Fake financial institution HTTP API

Perfectstay technical test

Simon Cariou

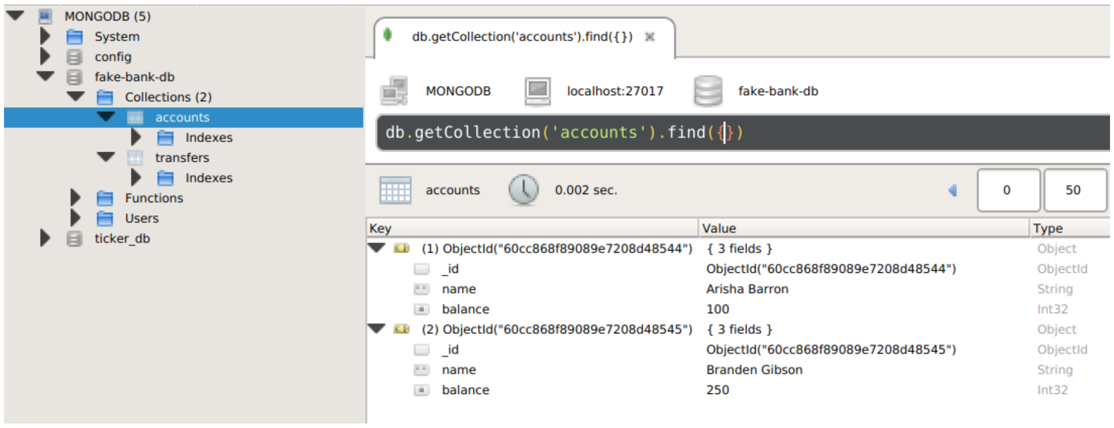
Overall

Python3 virtual environment with:

- Mongodb for the Database
- Flask for the API Framework

MongoDB to store JSON formatted documents:

- 1 database:
  - db
- Collection:
  - Customers:
    - Name (String)
    - Amount (Double)

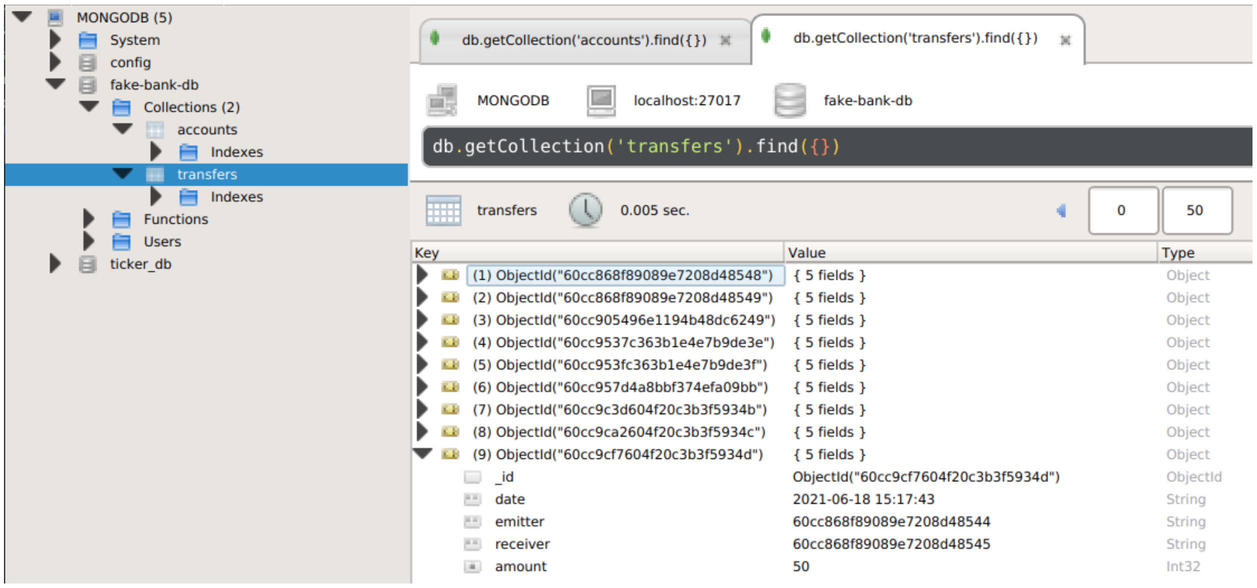


```
MONGODB (5)
  System
  config
  fake-bank-db
    Collections (2)
      accounts
        Indexes
      transfers
        Indexes
    Functions
    Users
  ticker_db
```

```
db.getCollection('accounts').find({})
MONGODB   localhost:27017   fake-bank-db
db.getCollection('accounts').find({})
accounts   0.002 sec.   0   50
```

