Samuel U, Andrew Meyer, Carolina Blanco

Dr. Akhlaghi

December 15, 2015

**Airline Reservation System Report: GitHub CIS3270\_Project**

The requirement for the project was to create a functional application that mimics an airline reservation system. The airline reservation is to have the basic functions of a reservation system like expedia.com or cheaptickets.com. When starting up the program, the end user is introduced by a splash screen to show that the application is loading. This will then bring up the login screen, where the user will have three choices to make: to login with their credentials, and if they are not a user, to click on the reservation button to create a user. The third choice is for the user to retrieve their password, in case they forgot about it. The user should be able to look at all the available flights pulled up from a database, reserve a spot for a flight they are interested in, and to cancel a flight. These privileges will be given to the users who are customers. Another type of users will be the administrators, who have all the privileges the customer has, plus the privilege to add or update a flight.

**Database Model**

The above diagram illustrates the data models for the airline reservation database. The above tables were created in MySQL. The Reserve table contains a constraint for the idFlight and username attributes. This means idFlight will not be able to be booked by the same username. The parent key in the Plane table becomes a foreign key to the Flight table. This allows the user to pull information on their flights and how many seats are available for that flight. The admin attribute in the User table has a value of either 1 or 0. If a user’s admin attribute is 1, that person is an administrator. This means that the administrator is able to add and update flights as stated previously. On registration, the user is required to input a security question and password. This is used to retrieve lost password if the user forgets their password.



**Project, Login, ForgotPassword, newRegister, MainUI, Flight, ListFlight, FlightForm**

Upon executing the program from the Project class, a splash screen should show which transition to the Login page. The login page give the user the choice to login with their username and password, click the “forgot password” button to retrieve password, or click “register” to create a new user account. Upon clicking the login page, the program will look into the database to see if credentials match, and if it does match, the user will be brought to the MainUI page. The MainUI page consists of three or four button. A regular Customer user will have access to the “List Flight” button, the “My Flights” button, and the “Logout” button. If the Customer clicks Logout, the MainUI will close and the Login page will load again awaiting for user input.

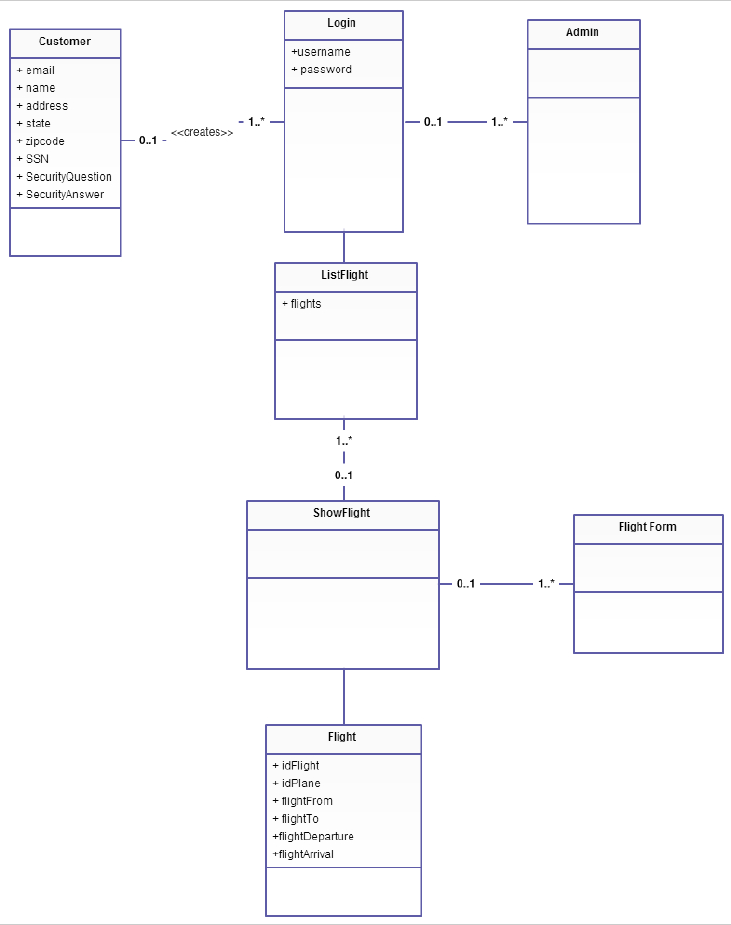
The “List Flight” button will take the user to a new page pulled from the ListFlight class. This page pulls up all the flights in the database from MySQL and gives the user the ability to book a flight by entering the flightID number in the textbook and clicking the “Book” button. By clicking the book button, the code will check if there are any time constraints with any other booked flights made by the customer, or if the user has already booked the chosen flight. It will also check if the plane is under capacity, so the user can book the flight. By clicking the “Main Menu” button, the user will be taken back to the MainUI page.

The “My Flights” button takes the user to a new page pulled from the ShowFlight class. This page searches through the database and pulls up all the flights the user has booked. In future implementations, this page will also give the user the ability to cancel a booking that was booked previously. By clicking the “OK” button, the user will be taken back to the MainUI page.

The fourth button in the MainUI page is only visible to a user who is an administrator. The button is labeled as “Add/Update Flights.” In this page, the administrator is taken to a page that pulls up a form created using the FlightForm class. In this page, the administrator has the ability to populate the given fields to add flights to the database for users to book. After populating all the fields, the “Add” button creates a new Flight object which uses the constructor in the Flight class. Upon clicking the “Add” button, the program will check if all inputs are valid and then populate the database using an insert statement in Java using the JDBC.

**User, Customer, Administrator, ShowFlight, NewRegister, dbModify**

The User class implements the dbModify Interface, which consists of 4 methods. The initialized() method is a method that makes a connection to the MySQL database. The User class implements the method to program in Java with JDBC. The add() method is used to create users when they are register from the new register page. This new register page can be seen in NewRegister class. In future implementations, the User page will be able to delete() a listing, such as a booked for customer. The user also contains the getInformation() method which uses a statement within java to write queries to the database. The getInformation() pulls up the user’s information and sets it’s fields to a customer object. The bookFlight() method implements the ability for users to book a flight from a list of flights that is pulled up from the database. The getFlight() method pulls up all the flight information that the user has booked and can be viewed on the ShowList class.



(Incomplete class diagram)

**Problems with Implementation and Lessons Learned**

There were many obstacles in the implementation of the application. One of the biggest problems was the usage of GitHub. The group had to create a brand new project because many issues came about from the first project. This ended up having to bulk insert the project and then making sure all users knew how to commit properly.

Another problem from the project was the usage of frames. It was difficult to understand how to move from one frame to another frame, and then going back to the previous frame. Upon many lines of code, one member researched online and found that it was bad practice to have multiple frames in an application. A better way of implementing the pages was to make panels in each class that required a page. From there, have one form that switches between each panel depending on what panel is called.

There are many areas in this program that was not finished due to time constraints. The splash does not really work how we wanted it to. It has the problem of loading at the same time as the login screen. There are many redundant codes throughout the application, and could have been organized and maintained properly. This would have saved time and code space, as well as ease of readability of the code. The application is also missing the ability to create a new administrator from the registration form. The use of regular expression has not been implemented due to time, but will be future implemented to make sure all user inputs in every form is valid for inserting into the database.

Finally, the application is missing the requirement for the user to be able to search for flights under certain criteria, and the ability for an administrator to delete or update a flight from the FlightForm page.

The project was a big task to accomplish, and there were many points in the project that got the team frustrated. It is a good experience to go through. One thing the team would change about the project is to get access to the project earlier. This will help in that it will give the team more time to implement the project.