

|  |  |
| --- | --- |
| Group 3: re-ordering | Max Cornilly  Student Number: s0191229  max.cornilly@student.uantwerpen.be |

|  |
| --- |
| **6 – Distributed systems**  **Practicum Report – Lab 2** |

Contents

[REST-based server-client banking application 3](#_Toc100064442)

[Code 3](#_Toc100064443)

[Testing REST-commands using Postman 4](#_Toc100064444)

[GET command 4](#_Toc100064445)

[POST command 4](#_Toc100064446)

[PUT command 4](#_Toc100064447)

[DELETE command 4](#_Toc100064448)

[Remote nodes 4](#_Toc100064449)

[Github 5](#_Toc100064450)

# REST-based server-client banking application

## Code

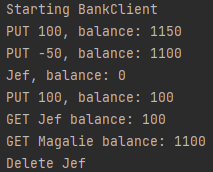
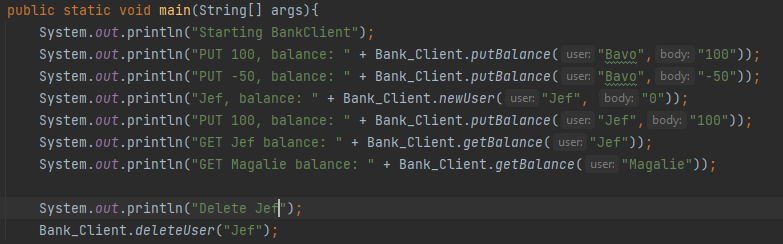
Our task for this lab session was to create a REST-based server-client banking application. We created this using the Maven framework. The application needed to have a set of requirements. We implemented all the requirement so that the client can get their balance from account (GET), add and get money from the account (PUT), adding a user (POST), and delete a user (DELETE). We implemented these on both server and client side using the java.net.http and com.sun.net.httpserver libraries. We have a Bank\_Client class in which we test some of our functionalities. We recreate all the rest commands using the REST\_Client class.

Figure 2: Bank\_Client result

Figure 1: Bank\_Client Test

To enable these REST-commands between client and server we created an abstract class REST\_Handler. We did this because we can separate the banking logic in a different class. This means when the client sends a REST-command, the REST\_Handler will invoke the getURI method in the Bank\_Server class.

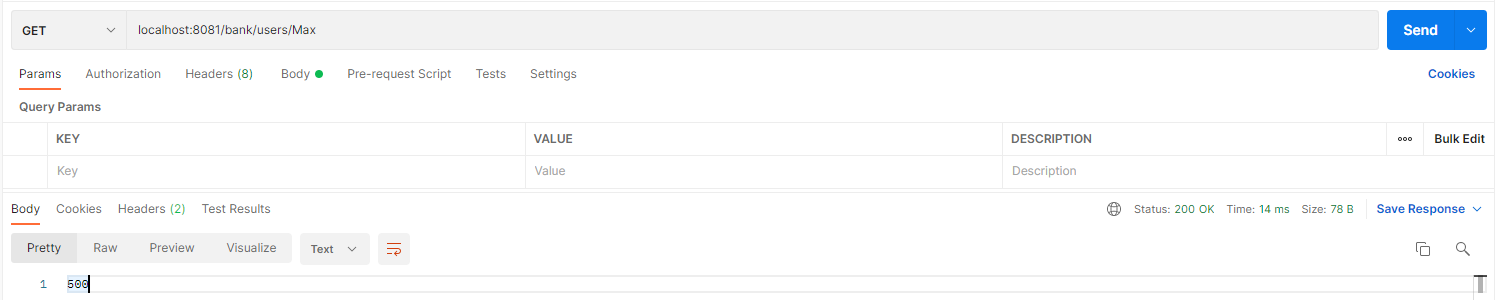
Another requirement was to have a joint account. This was a simple addition to our code because we used a Hashmap to link user and account.



