Money Transfer Application

A Java RESTful API for money transfers between users accounts.

This application is build by using sparkjava simple and expressive Java/Kotlin web framework DSL built for rapid development. It starts embedded server on localhost port “4567” with H2 in memory database initialized with some sample users account data to view.

This API guaranties the data consistency in any case. Even if it will be a huge amount concurrent user. This ability was achieved by using of select ... for update database feature which helps to lock the object until all related objects will be updated/created

It uses just two entities:

**Transaction** - the money transfer transaction used to initialize the transaction

**Bank Account** - the bank account which has balance in the specified currency

Required Technologies

\* Java 8

\* Maven

\* JavaSpark lib

\* H2 in memory database

\* Log4j

\* Apache HTTP Client

\* Junit

\* TestNG

\* Allure

\* Jacoco

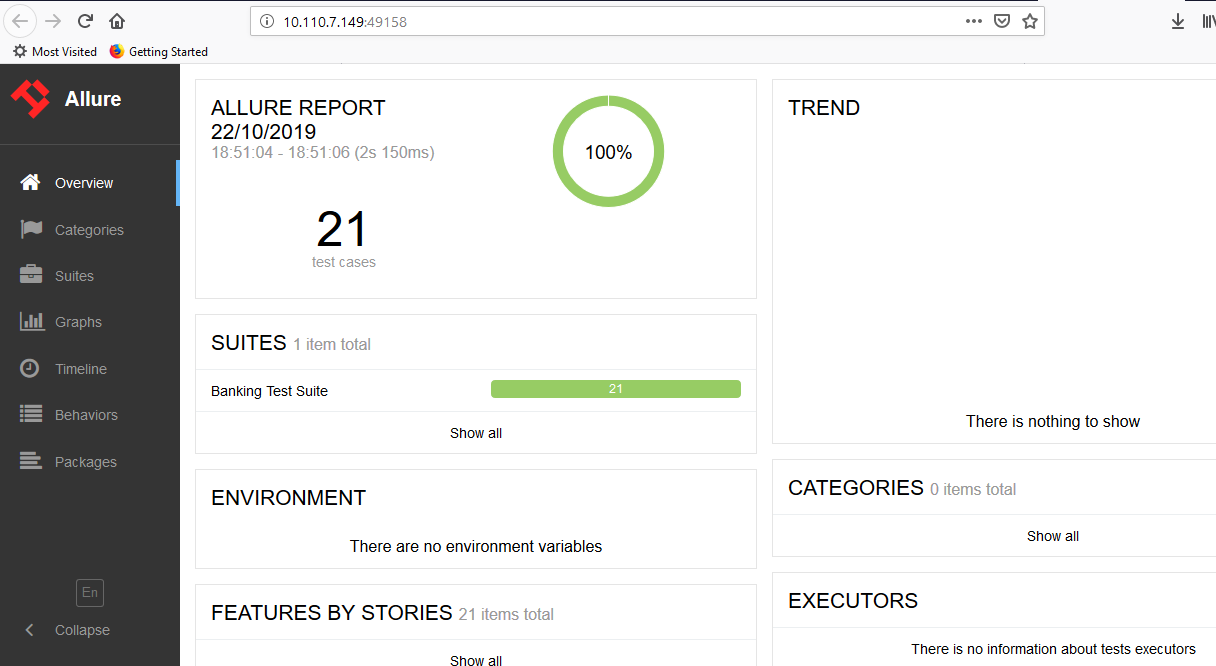
1.How to start

Once the application is fetched from git it can be built with maven by using this command.

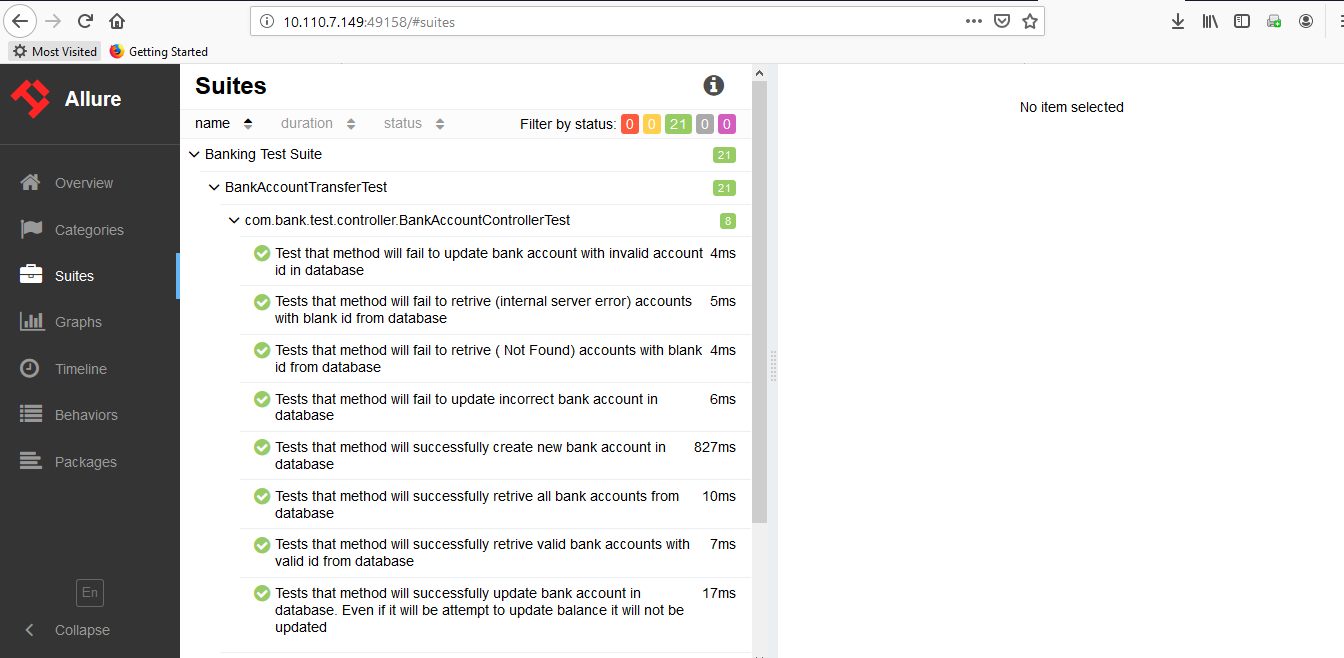
\Bank> mvn clean install test allure:serve allure:report

