Technische Universität München

Teamproject

EIST SoSe 22

42er

Flightsystem

Submitted by:

Carlo Bortolan
Fabian Fritz

Corrected by:

Alexandra Cara Marquardt

Carlo Bortolan
Fabian Fritz

FLIGHTSYSTEM 42

1. Introduction

1.1 Purpose of the system

An app for saving your planned flights, view detailed information and statistics as well as to entertain you during your flight.

1.2 References

See readme.md file for more information.

API's:

- Lufthansa API (https://developer.lufthansa.com/docs/read/Home)
- GoogleMaps API (https://developers.google.com/maps?hl=de)
- GoogleMaps API (https://github.com/googlemaps/google-maps-services-java)
- FlightRadar API (https://de.flightaware.com/live/flight/map/)
- OpenWeatherMap API (https://openweathermap.org/api)

1.3 Sources

- Scene-builder (https://gluonhq.com/products/scene-builder/)
- JavaFX (https://gluonhq.com/products/javafx/)
- Classdiagram:(https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagramtutorial/)
- Include and exclude relationships (https://www.educative.io/answers/what-are-include-and-extend-relationships-in-a-use-case-diagram)
- Spring Boot port configuration (https://stackoverflow.com/questions/21083170/how-to-configure-port-for-a-spring-boot-application)

2. System

2.1 Overview

It is important to keep in mind that all used and displayed data is real as the app gets its data from different API's (for more details view sources). Therefore, you can search for any existing city/airport/flight in the world and will get live flight data and current weather conditions all around the world.

The functionality of the app can be divided into five main functional areas:

- *Search for flights*
    - You can search for real flights by going to the "Search for flights" tab and input a start and destination city. This can be done by either writing the city's name and then clicking on one of the suggested airports or by simply typing the IATA Code of the Airport you want to start from / arrive to. You can also choose to select a certain date (if not, today's date will automatically be used) and filter direct flights by pressing the "Direct" Button. After pressing "Search" if no flights were found you will get a notification, otherwise there will appear all found flights based on your search criteria with the most important information such as start, terminal at start, destination, terminal at destination, departure time, estimated time of arrival duration and the flight number.
    - Each flight consists of at least one flight-object which represents a single direct flight. So, for example when you search from MUC to JFK many flights will first fly to FRA where you will have to change to another flight to JFK. On every flight you can choose to save the flight to your personal flight collection by pressing "add to my flights" and then going to the "My Flights" tab. Of course, if you have saved a delayed flight, you will be notified about it. Once you've added a flight you can also remove it by pressing "Remove from my flights".
- *Flight details*
    - View status: You can see the current status of a flight by pressing "View status". There you will be shown whether the flight is on time or if it is delayed (with the corresponding delay) as well as the original, expected and actual time of departure and arrival with the Gate and Terminal of both the start's and the destination's airport.

      If you like to, you can closely monitor the current position, altitude and velocity of your airplane by pressing "Follow Flight on Map" where a new window with a map and detailed statistics will appear.

  - View details: By pressing on "View details" you can see details of the city you're flying from/to such as the name, country, weather (with corresponding icon), current temperature, current temperature feels like, min and max temperature of the day, wind direction, wind speed and on top of all that also an image of the city.

- *Map / My locations service*
  - You can access a global map by pressing "View details" and then "Open on map" on any flight where you will be shown your current trim. From there you can view the city you're flying from/to and the app will provide you with a detailed description of the city, images and much more. You can also search for attractions/hotels/restaurants at your destination, filter them, read user feedback, book a hotel/restaurant by pressing the respective buttons and save selected locations to your personal collection. You can either access this collection by pressing "Show favorites" or by going to the "My Locations tab".

- *Media and entertainment system*
  - Watch flight instructions: Press on "Instructions" where a new window with a safety-instruction video will open.
  - Watch movies: Press on "Movies" where you will be able to watch any of the 17 preselected movies in HD-Quality by clicking on the cover of the movie you want to watch.
  - Listen to music: Press on "Music" and select the piece you want to listen to by clicking on its cover. You can also read the lyrics (or, if there are none - a detailed analysis of the piece) by clicking on the cover of the music player that appears once you've started a piece.

