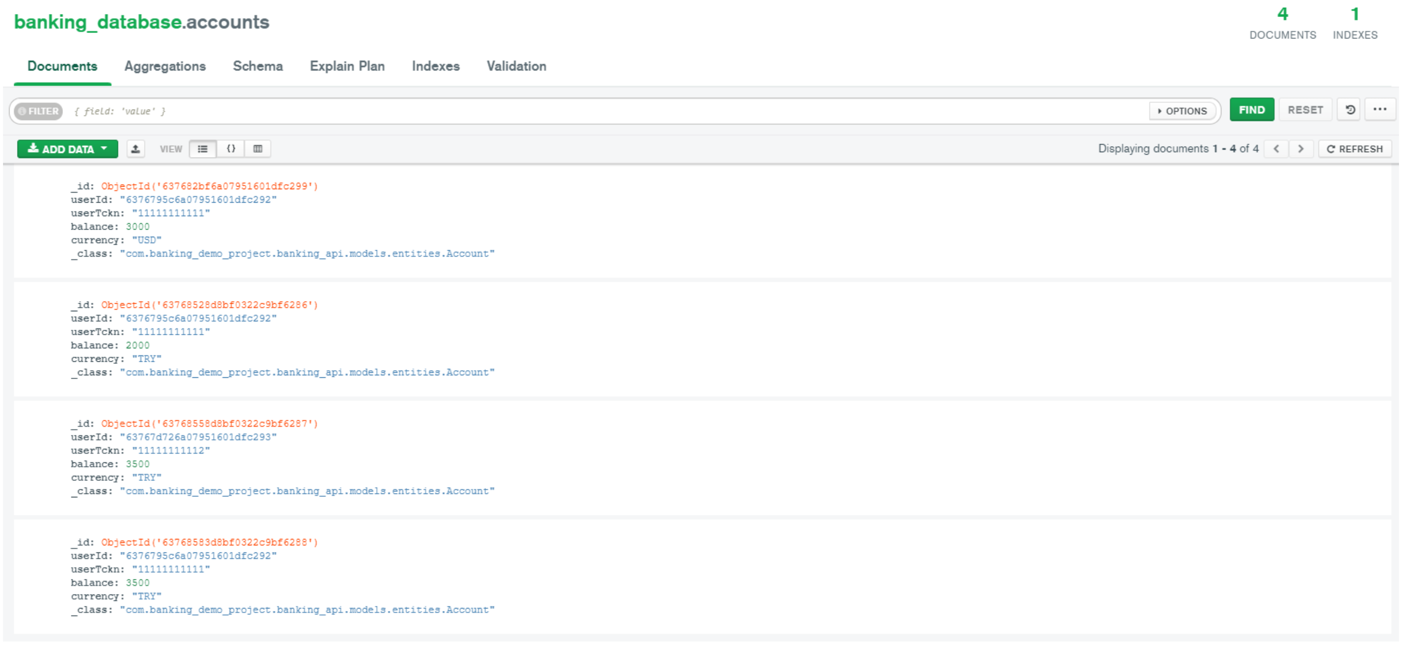
Figure 15: Database name: banking_database



accounts
Storage size: 20.48 kB Documents: 4 Avg. document size: 193.00 B Indexes: 1 Total index size: 36.86 kB

users
Storage size: 20.48 kB Documents: 2 Avg. document size: 181.00 B Indexes: 1 Total index size: 36.86 kB

Figure 16: Collections in database



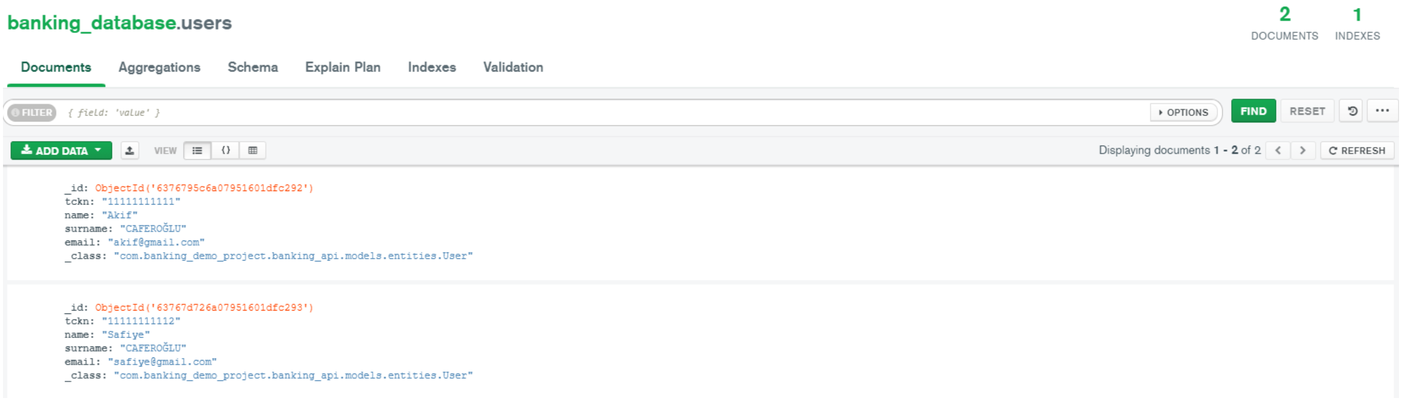
```
_id: ObjectId('637682bf6a07951601dfc299')
userId: "6376795c6a07951601dfc292"
userTckn: "11111111111"
balance: 3000
currency: "USD"
_class: "com.banking_demo_project.banking_api.models.entities.Account"

_id: ObjectId('63768528d8bf0322c9bf6286')
userId: "6376795c6a07951601dfc292"
userTckn: "11111111111"
balance: 2000
currency: "TRY"
_class: "com.banking_demo_project.banking_api.models.entities.Account"

_id: ObjectId('63768558d8bf0322c9bf6287')
userId: "63767d726a07951601dfc293"
userTckn: "11111111112"
balance: 3500
currency: "TRY"
_class: "com.banking_demo_project.banking_api.models.entities.Account"

_id: ObjectId('63768583d8bf0322c9bf6288')
userId: "6376795c6a07951601dfc292"
userTckn: "11111111111"
balance: 3500
currency: "TRY"
_class: "com.banking_demo_project.banking_api.models.entities.Account"
```

Figure 17: Accounts collection



```
_id: ObjectId('6376795c6a07951601dfc292')
tckn: "11111111111"
name: "Akif"
surname: "CAFEROĞLU"
email: "akif@gmail.com"
_class: "com.banking_demo_project.banking_api.models.entities.User"

_id: ObjectId('63767d726a07951601dfc293')
tckn: "11111111112"
name: "Safiye"
surname: "CAFEROĞLU"
email: "safiye@gmail.com"
_class: "com.banking_demo_project.banking_api.models.entities.User"
```

Figure 18: Users collection