Flight and Weather Application Design Documentation

COMP.SE.110 Software Design

Teaching assistant: Eveliina Hämäläinen

Student name	Student number
Chu Duc Anh	50358922
Nguyen Quang Duc	151113370
Kalle Hirvijärvi	150218713
Hamza Rizvi	152275479

Table of Contents

1. Project Description

The goal of this project is to implement an application that gives the user information about commercial flights based on a given departure and arrival destination. The information shown includes date and time, airline, type of aircraft, price and possible layover. Additionally, the application will show the user the relevant weather forecasts of departure and arrival location based on outbound and return date. Searched flights can be saved and conveniently accessed later, and users can also choose a preferred currency, temperature unit and filter results based on number of layovers and maximum price of the trip. The search results can also be sorted based on flight length, price or departure time.

2. Application Structure

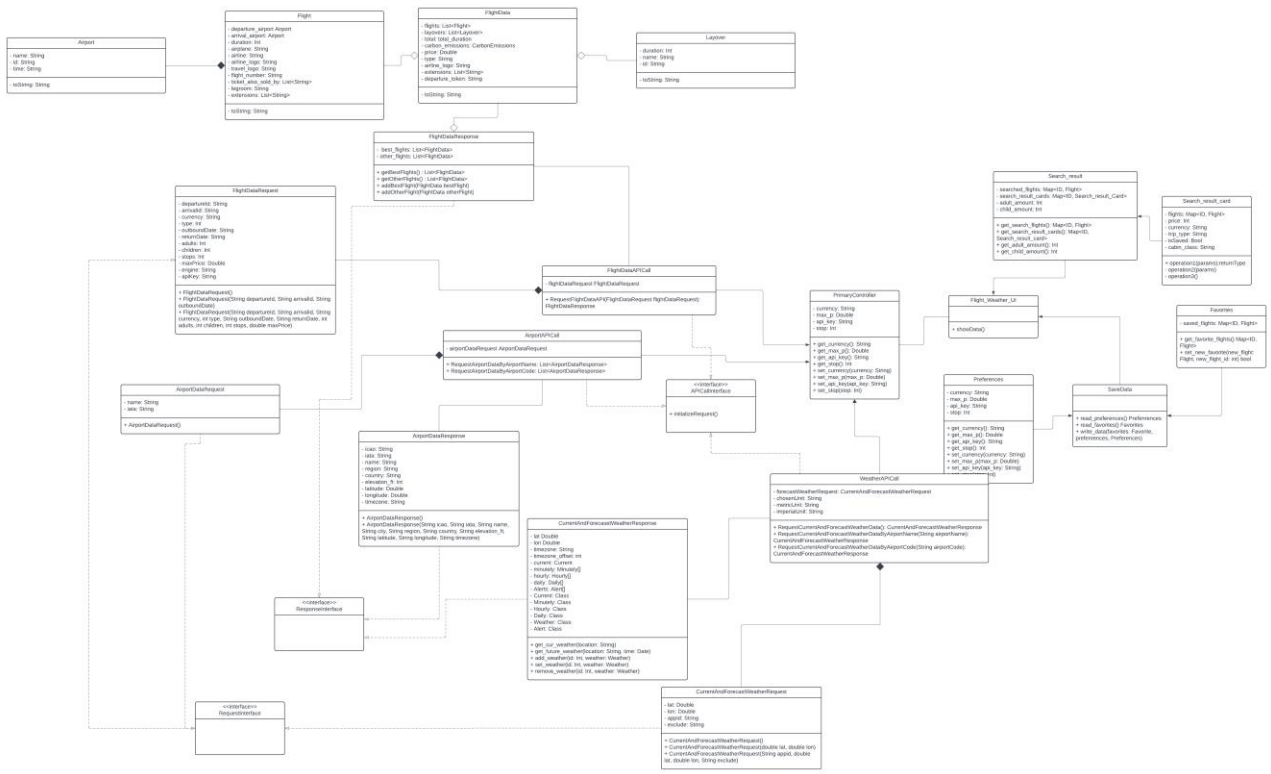

Figure 1: UML chart of the application.

The UML chart above illustrates the coding design structure which includes main components and interfaces such as Request classes, Response classes, API Call classes (FlightAPICall, WeatherAPICall and AirportAPICall), PrimaryController, SearchResults and Preferences.

