6 – Distributed Systems

Practicum Report – Sessions 2 and 3

Mohammad Wasefi
Student Number: s0156154
Mohammad.wasefi@student.uantwerpen.be

Group number: 7

Contents

Session 2: REST

In this project we explore REST which uses HTTP protocol. On the server side I use Spring Boot framework in Java which listens for port 80. On client side I use Postman.

Server

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

Deposit

```
@PutMapping(path = "/deposit",
        consumes = {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE},
        produces = {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE})
public String deposit(@RequestParam String username, String password, Double depositAmount)
{
    List<Account> accountList= getEntry(username,password);
     if (accountList.isEmpty())
         return "Account not found with given credentials";
     else
         return accountList.get(0).deposit(depositAmount);
}
```

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

```
@PutMapping(path = "/withdraw",
        consumes = {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE},
        produces = {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE})
public String withdraw(@RequestParam String username, String password, Double withdrawAmount)
{
    List<Account> accountList= getEntry(username,password);
    if (accountList.isEmpty())
        return "Account not found with given credentials";
    else
        return accountList.get(0).withdraw(withdrawAmount);
}
```

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

```java
@GetMapping(path = "/getBalance",
        consumes = {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE},
        produces = {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE})
public String getBalance(@RequestParam String username, String password)
{
    List<Account> accountList= getEntry(username,password);
    if (accountList.isEmpty())
        return "Account not found with given credentials";
    else
        return "balance is: "+accountList.get(0).getBalance();
}
```

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

GetEntry

```java
@GetMapping(path = "/getEntry",
        consumes = {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE},
        produces = {MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE})
public List<Account> getEntry(@RequestParam String username, String password) {
    return getDatabase().values().stream().filter(account -> account.getUsername().compareToIgnoreCase(username)==0 && account.getPassword().equals(password)).collect(Collectors.toList())
}
```

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

```
"C:\Program Files\Java\jdk-15.0.2\bin\java.exe" ...
Java HotSpot(TM) 64-Bit Server VM warning: Options -Xverify:none and -noverify were deprecated in JDK 13 and will likely be removed in a future release.

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.4.5)

2021-05-02 22:00:28.318  INFO 16452 --- [           main] b.u.RESTbanking.ResTbankingApplication   : Starting ResTbankingApplication using Java 15.0.2 on Dell-Inspiron with PID 16452 (C:\Users\Asif\
2021-05-02 22:00:28.321  INFO 16452 --- [           main] b.u.RESTbanking.ResTbankingApplication   : No active profile set, falling back to default profiles: default
2021-05-02 22:00:29.133  INFO 16452 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2021-05-02 22:00:29.140  INFO 16452 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2021-05-02 22:00:29.140  INFO 16452 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.45]
2021-05-02 22:00:29.196  INFO 16452 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2021-05-02 22:00:29.196  INFO 16452 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 818 ms
2021-05-02 22:00:29.675  INFO 16452 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2021-05-02 22:00:30.024  INFO 16452 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2021-05-02 22:00:30.033  INFO 16452 --- [           main] b.u.RESTbanking.ResTbankingApplication   : Started ResTbankingApplication in 1.985 seconds (JVM running for 2.692)
2021-05-02 22:00:37.101  INFO 16452 --- [nio-8080-exec-2] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2021-05-02 22:00:37.101  INFO 16452 --- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2021-05-02 22:00:37.102  INFO 16452 --- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
```

