

|  |  |
| --- | --- |
| Group number: 7 | Mohammad Wasefi  Student Number: s0156154  Mohammad.wasefi@student.uantwerpen.be |

|  |
| --- |
| **6 – Distributed Systems**  **Practicum Report – Sessions 2 and 3** |

Contents

[Session 2: REST 3](#_Toc70977110)

[Server 3](#_Toc70977111)

[Deposit 3](#_Toc70977112)

[Withdraw 3](#_Toc70977113)

[Get Balance 4](#_Toc70977114)

[GetEntry 4](#_Toc70977115)

[Spring Boot 4](#_Toc70977116)

[Client 5](#_Toc70977117)

[Deposit 5](#_Toc70977118)

[Withdraw 5](#_Toc70977119)

[Get balance. 5](#_Toc70977120)

[Multi-threading 6](#_Toc70977121)

[Run on remote server. 6](#_Toc70977122)

[Create jar file. 6](#_Toc70977123)

[Transfer to remote node 7](#_Toc70977124)

[Test on remote node 9](#_Toc70977125)

[Session 3: Naming Server 12](#_Toc70977126)

[Hash function 12](#_Toc70977127)

[Add node. 13](#_Toc70977128)

[Add node with existing node name. 14](#_Toc70977129)

[Send a filename and the IP address. 14](#_Toc70977130)

[Send a filename with a hash smaller than the smallest hash of the nodes. 14](#_Toc70977131)

[Send a filename with filename and at the same time remove the node. 14](#_Toc70977132)

[Ask from two PCs for an IP address of a filename. 14](#_Toc70977133)

[Questions 14](#_Toc70977134)

[GitHub Repo 16](#_Toc70977135)

[REST 16](#_Toc70977136)

[Naming server 16](#_Toc70977137)

[Bibliography 17](#_Toc70977138)

# Session 2: REST

In this project we explore REST which uses HTTP protocol. On the server side I use [Spring Boot](https://spring.io/projects/spring-boot) framework in Java which listens for port 80. On client side I use [Postman](https://www.postman.com/).

## Server

The server runs a simple bank application where one can deposit, withdraw, and get balance from the account.

### Deposit

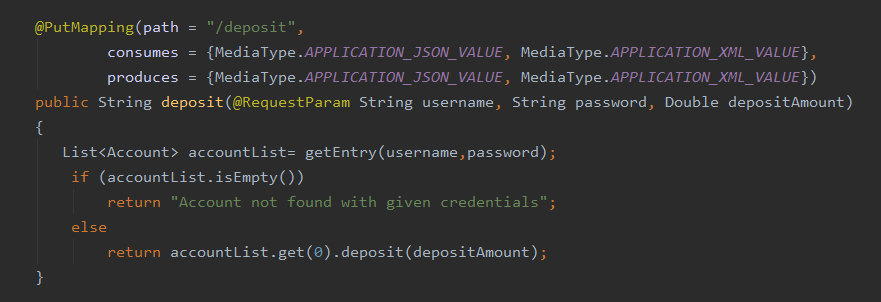


Figure 1: I use a single database which holds Hashmap of String (key) and Accounts (value). Key is the unique username of the account. The account is retrieved by getEntry(username, password) method.

### Withdraw

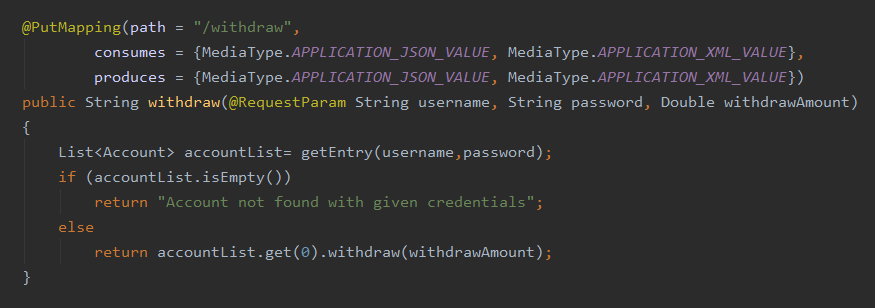


Figure 2: withdraw also uses getEntry(username, password) to retrieve the account from the database. It is then used to withdraw money from.

### Get Balance

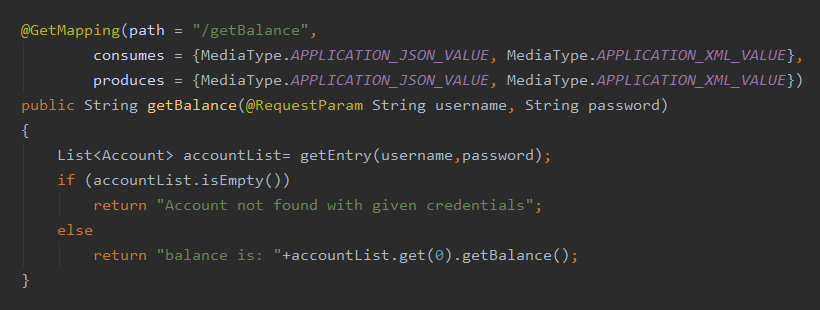


Figure 3: this method is like deposit/withdraw. It uses getEntry(username, password)

