

Table of Contents

Introduction Page 3

System design Page 3

Implementation Page 6

Testing Page 11

Reflection and responsibilities Page 16

Conclusion Page 17

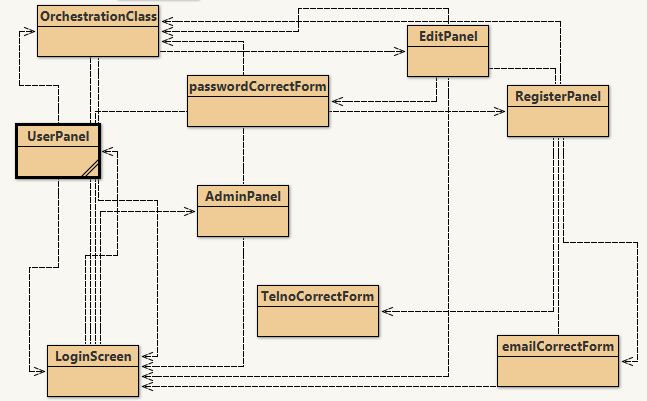
Appendix Page 17

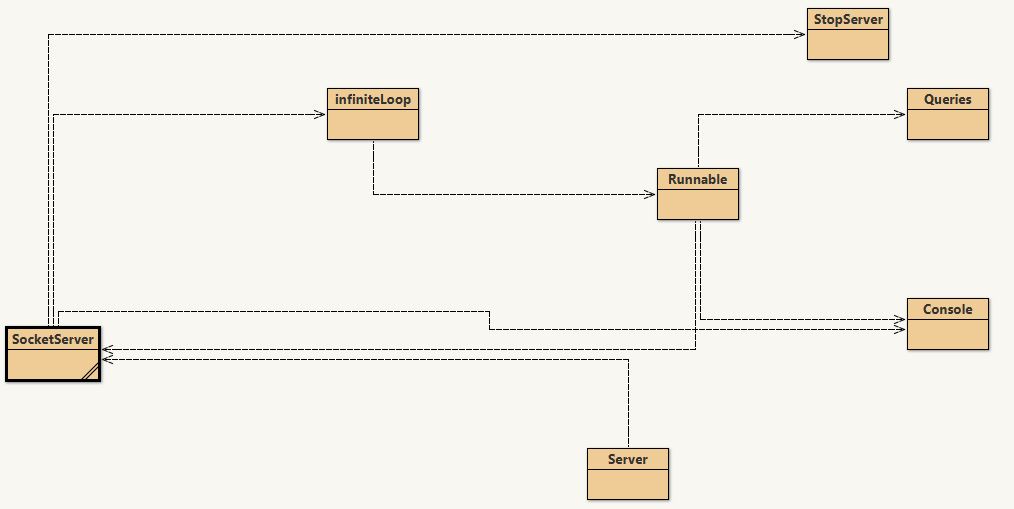
List of Group Members

**Introduction**

According to the task given a client server application is required, this will be a hotel booking system that allows customer to book online their futures trips as well as edit them. The main requirements for the task is a system that allows the customer to register an account with his details, view his upcoming bookings and edit or delete some of them. Also tasks requires a hotel manager to be able to assign rooms to reservations as well as viewing bookings per room. The task also request that customer or admin will have to login in each time the want to run the program. Finally a database is required to hold data such as user information, reservations, rooms room types and assigned rooms to reservations. Trying to implement an application that covers all of the above mentioned GROUP 24 created a three tier application (CLIENT-SERVER-DATABASE), the application allows user to login as a customer or as an admin by having a different GUI for each type of user. New users are required to register before they are able to login in, so a register page has been created with all the required Fields. After a user has successfully logged in he can see a GUI that allows him to check availability in a specific date range and make bookings. Also user is able to view his booking and change dates-or amount of rooms. When an Admin logs in the system he is forwarded in an admin GUI which gives him the ability to check the reservations of a room (future and past reservations) and also to assign rooms to a reservation when a customer arrives at the hotel. In the following sections we are going to go through all the futures and the design of the application described above.

