Parallel & Distributed Processing

JDBC — Setting up The Database [A2]

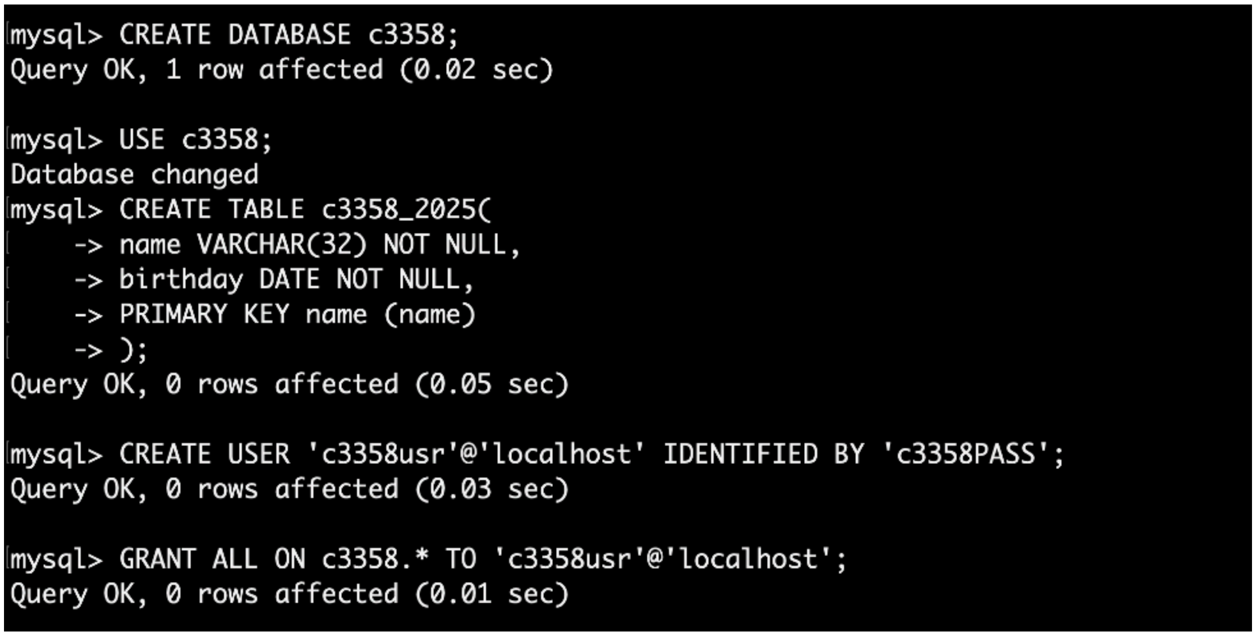
The provided Java application allows for user login/registration, saving user records to a MySQL Database.

Setup & Execution [Part 1]

1. Download the mysql-connect-j plugin forJDBC. See assignment handout for details.

2. Download and install MySQL (server and client—see assignment handout for further details).

3. Setup the database as shown in the figure below. The schema for the table (along with the other needed fields) can be seen in the figure below. The fields shown in the figure can be edited to your liking in the script provided.



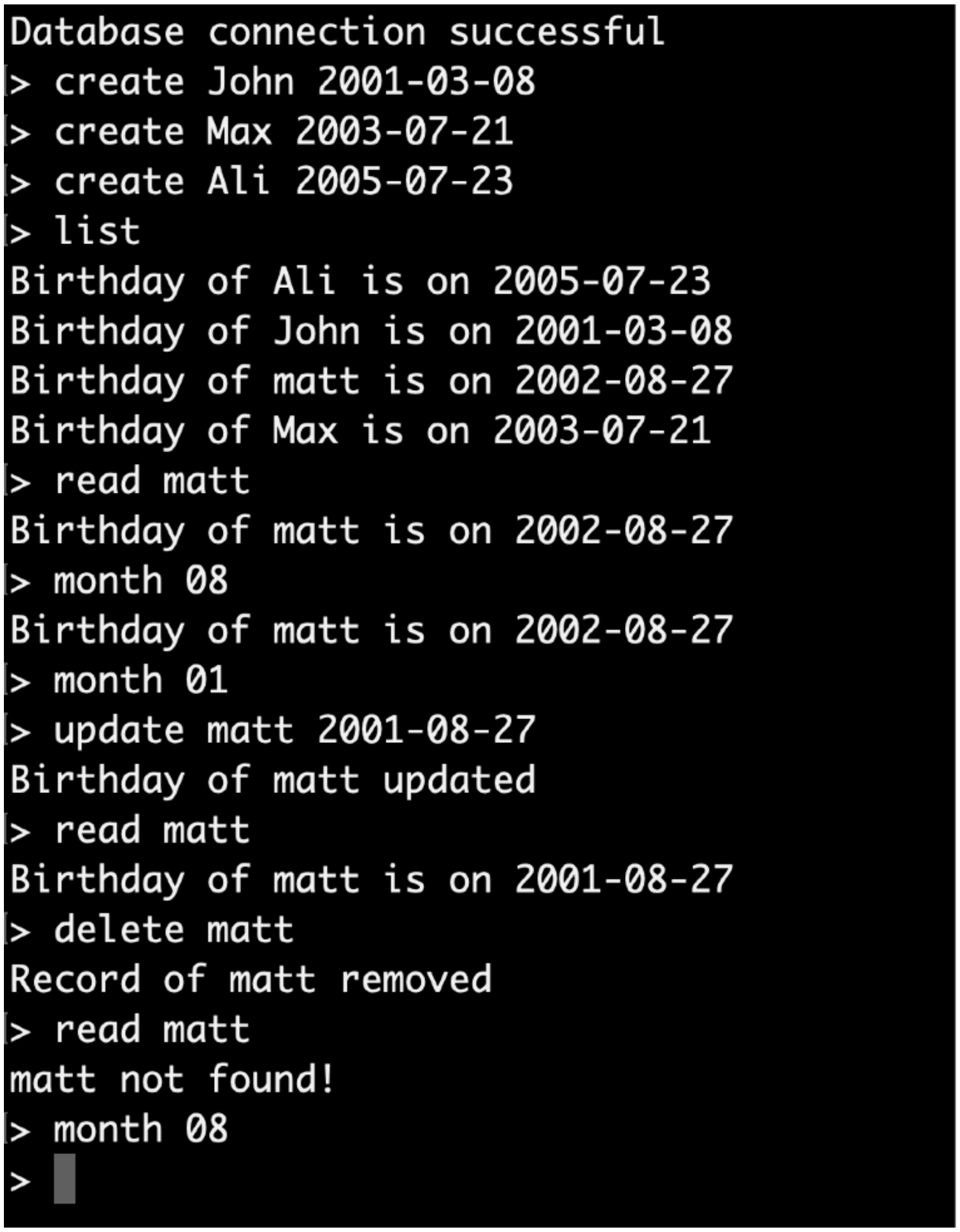
```
mysql> CREATE DATABASE c3358;
Query OK, 1 row affected (0.02 sec)

mysql> USE c3358;
Database changed
mysql> CREATE TABLE c3358_2025(
    -> name VARCHAR(32) NOT NULL,
    -> birthday DATE NOT NULL,
    -> PRIMARY KEY name (name)
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql> CREATE USER 'c3358usr'@'localhost' IDENTIFIED BY 'c3358PASS';
Query OK, 0 rows affected (0.03 sec)

mysql> GRANT ALL ON c3358.* TO 'c3358usr'@'localhost';
Query OK, 0 rows affected (0.01 sec)
```