This will fetch dependencies and run all tests. It will compile and run all Junit test cases and open Allure report automatically in browser using random port.

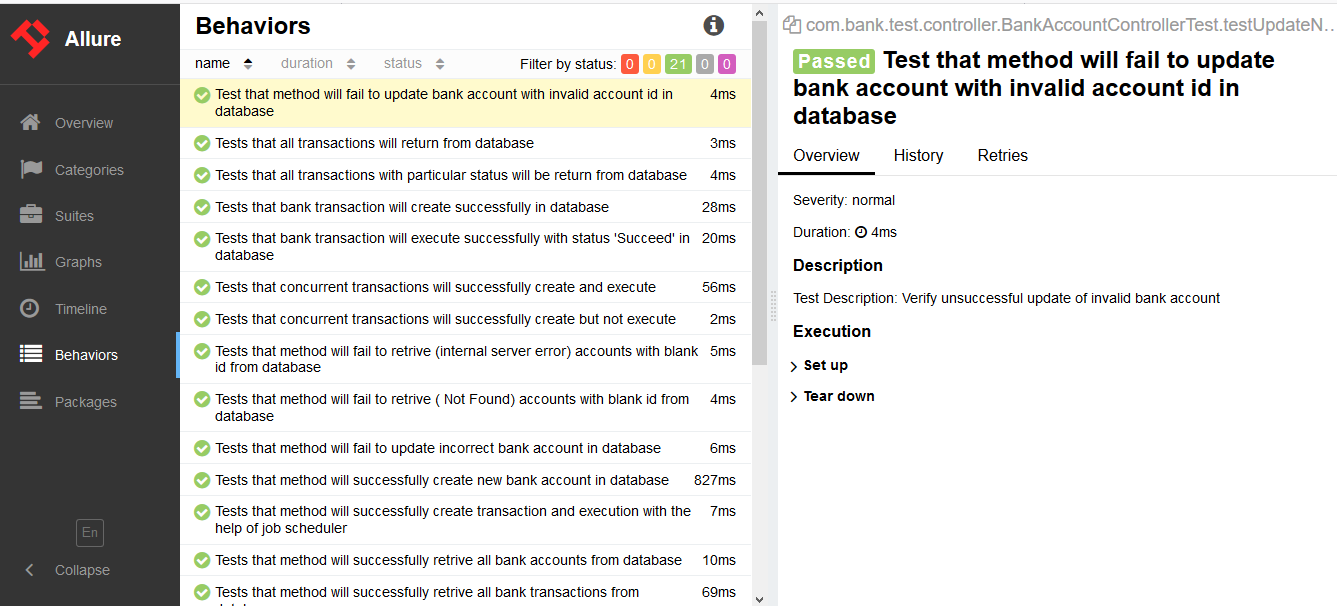
2. Overall Test Report



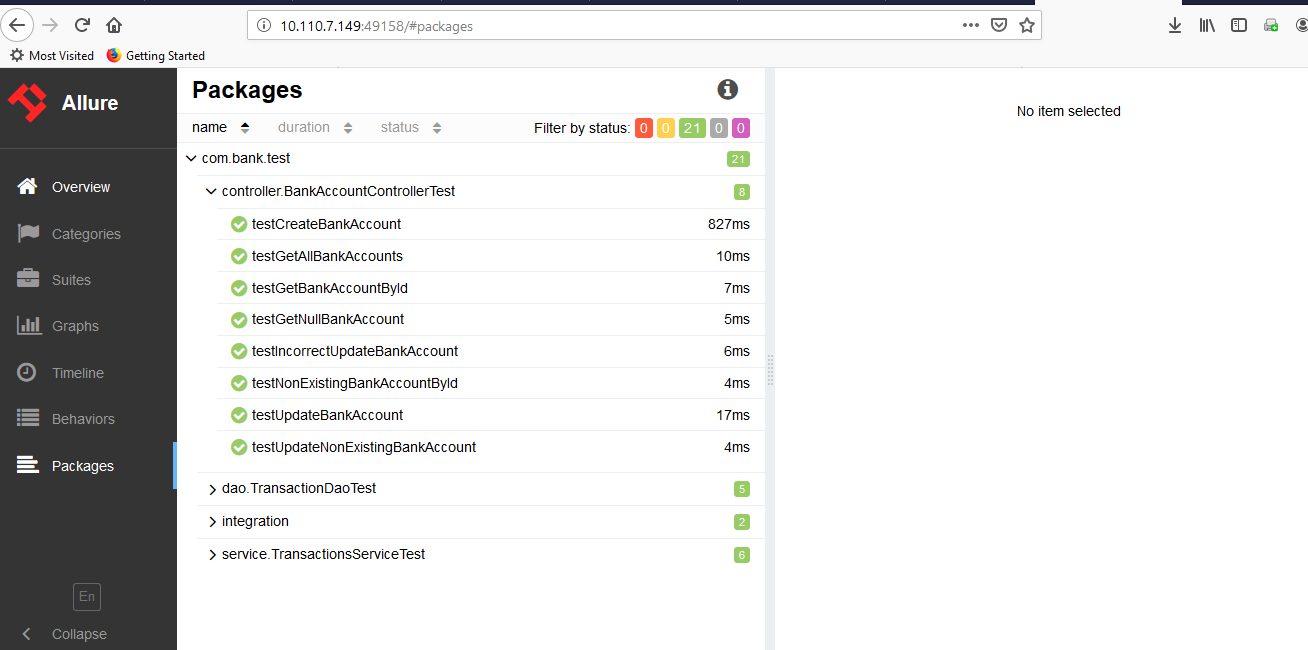
