Assignment 3 - Full Stack

Felipe Orihuela-Espina & Archie Powell

Contents

1 About the assignment

- **Deadline**: Monday 22nd-April-2024 at 16:00 (UK time).
- **Late submissions policy**: No late submissions allowed.
- **What to submit**: A single `.zip` file as explained in Section 6
- **Learning outcome**: Integrate a database, server, and front-end into a full software stack.
- **Where has the skill been learned in the module**:
  - Weeks 1-2 covered front end interface creation using JavaFX.
  - Weeks 3-7 covered SQL querying.
  - Week 8 covered concepts and code on Threads' creation, execution and termination.
  - Week 9 covered concepts and code on Client-Server Architecture, including a lab exercise using TCP.
  - Week 10 covered concepts and code on JDBC.

- **Note**: As this assignment is being released a few days before the Week 10 lectures on JDBC, you have been given a long submission deadline in order to give you plenty of time to get familiar with the concepts.

2 Problem statement

In this assignment you will be implementing a complete full stack application by "putting together" all the bits and pieces you have been studying over the course. The emphasis though is on JDBC and Java networking aspects, as the rest has been assessed in previous assignments.

In particular, you will be developing a 3-tier TCP-based networking multi-threaded client-server application to consult a database about vinyl records. The application features a client that offers a JavaFX based graphical user interface to request the service and communicate with a intermediate server providing a query specific service. The server, located in the local host, will be composed of two parts or classes; one being the main server which attend request as they arrive on an infinite loop, and the server's service provider that are created to attend each service. It is the server's service provider which does connects to the database using JDBC, retrieves the outcome of the query and sends back the outcome to the client.

Although each service provider thread will only attend 1 request and then stop, the server will attend any number of request by creating new service threads.

The service provided is a fixed but parameterizable query; Given an artist's last name and a record shop's city, the query must retrieve the list of records available from the author at the indicated record shop. For each record available, the query must retrieve, the record's title, the music label, the genre and the recommended retailer's price together with the number of copies available for that record at that record shop. Records from the artist that are not available in the record shop (i.e. 0 copies) should NOT be listed. For artists that do not have a surname, e.g 'Metallica', the band name should be used.

Both, the client and the server applications should be able to work without errors when the query returns no entries. The server will be robust to SQL injection.

In *all* classes, you are requested to treat exceptions in the method that first can raise them i.e. do not rely in `throws` clauses to deal with exceptions at a later stage. If some exceptions requires you to stop the execution of either the servers, or the client without a correct resolution of the request,

make sure that in those cases you exit the program with code 1. Exit with code 0 if program execution is correct.

Feel free to use any IDE of your choice, but tests will be carried from the command line. The back end will be implemented in postgreSQL using `postgresql-42.6.0.jar` driver. Tests will be run in a Linux machine.

3 Setting up the database

For this assignment you will use the 'FSAD2024_Records' database. This is provided to you. Download the file `records.sql` from Canvas. Open as postgres shell window and connect with your own credentials. First, you need to state the path on your own computer where you have the file. Then import this file is as follows.

```
\i records.sql
```

Once you have imported the tables, you can list them by performing the following command in PostgreSQL:

```
\dt
```

Note that this will list all of the tables you may have created in this database. Run SQL commands to view the contents of each table by doing

```
SELECT * FROM tablename;
```

replacing tablename with the name of the table you want to look at.

4 Templates

You are provided with templates for all classes in canvas. The exercise requires you to fill the gaps clearly identified with;

```
//TO BE COMPLETED
```

The templates already contain all the atributes, methods, and correct signatures. There is no need to create more attributes for the classes, nor additional methods, nor to import new classes or libraries, but you can declare local variables within the methods as you may need. Do not alter the signature of the methods as the automarker will rely on these.

5 Methods description

Only move to this stage when you have set up the database, as above.

5.1 Credentials.java

The Credentials.java contains the credentials for the database (username, password, url) and for the server connection (host address, port number). You must replace the credentials with your own. Submitting this class is optional, as we are going to use our own credentials for testing your code. Therefore, you can delete this class from the src folder of your `.zip` file before submitting your assignment.