Figure 1. Setting up the MySQL Database

4. Export the path to `*mysql-connector-j-9.3.0.jar*` under the variable *CLASSPATH* into the environment using possible a command like:

```
export CLASSPATH=$CLASSPATH:"/../mysql-connector-j-9.3.0.jar"
```

5. Compile `*JDBCDemo.java*` and run the program. A sample run for the script is shown in the figure below.



```
Database connection successful
> create John 2001-03-08
> create Max 2003-07-21
> create Ali 2005-07-23
> list
Birthday of Ali is on 2005-07-23
Birthday of John is on 2001-03-08
Birthday of matt is on 2002-08-27
Birthday of Max is on 2003-07-21
> read matt
Birthday of matt is on 2002-08-27
> month 08
Birthday of matt is on 2002-08-27
> month 01
> update matt 2001-08-27
Birthday of matt updated
> read matt
Birthday of matt is on 2001-08-27
> delete matt
Record of matt removed
> read matt
matt not found!
> month 08
> 
```

Figure 2. Sample execution of JDBCDemo

Setup & Execution [Part 2]

The application is bundled as a package titled `*Twenty Four Game*`. For the convenience of the user/tester "build scripts" have also been provided. The package has been thoroughly tested on both MacOS (x86) & Ubuntu. To build and execute the program, follow the proceeding steps:

1. Change your directory to the one which contains the package along with the build scripts and the JDBC driver [see figure below].

