Project Report

Fly High – Airline Management System

Created by:

Cristina Ailoaei (266543)

Dragoș Sîrbu (266500)

Michał Jurewicz (266892)

Michał Podgórni (267128)

Supervisors:

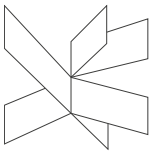
Ib Havn

Jens Cramer Alkjærsig

Mona Wendel Andersen

Steffen Vissing Andersen

VIA University College



Number of characters: 24,978

Information and Communication Technology Engineering

2nd Semester

IT-SEP2Y-A17

Group I

08.06.2018

I hereby declare that my project group and I prepared this project report and that all sources of information have been duly acknowledged

Michal Podgorni Cristina Ailoaei Michal Jurewicz Dragos Sirbu

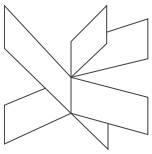
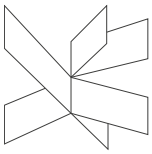
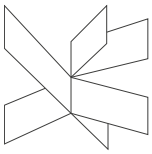
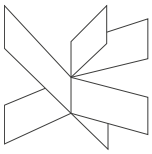


Table of content



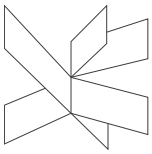


List of figures and tables:



Executive summary

The purpose of the project is to create a java application for *Fly High*, a fictive airline company. The system uses a database connection and a client-server architecture. The application is supposed to store and manage data about flights, airports, airplanes, crew, club members and passengers. This study analyses the process of developing the project step by step. Its main goal is to provide an answer to the project problems and to communicate the ideas and methods used for reaching the final product.



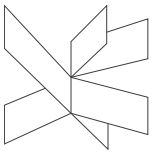
Abstract

The use of flights, both in terms of travelling and airfreight, has greatly increased over the past decades, air travel becoming one of the most important modes of transport nowadays. The air transport is and has demonstrated itself to be a thriving market which doubles its volume of passengers every 15 years. Fly High is a fictive airline company that brought the request to the team to build a system which could help them storing everything in a well-organized way and allow the customers to use their services at once.

The unified process was used in order to complete the project, meaning that more parts of the project were done concurrently with different intensity.

Inception phase was the one where the project group completed most of the business modelling, set the goals and made sure that the team and the airline company have the same expectation from the product to be created. Then, in elaboration phase the group focused more on the analysis and design parts. During the construction phase the majority of the implementation were created. At the end, in the transition phase, the group tested the application, tried to solve the minor issues and made sure that all requirements are met.

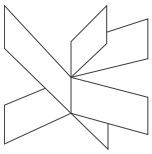
It can be concluded that the final version of the project is functional and meets the set goals to a big extent.



1 Introduction

Air traffic is an essential branch of transport sector nowadays. Although it can be named a relatively new option of transport, this market has already achieved a lot, promising even much more than that. Studies show that until 2036, the traffic flow in Europe will expand 2.6 times than in present. (AIRBUS S.A.S., 2017) The reasons of its enhanced popularity are mainly supported by the title of the fastest form of transport and a good safety record for commercial air transport.

One specific Danish airline, Fly High, founded by Tobias Jensen and William Christensen in 2009 is a company headquartered in Vejle, which initially operated domestic flights. Now, the airline wants to expand its flights across Europe, which brought the request for a new management system. While operating internal flights, the only way for booking a ticket for a specific flight was calling the company and discussing with an employee all the details of the flight, starting with destination and ending with check-in, the ticket being sent afterwards via email. Due to the difficulty in booking tickets it is not surprising that “Fly High” has been left behind and it is not very popular. Therefore, expanding across Europe in the lack of a better management system would be totally ineffective, as the current state of things implies that both the clients and employees have to put a lot of effort into it. Plus, a delimitation of the project would be that the application will be available solely in English. Therefore, it will not have multi-language support. However, if a more convenient way of purchasing the flight will not be created, most of the people would be very much tempted to choose other companies for flying. In the context of today’s society, something that does not put in use the available technological possibilities will not achieve the biggest success. Regarding “Fly High”, once the amount of data started to increase in size (new flights, new planes, change in the crew), operations such as storing and managing company’s information became very difficult to be handled on files. A simple scenario of an admin who wants to change the date of a flight could create serious problems in such a system. And things can become even more dangerous, as the slightest issue can lead to cancellations and delays, risking the future of the company.



2 Literature survey

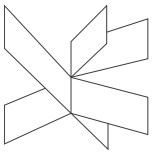
An important part in creating the application and documenting it properly was using the next three sources of knowledge:

- Lectures conducted by teachers and exercises at school
- Books:
  - Craig Larman – “Applying UML and patterns”
  - Thomas Connolly – “Database systems”
- Online research:
  - Java 8 Documentation
  - SQL Part of W3schools website
  - Stack Overflow website
  - MOUNTAIN GOAT SOFTWARE

In the process of discovering the technical and business planning issues, the oral presentations conducted during classes played an important role. The information gained in Software Development with UML and Java 2 (SDJ2) allowed the team to implement most of the application while Database System (DBS1) course provided the knowledge for creating the database, which is a necessary part of the project. The Software Engineering (SWE1) course helped making sure that the diagrams are correctly created and documented. For organizing the work properly and using an adequate way of developing the project, the Semester Project (SEP) course played a crucial part.

Regarding the books used, both of them have a wide range of content, therefore the project group has not used everything that could be learnt from them, but mostly the parts connected to the project. They were especially helpful for including the design patterns in the java application and for the most complicated parts of the database.

Online research has been used for assistance regarding minor issues and clearing misunderstandings (Java Documentation, Stack Overflow or SQL Part of W3schools), but also for staying informed regarding software practices (MOUNTAIN GOAT SOFTWARE), allowing us to be updated and complete the gaps in the process of creating the project.



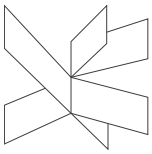
3 Methods

Because the requirements of the project were quite specific from the beginning, the project team could know from the early beginning what the exact tasks will be and how the work should be divided. The system had to be implemented in Java and use the client-server architecture. The group members started working with IntelliJ IDEA, because creating the Graphical User Interface (GUI) was more efficient this way. Also, the team decided to use the JavaFX package, for its simple and efficient working style.

As for the work planning, the group used the Unified Process, which assumes that more steps in developing the project can be done concurrently. However, it still divides all the process into four main phases which are next listed:

1. Inception
2. Elaboration
3. Construction
4. Transition

Requirements (3.1.1) were first to be started, in a wish of making sure that the proper goals are set. The main functions of the system are displayed in *Use Case diagram (3.1.2)* together with the *Use Case descriptions (3.1.3).* A graphical way of showing the behavior of the system was covered by the *Activity diagrams (3.1.4)*. The exact classes together with their methods that explain the operational part of the program were illustrated on *Analysis class diagram (3.1.5)* while the *Design class diagram (3.2.1)* gives an overview on the whole application. Connection between client and server side is presented on the *TCP connection diagram (3.2.2).* The *Sequence diagrams (3.2.3)* explain how objects operate one with another for the most complex methods. The graphical representation of all the system functionalities is shown on the *GUI Design part (3.2.4).* All possible scenarios of the application's behavior are discussed and shown in the *Test cases (3.4.1).*

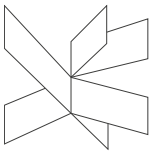


3.1 Analysis

3.1.1 Requirements

3.1.1.1 Functional requirements

1. An administrator should be able to add airports to the system. While adding a new airport, the administrator has to specify the code, name, city, postcode, country, number of gates.
2. An administrator should be able to add airplanes to the system. While adding a new plane, the administrator has to specify the number, model, number of seats.
3. An administrator should be able to add crew members to the system. While adding a new crew member, the administrator has to specify the name, position, address, birthdate, id, phone number, e-mail.
4. An administrator should be able to add flights to the system. While adding a new flight, the administrator has to specify the number, departure time, arrival time, departure place, arrival place, plane, crew, price.
5. A customer booking a flight should specify all of the following: name, birthdate, nationality, type of ID, ID number, expiration date.
6. A head administrator should be able to delete data from the system.
7. A head administrator should be able to cancel flights.
8. An administrator should be able to change data for club members, crew, flights, airplanes and airports.
9. A customer should be able to choose a seat number, luggage size, payment method in order to book a ticket.
10. A customer should be able to select departure and destination airport and the departure and return date (or departure only) for flights in order to get the available flights.
11. An administrator should be able to get a list of all flights, airports, airplanes, crew and club members.
12. An administrator should be able to set the annual fee for club members.