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

Client

Deposit

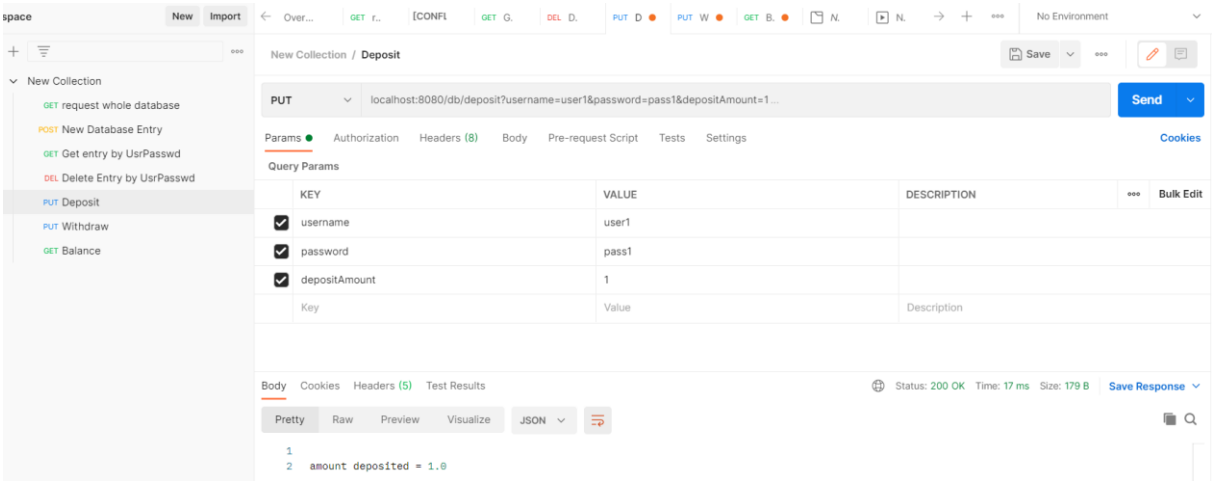

Figure 6: on client side (=Postman) one can deposit some amount on a bank account using PUT request. The deposit only succeeds if the username and password match.

Withdraw

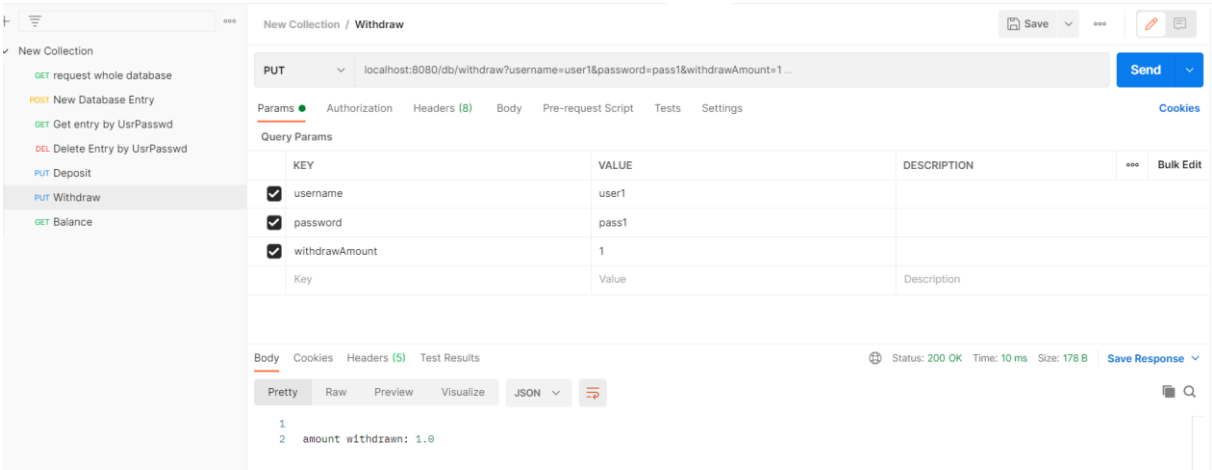

Figure 7: withdraw option uses PUT request. The authentication happens via username and password combination. Any amount higher than the balance cannot be withdrawn.

Get balance.