### GetEntry

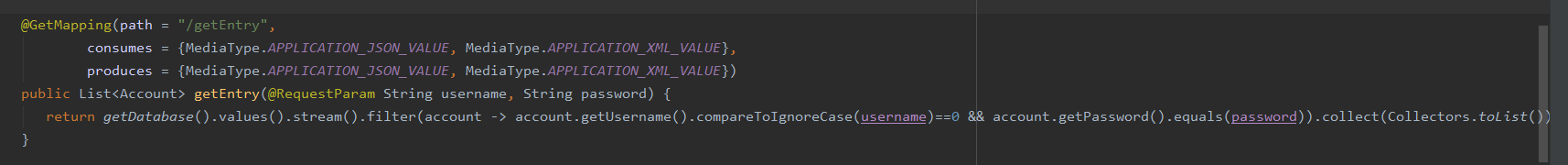


Figure 4: this method returns a list of all accounts which match with the given username and password. Of course, this would return a single account if any.

### Spring Boot

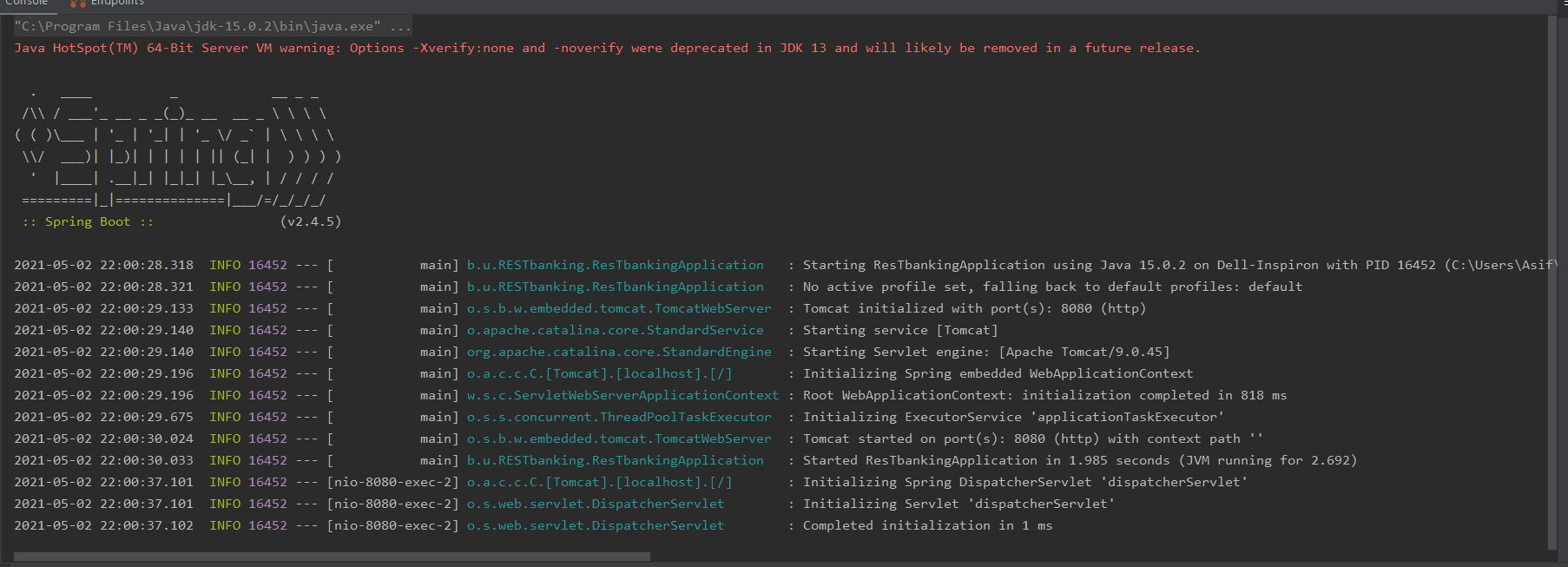


Figure 5: Spring boot server running on Java in IntelliJ IDE. The server listens to REST requests on HTTP port.

