EECE 350 Project Final Report

Section: 1
Group: 6
Names: Sarah Awar, Christophe El Haddad, Emmanuel Nassour

Description

We are implementing a hotel reservation system based on the Client - Server model. Our philosophy is to provide users with a secure, intuitive, and rich experience, where they would have control over their account, hotel transactions and have a clear idea of the hotel policies.

“Enjoy a memorable getaway at our hotel. Famous for our magnificent pool and luxurious SPA, we promise you a very comfortable stay. Our hotel includes a variety of rooms to accommodate for family and friends.” *Hotel Description*

Hotel Policy

- The Hotel houses a total of 120 rooms: 60 rooms are Regular and the other 60 are Deluxe
- Available bed types: Single, Double or Family (1 Double + 1 Single)
- Reserving for a Deluxe room would benefit you with:
    - Access to the luxurious SPA
    - Desks for work or study in your room
    - A more spacious room with more comfort

- **Prices for each room and bed types are as follows**

Room Type	Bed Type	Price per Night
Regular	Single	50 $
Regular	Double	60 $
Regular	Family	70 $
Deluxe	Single	100 $
Deluxe	Double	120 $
Deluxe	Family	140 $

- Your reservation can be either Confirmed or Unconfirmed
    - An Unconfirmed reservation can be edited and cancelled (at no cost)
    - A Confirmed reservation can't be edited and if cancelled would add a $30 penalty
    - If you are reserving for a check-in that's in less than 4 days, your reservation would automatically be Confirmed
- Loud sounds are prohibited after 11PM

Implementation

The hotel reservation system is done using the client-server model. Using socket programming, both the Client and Server communicate via Port 1927 and send/receive lines of requests/responses.

The "hard-work" is done on the Server's side (communication with the MySQL database, handling of multiple users...) while the Client makes sure that it sends "correct" queries to the Server (avoid empty strings, overlapping dates...). These ideas are developed in the Server and Client sections.

The Client and Server are coordinated: they have been designed together, so that each one expects a certain input from the other at a certain order. For instance, if the Client sends a line to the Server, the Server is expected to read that line and then send lines to the Client (should the latter expect them).

MySQL Tables

All our tables are in the "sys" scheme inside the database. Please note that our database is on port 3306 of localhost and has user "root" and password "login".
Here are the tables (Note that the first column is the Primary Key):

1. usernames-pass:
    - Handles the accounts credentials
    - Columns: user (VARCHAR(45)), password (VARCHAR(45)) and isLoggedIn (TINYINT(4))

2. profiles:
    - Contains the account information for each user
    - Columns: user (VARCHAR(45)), firstName (VARCHAR(45)), lastName (VARCHAR(45)), gender (VARCHAR(45)), phone (VARCHAR(45))

3. invoices:
    - Stores the penalties of each user
    - Columns: user (VARCHAR(45)), penalties (INT(11))

4. reports:
    - Keeps track of every activity
    - Columns: reportNumber (INT(11)), date (DATETIME(1)), user (VARCHAR(45)), roomNumber (INT(11)), entry (TEXT)

5. rooms:
    - Stores the rooms along with their type, bed type and price
    - Rooms 1 to 60 are Regular and Rooms 61 to 120 are Deluxe
    - Columns: roomNumber (INT(11)), type (VARCHAR(45)), bedType (VARCHAR(45)), price (INT(11))

6. reservations:
    - Stores the reservation details with the username, room number, dates, status and smoking
    - Columns: reservNumber (INT(11)), user (VARCHAR(45)), roomNumber (INT(11)), smoking (TINYINT(4)), status (VARCHAR(45)), checkIn (DATE), checkOut (Date)

Server

- Communicates with the MySQL database with the url “jdbc:mysql://localhost:3306/sys”, user “root” and password “login”
- The Server is Multithreaded and can serve multiple guests at the same time (by using the ServerThread class implementing Runnable)
- The Server’s structure is similar to a Finite State Machine: in a while loop breaking with the “EXIT” instruction, the Server first checks what’s the clientSentence (indicating the instruction) and then acts accordingly depending on the instruction (set variables, get input from client, send output to client)
- The server stores the username in a string as soon as a user is logged in, so that it keeps track of it for its actions
- In most of the Server functions, the insertlog function is called, which inserts a corresponding log entry into the reports table
- Some of the Server functions
    - boolean findp(String, user, String password) ***Instruction: L***
        - Returns true/false if it finds the user and password in the usernames_pass Table
    - int reserve(String type, String bed, Date checkin, Date checkout, boolean smoking)
        - Used in the multipleReserve function
        - Get an available room with the query: “SELECT roomNumber FROM rooms WHERE roomNumber NOT IN (SELECT roomNumber FROM reservations WHERE (checkin < checkout AND checkout > checkin)) AND type = type AND bedType = bed”

        - Explanation: we find a room that's **not overlapping with a reservation** and meets the requirements
        - If the resultSet is empty, it returns -1
        - Otherwise, it inserts the corresponding reservation row in the reservations Table
    - void getCurrentReservations() ***Instruction: Get Reservations***
        - Get the reservations with the query: "SELECT checkIn,checkOut,roomNumber,reservNumber,status,smoking FROM reservations WHERE user = username AND checkin >= CURDATE()"
        - Explanation: find all the reservation rows of username and checkin after Today
        - Get the resultSet size and send it to the Client
        - In a loop depending on the size (if it's zero the loop isn't started), the Server successively sends the reservation details to the Client (which expects them according to the size it received)
    - void multipleReserve(int num, String type, String bed, Date checkin, Date checkout, boolean smoking) ***Instruction: Reserve***
        - Tries to do num reservations with those specifications and store the reservation numbers in an ArrayList
        - As soon as an error occurs (no more rooms available), the Server deletes the reservations stored in the ArrayList and sends "-1\n" to the Client
        - If all the reservations were made, the Server send their numbers to the Client (1st send the number of reservations then each reservation ID)
    - void getLogFromDates(Date first, Date last) ***Instruction: Get Log Dates***
        - Search logs matching the dates and sends them to the Client (first send the number of logs, then each log and its details)