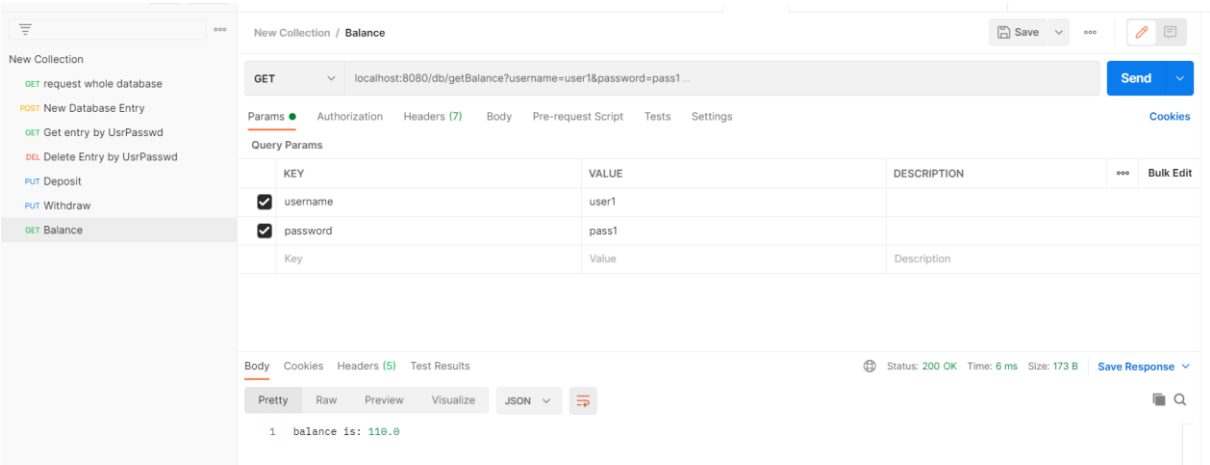

Figure 8: this GET request returns the balance of a bank account. Username and password are needed for login.

Multi-threading

```
Account in use, cannot deposit
Account in use, cannot withdraw
Account in use, cannot withdraw
Account in use, cannot withdraw
Account in use, cannot withdraw
Account in use, cannot withdraw
Account in use, cannot withdraw
Account in use, cannot withdraw
Account in use, cannot withdraw
Account in use, cannot withdraw
Account in use, cannot withdraw
104.0
104.0
Account in use, cannot withdraw
Thread-3 ended
Thread-4 ended
106.0
Thread-6 ended
108.0
Thread-5 ended
110.0
Thread-7 ended
112.0
Thread-8 ended
114.0
Thread-9 ended
116.0
Thread-10 ended
```

```
Thread-10 ended
118.0
Thread-11 ended
120.0
Thread-12 ended
118.0
Thread-14 ended
118.0
Thread-15 ended
117.0
Thread-16 ended
116.0
Thread-17 ended
115.0
Thread-18 ended
114.0
Thread-19 ended
113.0
Thread-20 ended
112.0
Thread-21 ended
111.0
Thread-22 ended
110.0
Thread-23 ended
```

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.

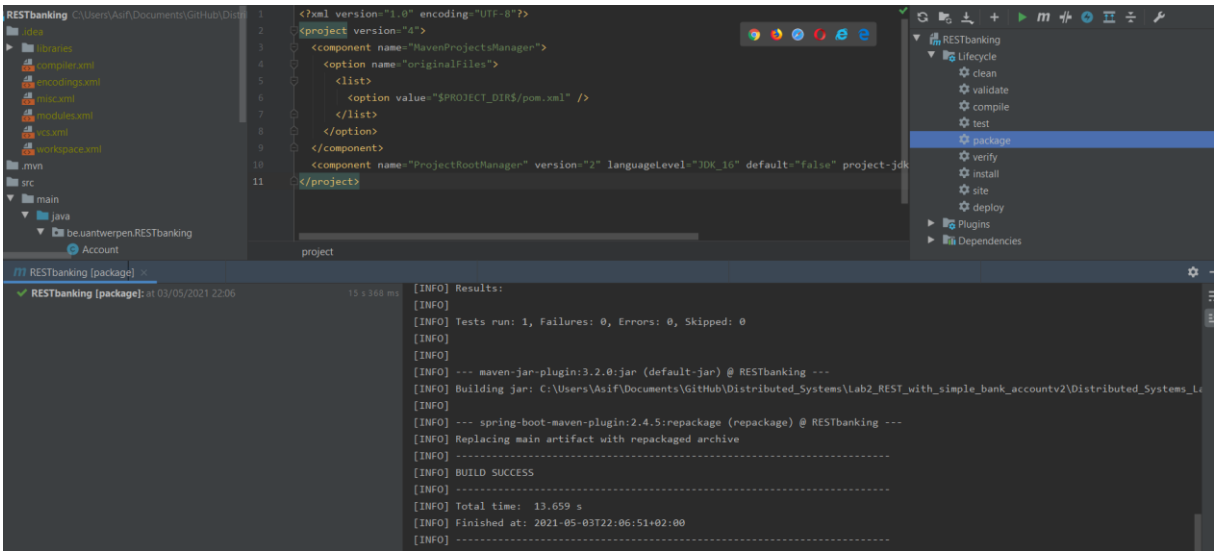

Figure 10: we create a jar file by using MAVEN packaging. This will produce a jar file in target directory.