13. A customer should receive the ticket via email.
14. A customer should be able to become a club member in order to get discounts.
15. A club member should be able to search only for cheap flights from his/her city.
16. A customer should be able to subscribe to the newsletter in order to receive new information regarding flights and offers via email.
17. An administrator should be able to log in the system in order to manage data.
18. A head administrator should be able to see the profiles of all administrators.
19. A head administrator should be able to create or delete an administrator account in order to ease the management of accounts.

3.1.1.2 Non-functional requirements

20. The system has to use the client-server architecture.
21. The system has to store persistent data using a database.
22. The system has to have a GUI.
23. The system has to be implemented in Java.
24. The system and the system development process have to be documented.

3.1.2 Use Case diagram

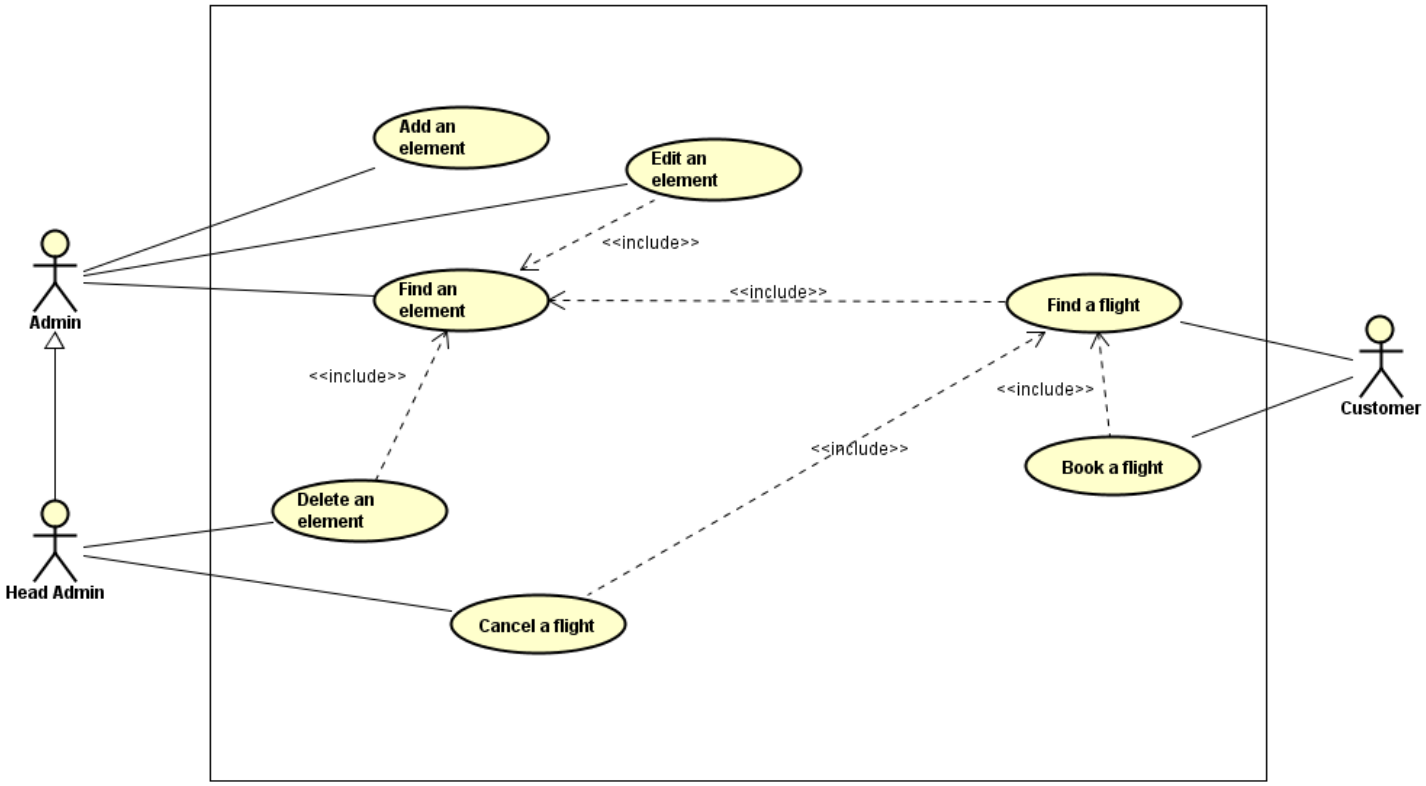
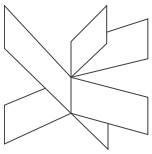


Figure 1 - Use case diagram

The use case shown above (Figure 1) presents all the functional features that users of Fly High application may perform. The use cases are the following:

- **Add an element** – The administrator or the head administrator can add an airplane, airport, crew member or flight to the system.
- **Find an element** – The administrator or the head administrator can search for an existing airplane, airport, crew member or flight in the system.
- **Edit an element** – The administrator or the head administrator can edit the data of an airplane, airport, crew member or flight in the system.
- **Delete an element** – The head administrator can delete an airplane, airport or crew member from the system.
- **Cancel a flight** – The head administrator can cancel a flight.
- **Find a flight** – The customer can search for an existing flight in the system.
- **Book a flight** – The customer can book an existing flight.



3.1.3 Use case description

UseCase	Book a flight
Summary	A customer books a flight
Actor	Customer
Precondition	None.
Postcondition	The flight is booked and the changes are stores in the database.
Base Sequence	1. The person goes through find a flight use case. 2. The person enters all the required personal data: name, birthdate, nationality, type of ID, ID number, expiration date, seat number, size of luggage, method of payment. 3. The person confirms the decision to book the given flight. 4. If one or more of the entered data is not valid then go to step 2 else the decision is confirmed and the given flight becomes booked, changes are stored in the database, the person is redirected to another page in order to make the payment and the use case ends.
Branch Sequence	
Exception Sequence	The entered data could not be valid: 4 as base sequence The system informs that the entered data is not valid
Sub UseCase	Find a flight
Note	