**System Design**

Illustration 1: Client side Classes

Illustration 2: Server side classes

**Classes**

As we can see in illustrations 1 and 2 representing the class diagrams we tried to make an easy maintainable program by separating in classes, most of the classes on the client side represent an interface such as customer interface, admin interface or edit booking interface and we can also see some classes representing validators. On the Server side we can see that we we have as well separated everything and there are classes like queries where we have the connection with our database there and all of our queries, we can see the Server Logic class and the console which prints out all the requests and the responses in order to help server admin to be able to solve and issues and the server sockets class which creates the threads.

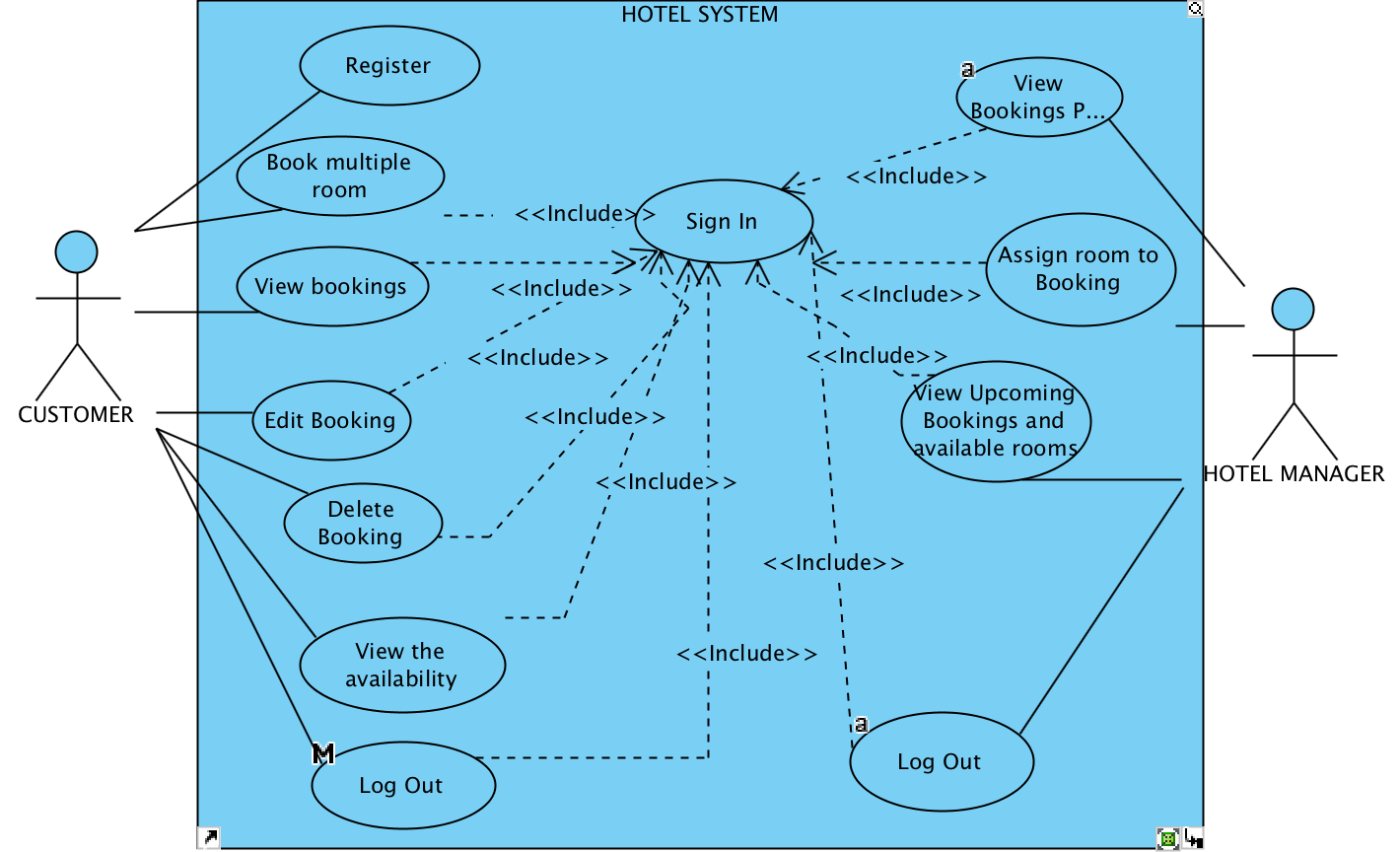


Figure 3

Use case diagram

**Illustration 3 :** demonstrates the functionalities of the application. As described in requirements a customer can register an account in the register page where is ask to enter informations as name, surname , address, mobile, email password and card number. All users customers or admins have to login before they have access to other functionalities of the program,

after a user has login he is able to check the availability of rooms in s specific range of dates, he can also book a room. Customer is also able to view all his reservations(future and past) and has also the option to delete or edit them. Admin can See the reservations that have been assigned to a room

(future and past reservations) and all the information about each reservation. Also admin can assign rooms to each reservation and see all the reservations that have not been assigned to rooms yet.

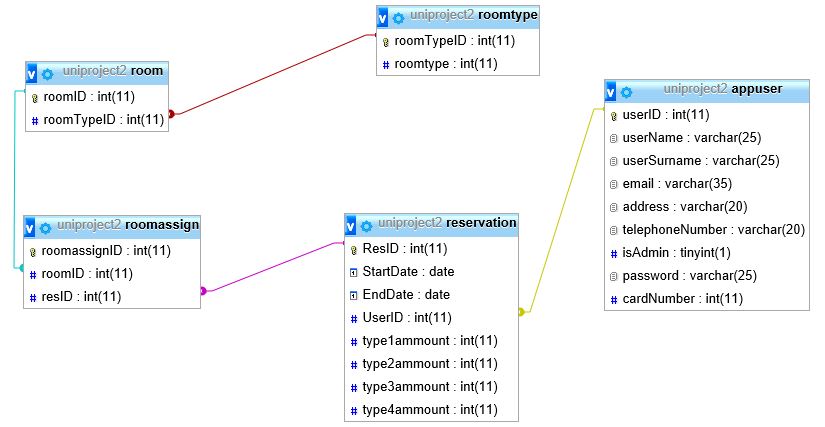


Figure 1 Databse design