Transfer to remote node

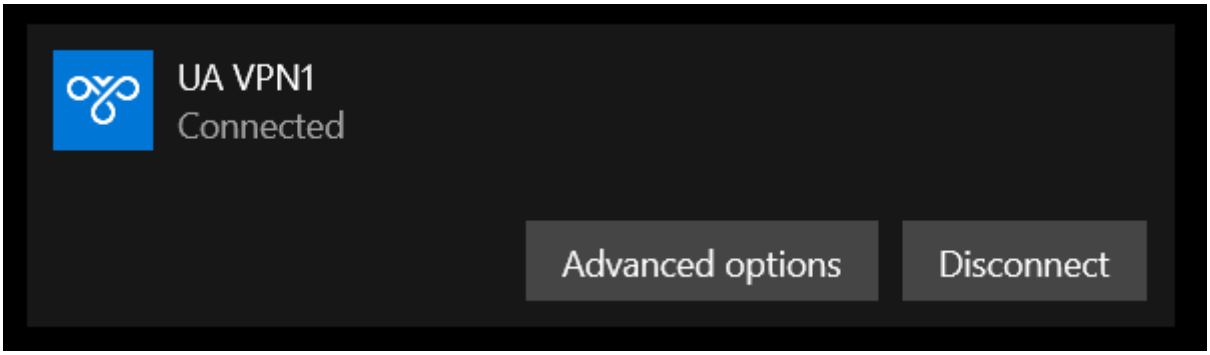

Figure 11: to access the remote server we need to connect to UA network via VPN.

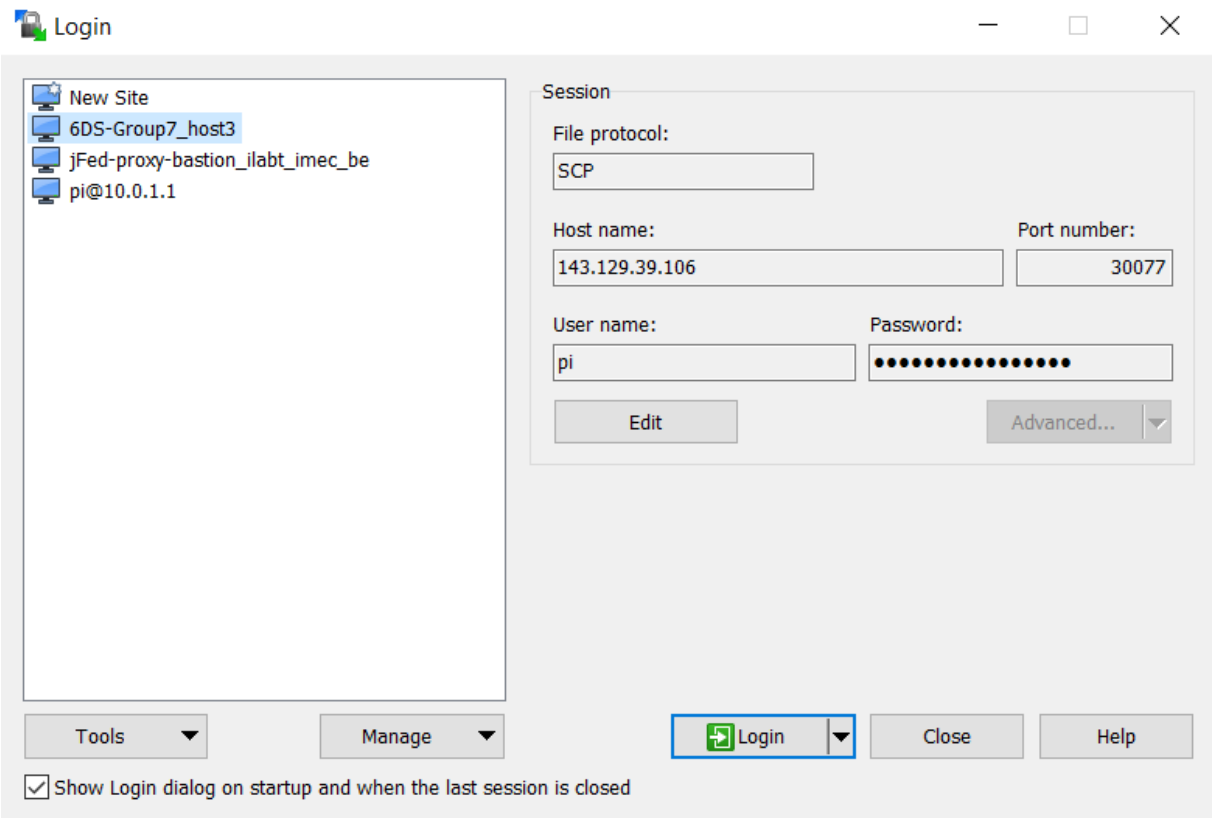

Figure 12: we use WinSCP to transfer files via SSH. The authentication details are already filled in.