Figure 2 - Book a flight use case description

3.1.4 Activity diagram

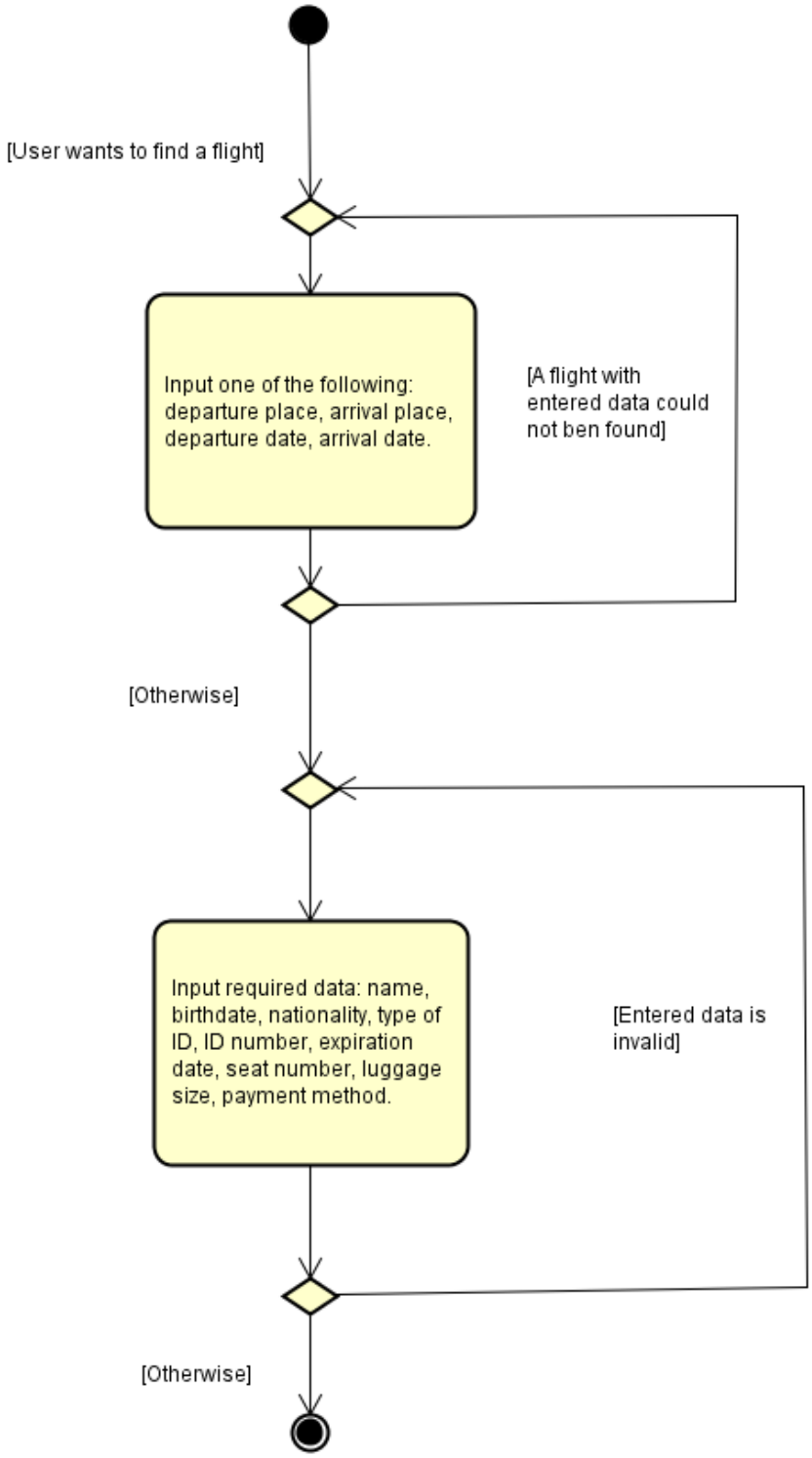


Figure 3 - Booking a flight activity diagram

One of the crucial functionalities of the system, booking a flight, works as it follows:

- The customer initiates an option to find a flight.
- All necessary data (departure place, arrival place, departure time, arrival time) is entered.
    - If the system cannot find a flight with given details, it displays an appropriate message and asks the customer to fill the new data.
    - If the system can find a flight with given details, it is displayed for the customer.
- The customer is asked to fill all his personal details (name, birthdate, nationality, type of id, id number, id expiration date) and flight details (seat number, luggage size, payment method).
    - If any of the entered data is not correct, the system displays an appropriate message and asks to fill the new data.
    - If all the entered data is correct, the system makes a reservation which is stored in the system and the customer is informed about it. The flow ends.

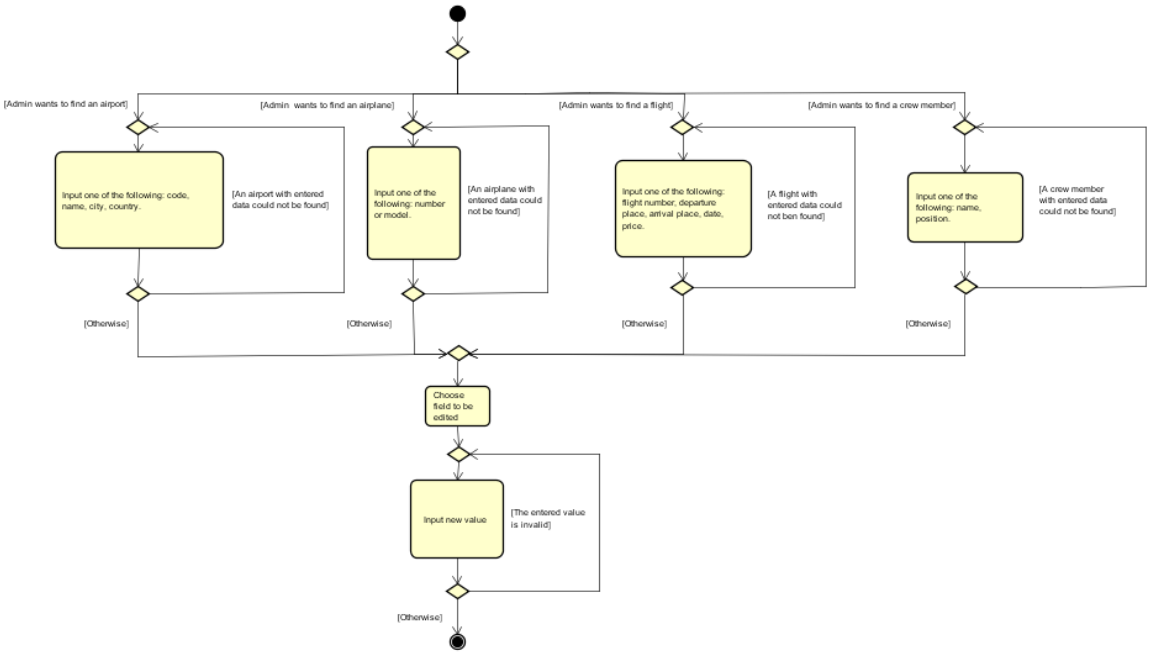
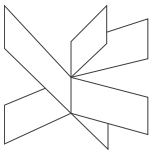


Figure 4 – Edit an element activity diagram



Another crucial part of the system is represented by editing different elements by administrators:

- The administrator initiates an option to find an element.
- If the element is an airport, one of the following is entered: code, name, city, country. If the element is an airplane, one of the following is entered: number of model. If the element is a flight, one of the following is entered: flight number, departure place, arrival place or date. If the element is a crew member, one of the following is entered: name or position.
    - If the system cannot find an element with given details, it displays an appropriate message and asks the administrator to fill the new data.
    - If the system can find an element with given details, it is displayed for the administrator.
- The administrator chooses the field to be edited.
- The administrator inputs a new value in the specific field.
    - If the entered data is not correct, the system displays an appropriate message and asks to fill the new data.
    - If the entered data is correct, the system makes saves the changes in the system. The flow ends.

3.1.5 Analysis class diagram

The analysis class diagram represents the part of the program that handles all the operations between the most essential classes and provides an easy way to understand basically all the functionalities. Due to the fact that the project group has used MVC pattern in order to implement the system, in this case the analysis class diagram represents the model part of the system.

All the content in the model is managed by the FlyHighModelManager. The program can access all the lists in order to add, search for, edit or delete an element. The only list that is not accessible directly from the Model Manager is PassengerList class, which is created each time for specific flight.

The most important class that connects all the others is Flight. It stores instances of airport, airplane, crew and passenger list.

