Design Document:

Anson Li and Satyen Akolkar

General Overview

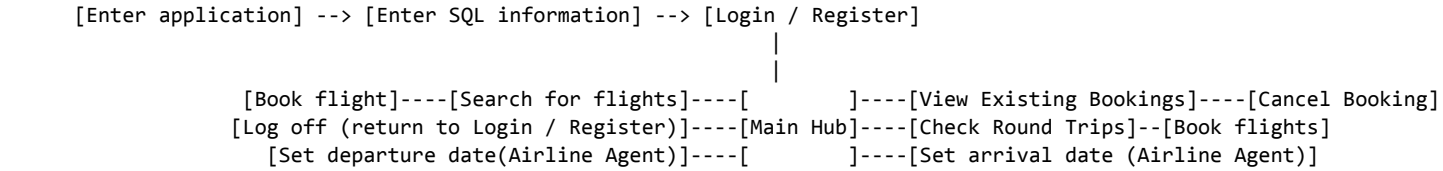
The application created for mini-project 1 is a Flight Registry Application - to which we have named Air Kappa. This application is designed for use of the following tasks for the user:

1. Register or login to the application using relevant login information.
2. View relevant flights for the user, and book flights.
3. View, or cancel bookings.
4. Find round trips, and book for them as well.
5. Log out of the application, and store the time in the application.

Additionally, the airline agents who use this application (who, within the context of this application, are also named powerusers) have the following functionalities:

6. Set the actual arrival date for a flight.
7. Set the actual departure date for a flight.

Process Diagram



User Guide

Entering the Application:

1. When you start the application, you will use your SQL credentials to enter.
2. When you enter valid credentials, you will enter the Login / Register page.
3. In this case, we will register an account for access to the application. Enter 'R' into the console to continue:
4. Enter a valid email and password, and verify the password.
5. Upon registering, the main hub is reached.

Booking a Flight:

6. First, let's search for a flight: enter 'S' into the console to continue.
7. Enter the ID of the flight you want to see to get more information and setup the booking process.
8. Book the flight with valid name and country credentials.
9. Select 'B' to book the flight. The system will return to the main page after booking is completed.

Cancel a Booking:

10. Select 'E' to view the booking.
11. Select '1' to view the first booking entry available, or input the number referring to the entry you want deleted.
12. Select 'C' to cancel the booking, removing the booking from the sql tables.

Record Departure:

13. Select ‘D’ to enter the option from the main menu.
14. Enter the flight number, eg. ‘AC017’.
15. Enter the departure time, or enter ‘C’ to enter the current time. The system will return to the main page after the departure time is updated.

Record Arrival:

16. Select ‘A’ to enter the option from the main menu.
17. Enter the flight number, eg. ‘AC017’.

18. Enter the arrival time, or enter ‘C’ to enter the current time. The system will return to the main page after the arrival time is updated.

Round Trip:

19. Select ‘R’ to enter the option from the main menu.
20. Enter the source airport, the destination airport, departure date, and the return date.
21. Select the booking that’s most suitable, and book the round trip. The system will return to the main page after the booking is completed.

Exiting the Application:

22. Select 'L' to logout.
23. Select 'E' to exit.

Design of Software Components:

The design of the software components will initially be a high-level analysis, and then will discuss each of the components identified for each functionality available in the application.

First, the sections relevant to the project are:

- FlightBooker.java
- SampleSQL.java
- SampleUI.java

These three files form the basis of the entire application. Their roles are as follows:

1. FlightBooker acts as the wrapper for the whole project; it takes the functionalities found in both SampleSQL and SampleUI and combine these to form the application. When a user runs the application, they run this file in order to begin. This file:
    a. Stores data that's used exclusively in generation.
    b. Generates the functionalities.
    c. Starts the user on the application.
2. SampleSQL acts as a wrapper for JDBC functionalities that are used in the application. It stores all the relevant SQL data and provides an abstraction for the other parts of the project to use. This file:
    a. Stores the connection and the statement, as well as its handlers.
    b. Stores the functionalities for JDBC.
    c. Provides code for the initialization and closing of the JDBC instance.
3. SampleUI stores all the processing and the front-end code for the application. This is where all the actual functionality is stored; as a result, the components will be identified here:

WelcomeScreen

The WelcomeScreen is where the login and register options are enabled. The user enters this function first; either by entering valid login details, or by registering a new account, a user can enter the application. This function redirects to Register(), or Login(), or exits depending on the user input.

Register

This separate function registers the user into the system. The user has to input the following in order to enter:

1. A valid email address (one that fits a regex).
2. A valid password (within 4 or less alphanumeric characters).

With respect to SQL, this function inserts the new user into the system after verification. Once these conditions are satisfied, the user enters the program through MainHub().

Login

This separate function logs in users into the system. The user has to input their login credentials correctly in order to enter (so long as the email and password match the ones found in the database). With respect to SQL, this function searches the database to find a matching email/password pair - if it's successful then the user is validated and directed to MainHub().

MainHub
This function acts as the base 'hub' for the user, in the same manner as a main page. In this function, users can choose to perform a multitude of options as identified in the function. In this function, the user can enter SearchForFlights(), ExistingBookings(), RoundTrips(), Logout(), RecordDeparture(), or RecordArrival() - of which these two can only be accessed by an airline agent.