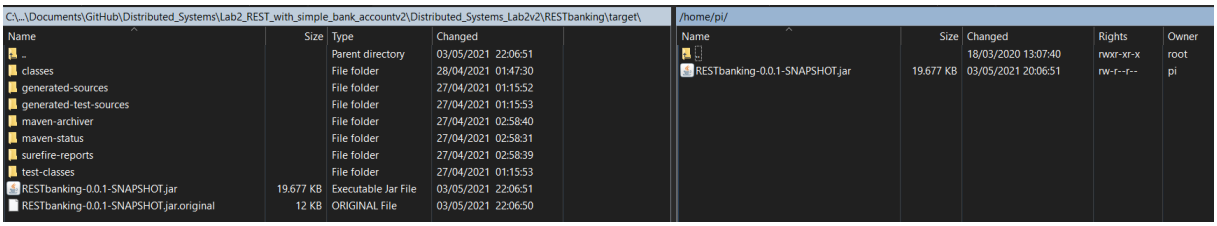

Figure 13: the jar file is being copied to remote server.

Test on remote node

Server

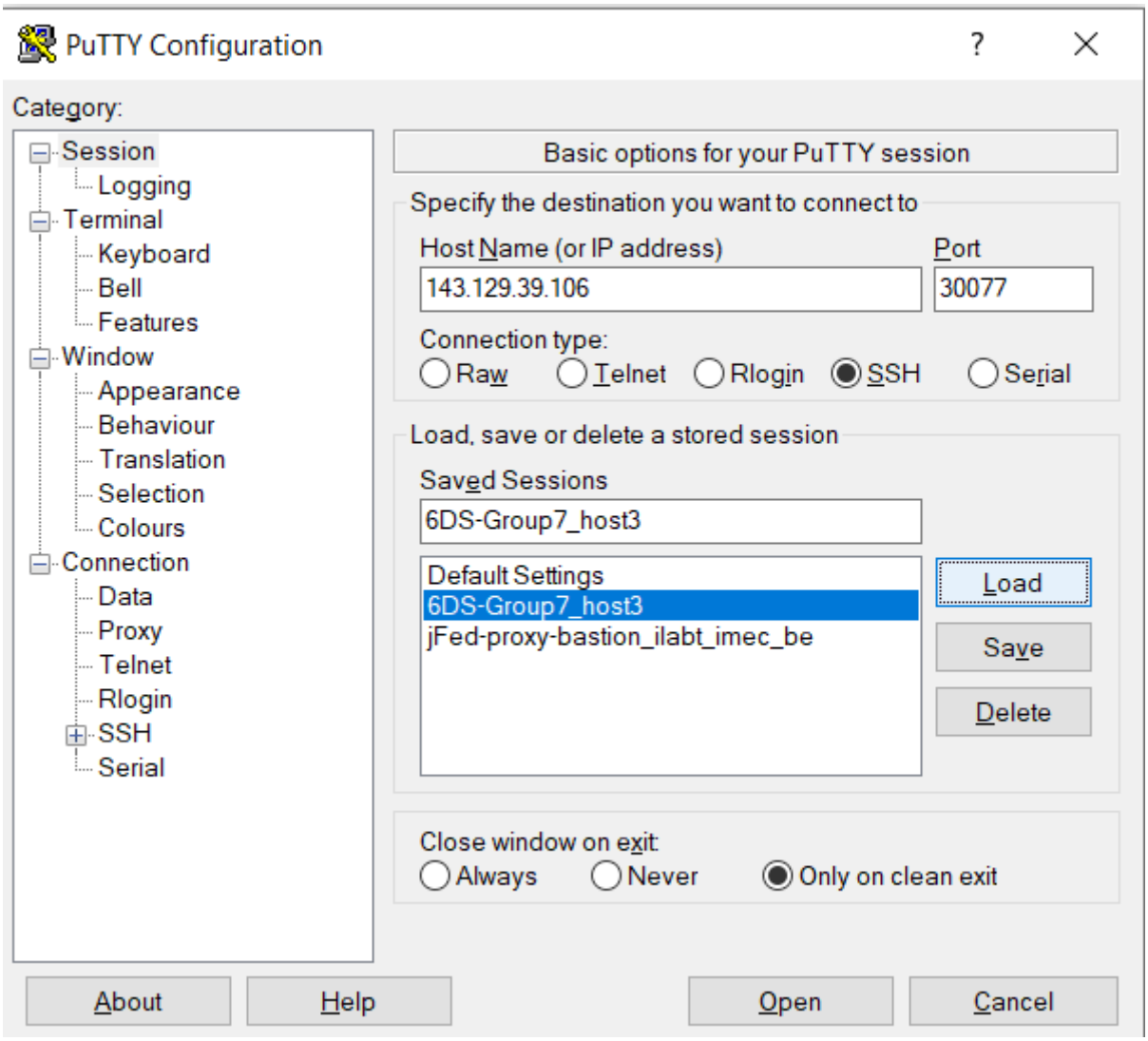

Figure 14: to connect to remote server an SSH connection can be established via SSH client such as PUTTY, etc.

```
pi@host3: ~
root@host3:/home/pi# java -jar RESTbanking-0.0.1-SNAPSHOT.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.4.5)

2021-05-03 21:20:56.867  INFO 15570 --- [           main] b.u.RESTbanking.ResTba
nkingApplication   : Starting ResTbankingApplication v0.0.1-SNAPSHOT using Java
16.0.1 on host3 with PID 15570 (/home/pi/RESTbanking-0.0.1-SNAPSHOT.jar started
by root in /home/pi)
2021-05-03 21:20:56.873  INFO 15570 --- [           main] b.u.RESTbanking.ResTba
nkingApplication   : No active profile set, falling back to default profiles: de
fault
2021-05-03 21:20:58.905  INFO 15570 --- [           main] o.s.b.w.embedded.tomca
t.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2021-05-03 21:20:58.921  INFO 15570 --- [           main] o.apache.catalina.core
.StandardService   : Starting service [Tomcat]
2021-05-03 21:20:58.921  INFO 15570 --- [           main] org.apache.catalina.co
re.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.45]
2021-05-03 21:20:59.005  INFO 15570 --- [           main] o.a.c.c.C.[Tomcat].[lo
```

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

Client

GET fetch database.

```
pi@host3: ~
root@host3:/home/pi# curl --location --request GET 'localhost:8080/db/usersList'
{"user1":{"accountHolder":{"firstName":"asif","lastName":"wasefi"},"balance":110
.0,"typeAccount":"joint","username":"user1","password":"pass1"}}root@host3:/home
/pi#
```

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

POST add.

```
pi@host3: ~
root@host3:/home/pi# curl --location --request POST 'localhost:8080/db/addEntry'
 --header 'Content-Type: application/json' --data-raw '{