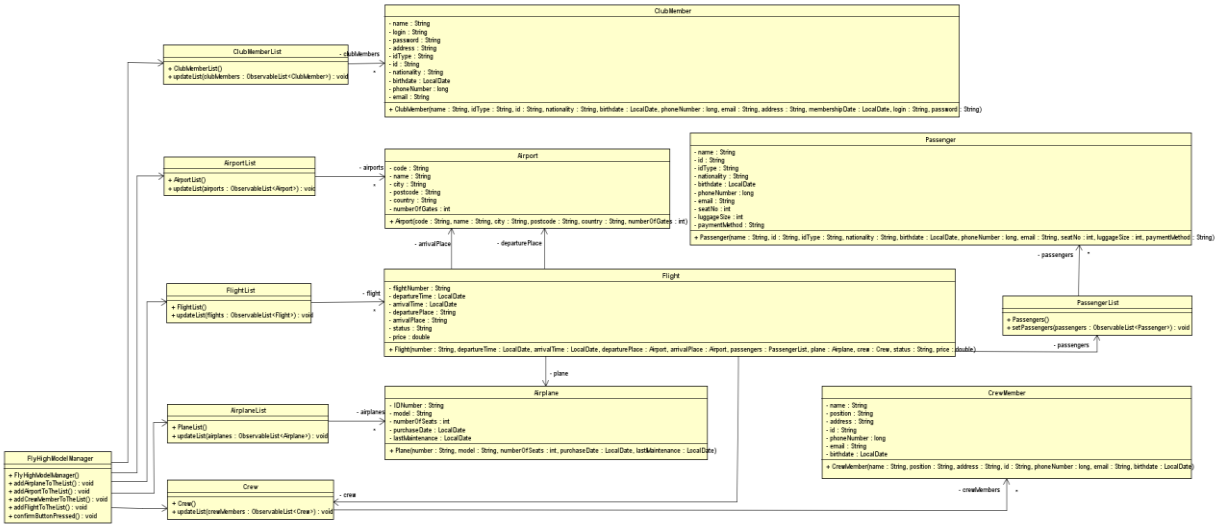


Figure 4 - Analysis class diagram

3.2 Design

3.2.1 Design class diagram

For a full overview of the system, the best way is to analyze the design class diagram, which shows all the packages, classes and the most important methods used. Throughout the development process, the project group was still learning new things and trying different approaches to creating the system. Therefore, the diagram were changed many times.

The whole system was supposed to be divided into two main packages: client and server.

The content of the server is designed as follows: All the basic classes are located in the model part, where there can be found most of the operations on the stored data. Then, the mediator provides a connection between the application and the database. The purpose of the controller is to direct the methods to proper places where they should be executed, while the view part is responsible for GUI.

On the client side there is no actual model part nor database because all the commands are being passed to the server, where they are executed in the proper places. The controller and the view parts are basically the same as on the server side.

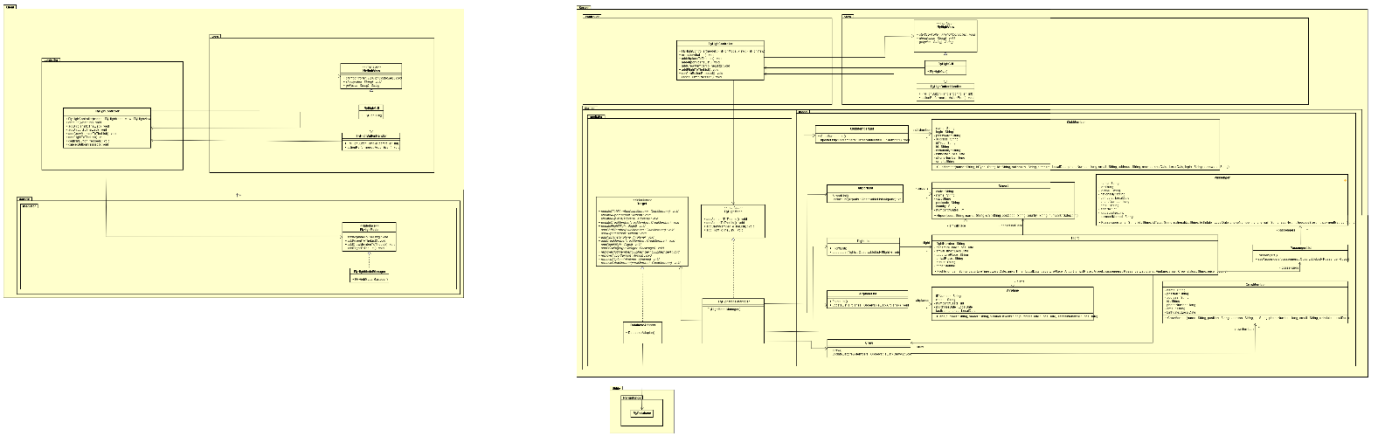


Figure 5 - Design class diagram

3.2.2 TCP connection diagram

In order to implement client-server architecture, the project group has used socket connection following the Transmission Control Protocol (TCP).

The procedure of client connecting to the server can be easily described. First, a proper request is being sent. Then, the server replies and requires some of sender's details. When the information is received, the client is registered.

If there is sent any other request than the required one, the connection is closed.

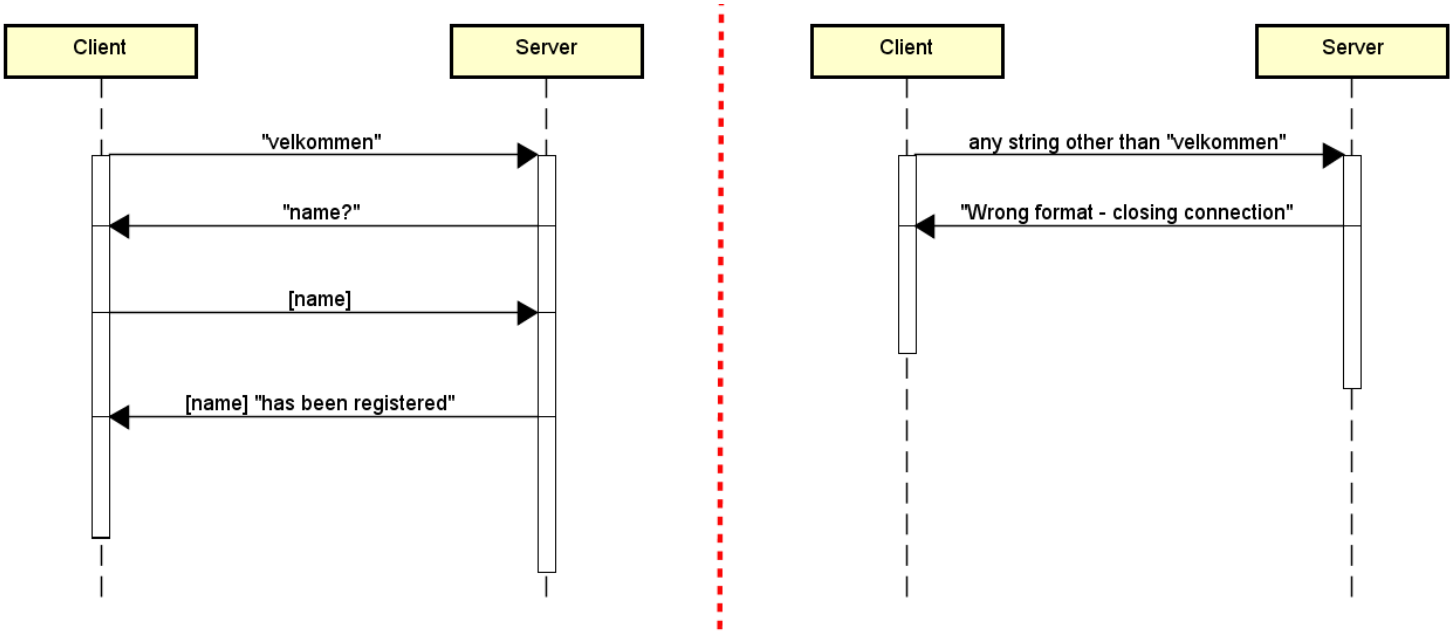


Figure 6 - TCP connection diagram

3.2.3 Sequence diagram

Regarding sequence diagrams, the project team decided to show how the method filterAirportsFrom() from class E_Flight in package Controller.Edit has been designed. First, the administrator enters *manage flights* section and decides to edit a flight. The diagram explains what is the process of editing the departure country.