- *In-flight services*
  - D

2.2 Functional requirements

Show flight information: The system shows flight information (flight number, start time, end time, gate, terminal, seat, airplane type, airline, etc.) of all the user's flights and notifies the user if a flight is canceled or delayed.

Add new flight trip: The user can search through available flights and can select the one's he has booked. The user can build flight journeys (trips) with multiple connecting flights or one return flight and can see the trips on a map.

Display destination information: The user can display important information such as POIs in the destination place (e.g., hotels, restaurants, attractions) and weather data. POIs should be shown in a map and the user should be able to filter them. When clicking on a POI, the user can see additional information and save the POI in the favorites.

Give feedback: A passenger can give feedback after he took the flight about the flight itself, catering, entertainment, service or comfort. Passengers who take the survey will be rewarded, e.g., by miles, souvenirs, coupons, or price drawings.

Request service: During a flight, a passenger can request the service. This notifies the service staff so that they can come to the passenger's seat.

Watch flight safety instructions: The user can watch the flight safety instructions.

2.3 Nonfunctional requirements

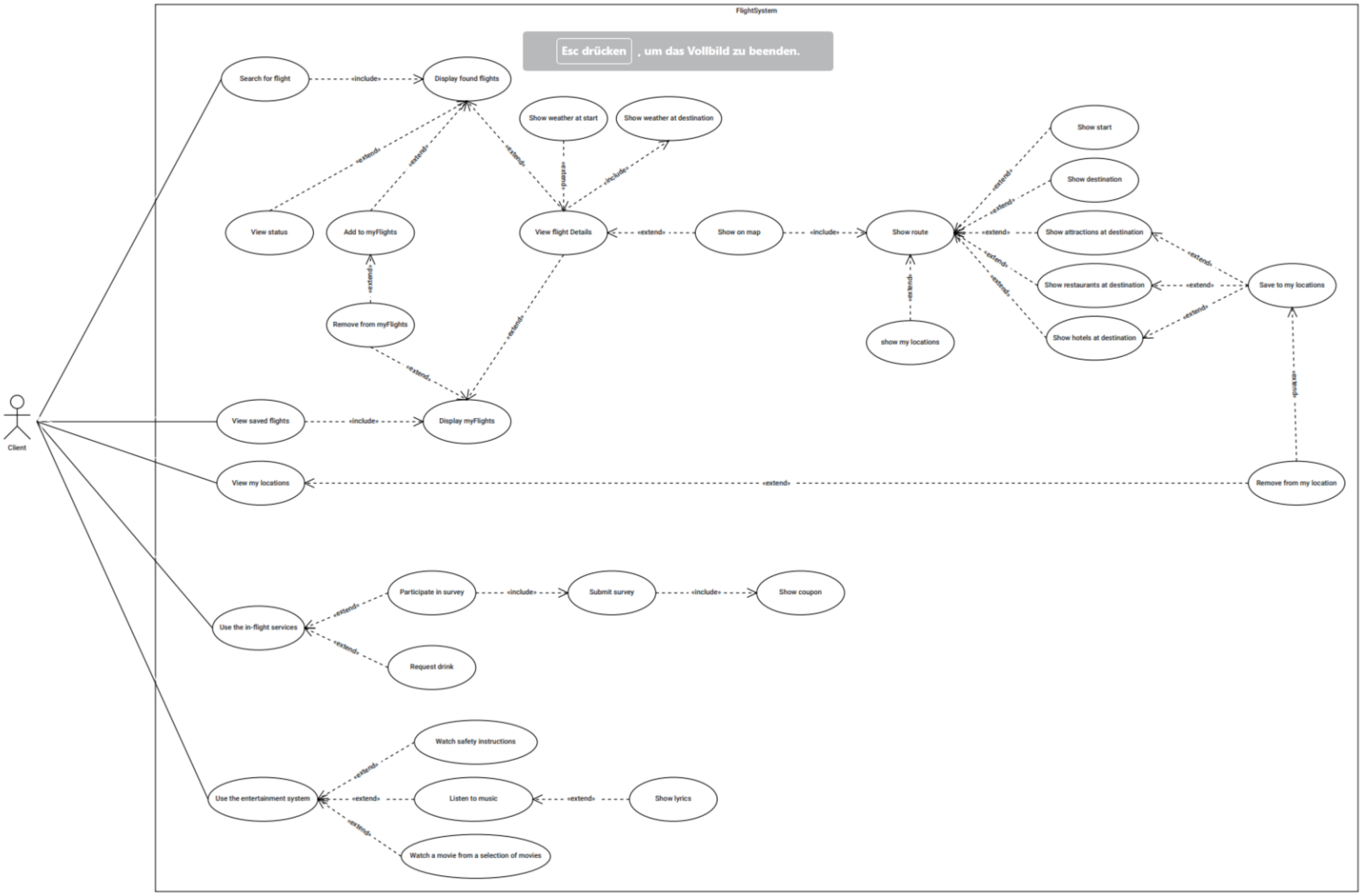
Usability: The system should be intuitive to use, and the user interface should be easy to understand. All interactions should be completed in less than three clicks.

