COMP.SE.110 Software Design

Flight-Weather-Application

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

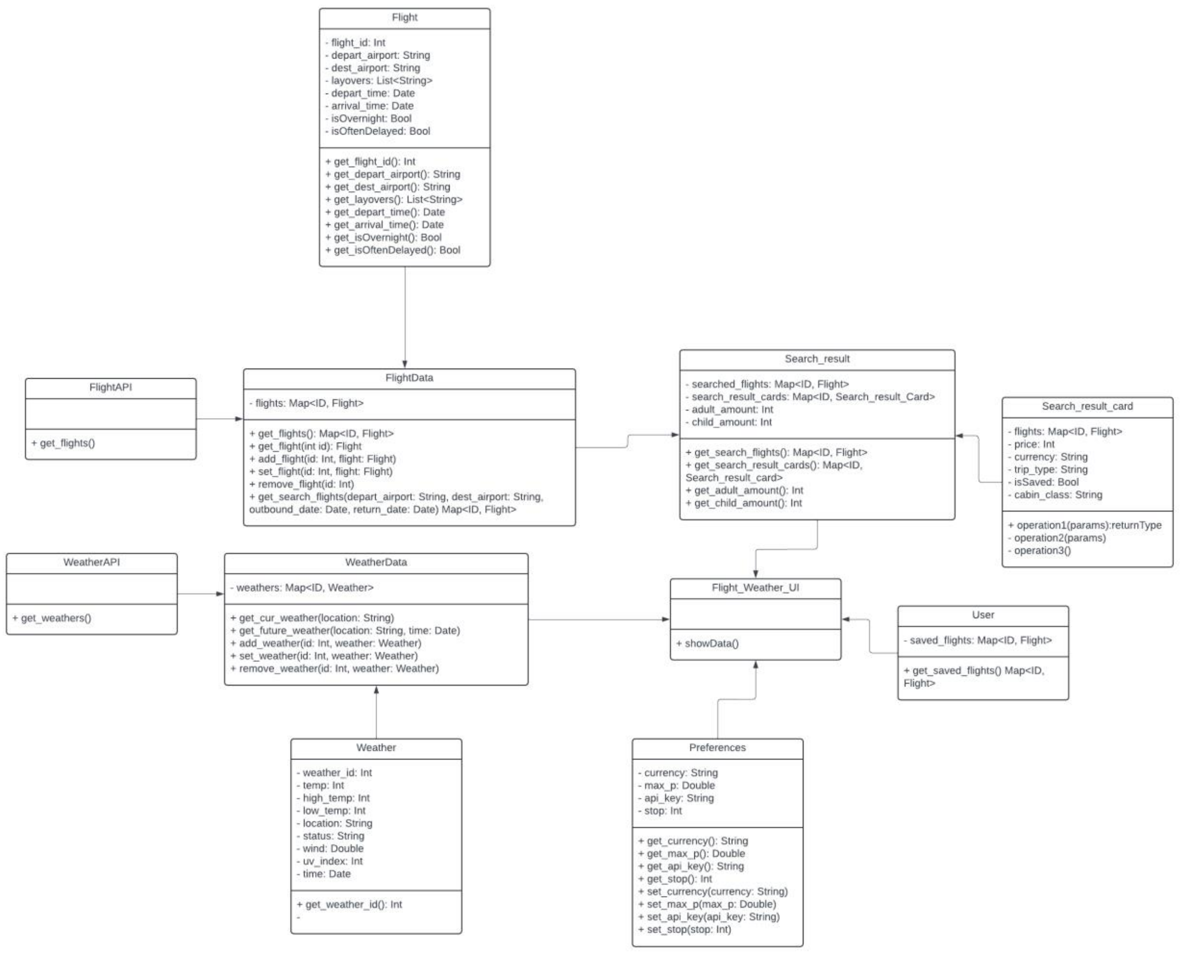
Table of Contents

1-Description 3
2-Structure 3
3-User interface 5
4-Design pattern and architecture 7
5-APIs 7

1-Description

The goal of the project is to create an application that gives the user information about commercial flights based on a given staring point and destination. The information shown includes date and time, airline, type of aircraft, price and possible layover. Additionally, the app will show the user the relevant weather forecasts based on local arrival and departure times. The user can also save favorite flights and conveniently access them later and choose a preferred currency and filter results based on number of layovers, number of passengers or the maximum price of the trip. The search results can also be sorted based on flight length, price or departure time.