## Client

### Deposit

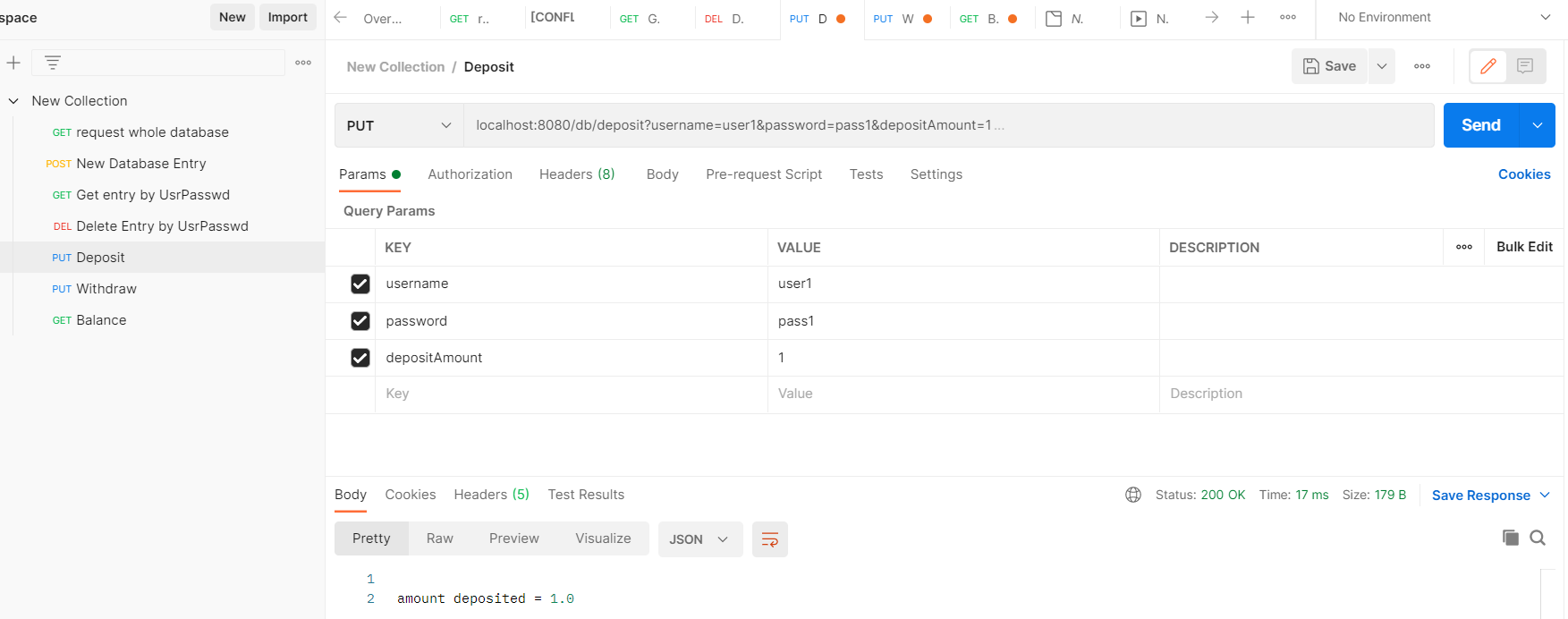


Figure 6: on client side (=Postman) one can deposit some amount on a bank account using PUT request. The deposit only succeeds if the username and password match.