>     "accountHolder": {
>         "firstName": "wout",
>         "lastName": "uytsel"
>     },
>     "balance": 40.0,
>     "typeAccount": "joint",
>     "username": "user2",
>     "password": "pass2"
> }'
entry added successfullyroot@host3:/home/pi#
```

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

GET fetch.

```
pi@host3: ~
root@host3:/home/pi# curl --location --request GET 'localhost:8080/db/getEntry?u
sername=user2&password=pass2' --header 'Content-Type: application/json'
[{"accountHolder":{"firstName":"wout","lastName":"uytsel"},"balance":40.0,"typeA
ccount":"joint","username":"user2","password":"pass2"}]root@host3:/home/pi#
```

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

DELETE remove.

```
pi@host3: ~
root@host3:/home/pi# curl --location --request DELETE 'localhost:8080/db/deleteE
ntry?username=user1&password=pass1' --header 'Content-Type: application/json'
removed successfullyroot@host3:/home/pi#
```

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

PUT deposit.

```
pi@host3: ~
root@host3:/home/pi# curl --location --request PUT 'localhost:8080/db/deposit?us
ername=user2&password=pass2&depositAmount=1' --header 'Content-Type: application
/json'

amount deposited = 1.0root@host3:/home/pi#
```

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

PUT withdraw.

```
pi@host3: ~
root@host3:/home/pi# curl --location --request PUT 'localhost:8080/db/withdraw?u
sername=user2&password=pass2&withdrawAmount=1' --header 'Content-Type: applicati
on/json'

amount withdrawn: 1.0root@host3:/home/pi#
```

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

GET Balance

```
pi@host3: ~
root@host3:/home/pi# curl --location --request GET 'localhost:8080/db/getBalance
?username=user2&password=pass2' --header 'Content-Type: application/json'
balance is: 40.0root@host3:/home/pi#
```

Figure 22: to get balance an authentication is needed.