2-Structure



Flight
- flight_id: Int
- depart_airport: String
- dest_airport: String
- layovers: List<String>
- depart_time: Date
- arrival_time: Date
- isOvernight: Bool
- isOftenDelayed: Bool
+ get_flight_id(): Int
+ get_depart_airport(): String
+ get_dest_airport(): String
+ get_layovers(): List<String>
+ get_depart_time(): Date
+ get_arrival_time(): Date
+ get_isOvernight(): Bool
+ get_isOftenDelayed(): Bool
FlightAPI
+ get_flights()
FlightData
- flights: Map<ID, Flight>
+ get_flights(): Map<ID, Flight>
+ get_flight(int id): Flight
+ add_flight(id: Int, flight: Flight)
+ set_flight(id: Int, flight: Flight)
+ remove_flight(id: Int)
+ get_search_flights(depart_airport: String, dest_airport: String, outbound_date: Date, return_date: Date) Map<ID, Flight>
Search_result
- searched_flights: Map<ID, Flight>
- search_result_cards: Map<ID, Search_result_Card>
- adult_amount: Int
- child_amount: Int
+ get_search_flights(): Map<ID, Flight>
+ get_search_result_cards(): Map<ID, Search_result_card>
+ get_adult_amount(): Int
+ get_child_amount(): Int
Search_result_card
- flights: Map<ID, Flight>
- price: Int
- currency: String
- trip_type: String
- isSaved: Bool
- cabin_class: String
+ operation1(params):returnType
- operation2(params)
- operation3()
WeatherAPI
+ get_weathers()
WeatherData
- weathers: Map<ID, Weather>
+ get_cur_weather(location: String)
+ get_future_weather(location: String, time: Date)
+ add_weather(id: Int, weather: Weather)
+ set_weather(id: Int, weather: Weather)
+ remove_weather(id: Int, weather: Weather)
Flight_Weather_UI
+ showData()
User
- saved_flights: Map<ID, Flight>
+ get_saved_flights() Map<ID, Flight>
Weather
- weather_id: Int
- temp: Int
- high_temp: Int
- low_temp: Int
- location: String
- status: String
- wind: Double
- uv_index: Int
- time: Date
+ get_weather_id(): Int
-
Preferences
- currency: String
- max_p: Double
- api_key: String
- stop: Int
+ get_currency(): String
+ get_max_p(): Double
+ get_api_key(): String
+ get_stop(): Int
+ set_currency(currency: String)
+ set_max_p(max_p: Double)
+ set_api_key(api_key: String)
+ set_stop(stop: Int)

Figure 1: UML chart

The UML chart shows the coding design structure which includes main components and interfaces such as FlightData, WeatherData, Search_result, and Flight_Weather_UI.

FlightData class:

- FlightData gets flight data from FlightAPI class and stores those data in a map data structure. A map will include multiple Flight components that represent one Flight unit.
- FlightData class can provide the Flight based on the flight's ID and can add, modify, and remove the Flight unit from the data structure.
- FlightData class can provide a searched flight based on departure airport, destination airport, and flight date.

WeatherData class:

- WeatherData gets weather data from the WeatherAPI class and stores data in a map structure. A map will include multiple Weather components that represent the Weather unit.
- WeatherData class can provide the current weather based on the given location.
- WeatherData class can predict the future weather for up to 8 days based on the given location.
- WeatherData can add, modify, and remove Weather units from the data structure.

Search_result class:

- Search_result class stores the searched flight in the map data structure.
- Search_result class has searched cards which will present searched flights in card form in the User interface.
- Search_result class stores the amount of adults and children in the flight.
- Search_result class will provide searched flights, cards, number of adults and children.

Flight_Weather_UI:

- 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.