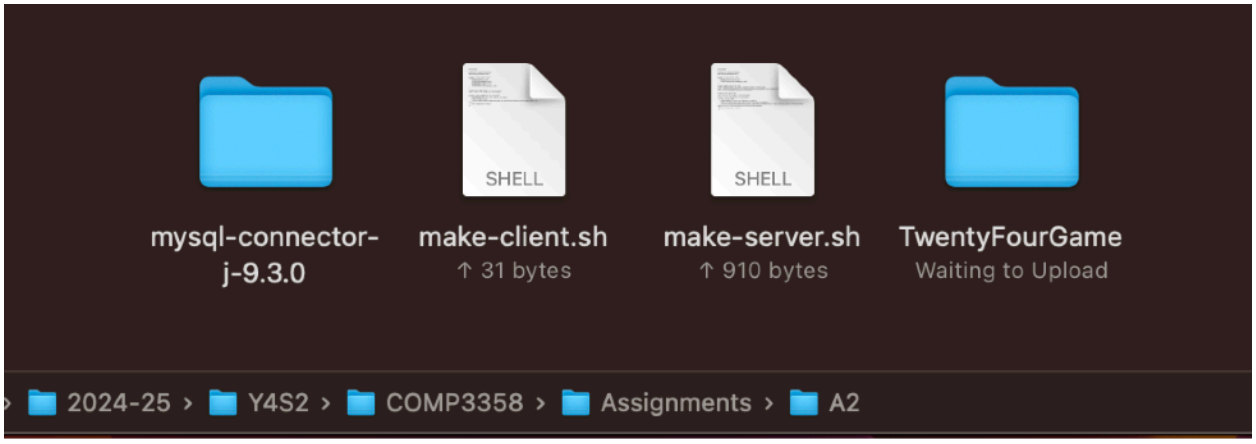


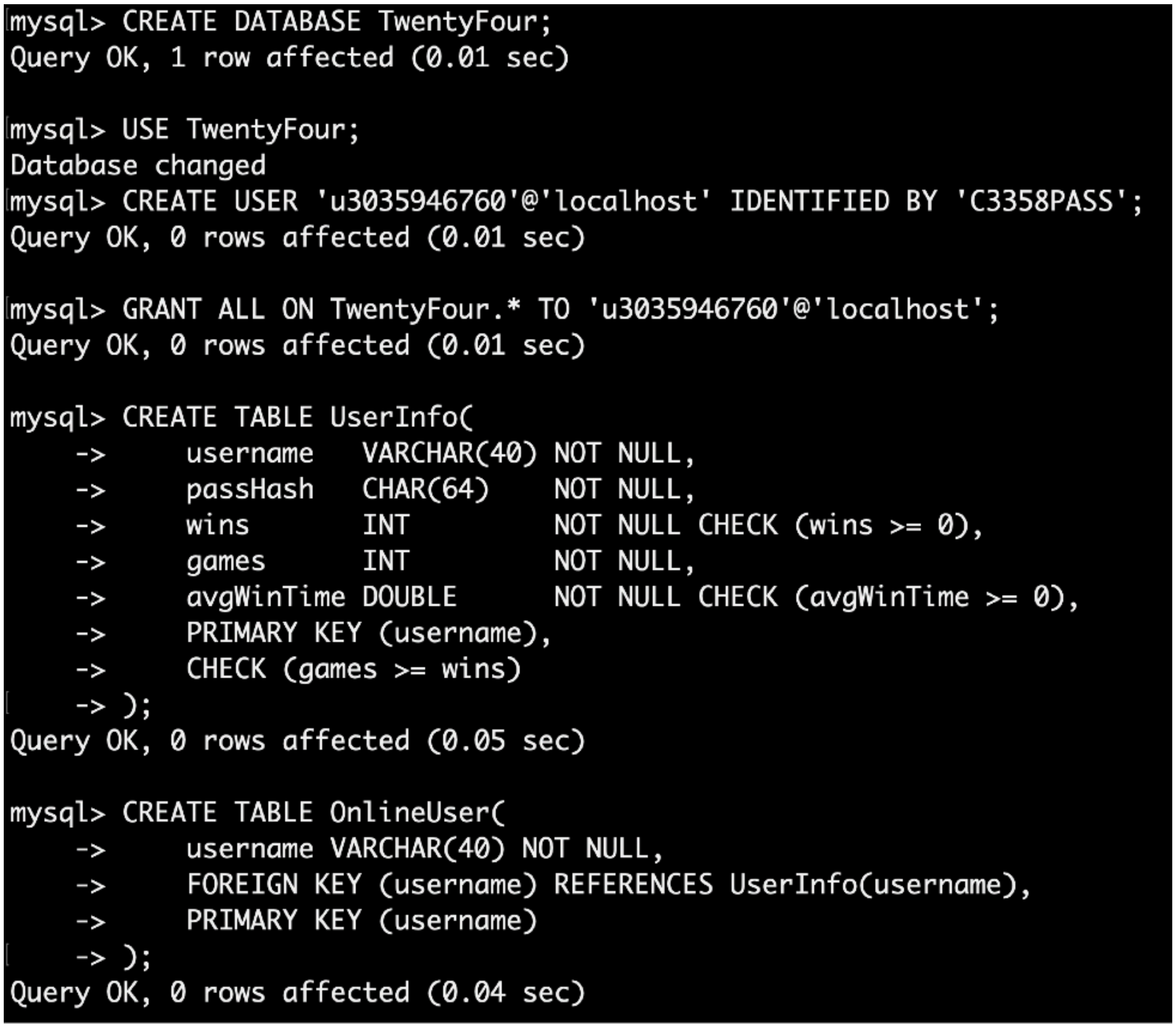
Figure 3. The necessary files to test the project

2. Setup the database. Ensure that mysql-server and mysql-client are installed on your machine. The commands executed to setup the database are shown in the Fig. 4—next page. The sql commands executed can also be found in the TableSchema.sql file attached in the submission.

Notice that we enforce certain constraints on the fields in the table UserInfo such as number of wins being no more than the number of games played or the average win time being non-negative.

We also do not store the rank of a player and instead compute it when it is required (using the number of wins).

The OnlineUser table has a single column namely the username which references the column from the UserInfo table so only registered users can be online (deletion doesn't cause funky behaviour).



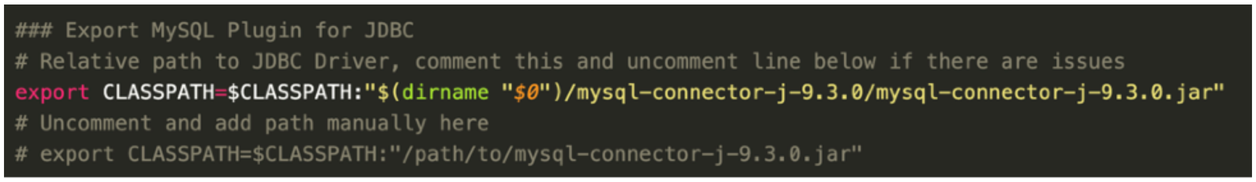
```
mysql> CREATE DATABASE TwentyFour;
Query OK, 1 row affected (0.01 sec)

mysql> USE TwentyFour;
Database changed
mysql> CREATE USER 'u3035946760'@'localhost' IDENTIFIED BY 'C3358PASS';
Query OK, 0 rows affected (0.01 sec)

mysql> GRANT ALL ON TwentyFour.* TO 'u3035946760'@'localhost';
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE UserInfo(
    ->     username   VARCHAR(40) NOT NULL,
    ->     passHash   CHAR(64)    NOT NULL,
    ->     wins       INT         NOT NULL CHECK (wins >= 0),
    ->     games      INT         NOT NULL,
    ->     avgWinTime DOUBLE      NOT NULL CHECK (avgWinTime >= 0),
    ->     PRIMARY KEY (username),
    ->     CHECK (games >= wins)
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql> CREATE TABLE OnlineUser(
    ->     username VARCHAR(40) NOT NULL,
    ->     FOREIGN KEY (username) REFERENCES UserInfo(username),
    ->     PRIMARY KEY (username)
    -> );
Query OK, 0 rows affected (0.04 sec)
```