SearchForFlights
This function searches for flights according to the project requirements description. Given source and destination airport information, as well as flight date, a user can find relevant flights with 0-1 connections. In the project, SearchForFlights provides the functionality with respect to the user input; the processing is identified in the next function, PrintFlightPlans(). With respect to SQL, this function finds the resultSet found within the query operations completed to find the flights, and then passes it to PrintFlightPlans().

PrintFlightPlans
This function prints flight plans for the project, given the queries completed in the SearchForFlights() function. At this point, users can select a flight to book (at which point the program will go to MakeABooking()), or return to MainHub().

MakeABooking
This function is reached when a flight is specified by the PrintFlightPlans() system. At this point, the system will confirm the booking to be made, and book the flight for the user given a user-input name and country. Once the query is completed, the user will be given a ticket number as reference for their booking. With respect to SQL, this function uses a transaction to complete the full process; if the flight was already booked prior to the transaction completing, the function will rollback and there will be no issues in SQL. Regardless, the function returns to MainHub() after completing.

BookingDetail
This function provides more details about the booking. It is requested by the MakeABooking() function. With respect to SQL, this function queries the bookings table for more information about the selected booking.

CancelBooking
This function cancels a booking. It is requested by the BookingDetail() function. With respect to SQL, this function deletes the entry in the bookings table that matches the user-selected entry. The function returns to MainHub() after completing.

Logout
This function logs out the user and saves the time they logged out of the system. With respect to SQL, this function updates the users table with the time they last logged in. This function returns the application to WelcomeScreen().

RecordArrival
This function lets airline agents record the arrival of a flight by setting its actual arrival time in the database. With respect to SQL, this function updates the scheduled flights table with the actual arrival time. The function returns to MainHub() after completing.

RecordDeparture
This function lets airline agents record the departure of a flight by setting its actual departure time in the database. With respect to SQL, this function updates the scheduled flights table with the actual departure time. The function returns to MainHub() after completing.

RoundTrips
This function lets users identify the round trips that are made (that also exist in the database). Additionally, this function lets users book round trips. This function mimics the functionality previously presented in SearchForFlights, but is used for round trip queries. The function returns to MainHub() after completing.

Testing Strategies
Various testing functionalities were used in order to ensure that the product was stable and consistent with respect to SQL queries, as well as UI functionalities.

One testing strategy we implemented is cause-effect graphs. Cause-effect graphs were implemented largely in the testing of the UI. Initially, for the design of the product, a simple cause-and-effect graph was completed, similar in design to the process diagram in page 1. With the implementation of a simple cause-and-effect graph, we could gain a basic understanding of the intended process of

the product. As a result, once we complete the implementation, we would have a solid understanding of the basic process flow of the operations. This testing method helped us solidify the high-level analysis of our application, which helped us develop the UI as well as our deadline goals.

Another strategy we used to test the application was through soft unit testing. Soft unit testing was implemented throughout our project. As developers of the program, we understand the underlying model behind our functionalities. As a result, we would have a clear view of the basic I/O of the software for all functions. Using this knowledge, we can test the functionalities using predetermined inputs, and physically verify each output as correct or incorrect using sqlplus. For example, when we tested the process of creating a booking, we checked the bookings table prior to the booking process, then checked the bookings table after the process. This process was effective at testing SQL queries and their implementation in our application; however, it required a lot of time in order to process each activity using all applicable test cases. Using this method, with respect to the major production development, we found roughly 18 bugs that occurred as a result of development of unknown requirements. One example of this strategy being used was when the password was developed; based on our initial requirements (less than 4 alphanumeric characters) we tested a variety of test cases, including 0-character passwords, invalid symbol passwords, 20-character passwords, etc,. As a result, through this testing method, we iterated on its design until we felt that the password verification method was complete. Functionalities that greatly benefitted from this testing method included testing user input in username and password development, and user input in airline arrival and departure time.

One last method of testing that we used is peer code review. Peer code review was implemented throughout our project, but largely within the end of the code production stage. Our team's development strategy was to produce code for each module separately, then integrate the code and examine whether or not it is consistent with the team's work (as well as the project requirements). As a result, all code produced has multiple examiners, which is useful for two purposes: first, at a cursory glance, it can be clear in some cases if a teammate's code is not consistent with the rest of the work. As a result, clear issues in the code can be easily identified and mitigated. Second, when testing the functionalities, the teammate who did not work on the code is viewing the functionality for the first time, or with 'fresh eyes'. As a result, the teammate is compelled to use their testing methods to attempt to break the functionality. In comparison, if the teammate was working on the functionality, they may be more compelled to use testing methods that would not break the functionality, or use more standard techniques in testing the methods. As a result, using peer code review was a significant benefit to our project, as we were then able to ensure that the program could not be broken by using non-standard testing methods. Functionalities that greatly benefitted from this testing method included adding bookings, selecting flights, and finding round trips.

Group Work Strategy

Our group work strategy was initially to handle the code development individually by separating the development of each module, then combining the modules and refactor the code together. The code development, for each module:

Anson's Tasks	Time Spent (Hr)	Satyen's Tasks	Time Spent (Hr)
UI Framework Development	5	SQL Wrapper Development	3
Code dev. for bookings, airline agents	16	Code dev. for Registration & Login Details	8
Finding flights development	3	Finding flights development	8
Testing	8	Testing	10
Design Documentation	3	Round-trips development	5
Refactoring	2	Refactoring	5

The method of coordination used to keep the project on track was frequent use of Google Hangouts, as well as multiple face-to-face meetings in order to confirm project schedules and soft development deadlines.

No code completed was out-of-specifications.