Assignment 3 - RMI & DBA

Network Programming, ID1212

Bernardo Gonzalez Riede, begr@kth.se

December 3, 2017

1 Introduction

High level goals for this assignment.

- Development of application which uses remote method invocation, RMI.
- Development of application which stores persistent data on a database.

2 Literature Study

Only the videos were used as re-introduction to JDBC and RMI since I have used JDBC and RMI in the past, although that was a few years ago. Important for RMI is the creation of Interfaces to which caller has access. The callee will have an implementation of it which gets called instead of the interface. Because its a subclass, it can be used everywhere where the subclass (the interface) is expected.

In regards to the persistent storage, prepared statements are best practice and should always be used.

3 Method

As in previous assignments, the mvc approach has been used again to layer the program.

The choice for a database fell on MySQL because of past works with it. The implementation was realized in Netbeans in Java 8 EE.

4 Result

Link to public Github repository with code: https://github.com/MemBernd/ID1212-RMI-DBA

4.1 Layering

The client is divided as follows:

- A startup layer for starting the client side.
- A view layer consisting of the, now familiar, classes similar to previous assignments.

The server consist of several layers.

- The *server.controller.Controller* acts as the view of the server side, initiating the events when called by the client through RMI. The *controller* accesses the *model* layer for functionality of retrieving the reference to the client, it's username etc.
- *ClientHandler* in the *model* layer has access to the storage layer to retrieve the data.
- *DBStorage* in the *storage* layer is the only class with actual knowledge about the implementation of persistent storage, in this case MySQL.
- The server gets started by a dedicated server.startup.ServerStartup.

A additional package *common* holds the interfaces needed for RMI, a *FileHolder* class and a custom Exception.

4.2 DBA

The persistent storage solution was implemented using MySQL. Every file has the following metadata:

- A unique filename.
- Size (in bytes).
- name of the owner.
- Permissions whether its a private or public file and if read or write is allowed for public access.

In conjunction with storing username and password for a given user, the resulting database schema is as shown in 1.

To increase performance, prepared statements have been used which get initialized at the creation of the server.storage.DBStorage (l. 148+).

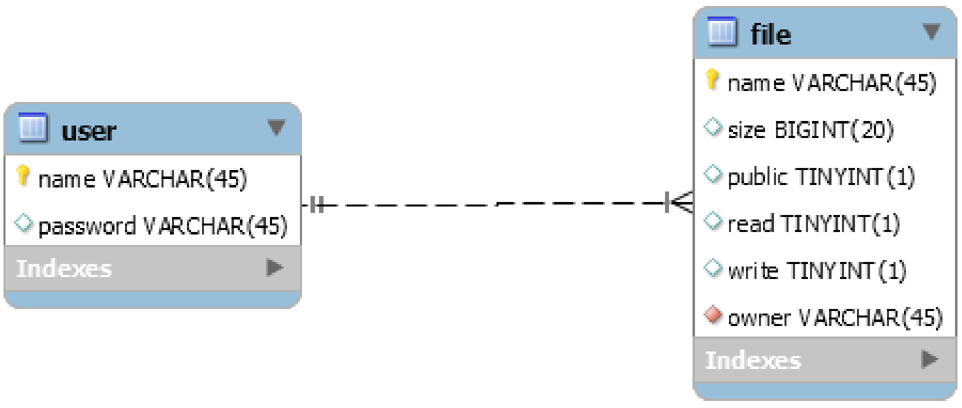


Figure 1: Table relations in database.

4.3 RMI

RMI is covered by registering an implementation of the common.Server interface in server.startup.ServerStartup, line 23. This interface and therefore its implementations include the following functions:

- *register()* to register a new user.
- *unregister()* to delete an existent user.
- *login()* for login a user in.
- *logout()* to logout.
- *list()* to list all the files the user is allowed to see.
- *upload()* to upload a new or a changed file.
- *download()* to download a file.
- *delete()* for deleting an existing file on the server.
- *addNotification()* to add a listener to changes in a file.

As for the Client, he implements common.Client which only holds functions to print something out on his command line. Its implementation client.View.Interpreter.ConsoleOutput, is passed when the user tries to login to the system and stored for future reference.

4.4 Permission management

When a user tries to login, the username and password are matched to the database in DBStorage (l. 95+). The controller uses the returned boolean (through the model) to either generate an random id (l. 54) to be returned to the client or to raise an AccessException.

This id is used as a token for the user to identify himself during the session. All other method calls to the server which require authentication require this token to be passed along.

ClientHandler (the model) holds three Hashmaps (l. 24 - 26):

- *clients* to be able to retrieve the rmi reference received when logging in.
- *clientNames* to be able to retrieve the username for owner matching of a file.
- *clientsToNotify* for knowing if there's someone who needs to be notified about a change.