### Withdraw

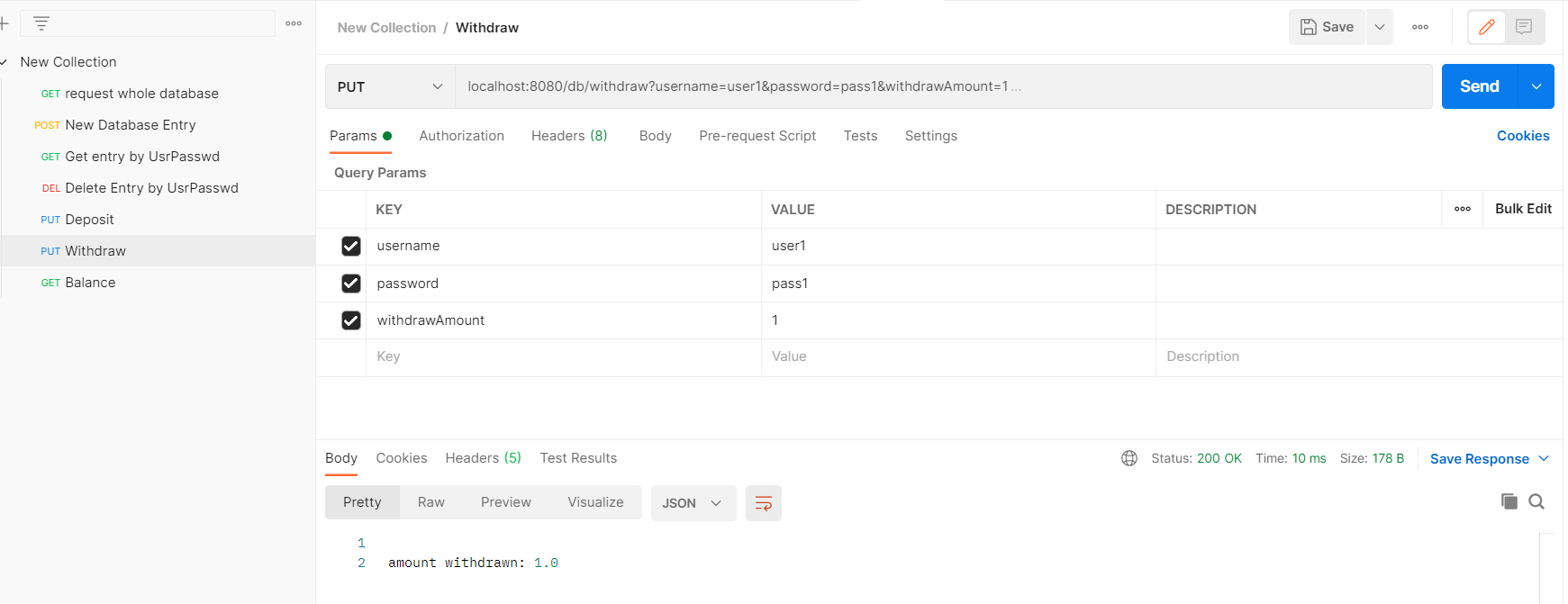


Figure 7: withdraw option uses PUT request. The authentication happens via username and password combination. Any amount higher than the balance cannot be withdrawn.

### Get balance.

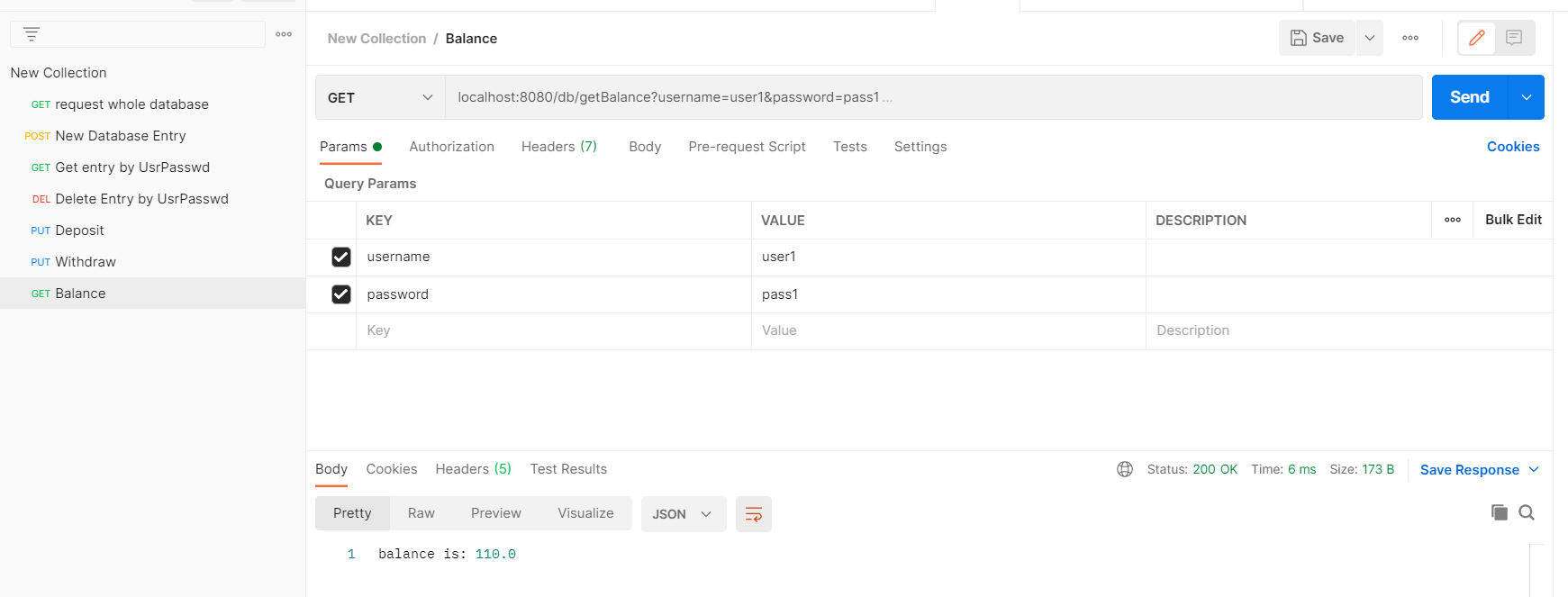


Figure 8: this GET request returns the balance of a bank account. Username and password are needed for login.

### Multi-threading

|  |  |
| --- | --- |
| Figure 9: Here I use 10 deposit and 10 withdraw threads. These threads access the same account. The access to critical sections (withdraw and deposit methods) is protected by semaphores. |  |