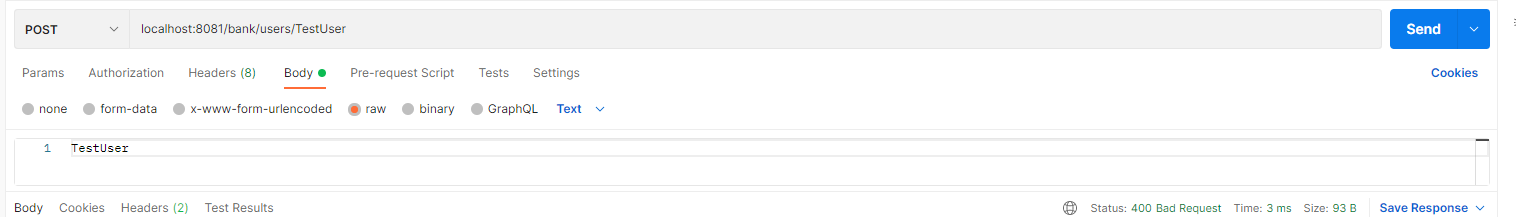
Figure 3: Joint Account

## Testing REST-commands using Postman

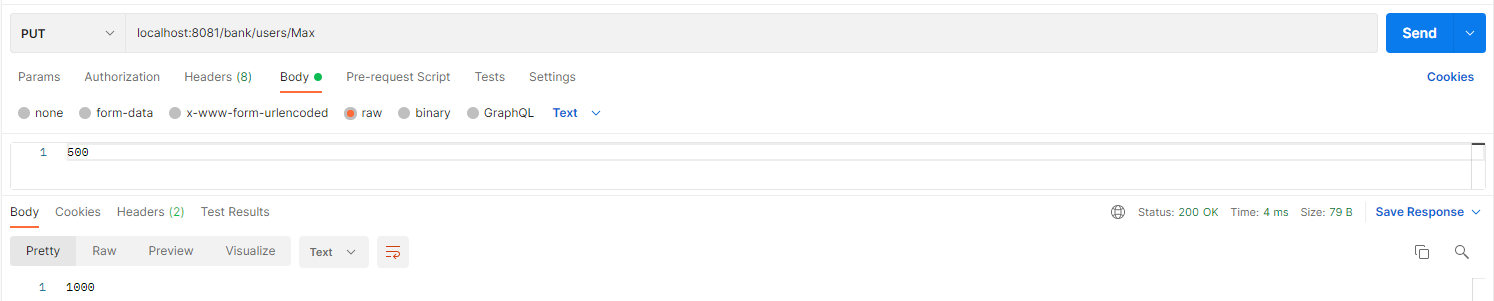
### GET command



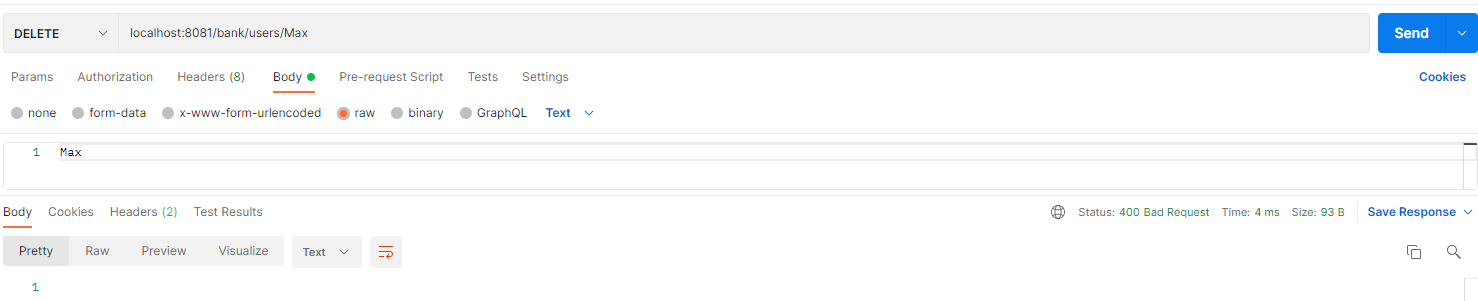
### POST command



### PUT command



### DELETE command



## Remote nodes

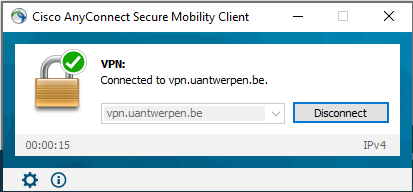
The next step was to connect to the remote hosts. To connect to them we need to be connected with the UAnwterpen VPN. I did this using Cisco Annyconnect application.