3-User interface

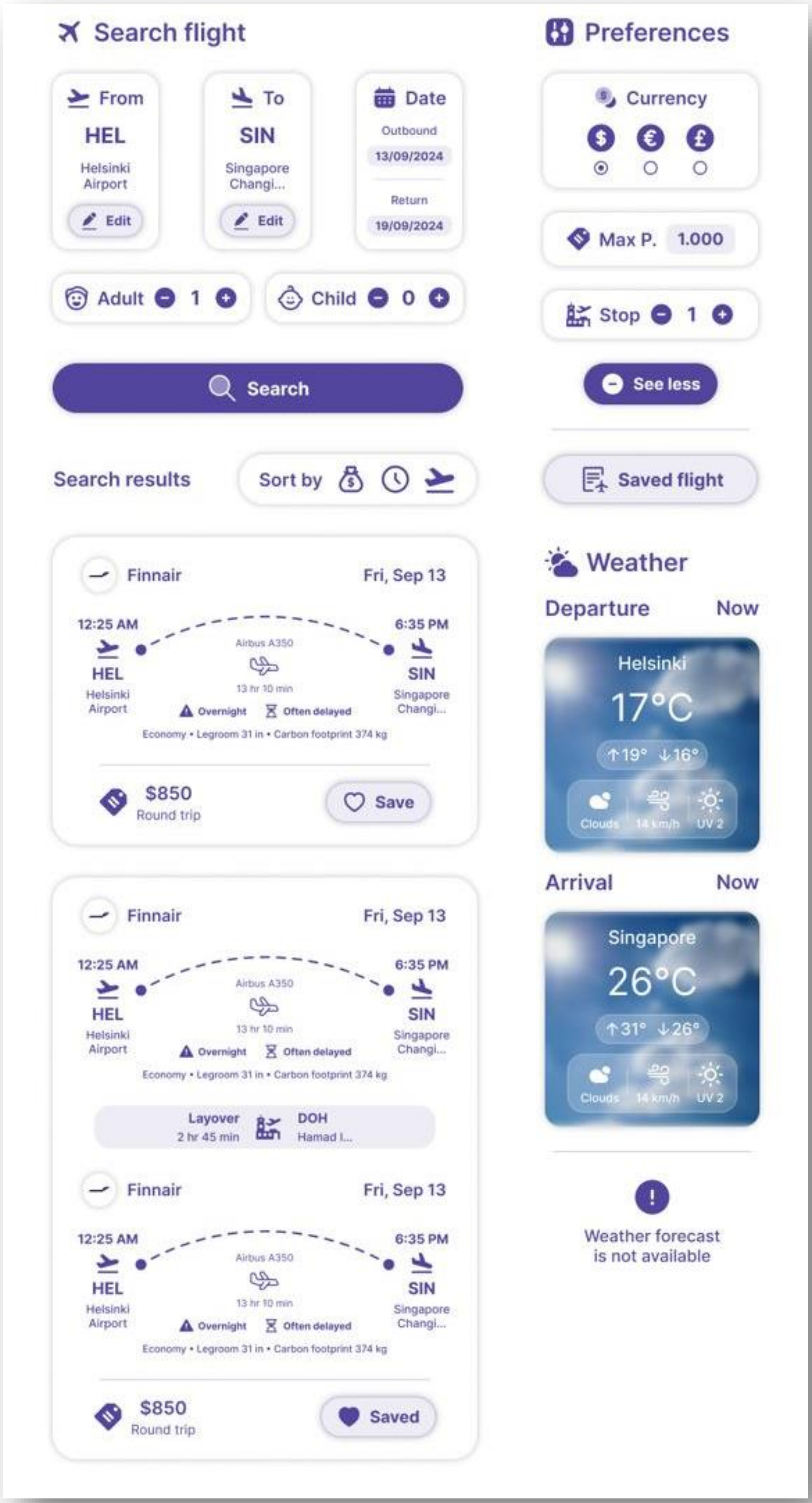


Figure 2: Main menu user interface

This is the main menu of the app. On the top of the screen the starting and destination points can be selected, as well as desired date and the number of passengers. In the top right corner, there are additional preferences for the search that can also be hidden by the "see less" button. Once you press the search button, the search results for the flight show up in the middle of the screen and also the weather forecast or the current weather (depending on if there is a forecast for the selected date) is displayed on the right side along with the search results.