## Run on remote server.

### Create jar file.

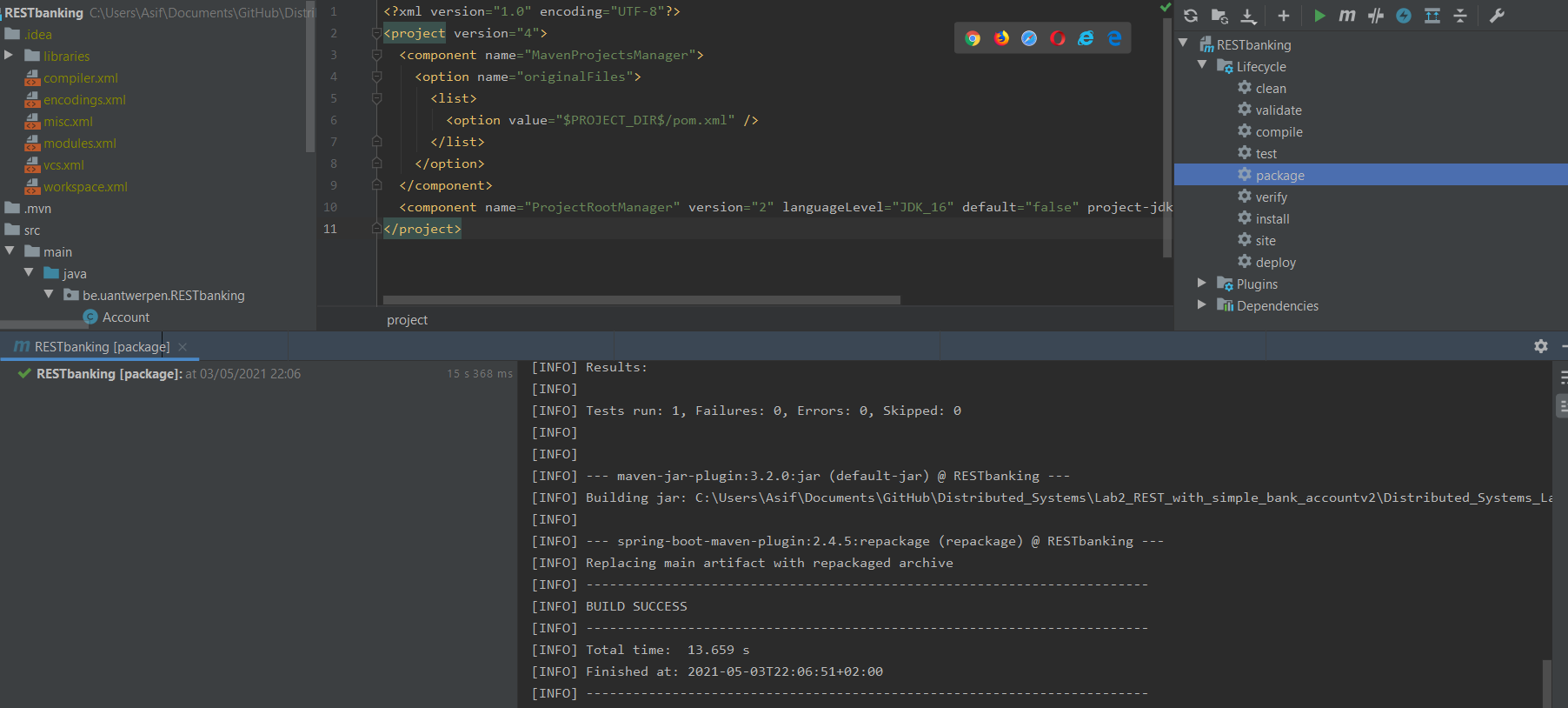


Figure 10: we create a jar file by using MAVEN packaging. This will produce a jar file in target directory.

### Transfer to remote node

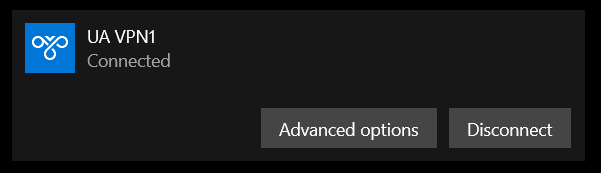


Figure 11: to access the remote server we need to connect to UA network via VPN.

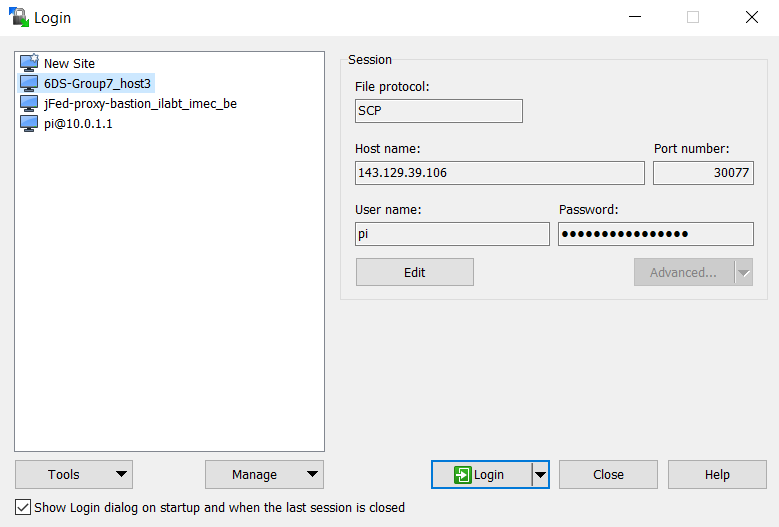


Figure 12: we use WinSCP to transfer files via SSH. The authentication details are already filled in.