**Illustration 4 :** Database followed a simple design with 4 tables. Table users will hold both customers and admins since at this stage we dont have any extra details that need storing for the admins. Admins and customers are separated by the boolean isAdmin in the appuser table, if it is true the user is an admin and is redirected to the admin interface if it is false the user is redirected to the customer interface.

**IMPLEMENTATION**

As mentioned above the application is a three tier architecture system, we have a server a client and a database. The way application works is client(GUI) sends a String to the server via print writer, the string structure is words separated by commas, the first word is always the request, depending on the request the server reads the string with bufferedreader and splits it where there are commas, then server creates a list and uses the indexes of the list 1 to n as arguments in methods in the Queries Class. Queries class will handle all the database queries this class is going to set a connection with database and execute the queries. There are different situations that Queries class will have to deal with are insert,delete,update,select and sum queries. The update,delete and insert queries are just executed without returning anything to the client GUI, but the select and sum queries need to return the data back to the client in a way. The way we return the data is the same way we receive, we construct a string with words separated by commas and we set the first word to the response and then the arguments following. After the server sends back a response client listener does the same procedure

and manipulates the string accordingly, after the main loop of the GUI reads the string it will call the appropriate methods to pass the data in the GUI. The way the server communicates with client is

with buffered reader and buffered writer. At the moment we run the server the server runs a

Server Socket and listens for new connections from client, each time a client connects to the server the Socket server creates a new socket to maintain a connection. For the server database connection we chose a JDBC connector in order to make it easier for the future if the hotel wants to change their database.