Session 3: Naming Server

Hash function

```java
public class Hasher
{

    public static int getHash(String input)
    {
        long max = 2147483648L;
        long min = - 2147483648L;
        long hashed = input.hashCode();  //return 32 bit integer

        return (int) (((hashed+max)*32768)/(max+Math.abs(min)));
    }
}
```

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

Add node.

```java
public boolean addNewNode(String hostname, ArrayList<String> files, String ipAddress)
{

    if (this.hostDatabase.values().stream().anyMatch(x -> x.getIpaddress().equals(ipAddress)))
        return false;

    int hash = Hasher.getHash(hostname);

    if(this.hostDatabase.containsKey(hash))
    {
        return false;
    }

    this.hostDatabase.put(hash,new Node(hostname, ipAddress));

    System.out.println("hostname: " + hostname + "=" + hash);

    files.forEach(x -> {
        localFileDatabase.put(Hasher.getHash(x),hash);
        System.out.println(x + "=" + Hasher.getHash(x));
    });

    outputXML();

    return true;
}
```

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.

```java
public boolean addNewNode(String hostname, ArrayList<String> files, String ipAddress)
{

    if (this.hostDatabase.values().stream().anyMatch(x -> x.getIpaddress().equals(ipAddress)))
        return false;

    int hash = Hasher.getHash(hostname);

    if(this.hostDatabase.containsKey(hash))
    {
        return false;
    }

    this.hostDatabase.put(hash,new Node(hostname, ipAddress));

    System.out.println("hostname: " + hostname + "=" + hash);

    files.forEach(x -> {
        localFileDatabase.put(Hasher.getHash(x),hash);
        System.out.println(x + "=" + Hasher.getHash(x));
    });

    outputXML();

    return true;
}
```

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.