2.1 Test Suite Report



2.3 Behaviors Report

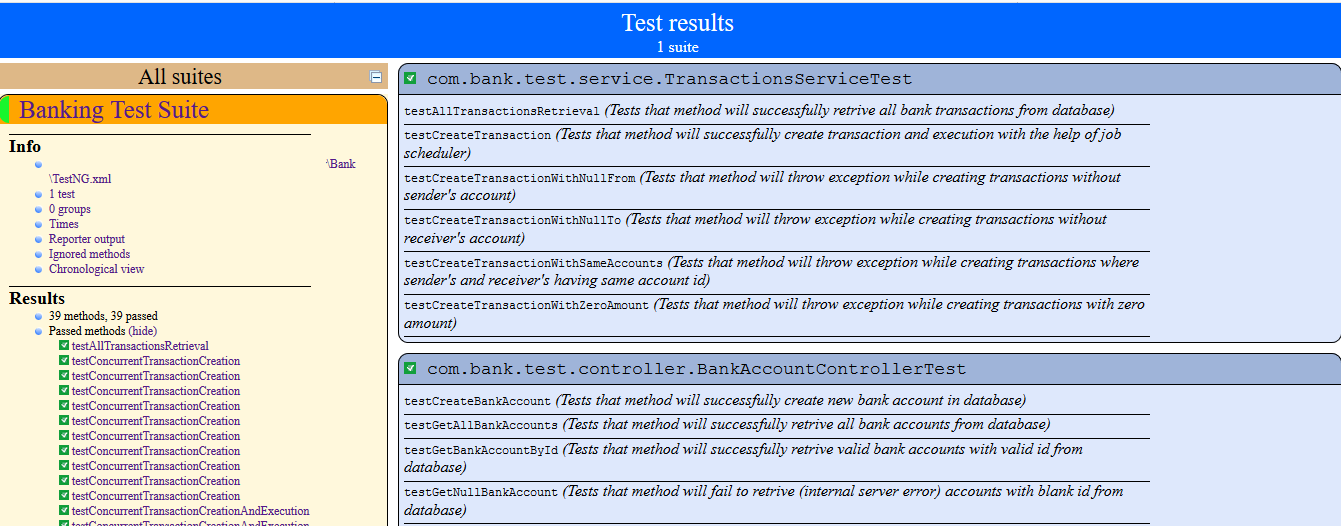


2.4 Overall Package Unit Test Report



2.5 TestNG Report:

Report Path: Bank\target\surefire-reports\index.html



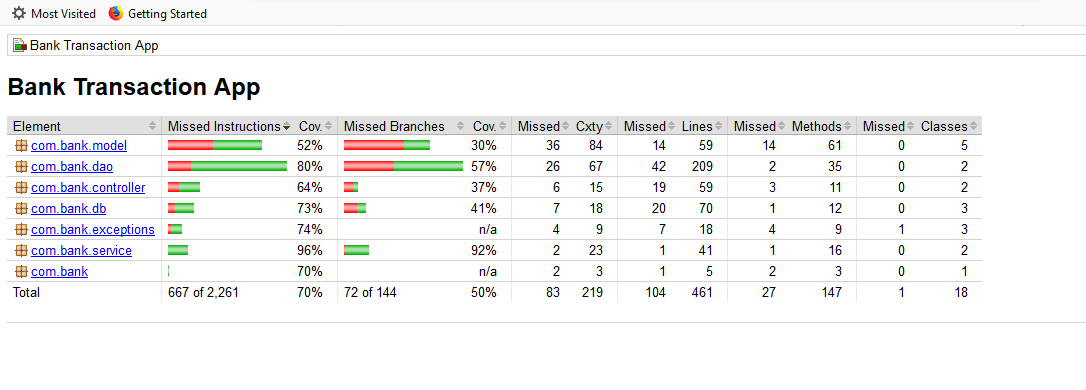
2.6 TestNG Report:

Report Path: Bank\target\surefire-reports\emailable-report.html.html



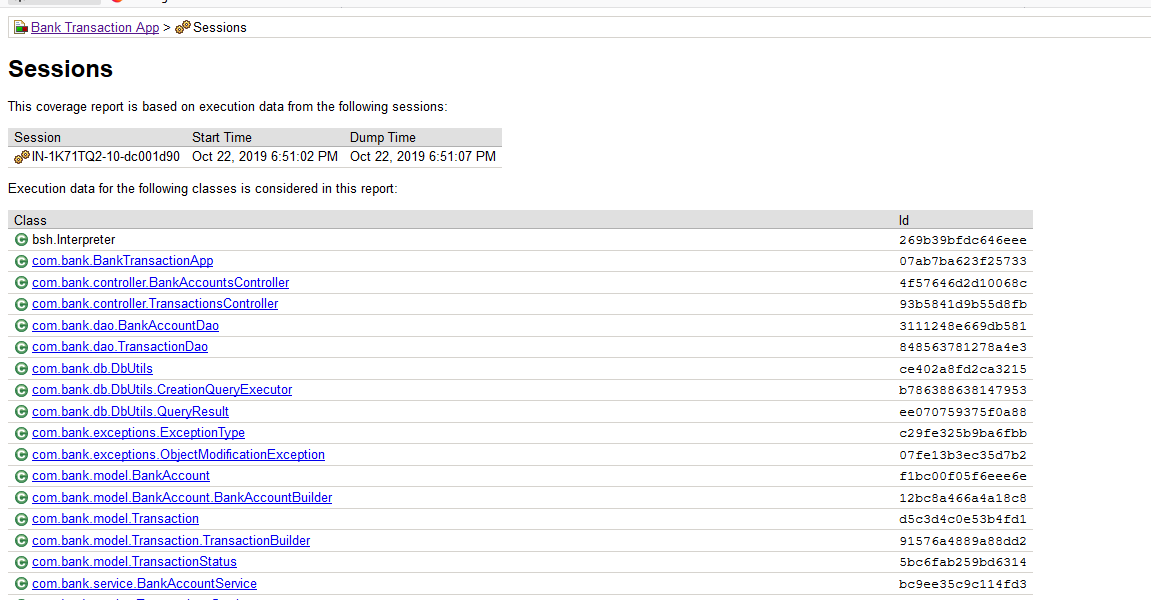
2.7 Code Coverage Report

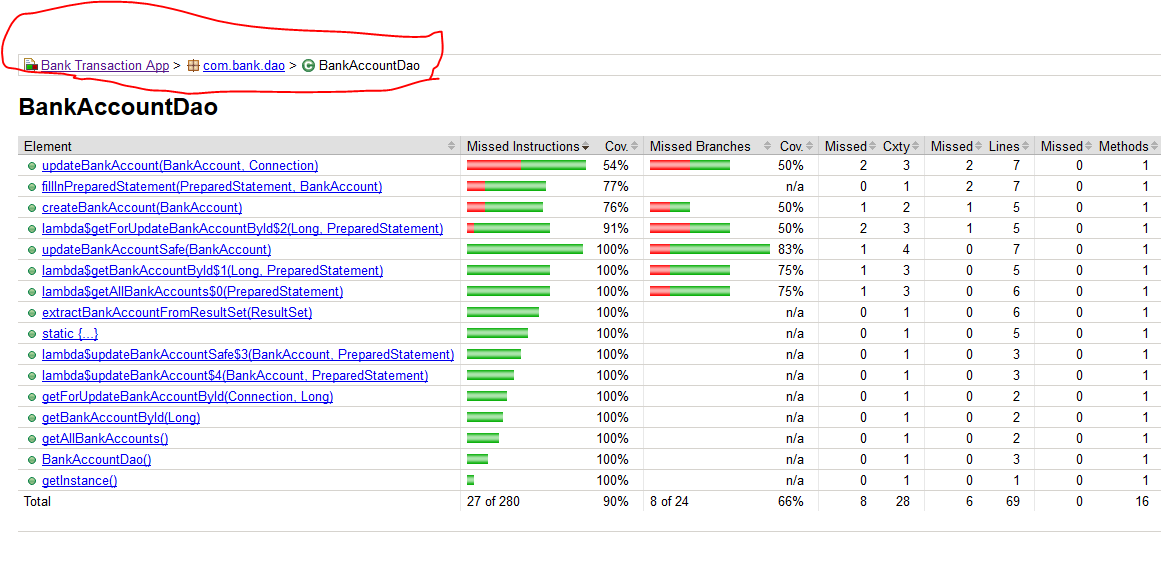
Report Path: Bank\target\site\jacoco\index.html