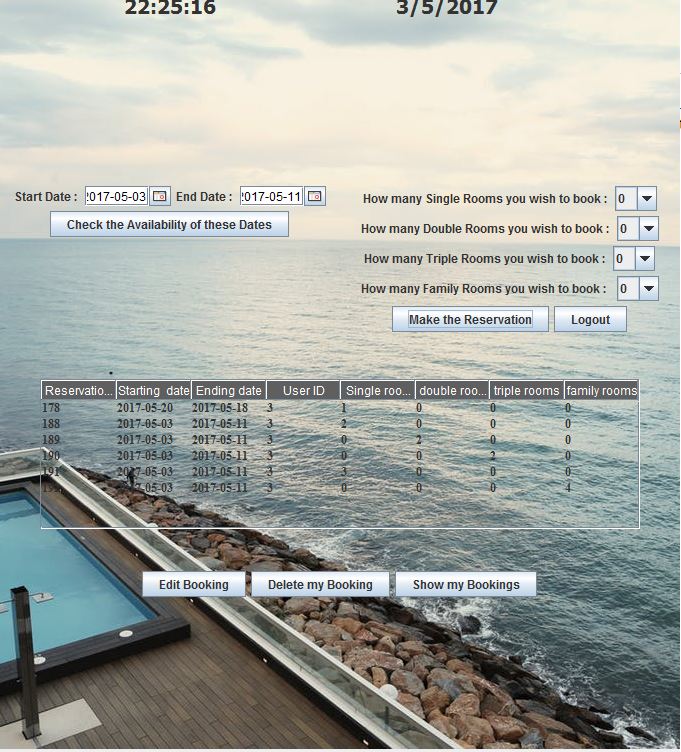
Illustration 4: Login Screen



Figure 2 Register interface

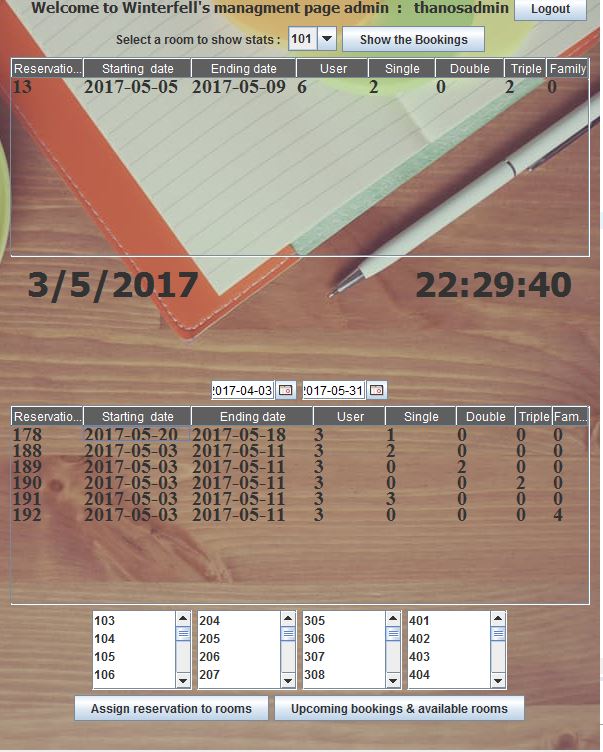
Illustration 5: Register interface

Illustration 6: Edit reservation interface

Illustration 7: Customer interface.

**JTABLES**

In order to project the bookings of a customer on our GUI the bookings per room in admin panel and the reservations to be assigned we used Jtables and set them to uneditable so the user or admin cannot modify the cells and possibly create bugs. The rows of each table are populated by 2D lists, each inner list is used to populate each row. A nice option was to make the Jtables transparent so we maintain a nice appearance.

Illustration 8: Admin interface with the Jtables and Jlists populated with data.

**Combo boxes**

For the user panel and the edit panel we used combo boxes in order to allow the user to chose an amount of rooms for each type of rooms. We populate the Jcomboboxes dynamically depending on how many available rooms we have, this allows us to save some time in implementing checks and have a solid protection in this part from SQL injection attacks.

**GUI design.**

For the GUI design the application comes with an undecorated Jframe and we set mouse listeners in order to make the GUIs draggable.