Key	Value	Type
(1) ObjectId("60cc868f89089e7208d48544")	{ 3 fields }	Object
_id	ObjectId("60cc868f89089e7208d48544")	ObjectId
name	Arisha Barron	String
balance	100	Int32
(2) ObjectId("60cc868f89089e7208d48545")	{ 3 fields }	Object
_id	ObjectId("60cc868f89089e7208d48545")	ObjectId
name	Branden Gibson	String
balance	250	Int32

  - Transfers:
    - Date + Time (String)
    - Amount transferred (Double)
    - Account ID that sends the money (String)
    - Account ID that receives the money (String)



```
MONGODB (5)
  System
  config
  fake-bank-db
    Collections (2)
      accounts
        Indexes
      transfers
        Indexes
    Functions
    Users
  ticker_db
```

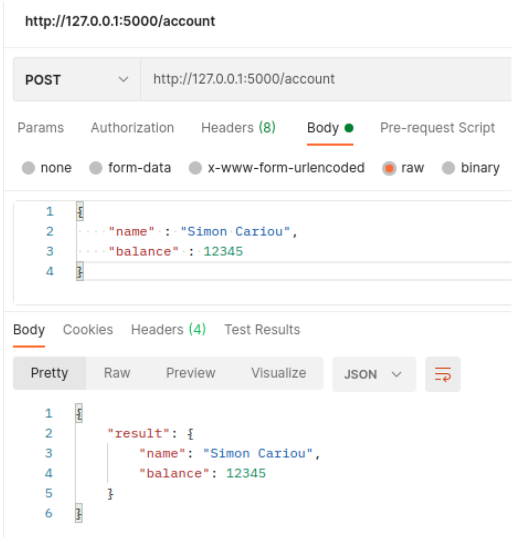
```
db.getCollection('accounts').find({})   db.getCollection('transfers').find({})
MONGODB   localhost:27017   fake-bank-db
db.getCollection('transfers').find({})
transfers   0.005 sec.   0   50
```

Key	Value	Type
(1) ObjectId("60cc868f89089e7208d48548")	{ 5 fields }	Object
(2) ObjectId("60cc868f89089e7208d48549")	{ 5 fields }	Object
(3) ObjectId("60cc905496e1194b48dc6249")	{ 5 fields }	Object
(4) ObjectId("60cc9537c363b1e4e7b9de3e")	{ 5 fields }	Object
(5) ObjectId("60cc953fc363b1e4e7b9de3f")	{ 5 fields }	Object
(6) ObjectId("60cc957d4a8bbf374efa09bb")	{ 5 fields }	Object
(7) ObjectId("60cc9c3d604f20c3b3f5934b")	{ 5 fields }	Object
(8) ObjectId("60cc9ca2604f20c3b3f5934c")	{ 5 fields }	Object
(9) ObjectId("60cc9cf7604f20c3b3f5934d")	{ 5 fields }	Object
_id	ObjectId("60cc9cf7604f20c3b3f5934d")	ObjectId
date	2021-06-18 15:17:43	String
emitter	60cc868f89089e7208d48544	String
receiver	60cc868f89089e7208d48545	String
amount	50	Int32

API Documentation

1. `@app.route('/account', methods=['POST'])`

POST - Add a customer account. Providing the name of the owner and the balance.

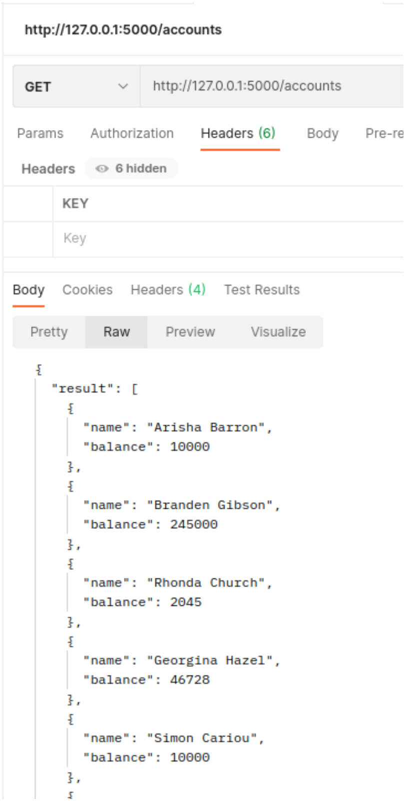


```
{
    "name" : "Simon Cariou",
    "balance" : 12345
}
```

```
{
    "result": {
        "name": "Simon Cariou",
        "balance": 12345
    }
}
```