5.2 RecordsDatabaseServer.java

The server main class. This server provides a service to access the Records database. The server application will run indefinitely until Ctrl+C (Windows or Linux) or Opt-Cmd-Shift-Esc (Mac OSX) is pressed. Add your student id in the file opening comments.

- **public RecordsDatabaseServer()**: Class constructor. Creates the server socket. Reads the connection credentials from class `Credentials`.
- **public void executeServiceLoop()**: Runs the service (infinite) loop. This method accepts the server's service requests. The method listens for incoming client requests and creates a service threads to attend the requests as needed.
- **public static void main(String[] args)**: Provided. No need to do anything. Execution.

5.3 RecordsDatabaseService.java

This class is the service provider or client handler. This class is the one responsible for reading the client request, connect to the database, make the query, retrieve the outcome of the query and sends it back to the client. We are using TCP networking, so if you have not completed the Week 9 lab exercise, it is **highly recommended** you do so. This class extends `Thread`. Add your student id in the file opening comments.

- **public RecordsDatabaseService(Socket aSocket)**: Class constructor. Initializes the server service's socket attribute and launches the service thread.
- **public String[] retrieveRequest()**: Receives the information corresponding to the client request. This method accepts the server's service requests. Upon receiving the service request, it parses the message to remove the termination character ('#') from the incoming message. The message will arrive in the form of a string. The remaining part of the

incoming message contains the two parameters of the query; that is, the artist's surname and the city of the record shop separated by a semicolon; e.g. 'Sheeran;Cardiff'. This method must parse the message splitting this message content into its two components and return a `String[]` with two elements; the first one being the artist's surname and the second being the record shop.

- **public boolean attendRequest()**: This method provides the actual service. It connects to the database, make the query, execute the query and retrieve the outcome and process the query. It should be robust to SQL injection. The connection, statement and result set should be closed properly afterwards. The processing of the query will transfer the contents of the `ResultSet` obtained from executing the query, into a `CachedRowSet` object that can survive the closing of the connection and can be sent over a socket (i.e. it requires to be serializable). The method returns a flag indicating whether the service has been attended sucessfully. Finally, to transfer the contents of the `ResultSet` into a `CachedRowSet` object you can use;

```
RowSetFactory aFactory = RowSetProvider.newFactory();
CachedRowSet crs = aFactory.createCachedRowSet();
crs.populate(rs); //with rs being the ResultSet
    variable.
```

- **public void returnServiceOutcome()**: Communicate the service outcome back to the client. This method returns the outcome of the request to the client and closes the service's socket. To send full objects through a socket you can use `ObjectOutputStream`. For instance;

```
ObjectOutputStream outcomeStreamWriter = new
    ObjectOutputStream(...);
outcomeStreamWriter.writeObject(...);
```

- **public void run ()**: Provided. No need to do anything. Execution.

5.4 RecordsDatabaseClient.java

This is the client or service consumer. It is a JavaFX application with a simple GUI (See Figure 1) with 2 major parts;

- An input area with two `TextField`s two insert the artist's surname and the record shop's city and a button to request the service every time it is pushed.
- An output area with a `TableView` to render the outcome of the last query.

Note that the same client can request the service many times by simply pressing the button again, and in every request, the current selected values

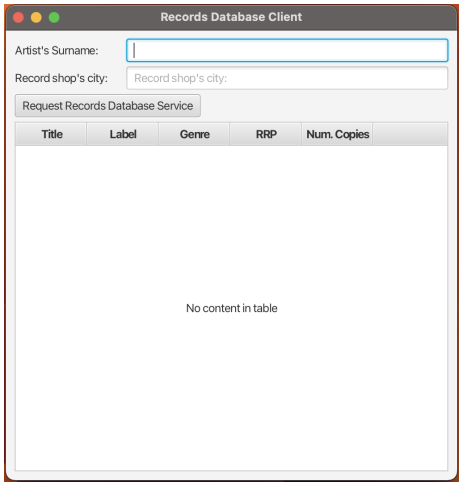

Fig. 1: Records database client GUI.

in the input boxes will be used for parameterizing the query. Note that you do not transmit the query, but only the values of the parameters.

In order to adequately populate the `TableView` with the result, the class also has an internal nested class `MyTableRecord` (Provided. No need to do anything.).