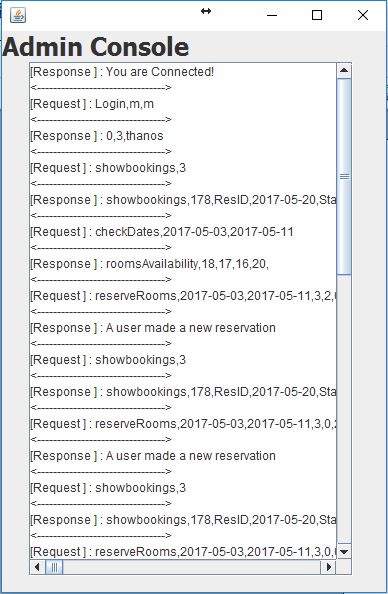
**Validators**

Something worth of mentioning is the validation of all the data entered through the Jtextfields email validators mobile number and always validating if the text fields are not blank.

**CONSOLE**

Another feature to make problem tracing easier was to add a console on the server side,

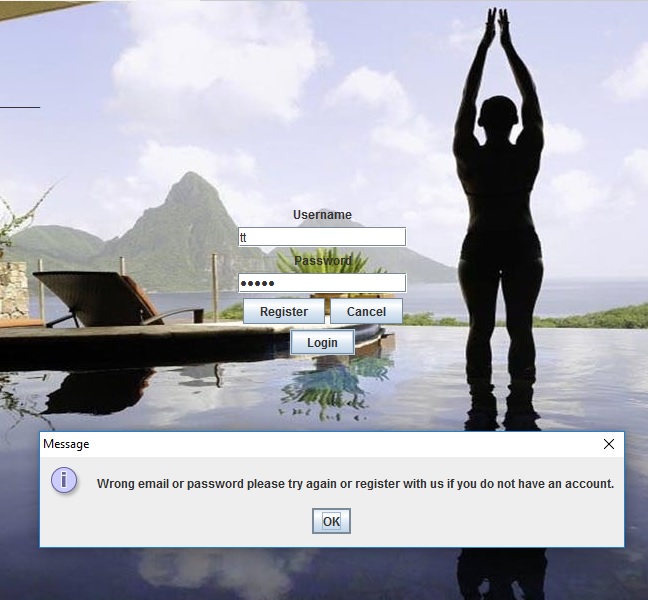
this console appends in a JtextArea the requests of the client and the responses of the server in order to make debugging easier by knowing the values we send or receive. This panel will help a server Admin to locate any possible packet attacks or SQL injection attacks.

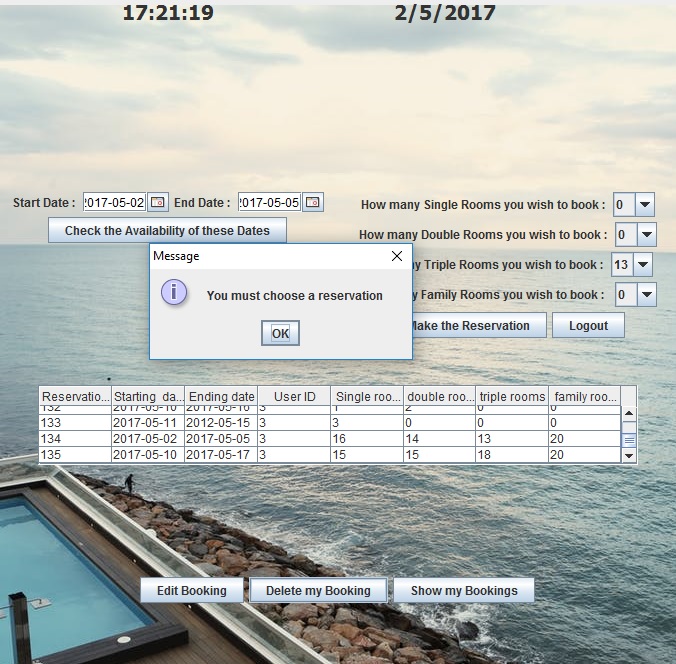
Illustration 9: Admin console is on the server side displaying requests and responses.