Conformance to guidelines: The design of the system should conform to the typical usability guidelines such as Nielsen's usability heuristics.

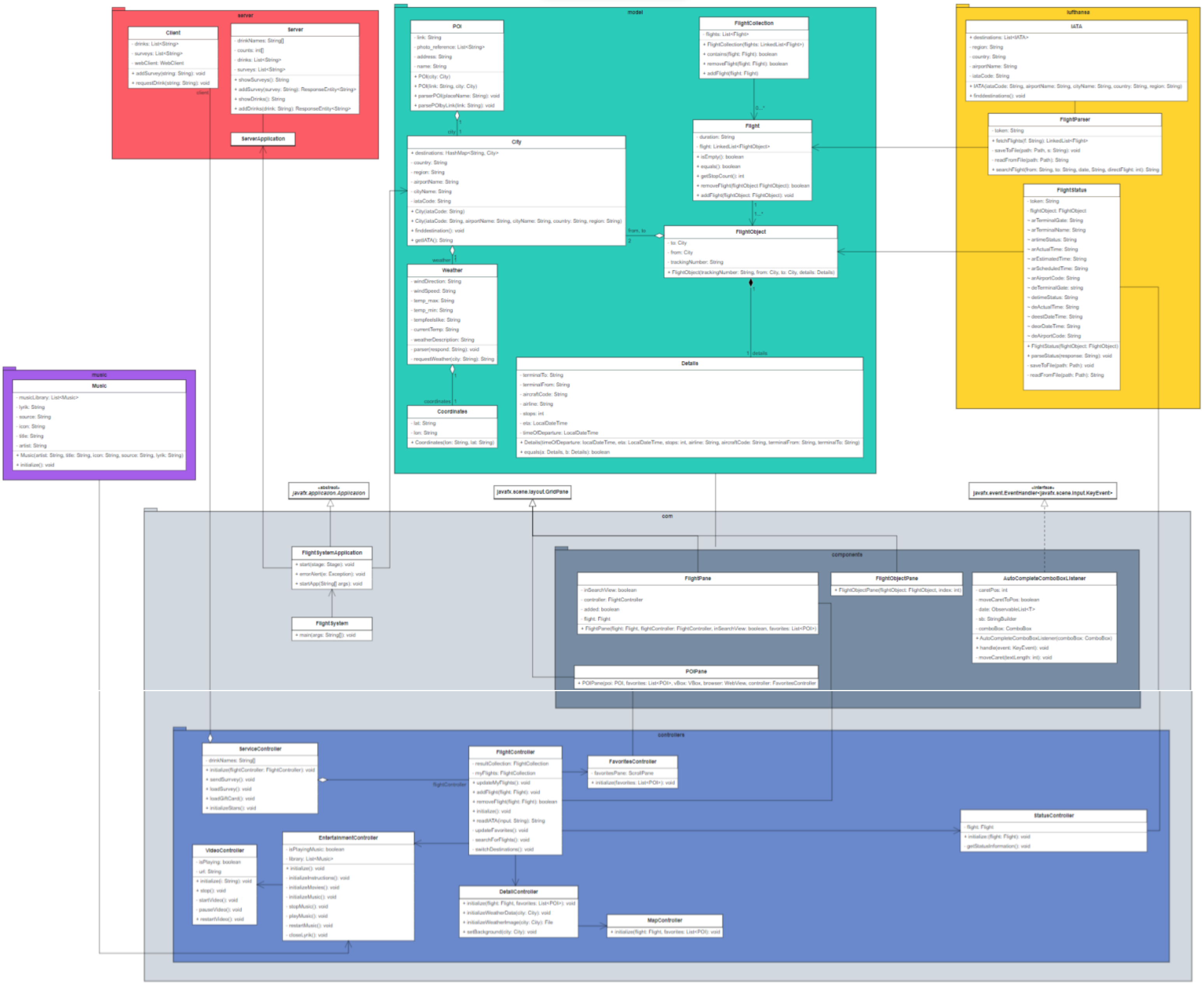
Server system: A server subsystem with a couple of services must be used in the system. However, additional services like destination information for weather and POIs should be obtained from external services.