Client GUI

- The Client is a JavaFX application consisting of different Scenes and their Controller
- The GUI's are designed in FXML files, and their logic is set in the corresponding Controller
- In order to pass objects between Controllers, and save the Connection Socket, we created a Logic class.
    - It is instantiated in the beginning of the application and passed to each Controller
    - Contains the Socket, input and output streams, and the stored username of the connected user
- We also passed to each Scene references to the Scenes that it will switch to
- **Switching between Scenes:** get the primaryStage from any GUI element of the scene with the getScene().getWindow() command. Then, we use primaryStage.setScene(*other_scene*) to switch Scenes. Sometimes, we also called the refresh() function of a Controller to refresh the content of the Scene

- **Dialogues:** we made use of the useful dialogues that allowed us to display information without switching Scenes (this was used for example for listing available rooms, asking for confirmation of the user for a reservation)
- Some of the Controllers:
    - NewReservationController:
        - Uses comboBox for selecting the appropriate reservation details
        - After checking if the inputs are correct (developed in Additional features Section), successively send lines to the Server starting with “Reserve\n” and following with the reservation specification
        - If the check-in is in less than 4 days, a dialogue informs the user that his reservation would be Confirmed and checks if he continues or not
        - Else, a dialogue asks the user if he wants a Confirmed or Unconfirmed reservation
        - Wait response from Server: if we receive “-1”, view message in red “Reservation wasn’t made!”. Else view message “Reservation made”
    - InvoiceController
        - Using a TableView, we show to the user his invoice by first getting the appropriate item quantities from the Server
        - The amounts due are summed and shown in the total

Additional features and Security features

- In the editReservation, since we can only edit an Unconfirmed reservation, we wanted to dynamically add buttons to the Scene. To do that, we created a GridPane (that looks like a Table) and according to the Server output queries, we dynamically added the Edit button (for Unconfirmed reservations) and the Cancel button (for all reservations) linked to the corresponding reservation
- In the main menu, we display “Welcome Mr. / Ms. firstName lastName” according to the gender of the User; this is more user-friendly
- Added colored buttons in the Main Menu for getting the User’s attention
- Logging in with the special admin User (username = admin password = admin), you have access to the View Activity Reports Menu which lets you retrieve logs by Date, Room Number or Username.
- **Security Features:**
    1. The isLoggedin entry in the usernames_pass table is set to 1 whenever the user logs in and is set to 0 when he logs out (this done by the Server either when the user manually logs out or when the thread closes because of an exception with a proper isLogged boolean flag in the Server). When a user tries to login, the Server first checks the isLoggedIn in the MySQL table: if it’s 1, prevent the login (*this might be an attack or someone else knowing the password*); if it’s 0 proceed with the login

2. Before signing up, a Human Verification Window is shown. A special function in the Server (generateCode) issues two integers from 1 to 15 and sends them to the Client. The User is then asked to enter the sum of these numbers and according to the result (*checked by the Server*), we allow the signup. This would prevent the creation of multiple fake accounts by robots

3. If the User quits the Human Verification dialogue without sending a query to the Server, the Server would be still waiting for the *sum answer*. To prevent that, whenever the dialogue is closed, the Client sends the query “0\n” which would either be counted as the *answer*, or be counted as an *Echo* request (the Server sends “EchoBack\n” to the Client)

4. In any Scene in the GUI, we make sure to prevent blank entries before sending the queries to the Server, which would be nasty for the MySQL database

5. In Signup, before sending anything to the Server, we make sure that:
   a. The User inputs his password correctly by with the Confirm Password field (alert the User if passwords don’t match)
   b. Check if the phone number is correct (“+ xxx xxx...xx” or “xxxxx”) where x are digits
   c. Making sure that everything is correct, we send the query to the Server and the latter alerts the Client if the username is already used (prevent duplicate users which would create major conflicts)

6. In Edit Profile Scene, first ask for old password if the User wants to change the password

7. In Reservation Scenes:
   a. Don’t allow reservations for more than 30 days
   b. Don’t allow a reservation before Today
   c. Don’t allow date overlaps (check-out before check-in)

8. In the View Activity Reports Menu for the admin, we present to the Client the end dates of the logs (and don’t allow sending dates outside these end dates). Also, we don’t allow choosing a room number outside the boundaries (0-120, with 0 corresponding to room-less reports)

Distribution of Load

Sarah Awar: Client GUI design (FXML), MySQL database design

Christophe El Haddad: Server implementation, Client GUI implementation (Controllers), MySQL database design

Emmanuel Nassour: Client GUI design (FXML), MySQL database design

Problems We Ran Into

- Dividing the load between all members was hard, since the Client is interdependent of the Server (knowing the order of the queries) which makes the coding very hard
- Initially, neither the Server or the Client would be able to read lines from each other, because we forgot to add a “\n” to the end of each string sent
- Knowing how to jump between multiple windows was hard at the beginning
- Passing values and objects from one Scene (or its Controller) to another was hard at first, but we resolved this issue by creating the Logic class passed to every Controller
- We didn’t know initially how to deal with different object types: LocalDate, Date (MySQL), ResultSet, int vs. Integer… We got to learn these Java objects
- Sometimes, variables having the same name would interfere with each other, so we had to rename them to fix the problems