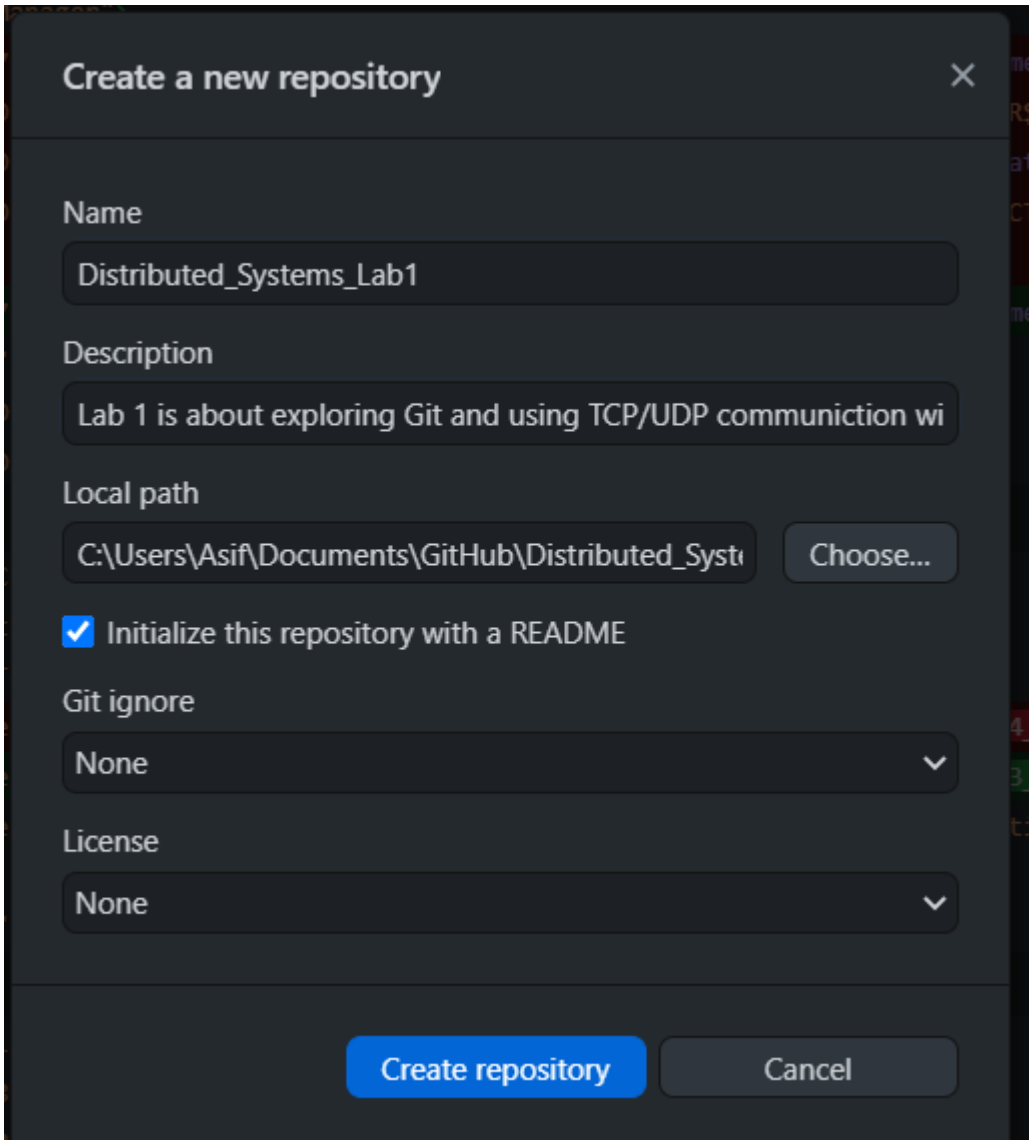

Figure 26: create a local repository on GitHub GUI.

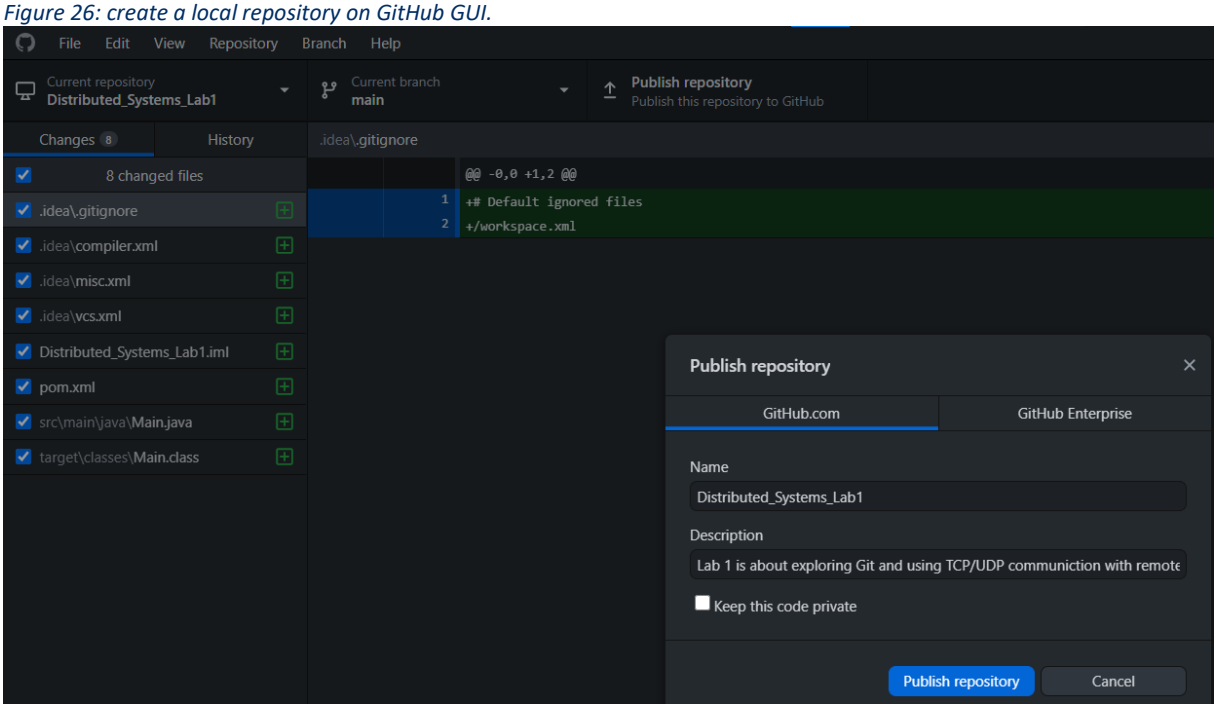

Figure 27: after creating the repository this can be made public by publishing it to remote repository.

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

   I used Spring Boots framework on server side which functions as a server. On client side I use Postman.

3. 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