2.4 System models

2.4.1 Use case model

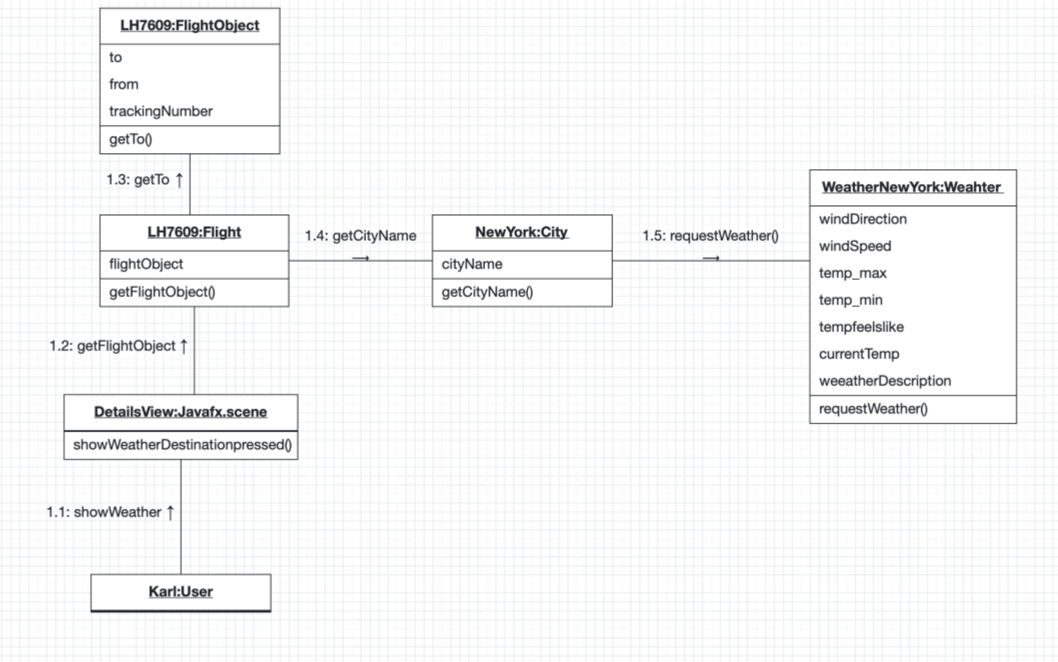


2.4.2 Object model

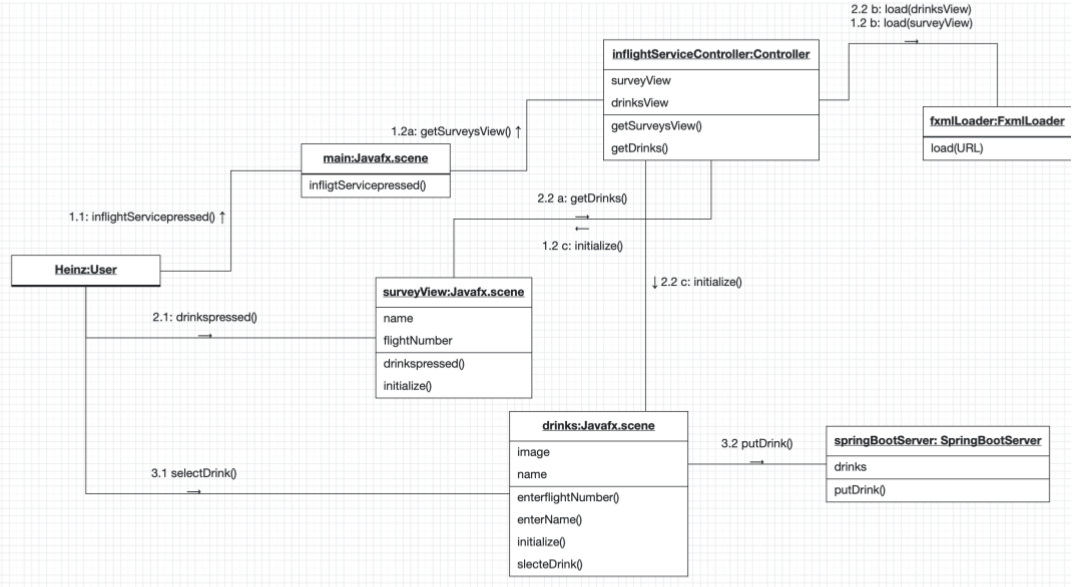

server
Client
- drinks: List<String>
- surveys: List<String>
- webClient: WebClient
+ addSurvey(string: String): void
+ requestDrink(string: String): void
client
Server
- drinkNames: String[]
- counts: int[]
- drinks: List<String>
- surveys: List<String>
+ showSurveys(): String
+ addSurvey(survey: String): ResponseEntity<String>
+ showDrinks(): String
+ addDrinks(drink: String): ResponseEntity<String>
ServerApplication
model
POI
- link: String
- photo_reference: List<String>
- address: String
- name: String
+ POI(city: City)
+ POI(link: String, city: City)
+ parserPOI(placeName: String): void
+ parsePOIbyLink(link: String): void
1
city
1
City
+ destinations: HashMap<String, City>
- country: String
- region: String
- airportName: String
- cityName: String
- iataCode: String
+ City(iataCode: String)
+ City(iataCode: String, airportName: String, cityName: String, country: String, region: String)
+ finddestination(): void
+ getIATA(): String
from, to
2
weather
1
Weather
- windDirection: String
- windSpeed: String
- temp_max: String
- temp_min: String
- tempfeelslike: String
- currentTemp: String
- weatherDescription: String
- parser(respond: String): void
- requestWeather(city: String): String
1
coordinates
1
Coordinates
- lat: String
- lon: String
+ Coordinates(lon: String, lat: String)
FlightCollection
- flights: List<Flight>
+ FlightCollection(flights: LinkedList<Flight>)
+ contains(flight: Flight): boolean
+ removeFlight(flight: Flight): boolean
+ addFlight(flight: Flight)
0...*
Flight
- duration: String
- flight: LinkedList<FlightObject>
+ isEmpty(): boolean
+ equals(): boolean
+ getStopCount(): int
+ removeFlight(flightObject FlightObject): boolean
+ addFlight(flightObject: FlightObject): void
1
1...*
FlightObject
- to: City
- from: City
- trackingNumber: String
+ FlightObject(trackingNumber: String, from: City, to: City, details: Details)
