Application Development I (Desktop)

Marie Andrews & Vishnu Narayanan

Final Project Deliverable 3: Airline Ticketing System



Contents

[Project Scope 3](#_Toc173679880)

[Users 3](#_Toc173679881)

[Clientele 3](#_Toc173679882)

[Project Functionalities 3](#_Toc173679883)

[Landing Page 3](#_Toc173679884)

[Show All Button 4](#_Toc173679885)

[Search with Filters 4](#_Toc173679886)

# 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

## Landing Page

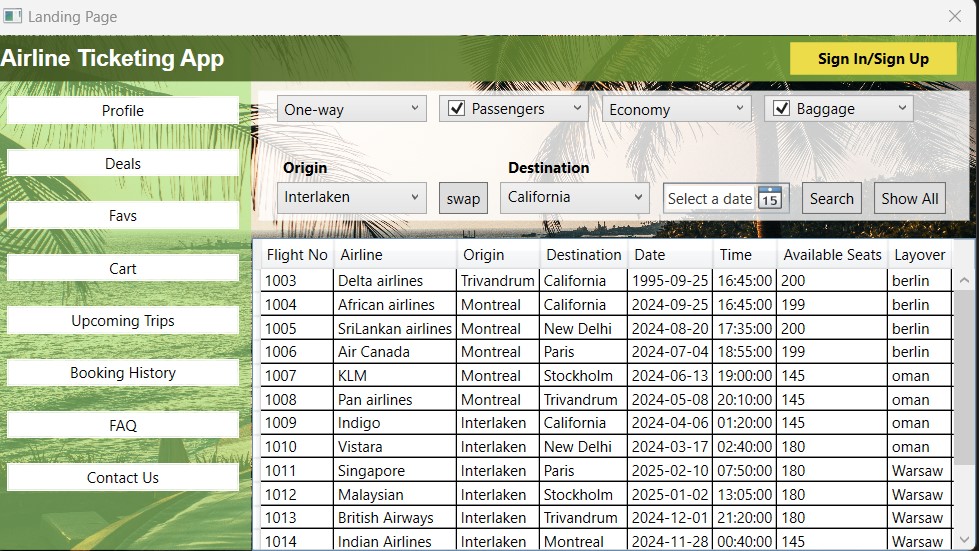
The application opens on the Landing Page:

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

## 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