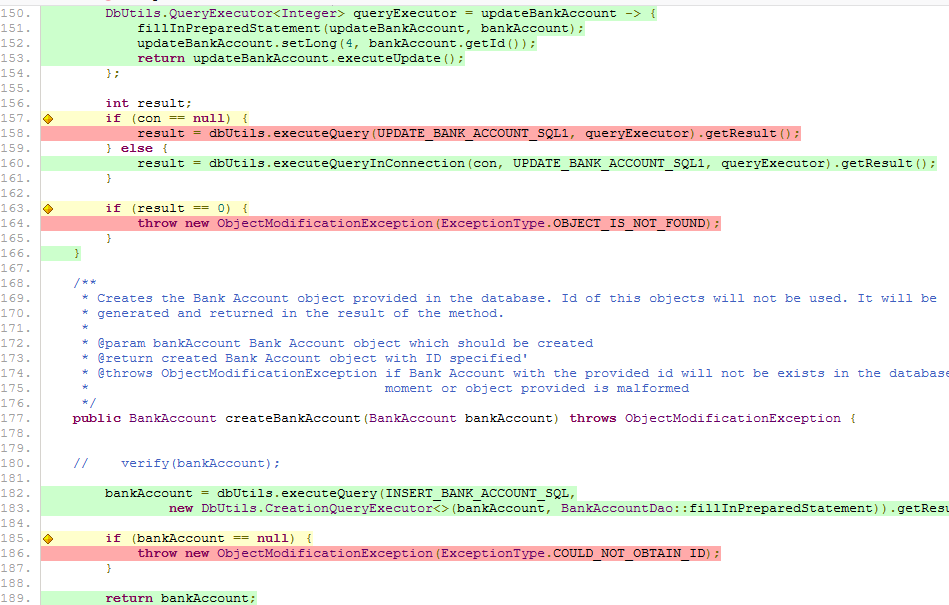


2.8 Session Report

Report Path : Bank\target\site\jacoco\ jacoco-sessions.html







3. To run only application to execute

Bank>java -cp target\Bank-0.0.1-jar-with-dependencies.jar com.bank.BankTransactionApp

The application will start on the localhost and will be listening to the port 4567

4. REST API Definition

**4.1 Bank Account Entity**

The bank account entity which has balance and could transfer the money if there is enough fund.

Structure

{

"id": <number>,

"ownerName": <string>,

"balance": <double>,

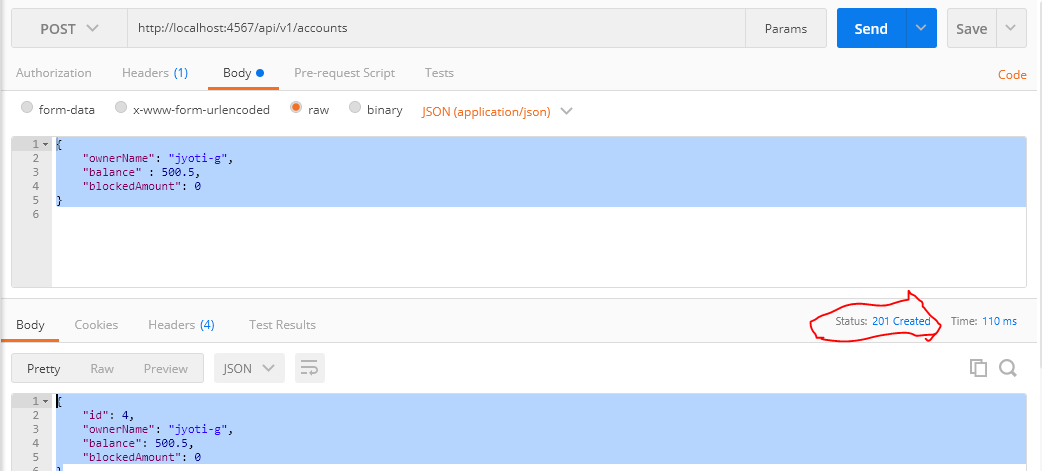
"blockedAmount": <double>,

}

**4.2 Create Bank Account**

HTTP POST request creates bank account and returns the created entity with Id specified.

