Application Development I (Desktop)

Marie Andrews & Vishnu Narayanan

Final Project Deliverable 3: Airline Ticketing System



Contents

[Project Scope 3](#_Toc173703709)

[Users 3](#_Toc173703710)

[Clientele 3](#_Toc173703711)

[Project Functionalities 3](#_Toc173703712)

[User Module 3](#_Toc173703713)

[Show All Button 4](#_Toc173703714)

[Search with Filters 4](#_Toc173703715)

[Flight Selection 5](#_Toc173703716)

[Sign In/Sign Up 6](#_Toc173703717)

[New User Registration 6](#_Toc173703718)

[New Agent Registration 7](#_Toc173703719)

[Log In 7](#_Toc173703720)

[Booking Module 9](#_Toc173703721)

[Initial Redirection When Logged In 9](#_Toc173703722)

[Booking History 10](#_Toc173703723)

[Admin Module 11](#_Toc173703724)

[Add Records 11](#_Toc173703725)

[Since Midterm Project Evaluation 11](#_Toc173703726)

[Project Setup 12](#_Toc173703727)

[Work Progress in Future 12](#_Toc173703728)

# Project Scope

## Users

The users of the Airline Ticketing System application will be both regular users seeking flights at the major airlines utilizing the platform as well as travel agents seeking flights for third party clients. Travel agents could particularly find the application useful as it could assist in streamlining aspects of their job duties, but all users should find the application suited to searching flights across a number of airlines based on personally selected filter parameters.

## Clientele

The primary clientele for the Airline Ticketing System application is the independent travel agent industry. Independent travel agents are not affiliated with major travel brands and often work from home. Our desktop application is ideal for this use case and would enable a small independent agent to compete with larger agencies volume wise by providing an ad-free, reliable, safe environment for acquiring the best rate on airline tickets.

# Project Functionalities

## User Module

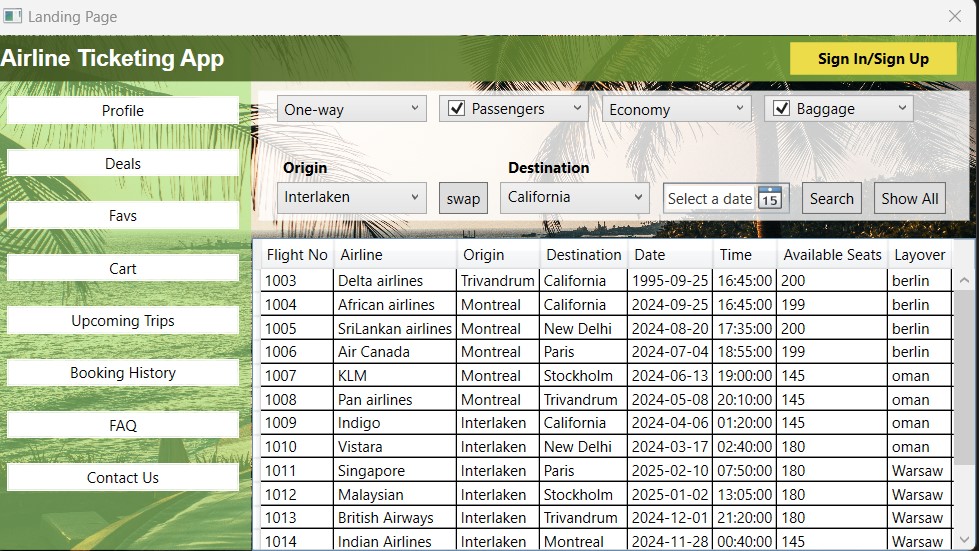
The application opens on the Landing Page:

A screenshot of a computer

Description automatically generated

### Show All Button

When a user clicks the button labeled Show All, the display grid is populated with all available flights in the database. The button click event calls method *refreshDisplayGrid().* This method is an asynchronous method that allocates the result of the REST Api method *getAllFlights()* into a List of Flight objects called allFlights. The *getAllFlights()* method queries the database to select all from the Flights table. List allFlights populates the display grid.



### Search with Filters

When a user selects filters from the available options and clicks the button labeled Search, the display grid is populated with only flights matching the selected filters. The button click event retrieves the data from the selected filters and creates a new FlightsFilter object. The new FlightsFilter object is passed to the REST Api method *GetFlightsByFilter().* This method queries the database for all records in the Flights table with attributes like the object attributes.The result of this asynchronous method is stored in a List of Flight objects called availableFlights. List availableFlights populates the data grid.

A screenshot of a computer

Description automatically generated

### Flight Selection

When a user selects a flight from the list on the display grid, a window box pops up with the details of the flight and asks the user if they want to confirm this selection. The user is able to click yes to confirm or no to go back to the list.

The display grid selection change event tries to retrieve the data from the selected flight into a new Flight object. If the object is not null, the window box will pop up with the flight confirmation, inserting the object attributes into the string of text displayed in the window box.

A screenshot of a computer

Description automatically generated

If the user is logged in, they are redirected to the Booking Module to finalize their flight confirmation. If they are not logged in, they are prompted to log in to complete their confirmation.

The application checks the value of the Boolean isLoggedIn, which is set to *true* when a user successfully logs in and *false* when a user successfully logs out. If the value of isLoggedIn is false, a new window box will alert the user to please log in to continue.