Figure 4. Setting up the database for A2 Part II

3. Change to the directory holding the build scripts and the `*TwentyFour*` package (as shown in step 1). Then run the following commands, in order:

```
rmiregistry &
bash make-server.sh &
bash make-client.sh localhost
```

If you encounter issues running the build scripts you can modify the make scripts. In the make-server.sh file, add the path to your installation of mysql-connector-j-9.3.0.jar (or whatever version you are using)—shown below in Fig. 5 if the JDBC driver is causing issues. If needed you can modify the path to the java compiler as needed in the scripts.



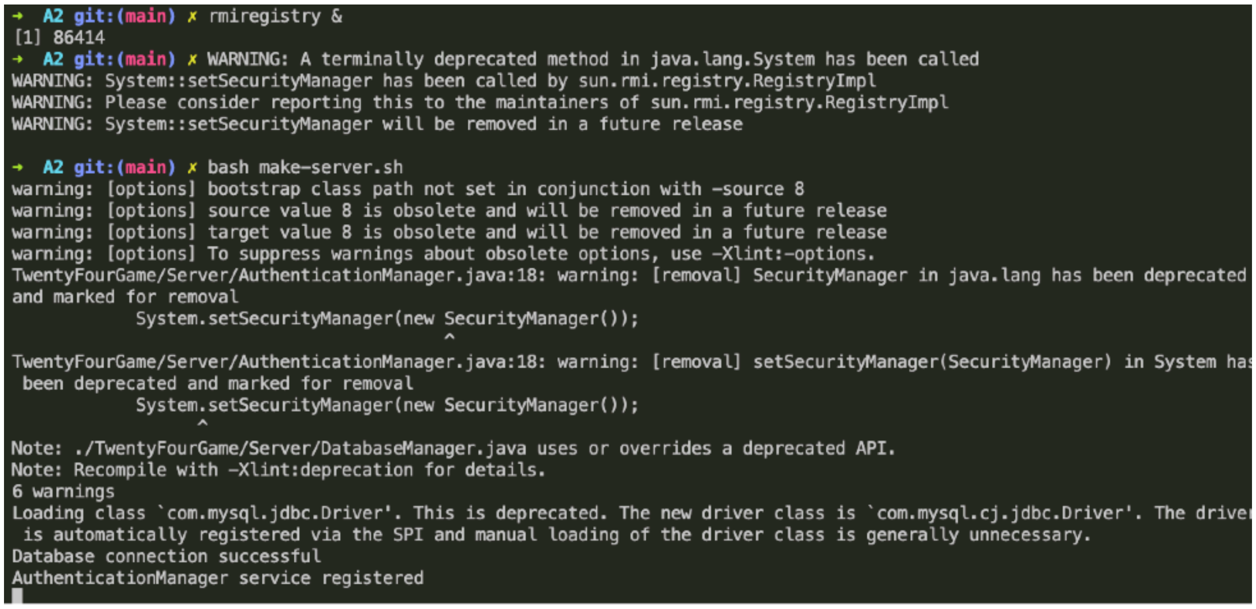
```
### Export MySQL Plugin for JDBC
# Relative path to JDBC Driver, comment this and uncomment line below if there are issues
export CLASSPATH=$CLASSPATH:"$(dirname "$0")/mysql-connector-j-9.3.0/mysql-connector-j-9.3.0.jar"
# Uncomment and add path manually here
# export CLASSPATH=$CLASSPATH:"/path/to/mysql-connector-j-9.3.0.jar"
```

Figure 5. Modify this part in *make-server.sh* if encountering issues with JDBC driver

If you are using a different version of Java you can modify the script to point to the specific version of Java 8 by modifying the script (e.g. replacing `Java` with `./path/to/my/Java_8/binary` in both the *make-client.sh* and *make-server.sh* scripts.

The first command (you ran in step 3) sets up the rmi-registry locally, while the second command sets up the server locally where the rmi registry is launched. This can be set up remotely as well. Then for the client program, we would be required to pass in the ip of the remote server. For convenience we are testing everything locally.

Upon successfully executing the first two commands you should see the messages `*Database connection successful*` *and* `*AuthenticationManager service*