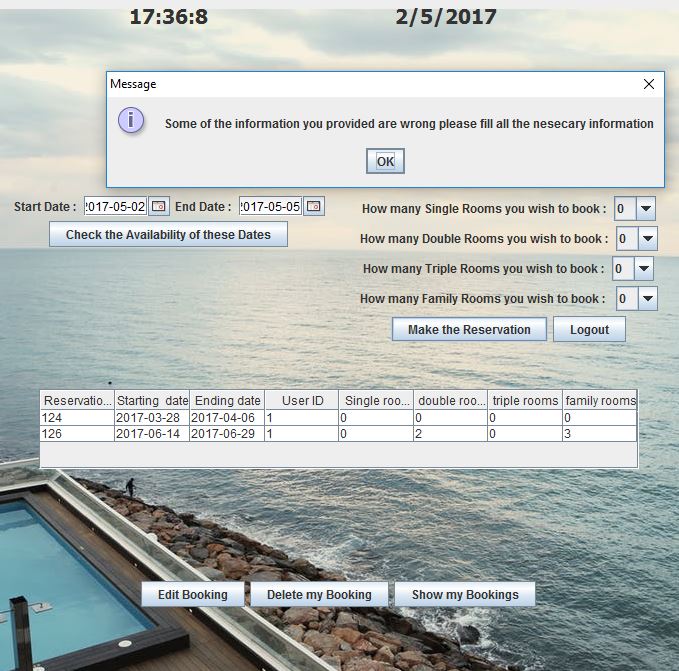
**TESTING**

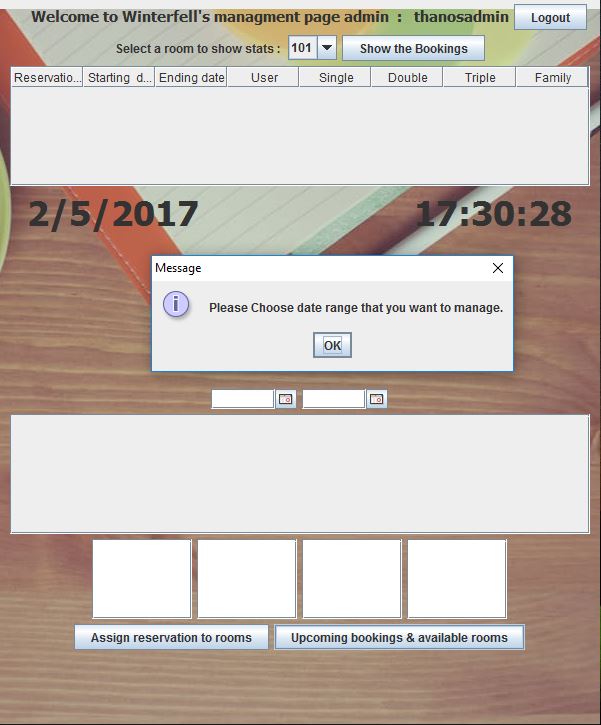
The target for this application was to minimise the possible errors that can be caused by the user and user input, this is why the application uses the comboboxes and jtables for the most part.

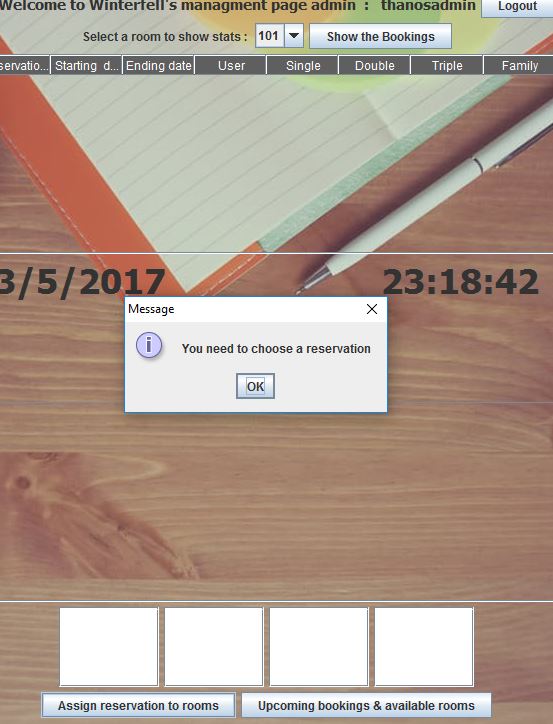
In the following illustrations we will demonstrate a series of tests that we successfully handled. Other than the application authors the application was handed to a small amount of people pretending to be a user or real life hotel admins in order to let them use the application and find possible mistakes we could not notice.