1
1
details
Details
- terminalTo: String
- terminalFrom: String
- aircraftCode: String
- airline: String
- stops: int
- eta: LocalDateTime
- timeOfDeparture: LocalDateTime
+ Details(timeOfDeparture: localDateTime, eta: LocalDateTime, stops: int, airline: String, aircraftCode: String, terminalFrom: String, terminalTo: String)
+ equals(a: Details, b: Details): boolean
lufthansa
IATA
+ destinations: List<IATA>
- region: String
- country: String
- airportName: String
- iataCode: String
+ IATA(iataCode: String, airportName: String, cityName: String, country: String, region: String)
+ finddestinations(): void
FlightParser
- token: String
+ fetchFlights(f: String): LinkedList<Flight>
- saveToFile(path: Path, s: String): void
- readFromFile(path: Path): String
+ searchFlight(from: String, to: String, date, String, directFlight: int): String
FlightStatus
- token: String
- flightObject: FlightObject
~ arTerminalGate: String
~ arTerminalName: String
~ artimeStatus: String
~ arActualTime: String
~ arEstimatedTime: String
~ arScheduledTime: String
~ arAirportCode: String
~ deTerminalGate: string
~ detimeStatus: String
~ deActualTime: String
~ deestDateTime: String
~ deorDateTime: String
~ deAirportCode: String
+ FlightStatus(flightObject: FlightObject)
+ parseStatus(response: String): void
- saveToFile(path: Path): void
- readFromFile(path: Path): String
music
Music
- musicLibrary: List<Music>
- lyrik: String
- source: String
- icon: String
- title: String
- artist: String
+ Music(artist: String, title: String, icon: String, source: String, lyrik: String)
+ initialize(): void
«abstract»
javafx.application.Application
javafx.scene.layout.GridPane
«interface»
javafx.event.EventHandler<javafx.scene.input.KeyEvent>
com
FlightSystemApplication
+ start(stage: Stage): void
+ errorAlert(e: Exception): void
+ startApp(String[] args): void
FlightSystem
+ main(args: String[]): void
components
FlightPane
- inSearchView: boolean
- controller: FlightController