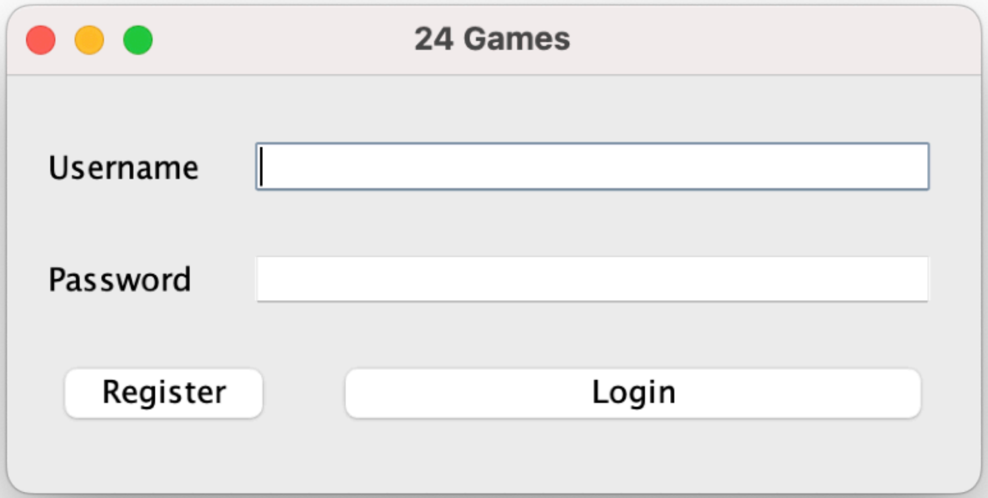


```
→  A2 git:(main) ✗ rmiregistry &
[1] 86414
→  A2 git:(main) ✗ WARNING: A terminally deprecated method in java.lang.System has been called
WARNING: System::setSecurityManager has been called by sun.rmi.registry.RegistryImpl
WARNING: Please consider reporting this to the maintainers of sun.rmi.registry.RegistryImpl
WARNING: System::setSecurityManager will be removed in a future release

→  A2 git:(main) ✗ bash make-server.sh
warning: [options] bootstrap class path not set in conjunction with -source 8
warning: [options] source value 8 is obsolete and will be removed in a future release
warning: [options] target value 8 is obsolete and will be removed in a future release
warning: [options] To suppress warnings about obsolete options, use -Xlint:-options.
TwentyFourGame/Server/AuthenticationManager.java:18: warning: [removal] SecurityManager in java.lang has been deprecated
and marked for removal
            System.setSecurityManager(new SecurityManager());
                                          ^
TwentyFourGame/Server/AuthenticationManager.java:18: warning: [removal] setSecurityManager(SecurityManager) in System has
 been deprecated and marked for removal
            System.setSecurityManager(new SecurityManager());
                  ^
Note: ./TwentyFourGame/Server/DatabaseManager.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
6 warnings
Loading class `com.mysql.jdbc.Driver'. This is deprecated. The new driver class is `com.mysql.cj.jdbc.Driver'. The driver
 is automatically registered via the SPI and manual loading of the driver class is generally unnecessary.
Database connection successful
AuthenticationManager service registered
```

Figure 6. Output for running rmiregistry and the server

registered`, along with other warnings about deprecation and security that can be ignored safely because our application has no users to steal money or data from.

Once these commands have run, a window running the application should pop up as long as your system has a GUI. If not please consider switch a system that has once to use the application.



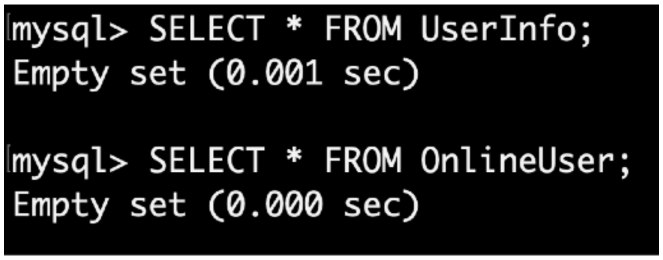
Once tired, you may want to run the command `*rm TwentyFourGame/*/*.class*` to clean the package.

Demonstration

The remaining part of the report demonstrates the instance of the application running on MacOS (for the convenience of taking screenshots, the application also runs identically on Ubuntu as testing verified).

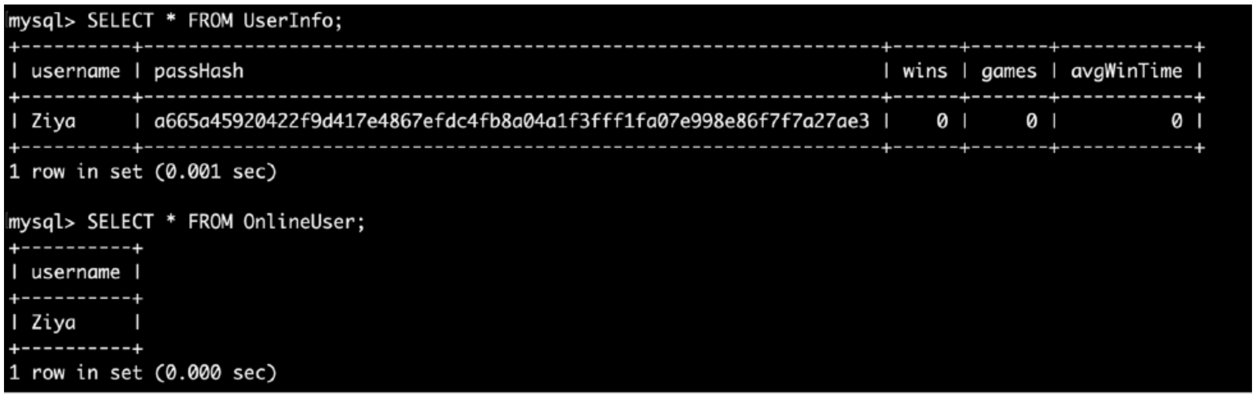
Initially, no registered users exist (see Fig. 7) so we transition to the register panel and register a user.

After registering the user, we can see the user in the UserInfo table as seen in Fig. 8. Since our application also logs the user in after registration (see Fig. 9), we can also see the user active in the OnlineUser table.



```
mysql> SELECT * FROM UserInfo;
Empty set (0.001 sec)

mysql> SELECT * FROM OnlineUser;
Empty set (0.000 sec)
```

Figure 7. Initial state of the database



```
mysql> SELECT * FROM UserInfo;
+----------+------------------------------------------------------------------+------+-------+------------+
| username | passHash                                                         | wins | games | avgWinTime |
+----------+------------------------------------------------------------------+------+-------+------------+
| Ziya     | a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3 |    0 |     0 |          0 |
+----------+------------------------------------------------------------------+------+-------+------------+
1 row in set (0.001 sec)

mysql> SELECT * FROM OnlineUser;
+----------+
| username |
+----------+
| Ziya     |
+----------+
1 row in set (0.000 sec)
```

Figure 8. The *UserInfo* & *OnlineUser* tables after registering a single user

Since we now have access to update methods (although unused in the application by default) we choose to initialise users to zeroed fields instead of randomly assigned data as this is the expected behaviour for real users.