A screenshot of a computer

Description automatically generated

### Sign In/Sign Up

When the user clicks on the button labeled Sign In/Sign Up, a new window opens. Each button click event opens a new window: either a registration window for a new user or new agent, or a log in window for a current user or agent. Clicking on the button labeled Back to Search closes the current window and re-opens the main window (the landing page).

A screenshot of a login form

Description automatically generated

### New User Registration

A new user registers by entering the requested information in the text boxes and clicking on the button labeled Save & Submit.

The submit button click event stores the user’s data into Strings which are added as attributes to a new User object. This User object is passed to an ApiRequest method *postUserApi().* This method queries the database with an insert statement to insert the User object attributes into a new record in the User table in the database.

A screenshot of a computer

Description automatically generated

### New Agent Registration

Travel agents are registered in the same manner as users, but they are also required to register with their agent license number. This license number is also verified at sign in via the same method used for username and password.

### Log In

Once a user is registered, they can log in by click the button labeled Log In from the same registration window. They can also click the button labeled Current Users Sign In in the Sign In/Sign Up window. These button click events open the user log in window and close the current windows.

The user then attempts to sign in with their chosen username and password. The Sign In button click event stores the username and password into Strings which are passed to the ApiRequest method *userSignInApi().* This method queries the database by selecting all records from the Users table where the username and password match the Strings from the user sign in. A new User object is created and its attributes set to the values returned from the database. This object is returned to the button click method. If the object is not null (and thus the database has found a matching user), a success message is displayed and the user is redirected to the Landing Page. At this point the object’s Boolean isLoggedIn is set to *true* and the User object declared in the Main Window is set to the logged in user.

A login screen with a beach and palm trees

Description automatically generated

If the object is null, a fail message is displayed instead and the user remains on the sign in page.

A login screen with a beach and palm trees

Description automatically generated

## Booking Module

### Initial Redirection When Logged In

If the user is logged in when they select a flight, they are redirected to the Billing Module.

When they click Yes to confirm the flight, a new booking window is opened. The user enters their information. Here they can select the button labeled For Myself? This button will prefill the user’s name in the Passenger Details section.

The user object’s name attributes are used to populate the name text boxes.

A screenshot of a computer

Description automatically generated

When the user clicks the button labelled confirm, the data is collected and saved. A window box opens letting the user know the booking is confirmed, their seat number, and advising them to navigate to Booking History for more information.

The confirm button click event creates a new BookingDetails object and sets its attributes to the data retrieved from the Booking window. The object is passed to the REST Api method *PostBookingTicket()* which queries the database for records in the Flights table matching the flightNo of the object. If the number of available seats is greater than 0, an insert statement is called on the database to insert the BookingDetails object attributes into the Bookings table. It then updates the number of available seats in the Flights table record by subtracting 1 from the current number based on selecting the record matching the BookingDetails object’s flightNo. If the number of available seats is 0, a window box opens alerting the user that their selection is not available.

The result of *PostBookingTicket()* is stored in a separate BookingDetail object within the event method. If this object is not null upon completion of the REST Api method, a window box opens with confirmation details taken from the new BookingDetail object.

A screenshot of a computer

Description automatically generated

### Booking History

If the user navigates back to the Landing Page, they can access the Booking History tab. Their previously confirmed flights will be displayed in a table, and if they select a flight, its details will be fully displayed.

The display grid is populated by List of type BookingDetail *bookingDetails* on opening the new window. This List is the result of a REST Api method *getBookingDetailsByUserId().* This asynchronous method takes in the main window user’s id and queries the database by selecting all records from the Bookings table left joined on the Flights table where the flightNos match and the Bookings id matches the user id. It initiates a new *bookingDetails* List composed of BookingDetails objects which contain Flight objects. If the List contains at least one object, it is successful.

Within the *initializeDisplay()* method in the Booking History window, the method *populateDetails()* is called to fill the text boxes with the BookingDetail and Flight object attributes. Both of these methods utilize an index declared outside to display the further details about the selected row of the grid. When there is a Display Grid Selection Change event, the *populateDetails()* method is called with the index set to the Display Grid’s selected index.

A screenshot of a computer

Description automatically generated

## Admin Module

### Add Records

When a user with Admin credentials signs in, they are automatically redirected to the Admin Module, opening with the Add A Record window. The user enters the requested information and click the button labeled Save.

The save button click event creates a new Flight object with attributes set to the data input by the Admin user. This object is sent to the ApiRequest method *postFlightApi().* This method queries the database with an insert statement inserting the Flight object values into the Flights table.

A screenshot of a computer

Description automatically generated

## Since Midterm Project Evaluation

Since the midterm project evaluation, the User Module has been fully realized, the Billing Module has been created and implemented, and the Admin Module has been implemented. The majority of the logic was created and the main functionality of the application was put into effect. The database was updated to include user type to allow a check for Admin users, as well as a table tracking all booking details.

# Project Setup

Database: Microsoft SQL Server Management Studio Version 19.3.4.0

Visual Studio Version 17.10.2

The project can run locally as is.

# Work Progress in Future

Additional features that can be added to the application include:

* Saved user profiles including preferences for flight filters
* Upcoming trips tab displaying a user’s upcoming confirmed flights
* Functionality to allow logged out user to maintain their flight search upon logging in