- **public RecordsDatabaseClient()**: Provided. No need to do anything. Class constructor. Initializes the attribute storing the application instance in javafx.
- **public void initializeSocket()**: Initializes the client socket using the credentials in class Credentials.
- **public void requestService()**: This methods requests the service. It adds a terminating character ('#') to the service request message and sends the message to the server via the client socket.
- **public void reportServiceOutcome()**: Receives the outcome of the service and renders such outcome in the output area of the interface. The service outcome is received by deserializing the `CachedRowSet` object using `ObjectInputStream`; e.g.

```
ObjectInputStream outcomeStreamReader = new
    ObjectInputStream(outcomeStream);
serviceOutcome = (CachedRowSet)
    outcomeStreamReader.readObject();
```

Upon receiving this outcome, the method iterates over outcome to update

the table contents. For this, you will use the aforementioned nested class `MyTableRecord`, e.g.

```
ObservableList<MyTableRecord> tmpRecords =
    result.getItems();
  //Here, result is the TableView object.
tmpRecords.clear(); //Clear the table content
while (this.serviceOutcome.next()) {
...
}
result.setItems(tmpRecords); //Update table contents
```

- **public void execute()**: This method implements the callback action to be execute when the button to request the service is pushed. In order to do so, it has to create a service request message containing the two parameters of the query by reading the values at the input `TextFields`; that is, the author's surname and the city of the library separated by a semicolon; e.g. 'Sheeran;Cardiff', and save this into the userCommand attribute. This method will then further proceed to initialize the socket, request the service and report the service outcome by calling the appropriate methods. Finally, it closes the connection with the server's socket.
- **public void start(Stage primaryStage)**: Provided. No need to do anything. This is JavaFX's main method. This method creates the GUI.
- **public static void main (String[] args)**: Provided. No need to do anything. This is the overall main method. It launches the client application.

6 What to submit?

This is a sumative assignment.

1. Compress into a single `.zip` file the three .java files corresponding to `RecordsDatabaseServer.java`, `RecordsDatabaseService.java` and the `RecordsDatabaseClient.java` classes. You may or may not include the `Credentials.java` file.
2. Submit the `.zip` file into canvas.

   Do NOT include the database file .sql or the postgres driver in the `.zip`.

☞ **IMPORTANT**: Do NOT use winrar for compressing but standard .zip. Do ONLY zipped the afore mentioned files. Do NOT create or organize the files into folders. If you develop your code in some IDE, e.g. IntelliJ, extract only the source files and do not submit the whole project. A penalty will be applied to your marks if any repackaging is necessary at our end.

7 Rubric

- RecordsDatabaseServer.java: 25%
- RecordsDatabaseService.java: 40%
- RecordsDatabaseClient.java: 35%

8 Sample outputs

Your outputs should match the sample outputs.

8.1 Server output

All server output is reported via the console. This combines the output of both the `RecordsDatabaseServer.java` and the `RecordsDatabaseService.java` as they together both form the server application. An exemplary output of the server application is shown in Figure 2. The numbers accompanying the "Service thread" line prefixes corresponds to the thread id. These will vary in each run.

8.2 Client output

The client output is reported via both, the GUI *and* the console. An exemplary output of the client application in the console is shown in Figure 3. The corresponding two services requests in such execution are presented in Figures 4 and 5 respectively. Also, Figure 1 shows how the output table should look when a query returns no records. A console output when a query returns no records will simply be empty e.g.

```
Client: Requesting records database service for user command
Franklin;Berlin

**** []
```

