

SOEN 6441 Project

Flight Schedule Search Software Development Project

|  |  |
| --- | --- |
| **Name** | **ID** |
| **Smita Deshpande** | **40198521** |
| **Rohan Dhiman** | **40230689** |

Project Overview

In this project, we have built a software to search flights schedule for given 5 different cities. User is given a choice to enter the origin city, destination city, departure date, return date if applicable and if he or she belongs to some category of the user. Based on the users input, the application pings the API, processes the data received and displays the data to the user in human readable format. Our application is using the Skyscanner’s partner public APIs to read available data from Skyscanner’s dataset for the provided user’s input. Then it parses the json data and displays the flights of user interests. It also provides feature to save his favorite flights in the local database. So whenever he logs in back, he can retrieve his favorite flights from the database. Also, the user is given a choice to delete or add any new flight from his favorite flights stored in the database. We have used postgres sql as the relational database.

In our design we have used, object relational mapping, Table DataGateway and Strategy Design Pattern.

1. Flow

First Login page appears, and the user enters the username and password. If the user account already exists, then the login would be successful and the main page opens up. If entered credentials do not match the ones present in the database, the login won’t be successful. The user needs to create new account using the sign up page. The username is unique, so every user has a unique username. The login credentials for the new user are saved in the database in a UserInfo Create Table. As soon as the login page is created, it creates userInfo table in it if it does not exists.

After the user logs in successfully, main page opens up which in turn creates a favorite flights table in the database if it does not already exist.

In the main page, user can get its previously stored favorite flights by clicking on my favorites button or search for new flight. If user clicks on My favorite page, the software navigates to Favorite page, which displays all the saved flights in the database by the user. User can remove any flight from the favorites list by clicking on delete button. Else, to search a new flight, user needs to input origin city, destination city, departure and return date and Fare Type to get the available flights as per user choices. On the click of Search Flights button on the main page, the software, takes the user input, adds some few extra parameters required to read the API of browseRoutes from the Skyscanner API URL <https://skyscanner.github.io/slate/#browse-quotes>. If there are any flights available as per user choices, the flights are displayed on the main screen, outbound flights are displayed on the left and inbound flights are displayed on the right table. The software parses the json data and displays the relevant results to the user. The user is then given a choice, whether to add it to the database or not by clicking the Add to Favorites button on the main page.**Diagram, schematic

Description automatically generated**

Figure 1: Flow Diagram

1. Software Design:

Below mentioned software design techniques are implemented in our project.

1. **Object Relational Mapping**: Here, the UserInfo table in the database is used to fetch the object of UserInfo class. We use the user Id received from this class to fetch the favorite flights of the user based on the user id. The Object FlightInfo depends on userId which is used a foreign key referencing the part of attributes in favorite flights table.
2. **Table Data Gateway**: We have created two tables. One of them saves favorite flights of the user corresponding to his user id in this postgre sql database. Other table save user information like, username, password and user id. FavFlightGateway and UserLoginGateway are the two classes meant to access the DB to perform operations of insertion, deletion and searching flights based on user id and user information based on user credentials respectively. No other classes are meant to access the database directly. Thus, we have kept logic to perform sql queries decoupled from the domain logic.
3. **Strategy Design Patterns**: While deciding the price of the flights for the user, we have implemented strategy pattern. PricingStudent, PricingSeniorCitizen, PricingArmedForces classes implement the IPricingDiscounting interface. These classes calculate the prices of flights based on the type of user.
4. Class Diagram

A screenshot of a computer screen

Description automatically generated with medium confidence

Figure 2: Class Diagram

The overall software is divided into three layers,

1. Graphical User Interface
2. Middle ware connecting the database and the GUI
3. Database connecting layer.
4. Graphical User Interface

The GUI consists of four pages.

1. Main Page
2. Login Page
3. Sign Up page
4. Favorites Page

Login Page Class: It is a starting point for the software. It initializes IUserSearch interface and instantiates and uses the Sign-Up page class to navigate to Sign up page. Else it initializes the static instance of Main frame and uses it to navigate to main page on successful login. It uses IUserSearch to retrieve the data for the current user. Compares if user entered credentials match the one in local data base, if yes then login is displayed successful and navigated to main page.

Main Class: It creates a table of favorite flights if not exists. It initializes IFlightSearch interface using FlightSearch class. It retrieves the user data from gui and uses “UserOptions” class to send information of these constraints to IFlightSearch interface to search the flights. Once the IFlightSearch interface returns the list if flights, this class then populates the outbound and inbound flights. Main page also provides a feature to save the flights which user finds his or her favorite flights in the local database.