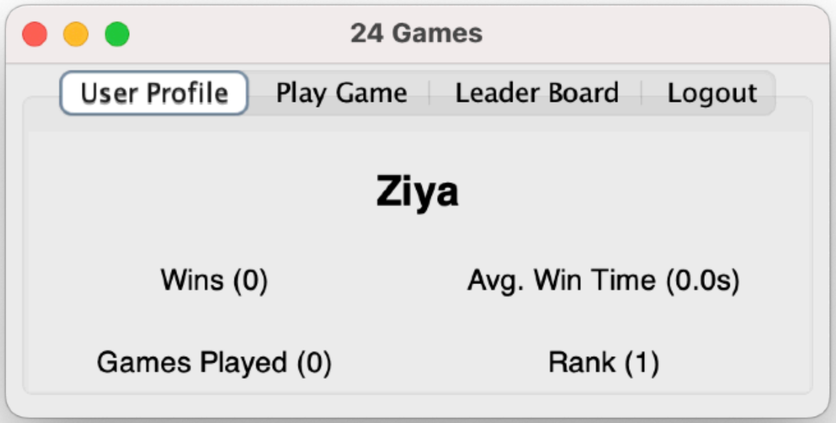
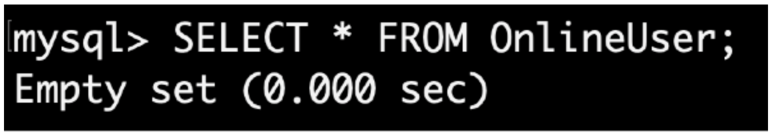


Figure 9. Automatic login after registration with zeroed fields

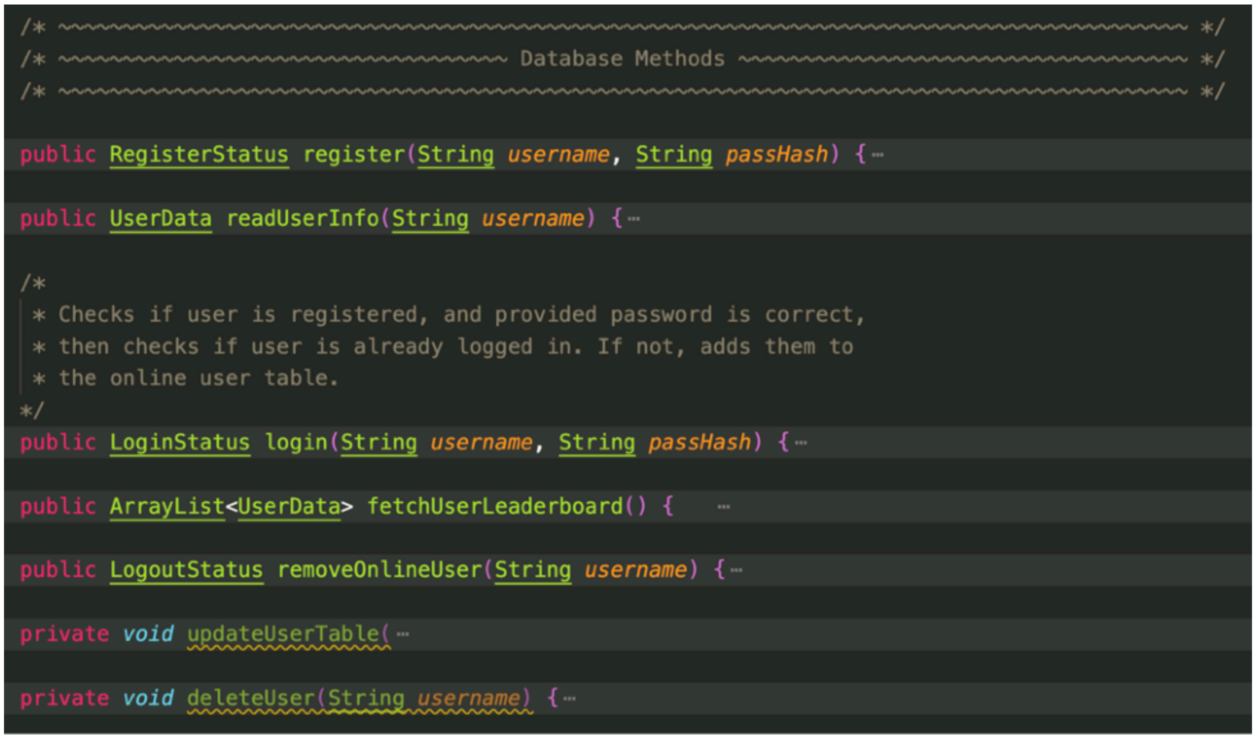
Then after logging out, the user is removed from the *OnlineUser* table as we can see in Fig. 10.



```
mysql> SELECT * FROM OnlineUser;
Empty set (0.000 sec)
```