POST <http://localhost:4567/api/v1/accounts>

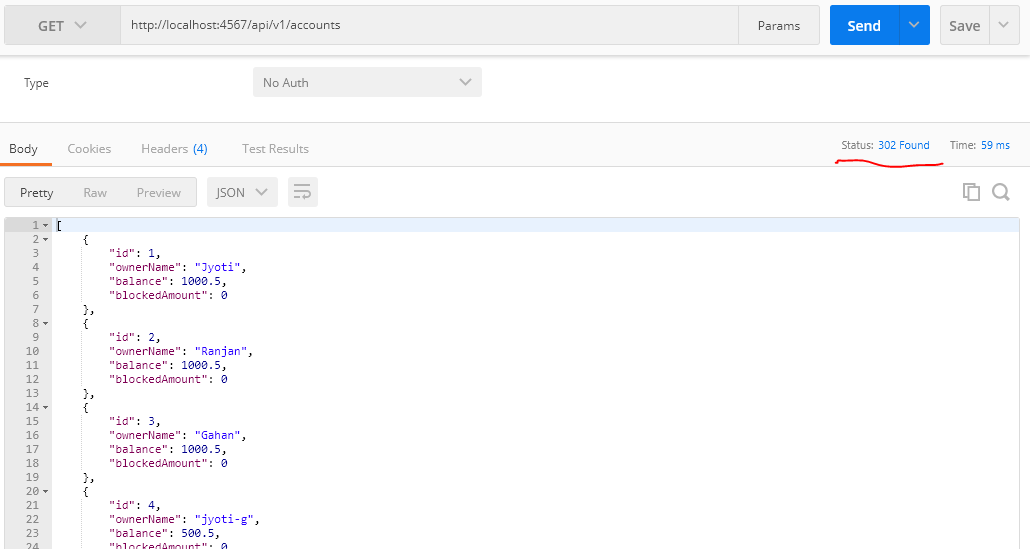


**4.3 Get All Bank Accounts**

HTTP GET request gets all the bank accounts that exist in the system.

Note: First three account created during application started.

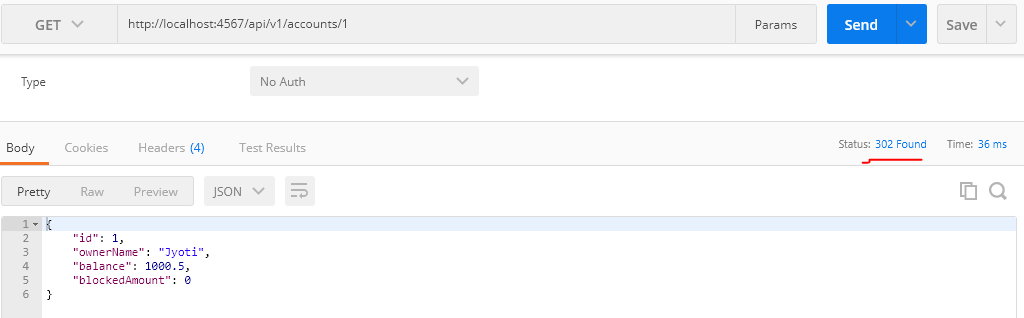
GET <http://localhost:4567/api/v1/accounts>



**4.4 Get A Particular Bank Account details**

HTTP GET request gets the particular account if it exists in the system

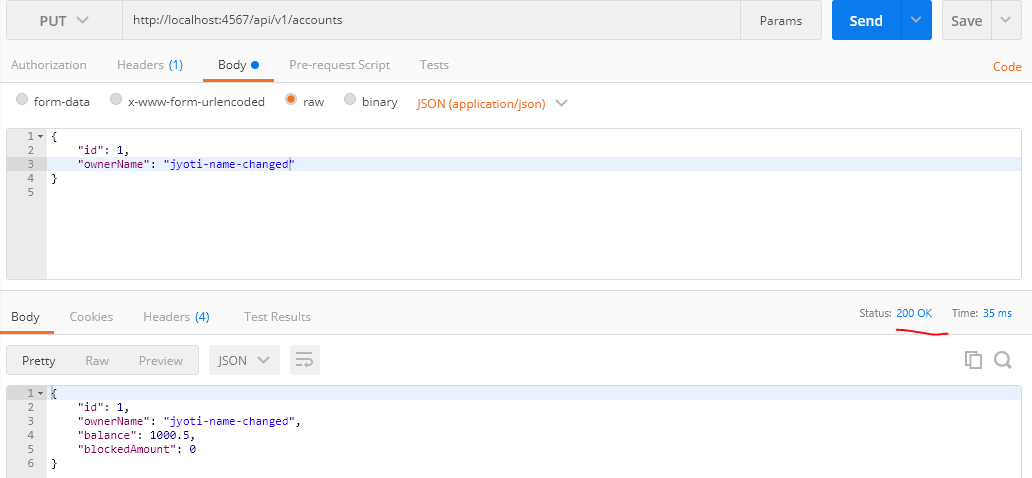
GET <http://localhost:4567/api/v1/accounts/1>



**4.5 Update A Particular Bank Account details**

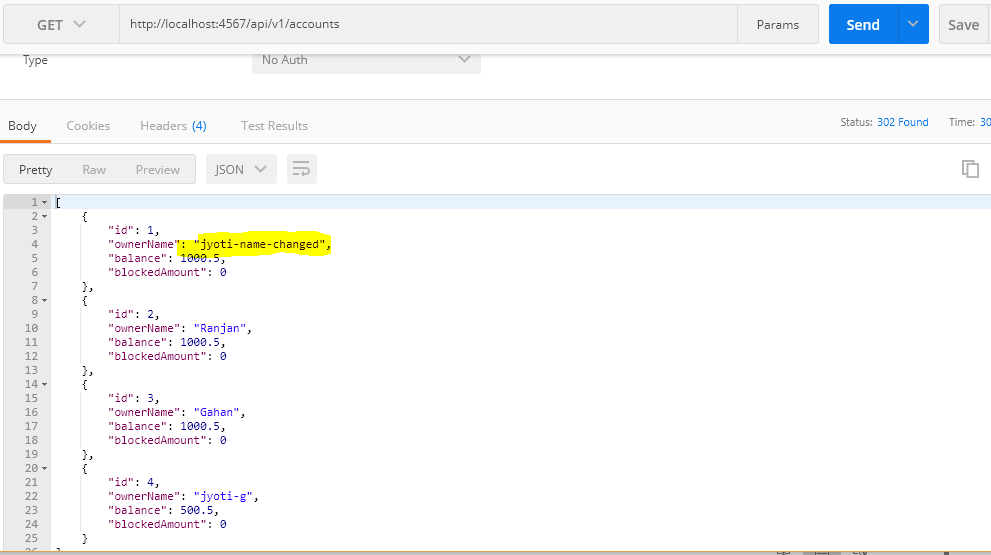
HTTP PUT request updates the details of the particular account if it exists in the system.You cannot update any field except "ownerName".