When the country is changed, the system first makes sure that the field has not been left empty and clears the field containing the name of the city corresponding to the airport already set. Then, a loop is used in order to check if the city of each airport is located in the new entered country. If the condition is true, then the city is being added to the list. If any condition is not fulfilled, meaning that the country field is empty or the city field is also empty, then null is returned.

In the end, the system displays a drop-down list of cities to choose.

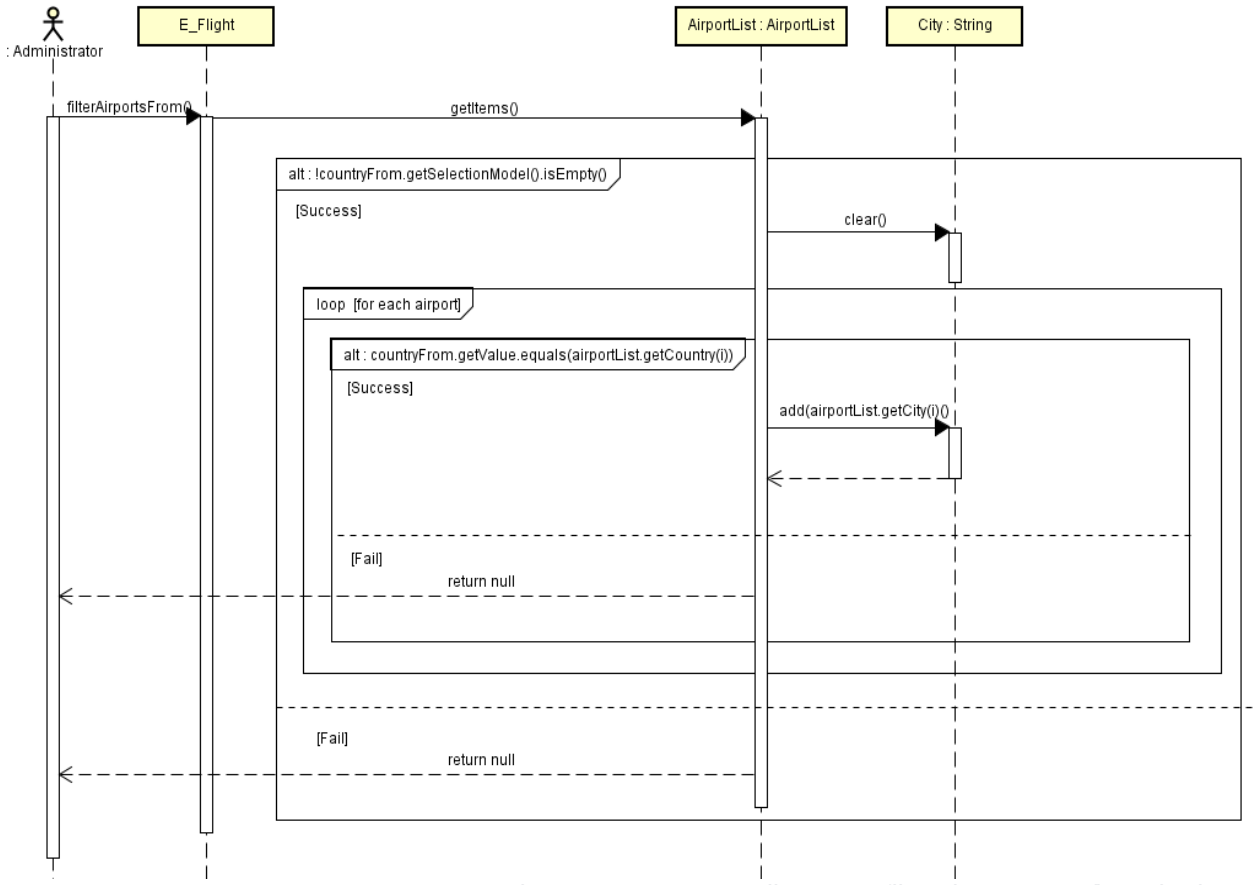


Figure 7 – Sequence diagram – filterAirportsFrom() method

3.2.4 GUI design

The GUI has been designed in a way to be operated easily and efficient and was strictly created for the use of administrators of Fly High and customers. Therefore, a log in is mandatory in order to keep the system safe. Customers have the option to register and create an account as club members or just skip and proceed further to finding and booking.

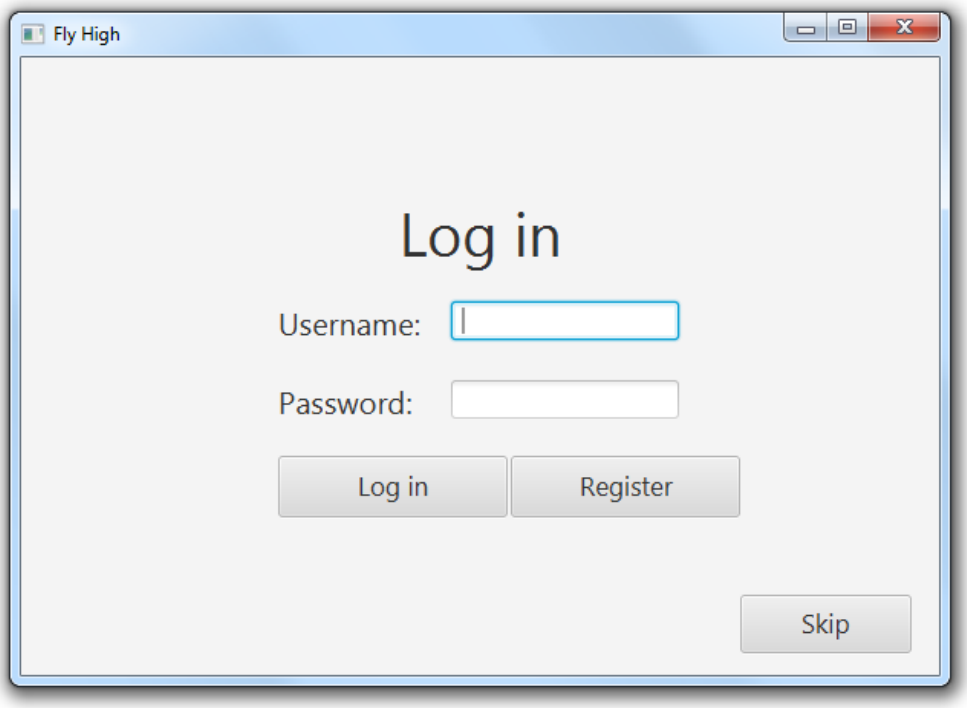


Figure 8 - Log in screen

After logging in, depending on the user's role, a view corresponding to an administrator, head administrator, club member or simple customer is loaded. The main difference between head administrator and administrator perspective is that the former has access to cancelling flights and deleting different kind of data from the system.