Favorite Page: This page uses the IUserSearch interface to retrieve his saved favorite flights from the local database and display it on the screen. It also lets user to delete any of those flights or return back to main page.

1. Middleware Layer
2. Interfaces:

IUserSearch interface – This interface provides three functions like checkIfUserNameExists, GetUser and SaveUser. It allows the UserSearch class from the User package to implement this interface. This interface is used in the login or signup functionality of the software

IFlightSearch interface – This interface provides 4 functions namely GetFlights, SaveFlights, GetFavFlights, DelFromFav, GetDiscountedPrices. The class FlightSearch implements this interface.

IPriceDiscounting: This interface provides the method UpdateFlightPrices.

1. Flight package: It contains below classes:

FlightSearch: It implements IFlightSearch interface. It instantiates FavFlightsGateway object and calls the functions of this class to access and operate on database. This class provides functionality to create http request and fetch the information from skyscanner browsequotes api based on user inputs. It parses the json data received in the response using json simple library. It prepares response using FlightInfo class and saves the list of these flights as its member variable. This is kept as member variable to provide functionality of saving the favorite flights selected by the user. This class also provides functionality to instantiate the Pricing class based on type of user and we are implementing strategy design pattern here.

FlightInfo: It contains attributes that describes the flight details to be shown to the user. The FavFlightTable row can be directly mapped to this class.

FavFlightsGateway: It contains methods to access database and do operations on Flights table. FlightSearch saves and retrieves favorite flights of the user from this table. Also, it is possible to delete any flight based on user’s choice. SQL queries like Select, Insert, delete ddl queries.

PricingForArmedForces: It implements IPriceDisounting interface. Applies 30% discount on flight prices.

PricingForSeniorCitizen: It implements IPriceDisounting interface. Applies 20% discount on flight prices.

PricingForStudent: It implements IPriceDisounting interface. Applies 10% discount on flight prices.

1. User package:

UserInfo: This class represents the row in the table UserInfo in the database.

UserOptions: It represents the class which initializes its attributes based on user input from the user and displays. This is provided to the FlightSearch class which uses it to form the request API parameters.

UserSearch: This class is present in the middleware layer and instantiates UserInfo object. It interacts with database using UserLoginGateway static instance and provides results to the “view” pages. It implements methods such as searching for valid user, if the username is unique, save new user information. The m\_id is considered as primary key for that table.

UserLoginGateway: This class interacts with database and performs sql query operations such as select and insert. Each row in the userInfo table maps to UserInfo class.

1. Database Interaction Layer

There are basically two tables in the database namely Flights table and UserInfo table.

* 1. Flights: The flights table consists of attribute fields such as m\_flightid, m\_id, m\_originairport, m\_destinationairport, m\_carrier\_in, m\_carrier\_out, m\_departuredate, m\_returndate, m\_price. Row of this table maps to FlightInfo class. Each row except userId is unique. It consists of composite primary key consisting of all attributes except for user id. FlightId is serial unique key. The userId is a foreign key referencing to m\_id in UserInfo table.
  2. UserInfo: UserInfo table consists of m\_id which is the userId referred throughout this document is the primary key and is incremented serially, username, password. Username is not kept unique in the database logic, we implement this in the domain logic to keep the username for each user unique.
  3. User package:

UserInfo: This class represents the row in the table UserInfo in the database.

UserOptions: It represents the class which initializes its attributes based on user input from the user and displays. This is provided to the FlightSearch class which uses it to form the request API parameters.

UserSearch: This class is present in the middleware layer and instantiates UserInfo object. It interacts with database using UserLoginGateway static instance and provides results to the “view” pages. It implements methods such as searching for valid user, if the username is unique, save new user information. The m\_id is considered as primary key for that table.

UserLoginGateway: This class interacts with database and performs sql query operations such as select and insert. Each row in the userInfo table maps to UserInfo class.

1. Database Interaction Layer

There are basically two tables in the database namely Flights table and UserInfo table.

* 1. Flights: The flights table consists of attribute fields such as m\_flightid, m\_id, m\_originairport, m\_destinationairport, m\_carrier\_in, m\_carrier\_out, m\_departuredate, m\_returndate, m\_price. Row of this table maps to FlightInfo class. Each row except userId is unique. It consists of composite primary key consisting of all attributes except for user id. FlightId is serial unique key. The userId is a foreign key referencing to m\_id in UserInfo table.
  2. UserInfo: UserInfo table consists of m\_id which is the userId referred throughout this document is the primary key and is incremented serially, username, password. Username is not kept unique in the database logic, we implement this in the domain logic to keep the username for each user unique.