Figure 10. *OnlineUser* table after logging user out

The database methods that are implemented for use in a later assignment (but are completely unused for the current implementation) are the update and delete methods for the *UserInfo* table. These are all other methods that interact with the database can be found in TwentyFour.Server.DatabaseManager.java in the source code. A snippet of their signatures can be seen in Fig. 11.



```
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Database Methods ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */

public RegisterStatus register(String username, String passHash) {…

public UserData readUserInfo(String username) {…

/*
 * Checks if user is registered, and provided password is correct,
 * then checks if user is already logged in. If not, adds them to
 * the online user table.
*/
public LoginStatus login(String username, String passHash) {…

public ArrayList<UserData> fetchUserLeaderboard() {   …

public LogoutStatus removeOnlineUser(String username) {…

private void updateUserTable(…

private void deleteUser(String username) {…
```

Figure 11. The methods that interact with the database

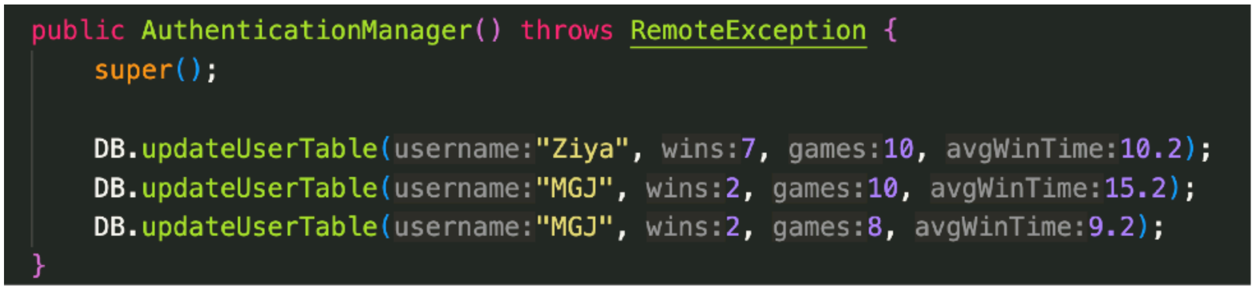
The register method handles insertion into UserInfo (detecting for duplication), while readUserInfo performs reads from UserInfo.

The login method performs a read on UserInfo reads/writes to OnlineUser. Since the atomicity of the two is required we implement these calls as a transaction.

The userLeaderBoard method also reads the UserInfo table but computes the ranks for all users. removeOnlineUser as the name suggests, performs a delete on OnlineUser.

The other two methods are not needed for this assignment (so are kept private) and as their names suggest perform update/deletes on UserInfo. We could also implement an update method for OnlineUser but it does not make much sense since OnlineUser is just a single column.

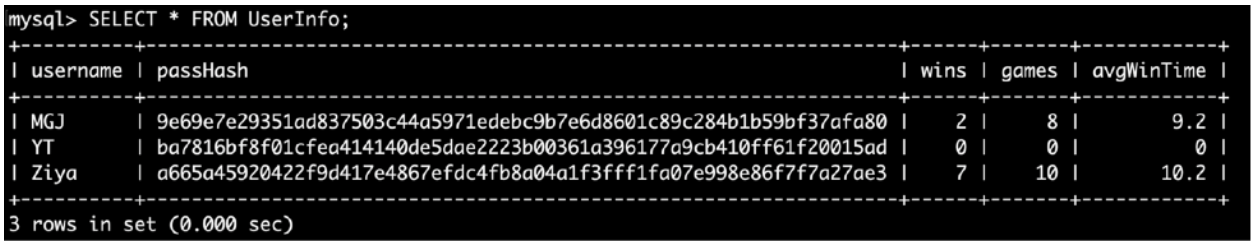
Then for the demonstration of ranking in leaderboard and the update/delete methods we register two more users (called MGJ & YT) with different passwords using the application and call update on the totality of the three users (as shown in Fig. 12).



```java
public AuthenticationManager() throws RemoteException {
    super();

    DB.updateUserTable(username:"Ziya", wins:7, games:10, avgWinTime:10.2);
    DB.updateUserTable(username:"MGJ", wins:2, games:10, avgWinTime:15.2);
    DB.updateUserTable(username:"MGJ", wins:2, games:8, avgWinTime:9.2);
}
```

Figure 12. Update the three users fields

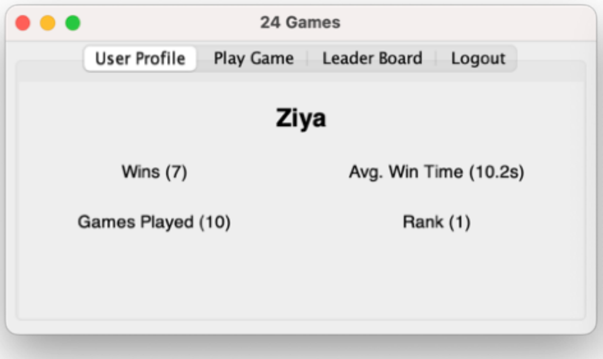
The result of this call can be seen in the database as shown in the figure below.



```
mysql> SELECT * FROM UserInfo;
+----------+------------------------------------------------------------------+------+-------+------------+
| username | passHash                                                         | wins | games | avgWinTime |
+----------+------------------------------------------------------------------+------+-------+------------+
| MGJ      | 9e69e7e29351ad837503c44a5971edebc9b7e6d8601c89c284b1b59bf37afa80 |    2 |     8 |        9.2 |
| YT       | ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad |    0 |     0 |          0 |
| Ziya     | a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3 |    7 |    10 |       10.2 |
+----------+------------------------------------------------------------------+------+-------+------------+
3 rows in set (0.000 sec)
```

Figure 13. The result of updating the users' fields

Then we log in as `*Ziya*` (see Fig. 14) to see that our fields are also viewable on the UI and that the leaderboard shows the correct ranks (see Fig. 15).