- added: boolean
- flight: Flight
+ FlightPane(flight: Flight, flightController: FlightController, inSearchView: boolean, favorites: List<POI>)
FlightObjectPane
+ FlightObjectPane(flightObject: FlightObject, index: int)
AutoCompleteComboBoxListener
- caretPos: int
- moveCaretToPos: boolean
- date: ObservableList<T>
- sb: StringBuilder
- comboBox: ComboBox
+ AutoCompleteComboBoxListener(comboBox: ComboBox)
+ handle(event: KeyEvent): void
- moveCaret(textLength: int): void
POIPane
+ POIPane(poi: POI, favorites: List<POI>, vBox: VBox, browser: WebView, controller: FavoritesController
controllers
ServiceController
- drinkNames: String[]
+ initialize(flightController: FlightController): void
+ sendSurvey(): void
+ loadSurvey(): void
+ loadGiftCard(): void
+ initializeStars(): void
flightController
FlightController
- resultCollection: FlightCollection
- myFlights: FlightCollection
+ updateMyFlights(): void
+ addFlight(flight: Flight): void
+ removeFlight(flight: Flight): boolean
+ initialize(): void
+ readIATA(input: String): String
- updateFavorites(): void
- searchForFlights(): void
- switchDestinations(): void
FavoritesController
- favoritesPane: ScrollPane
+ initialize(favorites: List<POI>): void
StatusController
- flight: Flight
+ initialize:(flight: Flight): void
- getStatusInformation(): void
VideoController
- isPlaying: boolean
- url: String
+ initialize(i: String): void
+ stop(): void
- startVideo(): void
- pauseVideo(): void
+ restartVideo(): void
EntertainmentController
- isPlayingMusic: boolean
- library: List<Music>
+ initialize(): void
- initializeInstructions(): void
- initializeMovies(): void
- initializeMusic(): void
- stopMusic(): void
- playMusic(): void
- restartMusic(): void
- closeLyrik(): void
DetailController
+ initialize(flight: Flight, favorites: List<POI>): void
+ initializeWeatherData(city: City): void
+ initializeWeatherImage(city: City): File
+ setBackground(city: City): void
MapController
+ initialize(flight: Flight, favorites: List<POI): void