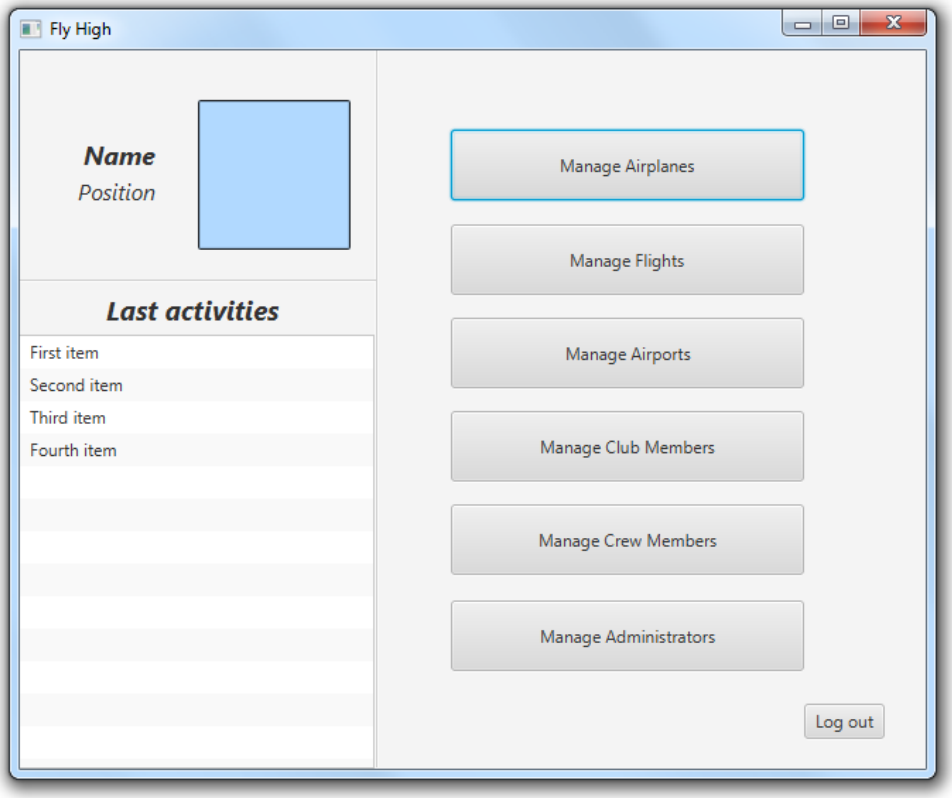


Figure 9 - Administrator view

As it can be observed in *Figure 8,* an administrator's view mostly includes managing options.

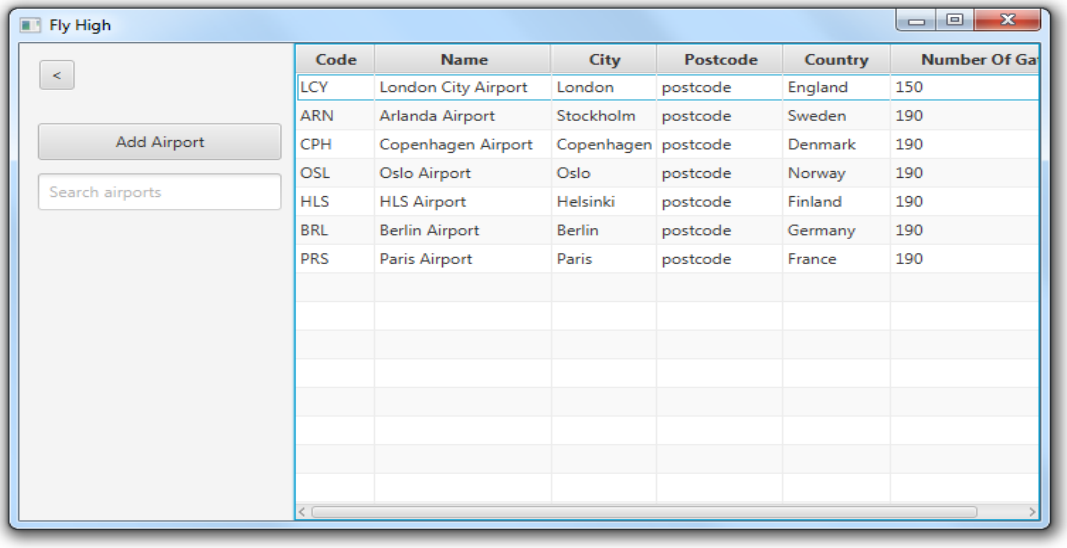


Figure 10 - Manage airports view

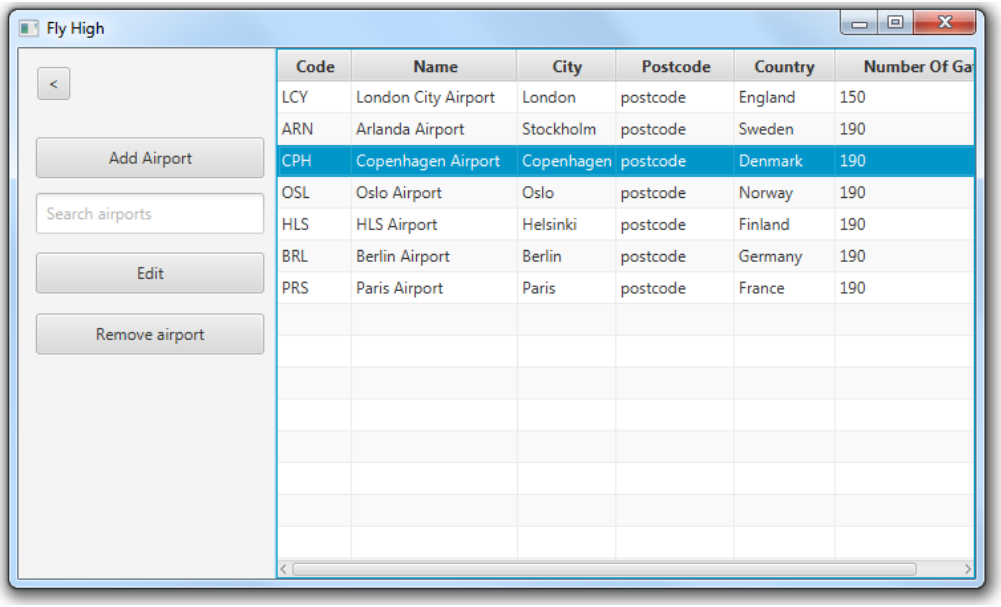


Figure 11 - Manage airports view (after focusing an airport)

Figure 9 and *figure 10* show a design preference of the team, as "Edit" and "Remove airport" buttons are only visible after focusing a specific airport.

The next figure (*Figure 11*) presents the fields to be filled by a customer who wants to become a club member.

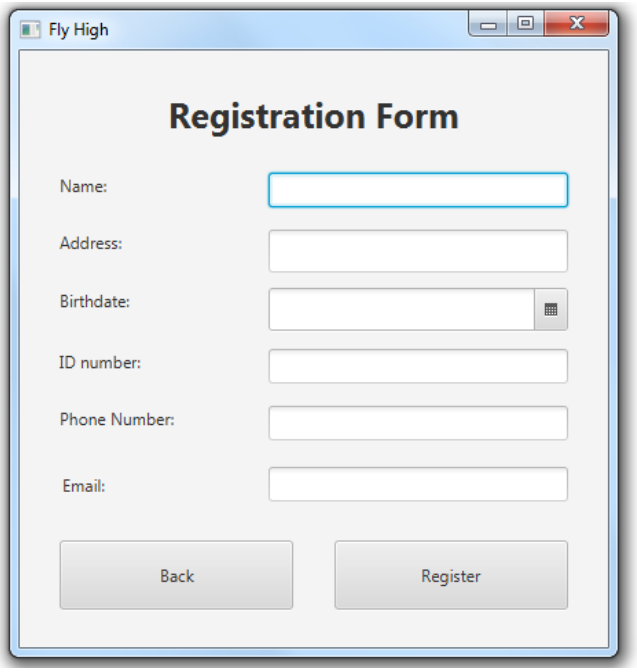
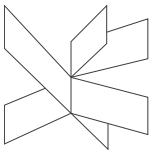


Figure 12 - Registration form view



In order to book a flight, the customer, whether it is a registered club member or not, has to specify the departure and arrival places and dates. When the person chooses a specific flight from the system, there is a possibility to book it. The difference between registered and unregistered customers is that the club members benefit from discounts on flights and do not have to enter the personal data every time booking a flight, as it is already stored in the system.