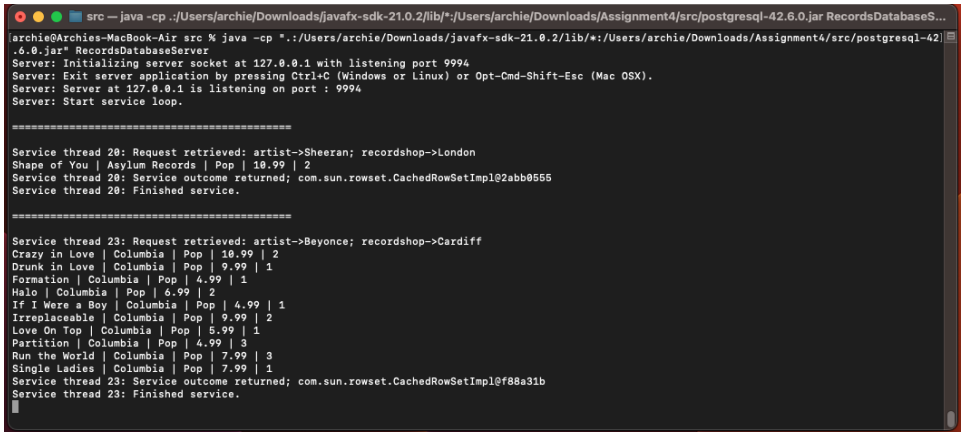

Fig. 2: Exemplary server console output.

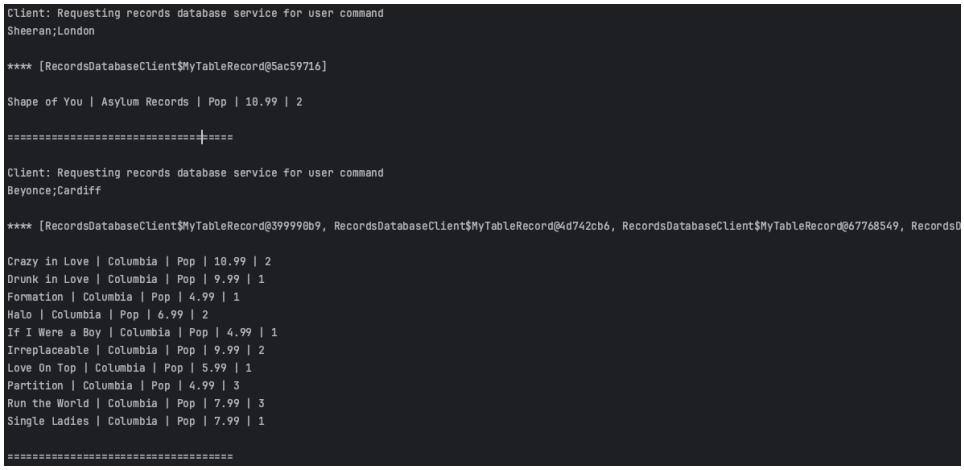

Fig. 3: Exemplary client console output.

==========

The lines preceded by **** list object identifiers in the outcome. These will vary in each run. It is optional to include these lines in the output.

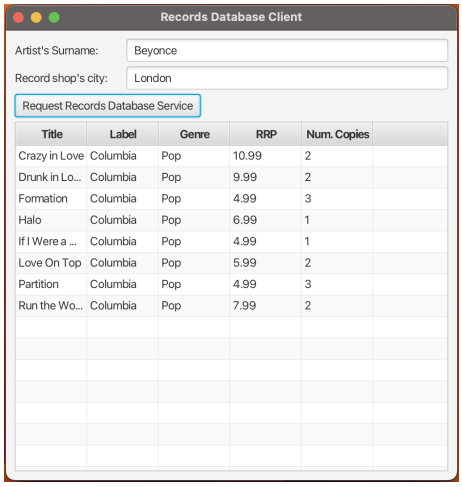

Fig. 4: Exemplary query output to input artist Beyonce and record shop in London.

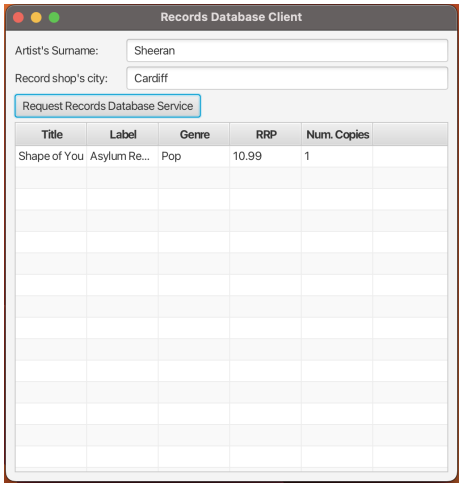

Fig. 5: Exemplary query output to input artist Sheeran and record shop in Cardiff.