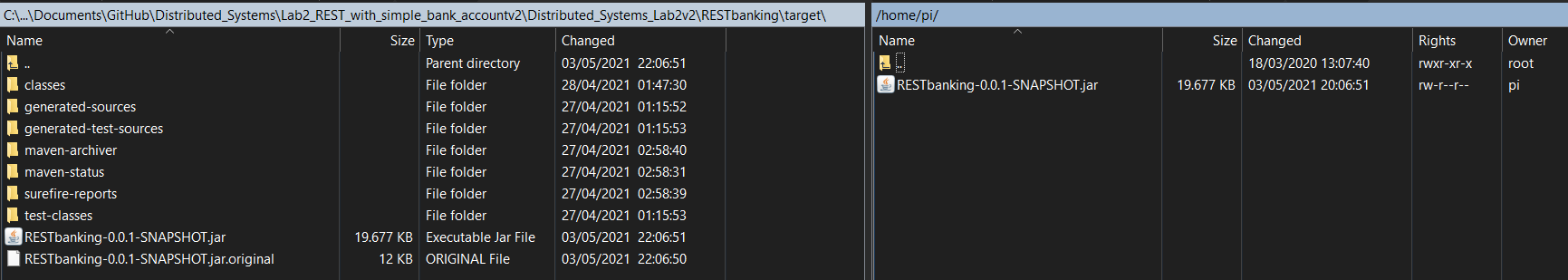


Figure 13: the jar file is being copied to remote server.

### Test on remote node

#### Server

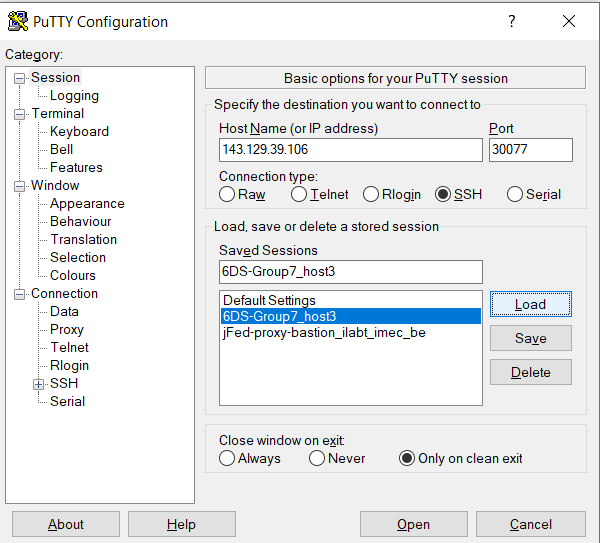


Figure 14: to connect to remote server an SSH connection can be established via SSH client such as PUTTY, etc.

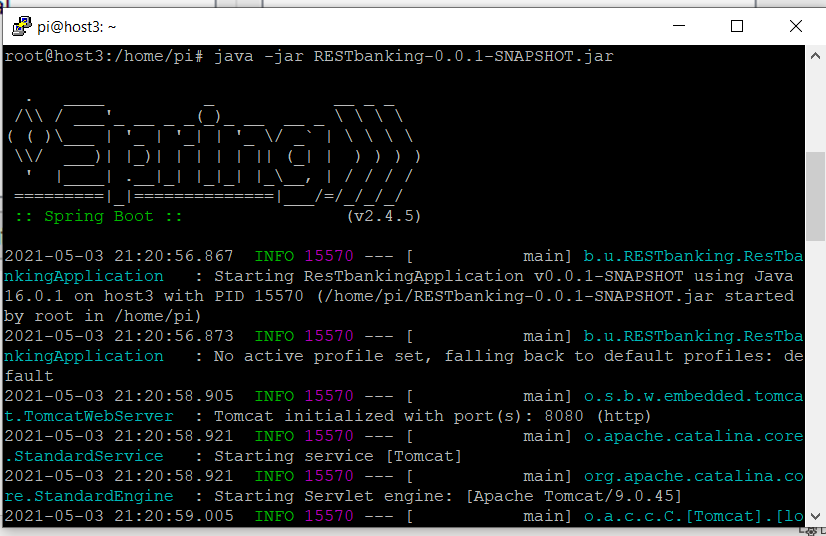


Figure 15: now we can run the project by “java -jar project.jar”

#### Client

##### GET fetch database.

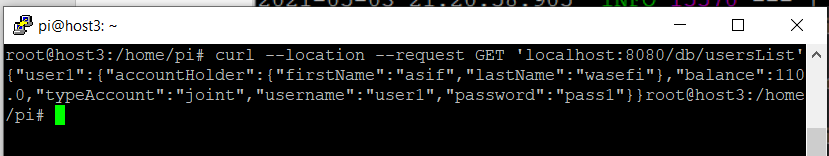


Figure 16: this cURL request is generated by POSTMAN. This retrieves the whole database.