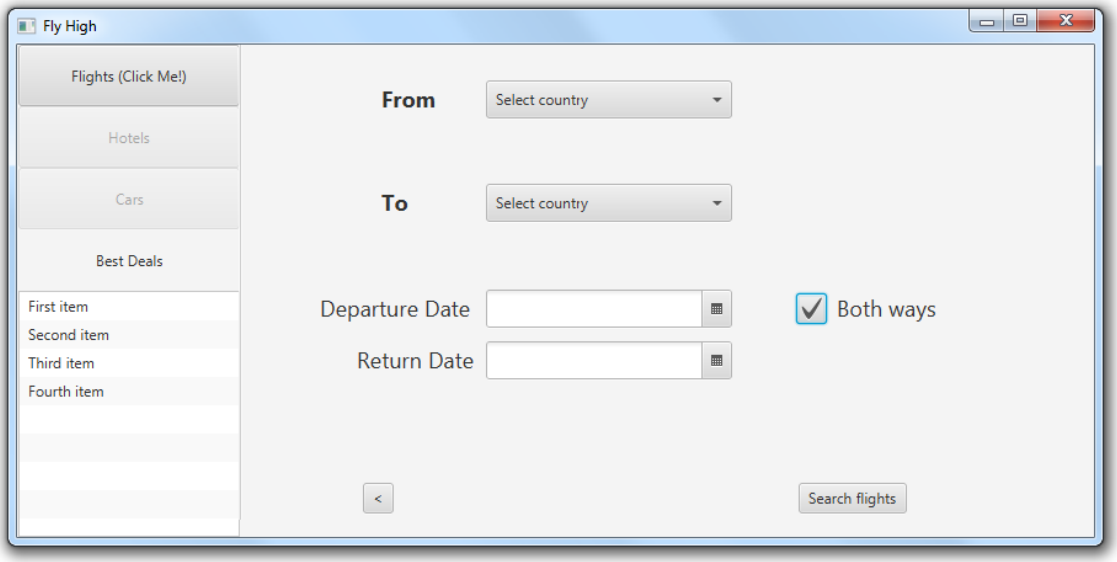
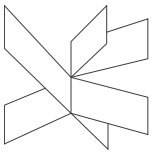


Figure 13 - Book a flight view



3.3 Implementation

There were many ways the system could be implemented. To fully understand an application, it is essential to know how the code works. In this section there are a few of methods presented and described.

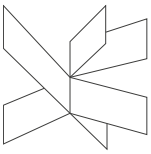
The method addCrewMemberButtonPressed() in the class M_CrewMembers is responsible for creating a button and assigning it to adding a new crew member into the system.

```
public void addCrewMemberButtonPressed() throws IOException {
    Stage window = new Stage();
    window.initModality(Modality.APPLICATION_MODAL);
    window.setTitle("Add New Crew Member Form");
    FXMLLoader loader = new FXMLLoader((getClass().getResource(
            name: "../../View/FXML/Administrator/Add/A_CrewMember.fxml")));
    window.setScene(new Scene(loader.load()));
    A_CrewMember controller = loader.getController();
    controller.setItems(crew.getCrewMembers());
    window.showAndWait();
}
```

Figure 14 - addCrewMemberButtonPressed() method

Firstly, a new stage is being created in a separate window. The next line limits the access for a user to operate any part of the application until the form is filled correctly and saved or cancelled. Then, the title of the form is set and the path to the form is specified using a new fxml loader. The next command loads the form onto the window. After the controller is defined and initialized using the FXMLLoader method, a list of already existing crew members is passed to it. Last line calls the method to show the window.

The purpose of the method updateFlight(Flight flight) in the class DatabaseAdapter is to update the database with the new entered flight data, as the name suggests.

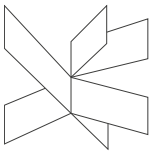


```
public void updateFlight(Flight flight) {
    connect();
    try{
        connection = dataSource.getConnection();
        connection.setAutoCommit(false);

        int tempId = Integer.parseInt(flight.getFlightNumber());
        statement=connection.createStatement();
        String sql= "UPDATE fly_high_database.flyhigh.flightlist  " +
                "SET departuredate ='"+flight.getDepartureDate()+"' WHERE flightNumber='"+tempId+"';"+
                "UPDATE fly_high_database.flyhigh.flightlist " +
                " SET departuretime='"+flight.getDepartureTime()+"' WHERE flightNumber='"+tempId+"';"+
                "UPDATE fly_high_database.flyhigh.flightlist" +
                " SET arrivaldate='"+flight.getArrivalDate()+"' WHERE flightNumber='"+tempId+"';"+
                "UPDATE fly_high_database.flyhigh.flightlist" +
                " SET arrivaltime ='"+flight.getArrivalTime()+"' WHERE flightNumber="+tempId+";"+
                "UPDATE fly_high_database.flyhigh.flightlist " +
                "SET departureplace ='"+flight.getDeparturePlace().toString()+"' WHERE flightNumber'"+tempId+"';"+
                "UPDATE fly_high_database.flyhigh.flightlist " +
                " SET arrivalplace='"+flight.getArrivalPlace().toString()+"' WHERE flightNumber='"+tempId+"';"+
                "UPDATE fly_high_database.flyhigh.flightlist " +
                " SET status='"+flight.getStatus()+"' WHERE flightNumber='"+tempId+"';"+
                "UPDATE fly_high_database.flyhigh.flightlist " +
                " SET airplaneidnumber='"+flight.getAirplaneIdNumber()+"' WHERE flightNumber='"+tempId+"';";

        statement.executeUpdate(sql);
        connection.commit();
        System.out.println("Updated successfully.");
    }catch (Exception e){
        e.getStackTrace();
        System.err.println(e.getClass().getName()+": "+e.getMessage());
        System.exit( status: 0);
    }
}
```

Figure 15 - updateFlight() method

The execution of the method starts with calling a private method used for creating a connection to the database. The try catch block establishes the connection and assures that any statement will be executed unless another one is completed. In the next line, a temporary variable of type int is created in order to keep the entered flight's number. Then, there is a command creating a SQL statement which checks if the flight data is the same as in the given one and the new data is set. Later, the statement created is being executed and the database is being updated. In the end of the try part, there is a message informing that the method worked properly. The purpose of the catch part is to ensure that no exception will disrupt the execution of this method.

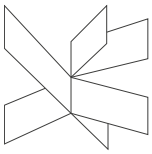
In order to get flights in given time ranged between specified airports the method getFlights() from BookFlightController class is used.



```
private ObservableList<Flight>  getFlights() {

    ObservableList<Flight> flights = FXCollections.observableArrayList();
    for (int i = 0; i <flightList.getFlights().size() ; i++) {
        if ((flightList.getFlights().get(i).getDepartureDate().equals(departureDate))
                && (flightList.getFlights().get(i).getDeparturePlace().equals(departurePlace))
                &&(flightList.getFlights().get(i).getArrivalPlace().equals(arrivalPlace))) {
            flights.add(flightList.getFlights().get(i));
        }
    }
    return flights;
}
```

Figure 16 - getFlights() method

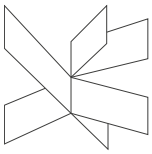
The controller uses the data that is initialized in a different method that calls the getFlights(). In the beginning, the observable list of flights is created. Then, in the loop, the system checks if the departure date, departure place and arrival place values match the ones specified before. If the conditions are met, the flight is added to the list. In the end, the observable list of flights is returned.