2.4.3 Dynamic model

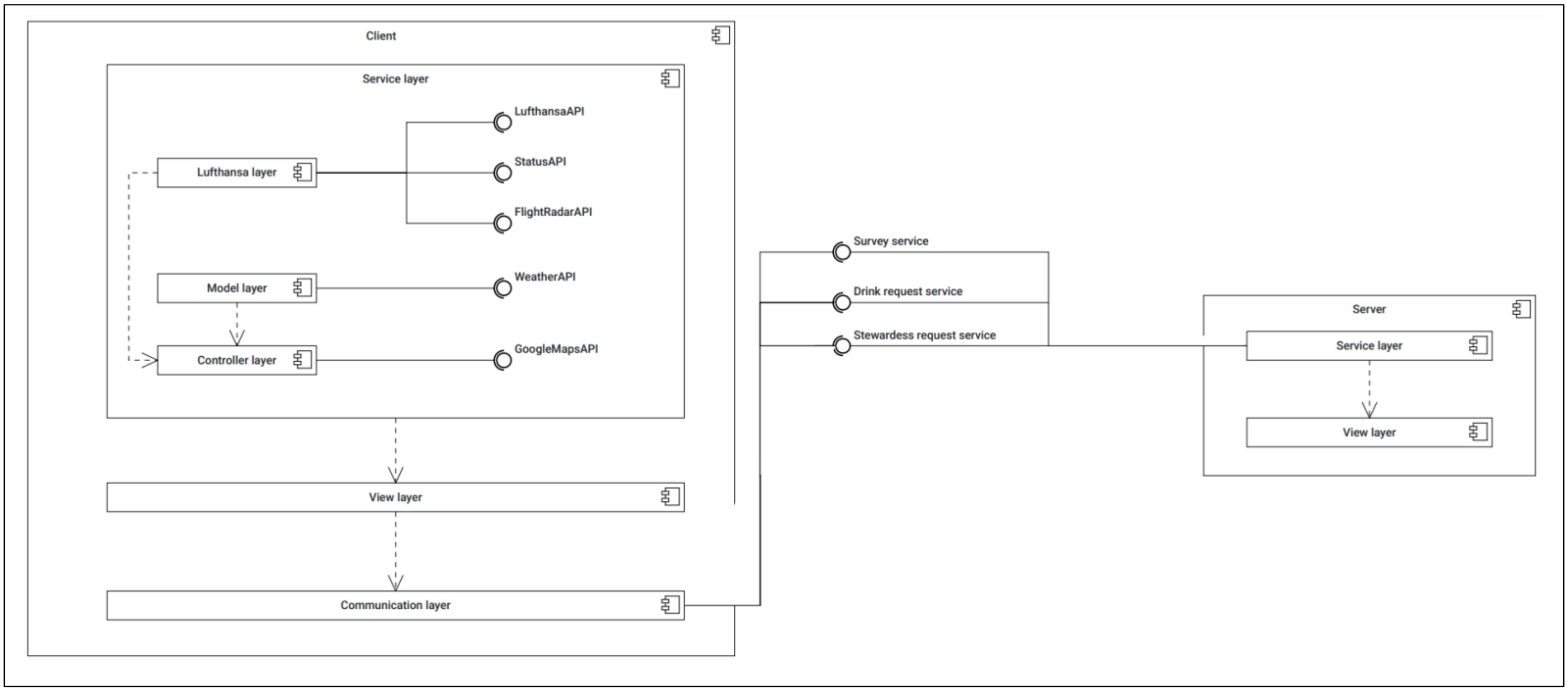
Karl = User, would like to see the weather.
To do this, he clicks on the showWeather() button in the DetailsView. Thereby the method accesses the flight from which one is in the DetailView. To determine what the destination is the FlightObject is needed (1.2), which contains the information about the destination (1.3). With the IATA code of the destination the destination city can be queried (1.4) and thus also the weather, which is contained in the city (1.5).



Heinz = User now wants to order a drink, he is in the main view.
Therefore he now clicks on the inflightService tab (1.1). Now the inflightServiceView opens, for this the method getSurveysView() is called to the controller of inflightService (1.2a), which in turn instructs the fxmLoader to open the view (1.2b). Afterwards the view is initialized (1.2c). Now Heinz clicks on Drinks (=Button | 2.1), which in turn causes the controller to call the fxmlLoader to open the view Drinks - same steps as before only with 2.X. Heinz decides for an apple juice, he clicks on the picture --> this causes the method selectDrink() (3.1) to be executed, which triggers a putRequest in the SpringBoot server and saves the order there under the path Drinks.



2.4.4 Component diagram


Client
Service layer
LufthansaAPI
StatusAPI
FlightRadarAPI
Lufthansa layer
Model layer
WeatherAPI
Controller layer
GoogleMapsAPI
View layer
Communication layer
Survey service
Drink request service
Stewardess request service
Server
Service layer
View layer