2. `@app.route('/accounts', methods=['GET'])`

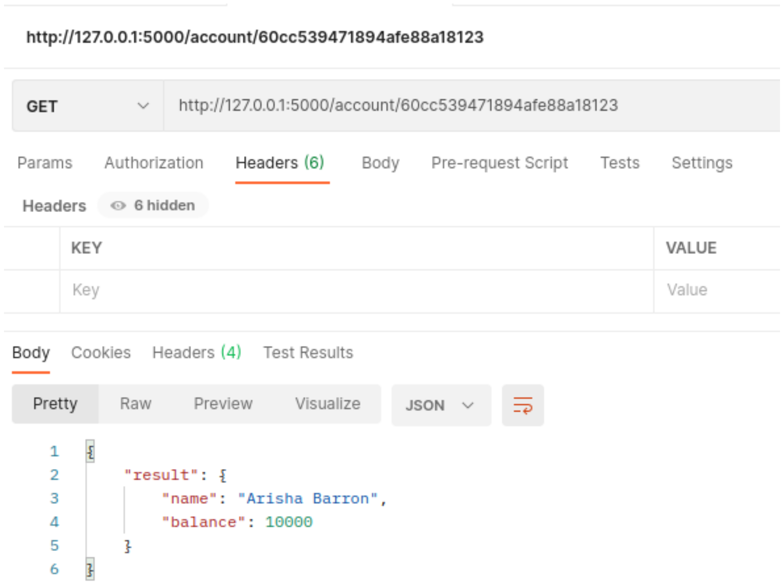
Returns a JSON containing off of the accounts present in the Accounts collection.



```
{
  "result": [
    {
      "name": "Arisha Barron",
      "balance": 10000
    },
    {
      "name": "Branden Gibson",
      "balance": 245000
    },
    {
      "name": "Rhonda Church",
      "balance": 2045
    },
    {
      "name": "Georgina Hazel",
      "balance": 46728
    },
    {
      "name": "Simon Cariou",
      "balance": 10000
    },
    {
```

```
3. @app.route("/account/<account_id>", methods=['GET'])
```

GET a specific Account ID information such as the owner's name and the balance:


http://127.0.0.1:5000/account/60cc539471894afe88a18123

```
{
    "result": {
        "name": "Arisha Barron",
        "balance": 10000
    }
}
```

```
4. @app.route('/balance/<account_id>', methods=['GET'])
```

Returns only the balance associated to the account of the queried id

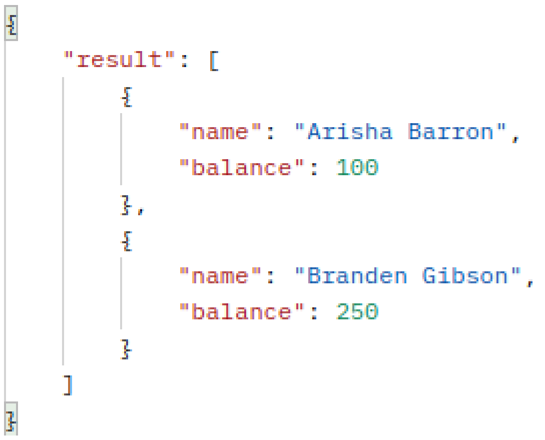
```
5. @app.route('/transfer', methods=['PUT'])
```

This method allows to transfer funds between 2 accounts: the emitter and the receiver
An entry is added to the transfer collection in order to keep track of all the transfers made by anyone.
The PUT request takes a JSON with key-values such as the following example:

```
{
    "emitter" : "60cc868f89089e7208d48544",
    "receiver" : "60cc868f89089e7208d48545",
    "amount" : 50
  }
```

<u>Example:</u>

We have the following accounts in our Accounts collection:



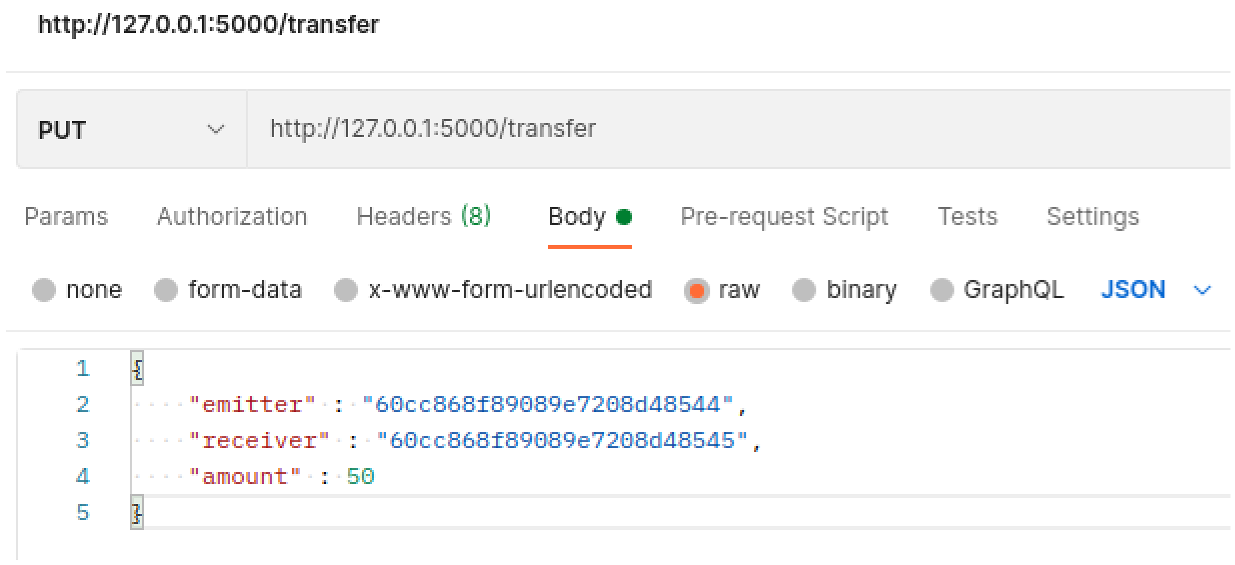
```
{
    "result": [
        {
            "name": "Arisha Barron",
            "balance": 100
        },
        {
            "name": "Branden Gibson",
            "balance": 250
        }
    ]
}
```

Arisha Barron owns 100€ and Branden Gibson 250€
In our Mongodb collection, we have unique IDs for both of these accounts, which are:

- Arisha Barron: “60cc868f89089e7208d48544”
- Branden Gibson: “60cc868f89089e7208d48545”

(ids are BSON ObjectIds which we conveniently cast as Strings for usability purposes)

We want Arisha to send Branden 50€, but since the same persons can have different bank accounts, we will query their uids in the request and we run the following PUT request:

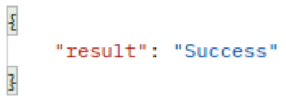


http://127.0.0.1:5000/transfer

PUT http://127.0.0.1:5000/transfer

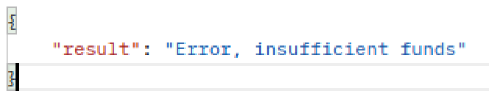
```
{
    "emitter" : "60cc868f89089e7208d48544",
    "receiver" : "60cc868f89089e7208d48545",
    "amount" : 50
}
```

Which returns:



```
{
    "result": "Success"
}
```

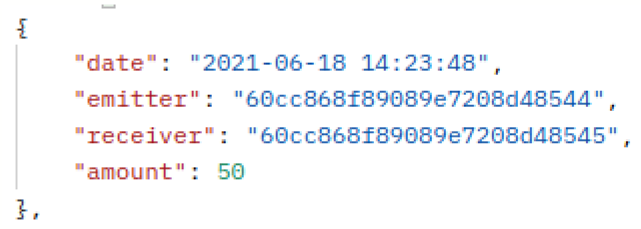
if the transfer goes through and:



```
{
    "result": "Error, insufficient funds"
}
```

If the emitter's balance is too low to send the desired amount.

When the transfer is complete and goes through, an entry is added to the Transfers collection to keep track of all of the transfers made between two accounts. The transfer is time stamped and contains the following information:

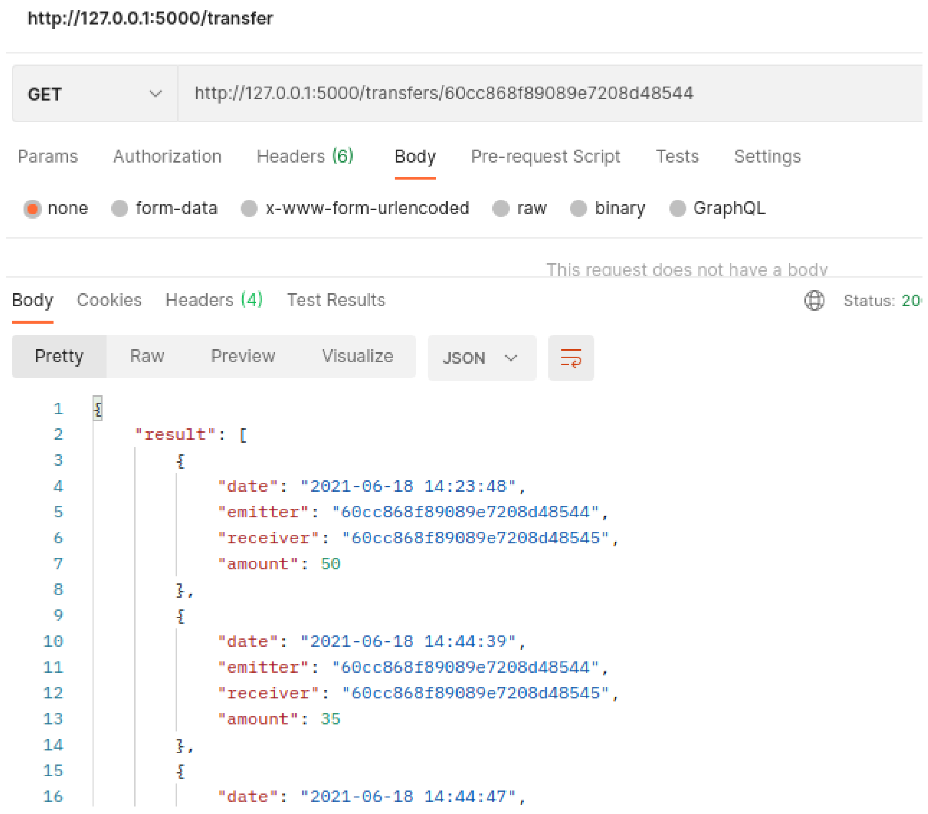


```
{
    "date": "2021-06-18 14:23:48",
    "emitter": "60cc868f89089e7208d48544",
    "receiver": "60cc868f89089e7208d48545",
    "amount": 50
},
```

We can verify that the transfer went through by executing the API call in 7.

6. `@app.route('/transfers/<emitter_account_id>', methods=['GET'])`

This method initiates a search in the transfers collection and will spit out all of the transfers matching the account_id. If we take the previous example:



http://127.0.0.1:5000/transfer

GET http://127.0.0.1:5000/transfers/60cc868f89089e7208d48544

```
{
    "result": [
        {
            "date": "2021-06-18 14:23:48",
            "emitter": "60cc868f89089e7208d48544",
            "receiver": "60cc868f89089e7208d48545",
            "amount": 50
        },
        {
            "date": "2021-06-18 14:44:39",
            "emitter": "60cc868f89089e7208d48544",
            "receiver": "60cc868f89089e7208d48545",
            "amount": 35
        },
        {
            "date": "2021-06-18 14:44:47",
```

(I also made another transfer which is why we can see the 35€ one).

Set up

Via docker-compose

```
sudo docker-compose build
sudo docker-compose up
```

How to use it ?

Then you are free to execute the API requests (API documentation in the next pages):
/!\ Mandatory to have the database populated. This creates 2 accounts which is enough to make money transfers:

1. `curl -iX POST -H "Content-Type: application/json" -d '{"name" : "Arisha Barron", "balance" : 10000}' "localhost:5000/account"`
2. `curl -iX POST -H "Content-Type: application/json" -d '{"name" : "Branden Gibson", "balance" : 245000}' "localhost:5000/account"`

Then we can perform the following API calls:

3. `curl -iX GET "localhost:5000/accounts"`

4. `curl -iX GET "localhost:5000/account/<str_account_id>"`

5. `curl -iX GET "localhost:5000/balance/<str_account_id>"`

6. `curl -iX POST -H "Content-Type: application/json" -d '{"name" : "Simon Cariou", "balance" : 50}' "localhost:5000/account"`

7. `curl -iX PUT -H "Content-Type: application/json" -d '{"emitter" : "60cefc98a90a1328ff3c56a4", "receiver" : "60cefc98a90a1328ff3c56a5", "amount" : 50}' "localhost:5000/transfer"`

8. `curl -iX GET "localhost:5000/transfers/<str_account_id>"`

The database is accessible on localhost:27017 with any db visualizer. I used Robot3T.