3.4 Testing

3.4.1 Test cases

In order to have documented and make sure that all the functionality of the system is safe regarding all possible events combinations which could create malfunctions in the system by being executed in a different way than they were meant, the project team developed a list of test cases. The test cases are based on the use cases requirements of the system and the main purpose for them is to verify all possible scenarios. Owing to the test cases, the system should not encounter any situations of conflict it would not know how to handle.

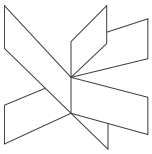


Scenario 1 – Successful booking a flight	Basic Flow	
Scenario 2 – Invalid entered customer data	Basic Flow	Alternate Flow 1
Scenario 3 – Doubled flight	Basic Flow	Alternate Flow 2
Scenario 4 – Incorrect birthday	Basic Flow	Alternate Flow 3
Scenario 5 – Expired document	Basic Flow	Alternate Flow 4
Scenario 6 – No tickets available	Basic Flow	Alternate Flow 5

Basic Flow	1. Finding the element – Customer goes through the finding a flight in the system UseCase. 2. Initiate Booking – Customer chooses an option to book a flight. 3. Filling the Data – Customer fills in all the required data: name, birthdate, nationality, type of ID, ID number, expiration date, seat number, luggage size, payment method. 4. Booking a Flight – Customer books a flight; System addes a new passenger to the given flight. The UseCase ends.
Alternate Flow 1 – Invalid data for booking a flight	In Basic Flow Step 3 –Filling the Data, if any of the entered data is not in a valid format or some data is not entered at all, an appropriate message appears and the system rejoins the Basic Flow Step 3 – Filling the Data.
Alternate Flow 2 – Doubled flight	In Basic Flow Step 3 – Filling the Data, if there already exists a booking for the same flight made by person with the same name and ID number, appropriate message appears and the system rejoins the Basic Flow Step 3 – Filling the Data.
Alternate Flow 3 – Incorrect birthday	In Basic Flow Step 3 – Filling the Data, if the entered birthday is after the date exactly 16 years before today's date, an appropriate message appears and the system rejoins the Basic Flow Step 3 – Filling the Data.
Alternate Flow 4 – Expired Document	In Basic Flow Step 3 – Filling the Data, if the expiration date is before the today's date, an appropriate message appears.
Alternate Flow 5 – No tickets available	In Basic Flow Step 4 – Booking a Flight, if another person have just booked the last ticket for given flight, an appropriate message appears.

TC ID#	Scenario / Condition	Valid format	Valid values	Unique element	Element found	Expected Result
CW1.	Scenario 1 – Successful booking a flight	YES	YES	YES	YES	Successful booking a flight
CW2.	Scenario 2 – Invalid entered customer data	NO	n/a	n/a	n/a	Warning message, return to Basic Flow Step 3 – Filling the Data
CW3.	Scenario 3 – Doubled flight	YES	YES	NO	n/a	Warning message, return to Basic Flow Step 3 – Filling the Data
CW4.	Scenario 4 - Incorrect birthday	YES	NO	n/a	n/a	Warning message, return to Basic Flow Step 3 - Filling the Data
CW5.	Scenario 5 - Expired document	YES	NO	n/a	n/a	Warning message
CW6.	Scenario 6 - No tickets available	YES	YES	YES	NO	Warning message

Figure 17 - Book a flight test case



4 Results/findings and Discussion

4.1 Results

The majority of the requirements were fulfilled and therefore the overall goal of the project has been achieved.

An overview of the accomplished results is that the application provides a log in, registration or skip option.

When choosing to log in, the user is redirected to one of the followings:

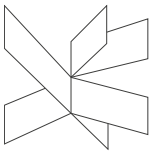
- Administrator profile: option to manage flights, airports, airplanes, crew or club members. In all cases manage means adding, finding or editing an element. There is an exception for club members, as there is no possibility to add new ones.
- Head administrator profile: option to manage flights, airports, airplanes, crew or club members and to cancel a flight. In all cases manage means to adding, finding, editing or removing an element. There are two exceptions: there is no possibility to add a new club member or remove a flight.
- Club member profile: option to search for a flight from and to a given airport in a period of time and possibility to book it. Also, there is an option to edit the account info.

When choosing to become a club member (registration option), the user is redirected to fill out a form containing all the necessary details.

When choosing the skip option, the user is redirected to a view where they can search for a flight from and to a given airport in a period of time and possibility to book it.

Fulfilled requirements:

- An administrator can add airports, airplanes, crew and flights to the system.
- A head administrator can delete airports, airplanes, crew from the system.
- A head administrator can cancel flights.
- A customer can book a flight.



- An administrator can edit airports, airplanes, crew and flights data in the system.
- An administrator can search for airports, airplanes, crew and flights in the system.
- A customer can become a club member.
- A customer can subscribe to the newsletters.
- The system has a log in option.
- The system stores persistent data using a database.
- The system has a GUI.
- The system is implemented in Java.
- The system and the system development process are documented.

Unfulfilled requirements:

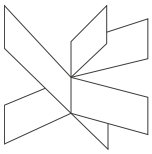
- The system uses the client-server architecture.
- An administrator cannot set the annual fee for club members.
- A customer cannot receive the ticket via email.
- A club member cannot search only for cheap flights from his/her city.
- A head administrator cannot create delete or get administrator accounts.

4.2 Discussion

One of the strongest features of the application is that the system is very simple to understand owing to the design of the GUI, which facilitates any action that the user wants to perform. For example, some of the options such as deleting an item from a list is hidden until the object is focused, in order to maintain the simplicity.

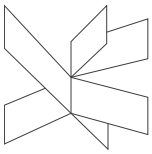
Also, when searching for an element, the system is saving time by refreshing automatically, retrieving the results after every entered character.

However, as listed in the results part, the project team has not implemented all the functionality which was initially planned. The biggest missing part is the client-server



architecture. The main reason why it is missing is because of bad planning resulted from wrong approximation of the time the task would take.

In the end, the application that the group developed is still functional and covers the most important requirements.

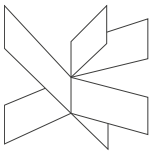


5 Conclusions

The project group focus was to create a user-friendly application with simple GUI design that has client-server architecture and connection to a database. The most important thing was to make sure that the final product will fulfill Fly High's needs: storing and managing all the necessary data as well as providing the customers with a way of booking flight tickets.

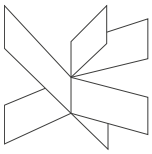
Taking into account the complexity of the system and the multitude of details to be considered, a plan was required to forecast the many different scenarios in advance to achieve the goal. The analysis part was the one that was changed most of the times, but thanks to SCRUM and UP, adjustments in the planning did not affect the product to a considerable extent. In the end, the implementation follows the all the documentation and diagrams, so anyone who wants to understand any part of the system, can use them in order to do so. All the files are categorized, so it is very easy to understand the logical structure of the project.

To encapsulate, the group achieved almost all the main goals excepting the client-server architecture due to some planning issues , but carried out a development process that resulted in a functional system.



6 Sources of information

1. AIRBUS S.A.S. (2017) *Growing Horizons*. Available at: http://www.airbus.com/content/dam/corporate-topics/publications/backgrounders/Airbus_Global_Market_Forecast_2017-2036_Growing_Horizons_full_book.pdf (Accessed: 5 June 2018).
2. *Java Platform Standard Edition 8 Documentation* (no date). Available at: https://docs.oracle.com/javase/8/docs/ (Accessed: 5 June 2018).
3. *SQL Tutorial* (no date). Available at: https://www.w3schools.com/sql/ (Accessed: 5 June 2018).
4. Developing General Java Applications - NetBeans IDE Tutorial (no date). Available at: https://netbeans.org/kb/docs/java/javase-intro.html (Accessed: 5 June 2018)



Appendices