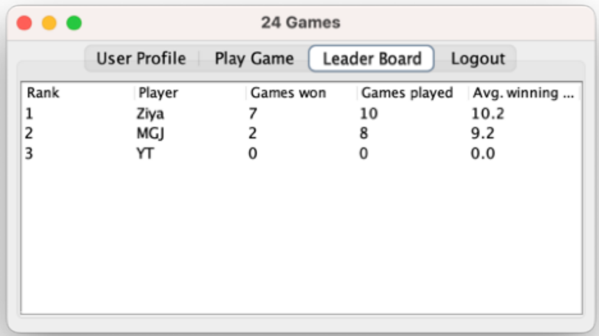
Ziya

Wins (7)

Avg. Win Time (10.2s)

Games Played (10)

Rank (1)

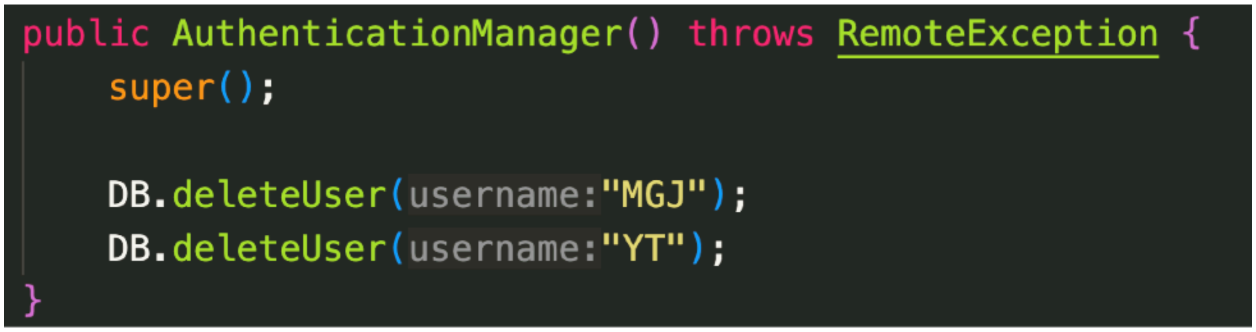


Rank	Player	Games won	Games played	Avg. winning ...
1	Ziya	7	10	10.2
2	MGJ	2	8	9.2
3	YT	0	0	0.0

Figure 14. Update UI for user *Ziya*

Figure 15. Leaderboard after more user added

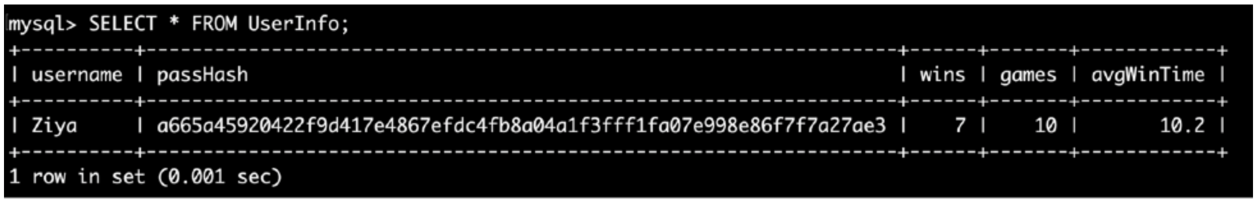
Then finally, we execute the following delete calls (see Fig. 16).



```
public AuthenticationManager() throws RemoteException {
    super();

    DB.deleteUser(username:"MGJ");
    DB.deleteUser(username:"YT");
}
```

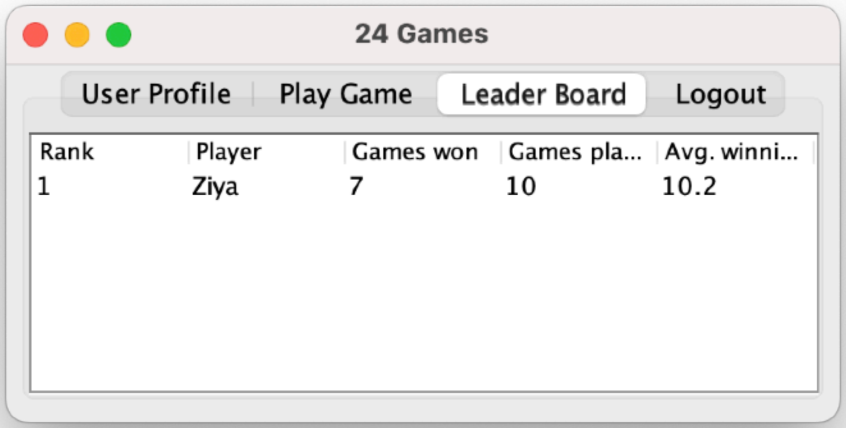
Figure 16. Code to delete the users *MGJ* and *YT*

The result can be seen in both the database (see Fig. 17) and is represented in the GUI (see Fig. 18).



```
mysql> SELECT * FROM UserInfo;
+----------+------------------------------------------------------------------+------+-------+------------+
| username | passHash                                                         | wins | games | avgWinTime |
+----------+------------------------------------------------------------------+------+-------+------------+
| Ziya     | a665a45920422f9d417e4867efdc4fb8a04a1f3fff1fa07e998e86f7f7a27ae3 |    7 |    10 |       10.2 |
+----------+------------------------------------------------------------------+------+-------+------------+
1 row in set (0.001 sec)
```

Figure 17. *UserInfo* table after deleting the users *MGJ* and *YT*



Rank	Player	Games won	Games pla...	Avg. winni...
1	Ziya	7	10	10.2

Figure 18. Leaderboard from user *Ziya* after deleting the two users