PUT <http://localhost:4567/api/v1/accounts>



**4.6 Get All Bank Accounts To Verify**

GET <http://localhost:4567/api/v1/accounts>



**4.7 Transaction Entity**

The money transfer transaction used to initialize the transaction. Once created will be executed automatically. If transaction can not be created by some reason the Error(HTTP 500 Internal Error) will be returned with details in the body. You can not update transaction object as it is controversial to the logic that transaction can not be modified once created.

Structure

{

"id": <number>,

"fromBankAccountId": <number>,

"toBankAccountId": <number>,

"amount": <double>,

"creationDate": <timestamp>,

"updateDate": <timestamp>,

"status": <string - one from "PLANNED", "PROCESSING", "FAILED", "SUCCEED">,

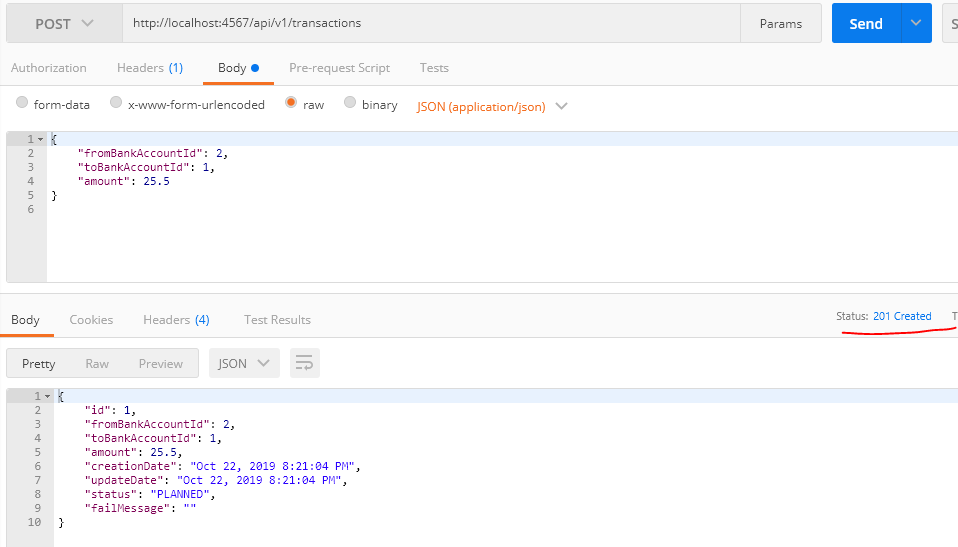
"failMessage": <string>

}

**4.8 Create a new transaction**

HTTP POST creates a new transaction if possible (valid Bank Accounts and parameters should be provided). Once id, creationDate, updateDate or status provided they will be ignored. You can obtain the generated values of these fields in the response of this call.

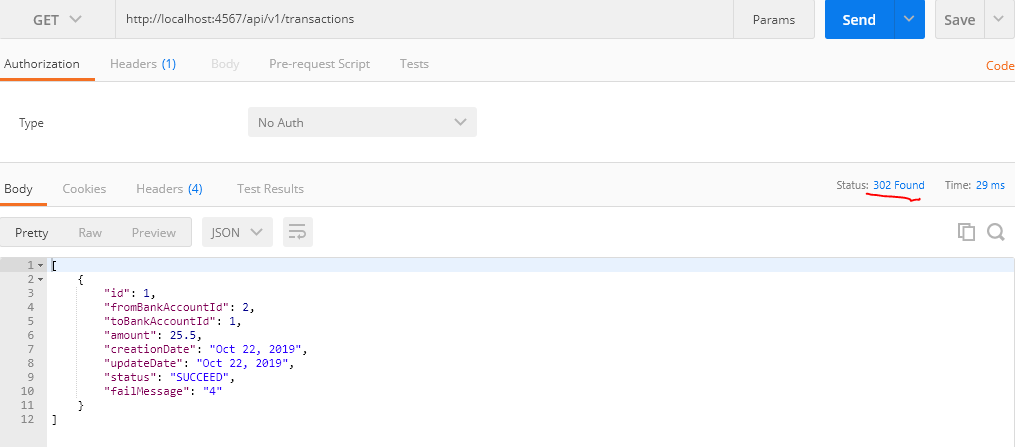
POST <http://localhost:4567/api/v1/transactions>



**4.9 Get all transactions**

HTTP GET request gets all the bank transaction that exist in the system.