For the request classes, which includes FlightDataRequest, AirportDataRequest and CurrentAndForecastWeatherRequest, there are important characteristics as follows.

- PrimaryController will initialize the Request and add its attributes partially or fully.
- Request class will contain parameters to form the URI of the API.
- For example: AirportDataRequest will contain the airport's IATA code and the airport's name.

In terms of APIs Call classes (FlightDataAPICall, AirportDataCall and WeatherAPICall), their implementation is described below.

- PrimaryController will modify the Request in the APIs Call classes
- The APIs Call classes will fetch the data from APIs (Google Flight API, Ninja API and Open Weather API)
- The APIs return the Response to the Primary Controller
- For example: AirportDataCall will form the http request based on the Request class the return the Response corresponding to the response from the API.

Response classes, which are FlightDataResponse, AirportDataResponse and CurrentAndForecastWeatherRespone, are assigned to conduct the actions shown below.

- PrimaryController will receive list of Response or a single Response.
- Response will contain data from the response of the API based on the Request.

Meanwhile, the class Flight_Weather_UI is responsible for the following tasks.

- Flight_Weather_UI class will present the user's data, user's preferences.
- Flight_Weather_UI class will present the searched results and the weather corresponding to the searched result's locations.

To handle the users' preferences, the class setting management is allocated to follow these instructions.

- Search preferences, saved favorite flights and the latest search result are locally stored in settings.json file in the root directory
- InfoCardStorage stores SearchResultCard objects in three sets where each set stores all the objects in a different order.
- Preferences stores users search preferences during runtime
- SaveData stores two InfoCardStorage objects for saving favorites and the latest search result locally and a single Preferences object. The previously saved data is read in the constructor and can be overwritten with a public method at any time.

3. User Interface

Figure 2 shows the user interface of the application's main menu. On the top of the screen, the departure and arrival airports can be searched and found. Furthermore, desired outbound and return dates and the number of adult and child passengers can be chosen with date pickers and dedicated buttons. In the bottom left corner, there are additional preferences for the search that can also be hidden by the "See less" button. These preferences allow the user to change the currency, temperature unit, maximum price and maximum layovers before the search for flights. Once you press the search button, the search results for the flight show up in the middle of the screen with details and option to be saved to later retrieval. Each flight of the search result contains information about airlines' icons and names, departure and arrival date and time, flight duration, airplane names, names and IATA codes of departure and arrival airports, layover duration and airports and lastly additional data such as carbon emissions, seat class and legroom size. For the weather data of departure and arrival destinations, they will be updated when both airports and at least outbound date are selected. However, if the selected date is more than one week away from the current date, the weather data would not be available. Below the preference settings, there is a button named "Saved flight", which will list the saved flights in the middle part of the interface. Lastly, it is noteworthy that users can sort the displayed flights by cost, duration and departure time using the sorting bar above the list of flights.