The first two use the id generated as a key, while the last one uses the file's name. Because the file name is unique and only the owner can request to be notified it was done this way.

Additionally, the ClientHandler includes two methods (l. 85 - 105) to be able to check if the user is allowed to read or write a given file. If the user is not allowed to realize the operation, a *AccessException* is raised with a message explaining what failed.

4.5 Notification

An assumption has been made; only the owner of a file can request to be notified about changes in his public files. If he's the owner, an entry in the *clientsToNotify* is made. This list is checked at every successful execution of uploading, downloading or deleting a file, e.g. server.controller.Controller (l. 89).

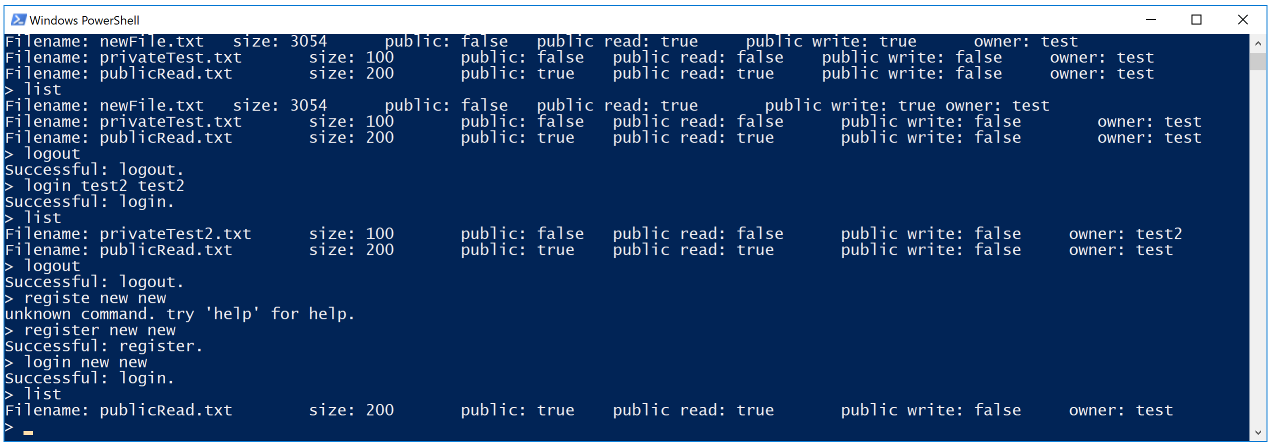
4.6 UI

Simplistic command line interface in 2.

5 Discussion

The server.controller.Controller needs to be cleaned. it executes the same sequence at the beginning of most of the methods to validate the user.

Moreover, I'm not sure about best practices regarding the handling of output from a method call to the server. I've used mostly exception which get caught by the client, but is this the ideal way to do it? Or is it better to use the reference provided by the client to write output to its command line? I'd appreciate a comment on this.



Windows PowerShell

```
Filename: newFile.txt   size: 3054      public: false   public read: true     public write: true      owner: test
Filename: privateTest.txt       size: 100       public: false   public read: false    public write: false     owner: test
Filename: publicRead.txt        size: 200       public: true    public read: true     public write: false     owner: test
> list
Filename: newFile.txt   size: 3054      public: false   public read: true       public write: true owner: test
Filename: privateTest.txt       size: 100       public: false   public read: false      public write: false        owner: test
Filename: publicRead.txt        size: 200       public: true    public read: true       public write: false        owner: test
> logout
Successful: logout.
> login test2 test2
Successful: login.
> list
Filename: privateTest2.txt      size: 100       public: false   public read: false      public write: false     owner: test2
Filename: publicRead.txt        size: 200       public: true    public read: true       public write: false     owner: test
> logout
Successful: logout.
> registe new new
unknown command. try 'help' for help.
> register new new
Successful: register.
> login new new
Successful: login.
> list
Filename: publicRead.txt        size: 200       public: true    public read: true       public write: false     owner: test
>
```

Figure 2: Several user listing files.

6 Comments About the Course

The time invested in this project were almost 12 hours.

- 7h 35 min for coding.
- 1h 50 min for the report.
- 2h 40 min watching videos, taking notes and thinking about how to tackle the problem.

Albeit being intimidated about the task at the beginning, because of the discussion about the length of the program, I'm surprised to find out that the time needed was actually lower than in previous assignment. Of course this is in regard to the work submitted, so the outcome will determine if it was indeed easier.

Disclaimer: I have been shown RMI and DBA to MySQL around 3 years before, resulting in an almost immediate understanding of the example code.