##### POST add.

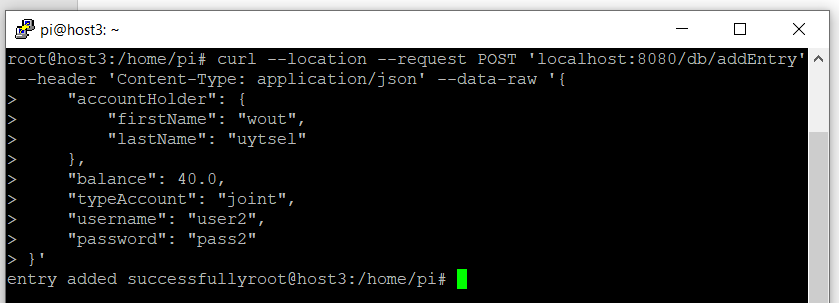


Figure 17: we use a POST request to add a new entry to the database.

##### GET fetch.

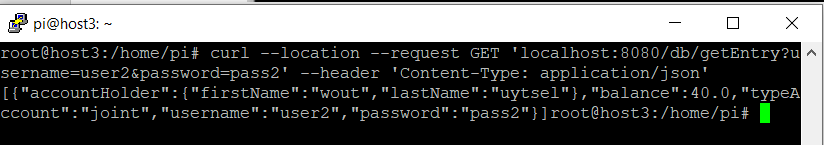


Figure 18: to GET an entry one must provide a username and password. An error message is returned upon invalid credentials.

##### DELETE remove.

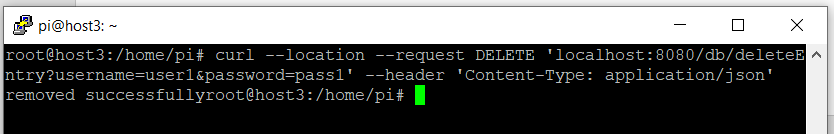


Figure 19: similarly deleting an entry needs username and password authentication.

##### PUT deposit.

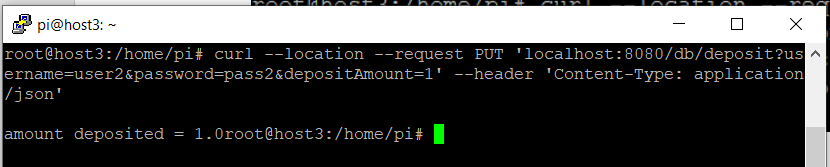


Figure 20: deposit also uses username and password otherwise an error is returned.

##### PUT withdraw.

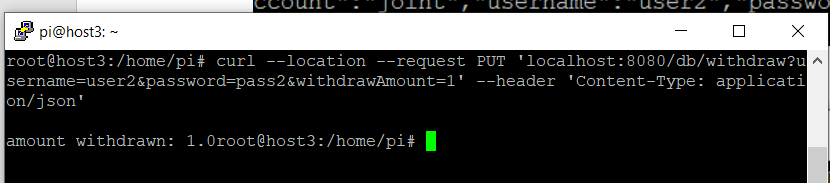


Figure 21: if withdraw amount is higher than balance an error is returned. The authentication happens by username and password.

##### GET Balance

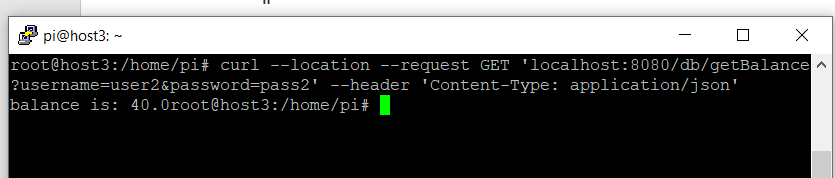


Figure 22: to get balance an authentication is needed.

# Session 3: Naming Server

## Hash function

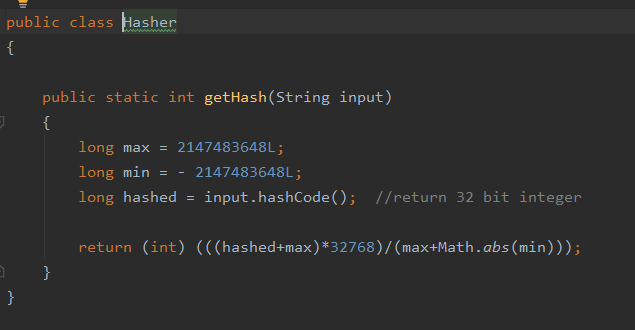


Figure 23: the hashing function maps values from min to max to range (0 to 32768)

## Add node.



Figure 24: adding a new node needs node name, a list of files(s) it hosts, and IP address of the node. The name of node will be hashed and saved as a key in the hostDatabase HashMap (value= node itself). The localFileDatabase is used for files where the key= hash of filename and value is hash of hosting node name.

## Add node with existing node name.



Figure 25: if the node already exists (=the node name hash already exists) the method will return false and add this node to the database (=overwriting the existing one).