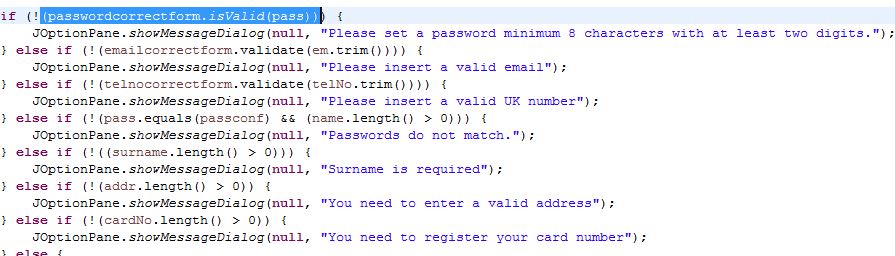
Illustration 10: Testing Login with wrong email or password.

Illustration 11: When user tries to delete a reservation without chosing one from the Jtable

Illustration 12: when no amount of any type of room entered.

Illustration 13: Admin tries to project upcoming bookings but did not specify date range.

Illustration 14: When admin clicks assign button but has not select reservation.

Illustration 15: All the validations for the register panel.

# **Reflection and Responsibilities**

Considering the time given to us to complete our final project we created a schedule which was divided in two parts. The first part was the group meetings where we were working on the project and the second part was research at home according to the problems we faced.

To begin with, in this project we acquire a considerable number of positives. First, on positive of this project is the fact that we did not have to change the starting idea because we managed to solve all the problems we dealt with. Moreover, the second and most important outcome of this project is that we broaden our horizons since we obtained a variety of new knowledge. More precisely, we learnt about models for tables combo-boxes and J-tables. Also, we found out how to read values from J-tables, and the technique of debugging.

All the previous positives mentioned, were accomplished while dealing with the following difficulties. To begin with, one problem we came across at the server was that we did not know how to include each client in one thread. In addition, we found it difficult to manage the requests into the main loop because, we had to split each request in a different way. Furthermore, we dealt with problems at the user class. One of these problems is that we did not know how to populate a combo-box and make it fed dynamic. What is more, we found it difficult to set our own design and position everything so we used a variety of layout managers. Additionally, we found it hard working on j-tables since we had to do a lot of research to be able to assign models, refresh and most tricky one was to populate them. Regarding the admin class rather than the j-tables we had a tough time trying to find the not assigned rooms and the reservations. Last but not least, we struggled with swapping interfaces. At the beginning, we were thinking of using multiple j-frames and because we could not maintain the connection with the server while swapping j-frames we end up having one j-frame and while using card layout we manage to change between multiple panels.

As regards the group and the time spent working together on the project, we were always working with awareness but we also held a friendly and a happy atmosphere. It is logical enough that not everything was going as planed and that we experienced some difficulties but luckily we handled them.

|  |  |  |
| --- | --- | --- |
| Name | Student ID | role |
|  |  | Group leader, JDBC and queries ,Admin interface, main documentation. |
|  |  | Database, User interface,documentation. |
|  |  | Edit reservation interface, documentation, Validators. |
|  |  | Login & register interfaces, Database. |
|  |  | Sockets implementation. |

**Conclusion**

Taking everything under consideration this is a complete application of a hotel booking system that meets all the task requirements in which we tried to minimalize the possibility of dealing with any errors. The projects it is three-tier architecture application which means that we have a server, a client and a database to store information. We used socket programming which is a networking application and moreover we used object oriented techniques.