Figure 4: VPN

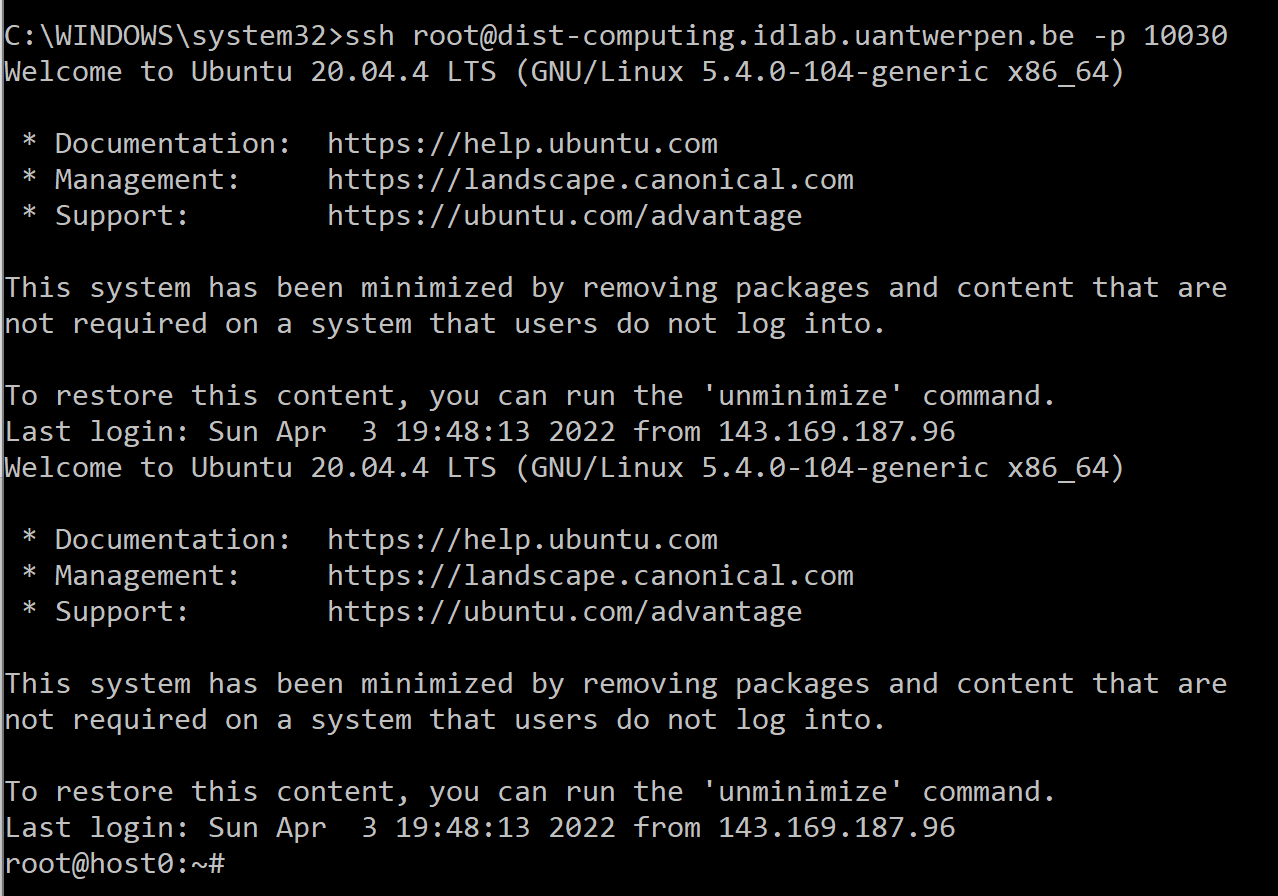
Then we need to ssh to the remote nodes using our terminal. Using this command (host0):

Figure 5: Remote node acces

|  |  |
| --- | --- |
| *ssh root@dist-computing.idlab.uantwerpen.be -p 10030* |  |
| This works because I gave my public. |  |

To run the REST server on the nodes we worked as a team. We uploaded one of our applications to one node and accessed it all to test. To put our application on the node we created a .jar file that we then later uploaded to a public repository in GitHub. Then we can pull the .jar file on the remote node with *git clone.* This is done on node 1 so the other nodes can act as client.

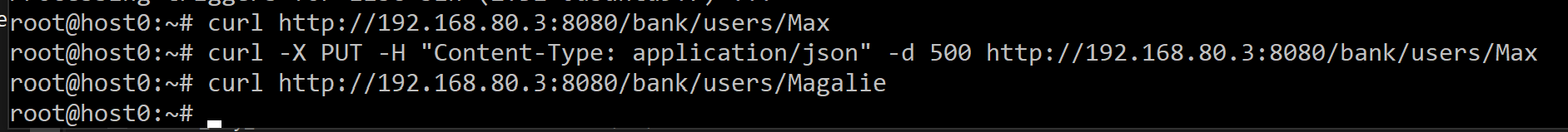
Before I can continue, I need to make sure our server is running on host1. After that I can start doing some REST-commands.

GET: curl http://192.168.80.3:8080/bank/users/Max

PUT: curl -X PUT -d ‘500’ http://192.168.80.3:8080/bank/users/Max

DELETE: curl -X DELETE http://192.168.80.3:8080/bank/users/Max

POST: curl -X POST -H "Content-Type: application/json" -d ‘500’ http://192.168.80.3:8080/bank/users/Max



# Github

https://github.com/mcornilly/Distr-Lab2-master-Max/tree/master/Lab2\_Max\_Cornilly/src/main/java/BankApp