Diagram

Description automatically generated

1. Refactoring

Refactoring, or code refactoring in full, is a systematic process of amending previously built source code, without introducing new functionalities or altering the fundamental workings of the subject software.

Refactoring Methods Used:

1. Extract Method

We used the extract method to refactor the functions and methods used in the code. We newly described the old methods into a more stabilized , clean and properly labeled ones in a way making it easier to decipher the code.

1. Simplifying Methods

We used the simplifying method to simplify the complex code and functions used in the application.

Reducing the code redundancy wherever possible.

1. Testing

We have used Unit Testing to test our application code.

What is Unit Testing?

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation.

Bubble chart

Description automatically generated

Tool Used

We used JUnit to perform the Unit Testing of our application.

JUnit is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks which is collectively known as xUnit that originated with SUnit. JUnit is linked as a JAR at compile-time.

Alternatively we could also use JHipster or AspectJ tools to test the Application.

Test Cases In Our Application

We divided the test cases into the following units:

1. User SignUp
2. User Login
3. Flight Search
4. Add Flights To Favorites
5. Delete Flight From Favorites
6. User SignUp

For checking this we created a JUnit File with a test case wherein we provide a username and a password to the function SaveUser() (function used for creation of user ) .

The function returns a Boolean value true if user is created and false if an exception was caught or the user is already existing.

We checked this with the expected value “true”.

Test Case was created as follows:

Graphical user interface, text, application, email

Description automatically generated

Output of testing on successful signup:

A picture containing shape

Description automatically generated

Output of testing when user is already present in Database:

Graphical user interface, application

Description automatically generated with medium confidence

1. User Login

For checking this we created a JUnit File with a test case wherein we provide a username and a password to the function GetUser() (function used for fetching the user if exists ) .

The function returns the user details if it exists. We check the if the user details from the table are received or not.

Test Case was created as follows:

Graphical user interface, text, application, email

Description automatically generated

Output of the Test Case On Success Sign In:

A picture containing application

Description automatically generated

Output of the Test Case on Failed Sign In Attempt (User Id or Password Entered Incorrect):

Graphical user interface, text

Description automatically generated

1. Flight Search

For testing flight search we created an object of User Options that would be entered by the user from the UI and passed it to the Flight Search function.

The function returns a list of flights for the provided input.

We check if the function has returned the list or not.

Test Case was created as follows:

Text

Description automatically generated with medium confidence

Output of the Test Case On Success Flight Search:

A screenshot of a computer

Description automatically generated

Output of the Test Case On Failed Flight Search:

A screenshot of a computer

Description automatically generated

1. Add flights to favorite

For testing the adding to favorite option we pass a particular user option that would be selected by the user from UI to the function SaveFlights() (function used to add flights to favorites) .

First we call the GetFlights function to get the flight details from the database for the particular user and pass it to saveflights to save it in favorites. The function returns a Boolean true value on successful add.

Test Case was created as follows:

Text

Description automatically generated with medium confidence

Output of the Test Case On Successful Adding of flight to Database:

Graphical user interface, application, Word

Description automatically generated with medium confidence

Output of the Test Case On Failed Addition of the Flight to Favorite Table in Database:

Graphical user interface

Description automatically generated with low confidence

1. Delete Flight From Favorites

For testing the Deletion of a favorite added flight we pass the user id of the user and the flight id of the flight to be deleted to the function DeleteFlight , which searched for the flight in the database for the particular user and deleted it from there.

The function returns a Boolean value true on successful deletion.

Test Case was created as follows:

Graphical user interface, text, application, email

Description automatically generated

Output of the test case on successful deletion of the flight from favorites:

A picture containing application

Description automatically generated

Output of the test case on failed deletion of the flight (flight not present in Database):

This test case doesn’t fail as the record to be deleted is searched using the delete sql query using the where condition to search using the user id and flight id and if no record is found nothing would be deleted , so unless there is any other exception caused by wrong data or sql connectivity error , the following testcase will always result in true output.

1. References

Skyscanner API:

<https://skyscanner.github.io/slate/#api-documentation>

Postgres:

<https://www.postgresql.org/>

JUnit:

<https://junit.org/junit5/>