# **Appendix**

All pictures used for background were downloaded from [https://www.pexels.com](https://www.pexels.com/) and they are licensed under the Creative Commons Zero (CC0) license. This means the pictures are completely free to be used for any legal purpose.

Clients main loop this is where client receives the data that server sends and calls appropriate methods to project them on our GUIs.

void whileChatting**()** **throws** IOException**,** ClassNotFoundException **{**

String message **=** "You are connected! "**;**

**do** **{**

message2 **=** **(**String**)** input**.**readLine**();**

**if** **(**message2**.**equals**(**"null"**))** **{**

JOptionPane**.**showMessageDialog**(null,**

"Wrong email or password please try again or register with us if you do not have an account."**);**

**}** **else** **{**

myListo **=** Arrays**.**asList**(**message2**.**split**(**","**));**

String response **=** myListo**.**get**(**0**);**

**if** **(**response**.**equals**(**"1"**))** **{**

cl**.**show**(**loginwallpaper**,** "5"**);**

setSize**(**600**,** 770**);**

userName **=** myListo**.**get**(**2**);**

userID **=** myListo**.**get**(**1**);**

**}** **else** **if** **(**response**.**equals**(**"0"**))** **{**

cl**.**show**(**loginwallpaper**,** "7"**);**

setSize**(**680**,** 770**);**

userID **=** myListo**.**get**(**1**);**

**}** **else** **if** **(**response**.**equals**(**"roomsAvailability"**))** **{**

userPanel**.**setIndexes**(**message2**);**

userPanel**.**clearComboboxes**();**

userPanel**.**feedComboBoxes**();**

**}** **else** **if** **(**response**.**equals**(**"roomsAvailabilityForEdit"**))** **{**

editPanel**.**setIndexes**(**message2**);**

editPanel**.**clearComboboxes**();**

editPanel**.**feedComboBoxes**();**

**}** **else** **if** **(**response**.**equals**(**"showbookings"**))** **{**

userPanel**.**appentToArea**(**message2**);**

**}** **else** **if** **(**response**.**equals**(**"BookingsPerRoomString"**))** **{**

adminPanel**.**fillAdmintablewithbookings**(**message2**);**

**}** **else** **if** **(**response**.**equals**(**"BookingsForAssignString"**))** **{**

adminPanel**.**fillAdmintableForAssign**(**message2**);**

**}** **else** **if** **(**response**.**equals**(**"notassignedString"**))** **{**

adminPanel**.**fillListsOfRooms**(**message2**);**

**}** **else** **if** **(**response**.**equals**(**"Not\_available"**))** **{**

System**.**out**.**println**(**"response is : " **+** response**);**

JOptionPane**.**showMessageDialog**(null,** "Some of the rooms you chose have been booked by another user,"

**+** "\n" **+** "Please chose a different amount of " **+** myListo**.**get**(**2**)** **+** "rooms."**);**

**}** **else** **if** **(**response**.**equals**(**""**))** **{**

System**.**out**.**println**(**"response is :" **+** response**);**

**}**

**}**

**}** **while** **(!**message**.**equals**(**"CLIENT - END"**));**

Mouse listeners are at ached on labels in order to make GUIS drag able.

bgLabel**.**addMouseMotionListener**(new** MouseMotionAdapter**()** **{**

// override the method

public void mouseDragged**(**MouseEvent evt**)** **{**

int x **=** evt**.**getXOnScreen**();**

int y **=** evt**.**getYOnScreen**();**

//OrchestrationClass.ki.setLocation(x - xMouse, y - yMouse);

myX **=** x **-** xMouse**;**

myY **=** y **-** yMouse**;**

OrchestrationClass**.**ki**.**setLocation**(**myX**,**myY**);**

**}**

**});**

//adding mouselisteners to a label will make it a drag field

bgLabel**.**addMouseMotionListener**(new** MouseMotionAdapter**()** **{**

// override the method

public void mousePressed**(**MouseEvent evt**)** **{**

xMouse **=** evt**.**getX**();**

yMouse **=** evt**.**getY**();**

**}**

**});**