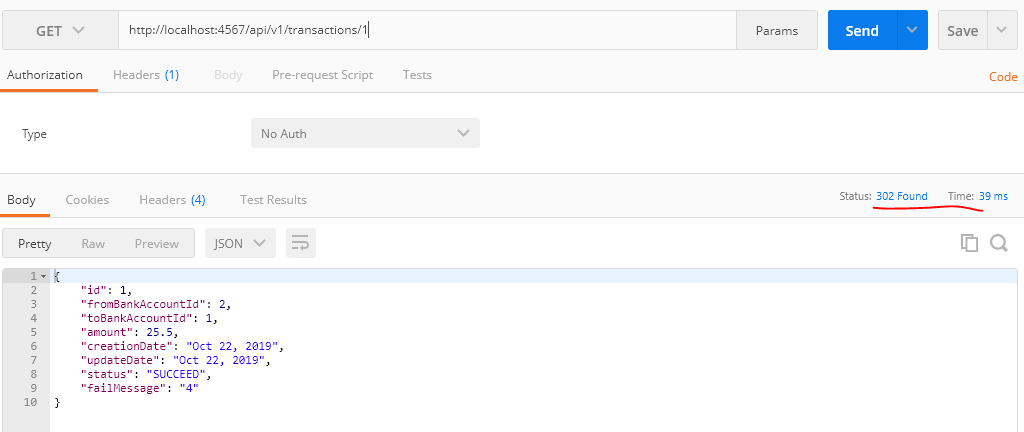
HTTP GET <http://localhost:4567/api/v1/transactions>



**4.9 Get a perticular transactions**

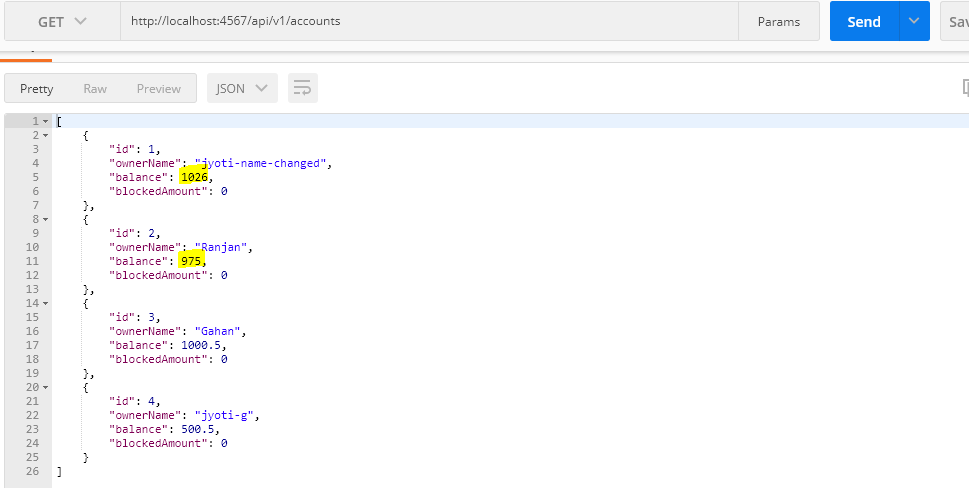
HTTP GET request gets the particular transaction if it exists in the system

HTTP GET <http://localhost:4567/api/v1/transactions/1>

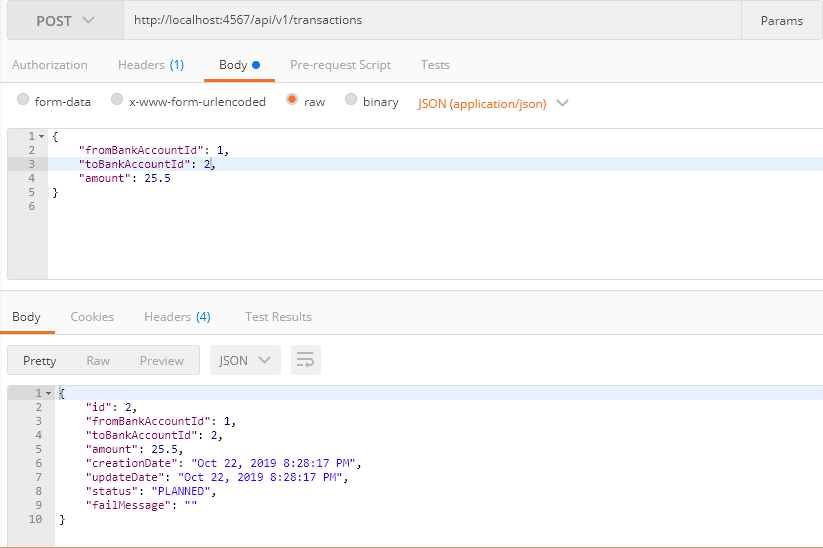


**4.9 Use Case: How to return money to sender ( ref above transaction)**

1. **First check the bank balance of both the party**



1. **Return money to Account holder having Account Id#1**



1. **Check the bank balance of both the part**

You can see amount 25.5 transferred from account holder having id# 2 to id#1