Below the additional settings there is a "saved flight" button that is used to navigate to an additional menu.

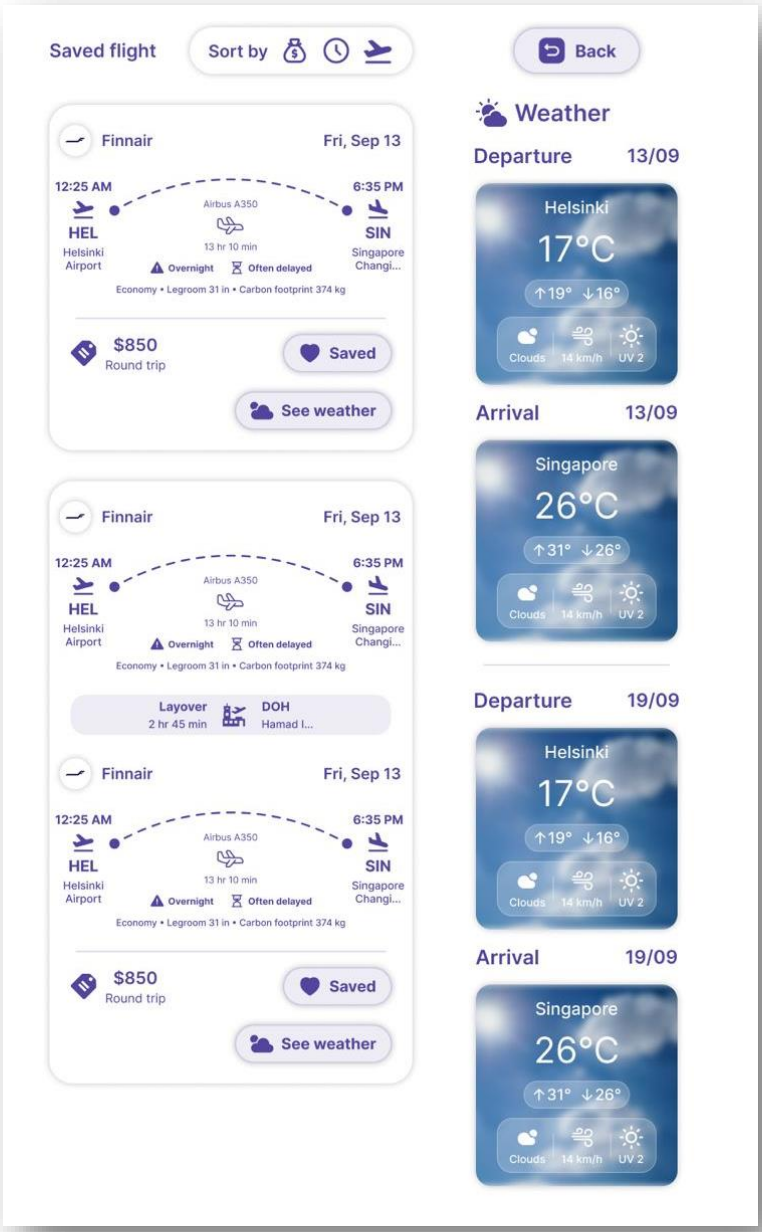


Figure 3: Saved flight window

Like in the main menu, the saved flights window displays the flight cards on a wider column on the left side of the screen and all the related weather information is displayed on the right side of the screen in a narrower column. The “back” button in the top right corner is used to navigate back to the main menu.

4-Design pattern and architecture

The design architecture that fits the application is the Model View Controller (MVC) design architecture.

View: The application will use an FXML file to define its layout and UI.

Controller: The application will use the Controller as Java classes which will handle user inputs, update the View, and interact with the Model. Examples of Java classes: Flight_Weather_UI class.

Model: The application will use the Model as Java classes which will contain the data that gets from the APIs (Google Flights API, and OpenWeatherAPI) and provide the data to the View. Examples of Java classes: FlightData class, WeatherData class.

5-APIs

Google Flights API: https://serpapi.com/google-flights-api

OpenWeather API: https://openweathermap.org/