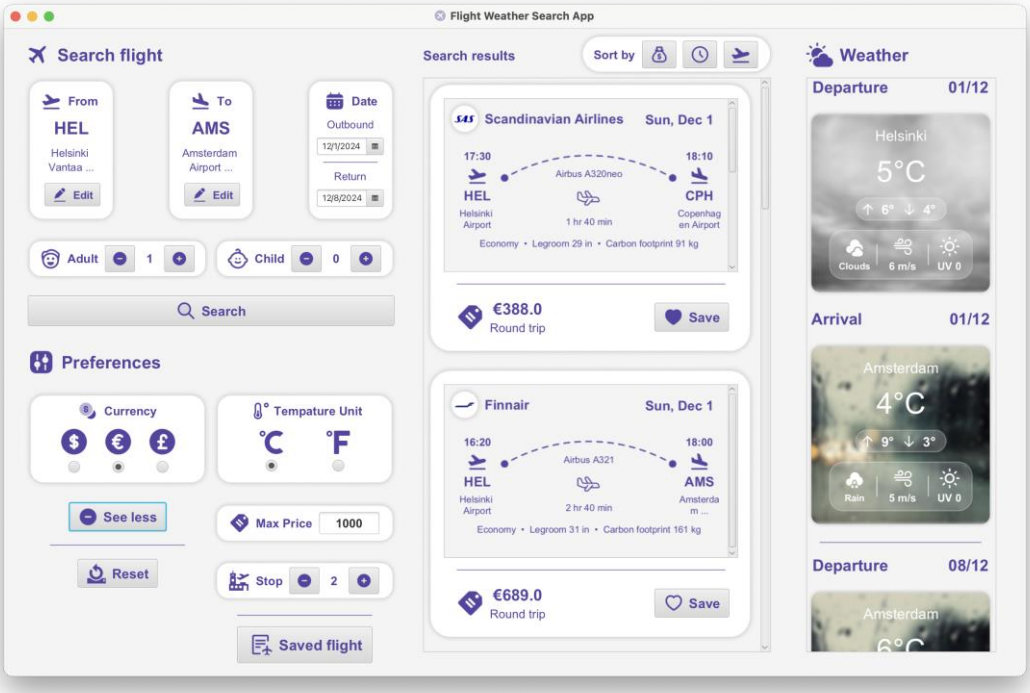

Figure 2: The user interface of main menu.

When the “Saved flight” button is pressed, all searching-related buttons will be disabled to avoid user from making erroneous interactions. Moreover, all the saved flights will be displayed in the middle of the window, and users can utilize the sorting bar to implement the sorting mentioned earlier to the saved flights. To remove the saved flights from the list, user can press the “Save” button one more time to make sure the heart icon becomes hollow, meaning the flight has been removed from the saved list. Another important trait of this saved list is that it allows flights will multiple different currencies to appear at the same time as shown in Figure 3. Finally, to be able to return to previously made search, the “Go back” button can be selected in the same position as the "Saved flight” button.

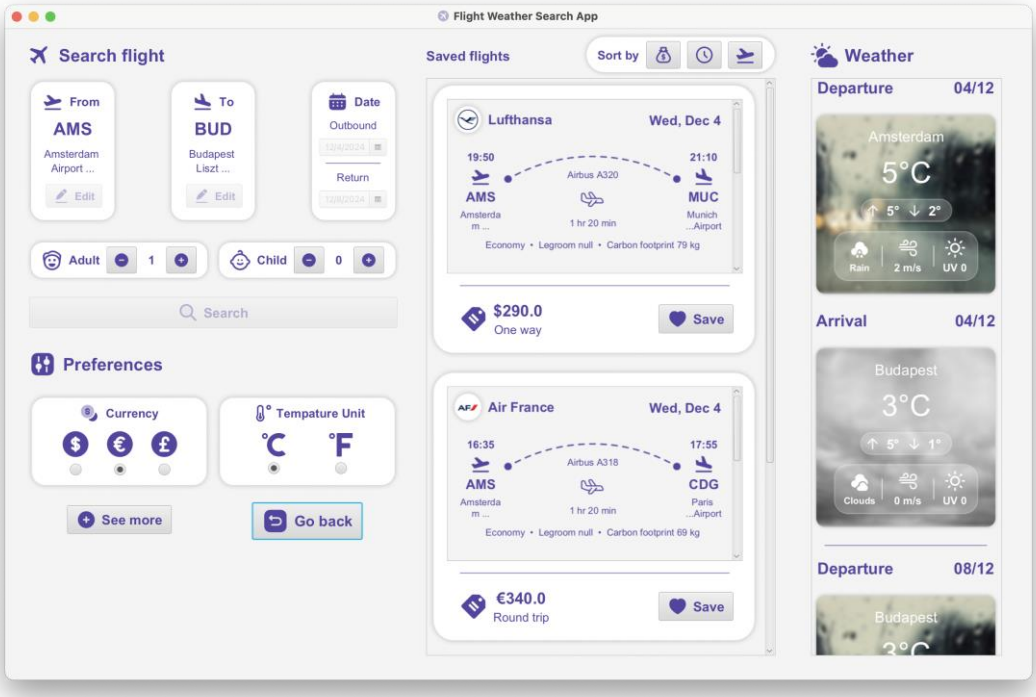

Figure 3: The user interface of saved flight.

During the designing process of the software’s interface, Figma is chosen as a design tool due to its user-friendly interface, collaborative features and flexibility. During the implementation phase, SceneBuilder is used to create and customize the FXML file, which is then compiled in JavaFX to generate the desired design.