## Send a filename and the IP address.

## Send a filename with a hash smaller than the smallest hash of the nodes.

## Send a filename with filename and at the same time remove the node.

## Ask from two PCs for an IP address of a filename.

# Questions

1. Explain the steps on how you managed to push your code to remote repository on GitHub.

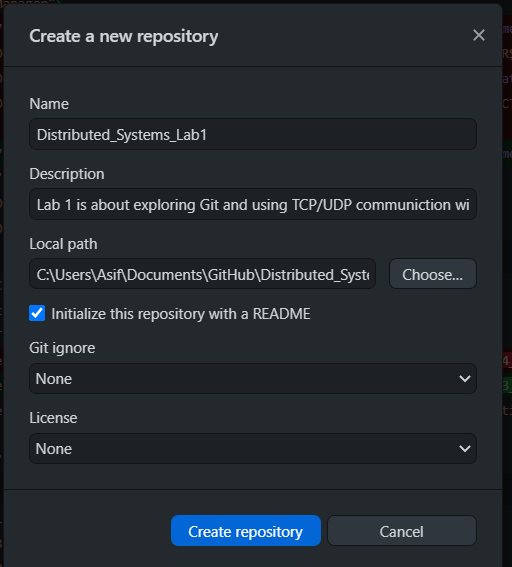


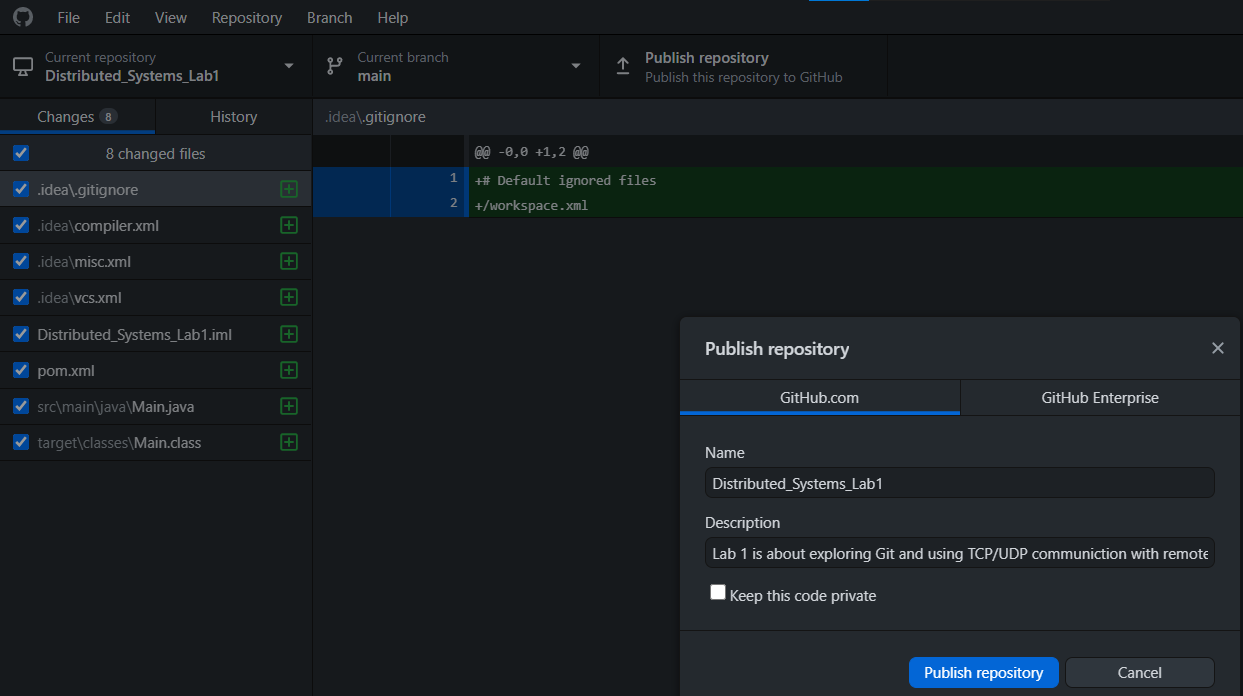
Figure 26: create a local repository on GitHub GUI. 

Figure 27: after creating the repository this can be made public by publishing it to remote repository.

1. What framework did you used to develop client-server application that communicates via TCP?

I used [Spring Boots](https://spring.io/projects/spring-boot) framework on server side which functions as a server. On client side I use [Postman](https://www.postman.com/).

1. Explain how you enabled multithreading.

I made a class ClientThread which implement extends Thread superclass. In the overridden “run” method of the classes I withdraw/deposit to an account whose username and password are passed as parameters. The deposit/withdraw amount is also given as parameter. The threads are created in Main class and run by Thread.start() method.

# GitHub Repo

## REST

<https://github.com/asifwasefi/Distributed_Systems_Lab2v2>

## Naming server

<https://github.com/TissieVA/Distributed-Systems>

# Bibliography

<https://spring.io/projects/spring-boot>

<https://www.postman.com/>

<https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Semaphore.html>

<https://www.geeksforgeeks.org/multithreading-in-java/>

<https://maven.apache.org/guides/getting-started/index.html#What_is_Maven>

<https://www.geeksforgeeks.org/multithreading-in-java/>

<https://www.oracle.com/java/technologies/javase-jdk16-downloads.html>

<https://phoenixnap.com/kb/how-to-install-java-ubuntu>