4. Design Pattern and Architecture

The design architecture that fits the application is the Model-View-ViewModel (MVVM) design architecture, and each component of this architecture will be presented as follows:

- For the View component, the application will use an FXML file to define its layout and UI. Moreover, the FXML components will handle all the events and will parse the information to the PrimaryController. After processing information from the APIs, the PrimaryController will provide data from Model to the FXML components to update.
- When it comes to ViewModel, the application will use the PrimaryController to connect with FXML. The main role of the PrimaryController is to receive input from the FXML components and parse it to model through APICall classes. After processing data from the model, the PrimaryController will check the data's validity and parse it to the UI.
- In terms of Model component, it has following important roles:
  - The Model contains Request, APICall, and Response classes.
  - The ViewModel will call the API Call classes and initialize the Request object.
  - With the Request object from ViewModel, the API Call classes will generate Http request to get data from APIs then return Response object.
  - FlightAPICall and WeatherAPICall contain static FlightData.

When examining the project through the lens of design pattern, Singleton design pattern is one possible option where:

- There will be only one FlightDataRequest, and CurrentAndForecastWeatherRequest object to be used throughout the entire application.
- The FlightDataRequest and CurrentAndForecastWeatherRequest will be stored in FlightDataAPICall and WeatherAPICall.
- SaveData is only created once to read all locally saved data after which it's only used to provide a method for rewriting the settings.json file
- There is only ever one Preferences object since the cannot be multiple sets of user preferences.

Another possible design pattern candidate is Builder design pattern, where:

- FlightDataRequest class uses the builder design pattern by initializing the builder to initialize the Request's properties.
- The FlightDataRequest contains multiple properties (departure_id, arrival_id, outbound_date, return_date,...)

5. Installation and Run Instructions

To be able to install and run the application, you should follow one of the two main ways to install and run the application:

1. The first method is to run through command line, which is considered quick and easy.
    - Firstly, you need to clone this repository into your local device.
    - Navigate to the application's JAR file, which is located in the 'Release' folder of the group's root directory.
    - Next, you can run it by using the command line 'java -jar "FlightWeatherApp.jar"' at the 'Release' folder's directory.
    - This command line is supposed to work in Windows operating system; however, we have not fully tested the software on MacOS counterpart yet.
2. The second method is to run through Netbeans IDE, which is a more reliable way.
    - Firstly, you need to clone this repository into your local device and configure it as Maven project.
    - Make sure you have installed Netbeans IDE and supported modules such as JavaFX with correct version stated at the end of the README file.
    - Open the project in Netbeans and select the "Run" button to start the application.
    - This method has been proved to work on both MacOS and Windows operating systems.

6. Implemented API Description

In the final product, we have successfully integrated a total of three separate Application Programming Interface (APIs), including Google Flights API, OpenWeather API and Ninja API. To be more specific, these APIs are utilized to retrieve all the flight data, forecast weather and airport information respectively. The names of these APIs and their URL leading to the documentation are listed below.

- Google Flights API: https://serpapi.com/google-flights-api/
- OpenWeather API: https://openweathermap.org/
- Ninja API: https://api-ninjas.com/api/airports/

7. Self Evaluation

The basic structure of the program remained mostly unchanged, however the code ended up being divided into more classes than originally anticipated, perhaps making the high-level structure of the code needlessly complex. Also, the introduction of new design patterns during the course changed our plans. For example, implementing MVVM-architecture was done when there was already an extensive plan for the design. Also, the original design lacked some of the data that we wanted to display, which needed to be filled in later.

When it comes to implementing the interface of programs, due to the dynamic nature of designed prototype, we must make several compromises in the final product while maintaining all crucial functionality. Particularly, the layout of prototype is optimized for portrait screen, which is not well supported on JavaFX and Scene Builder, leading us to adopt a landscape design where the interface is divided into three concrete columns. Furthermore, since the displayed information is rich in the original design, we ought to adopt two-layer vertical scroll panes in the flight list to properly illustrate the flight information.

Moreover, the use of multiple design patterns and interfaces help manage the classes easily and can extend the functionalities of the application in the future. The management of classes helps build the API request using Builder and store the response in coherent structure.

Generally, regardless of compromises made in the user interface, all the original functionalities are successfully implemented in the final product.

8. Acknowledgement

We mostly use Artificial Intelligence (AI) to optimize the code syntax as well as produce a getter, setter functions for the responses of the API Call. Moreover, AI is also utilized to explain how the design patterns work and suggest how the design patterns can be applied to our application. The detailed list of used AI tools during the project is listed below.

- GPT-4o mini provided by OpenAI.
- Microsoft Copilot provided by Microsoft.
- Gemini provided by Google.