**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**

****

**[Number of characters]**

**Information and Communication Technology** **Engineering**

**2nd Semester**

**IT-SEP2Y-A17**

**Group I**

**07.07.2018**

****

**Table of content**

[List of figures and tables v](#_Toc515751217)

[Executive summary vi](#_Toc515751218)

[Abstract vii](#_Toc515751219)

[1 Introduction 1](#_Toc515751220)

[2 Theory/literature survey 2](#_Toc515751221)

[3 Methods 3](#_Toc515751222)

[3.1 Analysis 4](#_Toc515751223)

[3.1.1 Requirements 4](#_Toc515751224)

[3.1.2 Use Case diagram 5](#_Toc515751225)

[3.1.3 Use case description 7](#_Toc515751226)

[3.1.4 Activity diagram 8](#_Toc515751227)

[3.1.5 Analysis class diagram 10](#_Toc515751228)

[3.2 Design 10](#_Toc515751229)

[3.2.1 Design class diagram 10](#_Toc515751230)

[3.2.2 TCP connection diagram 10](#_Toc515751231)

[3.2.3 Sequence diagram 10](#_Toc515751232)

[3.2.4 GUI design 10](#_Toc515751233)

[3.3 Implementation 10](#_Toc515751234)

[3.4 Testing 10](#_Toc515751235)

[3.4.1 Test cases 10](#_Toc515751236)

[3.4.2 JUnit testing 12](#_Toc515751237)

[4 Results/findings and Discussion 12](#_Toc515751238)

[4.1 Results 12](#_Toc515751239)

[4.2 Discussion 13](#_Toc515751240)

[5 Conclusions 14](#_Toc515751241)

[6 Sources of information 15](#_Toc515751242)

Appendices

# List of figures and tables

Figure 1 – Use Case Diagram x

Figure 2 – Book a flight Use Case Description x

Figure 3 – Book a flight Activity Diagram x

Figure 4 – Analysis Class Diagram x

Figure 5 – Design Class Diagram x

Figure 6 – TCP Connection Diagram x

Figure 7 – Sequence Diagram x

Figure 8 – GUI Log In Page x

Figure 9 – GUI Administrator Profile x

Figure 10 – GUI Head Administrator Profile x

Figure 11 – GUI Club Member Profile x

Figure 12 – GUI Registration Form x

Figure 13 – GUI Non-User Page x

Figure 14 – GUI Booking a Flight x

Figure 15 – Test Cases x

# Executive summary

The purpose of the project was to create a java application with a database connection that uses client-server architecture for Fly High Airline. The application was supposed to manage and store data about flights, airports, airplanes, crew members and club members. This study analyses the process of developing the project step by step. Its goal is to create a better way of understanding how java programs work and to cache and maintain information in a more efficient and modern way.

# 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. Dating back to 1903 with the first flight of two American engineering brothers, 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 an 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, but with different intensity.

At first, in the inception phase, was the one where the project group made most of the business modelling, set the goals and assured that the customer will accept the result of the work. Then, in elaboration phase the group focused more on the analysis and design parts. During the construction phase there was created the majority of the implementation. At the end, in the transition phase the group assured that everything meets the requirements, tested the application and tried to solve the minor issues.

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

# Introduction

Air traffic is a big and essential branch of transport sector nowadays. Although air travel 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. If there is not a more convenient way of purchasing the flight, most of the people would be very much tempted to fly with other companies. In the context of today’s society, something that does not put in use the available technological possibilities, it will not achieve the biggest possible 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 complicated to be handled on files. A simple scenario of a client 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.

# Theory/literature survey

For creating the project, the group had to develop a proper java application together with all documentation. In order to do so, it used 3 sources of knowledge, which are following:

* Lectures and exercises at school
* Books:
  + Craig Larman – “Applying UML and patterns”
  + Thomas Connolly – “Database systems”
* Webpages:
  + Java 8 Documentation
  + SQL Part of W3schools website
  + Stack Overflow website

Owing to the oral presentations in class, the project group acknowledged both the technical and business planning issues. Knowledge from Software Development with UML and Java 2 course allowed to implement the most of application, Database System course showed how to create the database, which is a necessary part of the project, while Software Engineering course helped with the documenting and assuring that everything is logical and straight. For organizing the work and using adequate way of developing the project, Semester Project course was the crucial one.

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

The internet sites were the place, where the project group could check how something works and find help with understanding some parts. Every time there was a minor issue that was not discussed in any of the books, the group used Java Documentation, Stack Overflow or SQL Part of W3schools websites. The online research allowed to complete all the gaps in the process of creating the project.

# Methods

Because of the requirements of the project were quite specific from the beginning, the project team could know very early 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. the team decided to use JavaFX package, owing to its simple and efficient work style.

For organizing the work, the group used the Unified Process which assumes that more than one step of developing the project can be done at the same time. However, it divides all the process into four main phases which are following:

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

After creating the *Requirements (3.1.1)* and making sure they are setting the correct goals, the group started creating the *Use Case diagram (3.1.2)* together with the *Use Case descriptions (3.1.3)*, which are the explanation of each use case. 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 with the other ones in 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).*

## Analysis

### Requirements

#### 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. An 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. An administrator should be able to select date/time range for flights in order to get flights in a specified range.
10. An administrator should be able to select cities for flights in order to get flights from/to the specified cities.
11. A customer should be able to choose a seat number, luggage size, payment method in order to book a ticket.
12. 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.
13. An administrator should be able to get a list of all flights and club members.
14. An administrator should be able to set the annual fee for club members.
15. A customer should receive the ticket via email.
16. A customer should be able to become a club member in order to get discounts.
17. A club member should be able to search only for cheap flights from his/her city.
18. A customer should be able to subscribe to the newsletter in order to receive new information regarding flights and offers via email.
19. An administrator should be able to log in the system in order to manage data.
20. A head administrator should be able to see the profiles of all administrators.
21. A head administrator should be able to create or delete an administrator account in order to ease the management of accounts.

#### Non-functional requirements

1. The system has to use the client-server architecture.
2. The system has to store persistent data using a database.
3. The system has to have a GUI.
4. The system has to provide a log in.
5. The system has to be implemented in Java.
6. The system and the system development process have to be documented.

### Use Case diagram

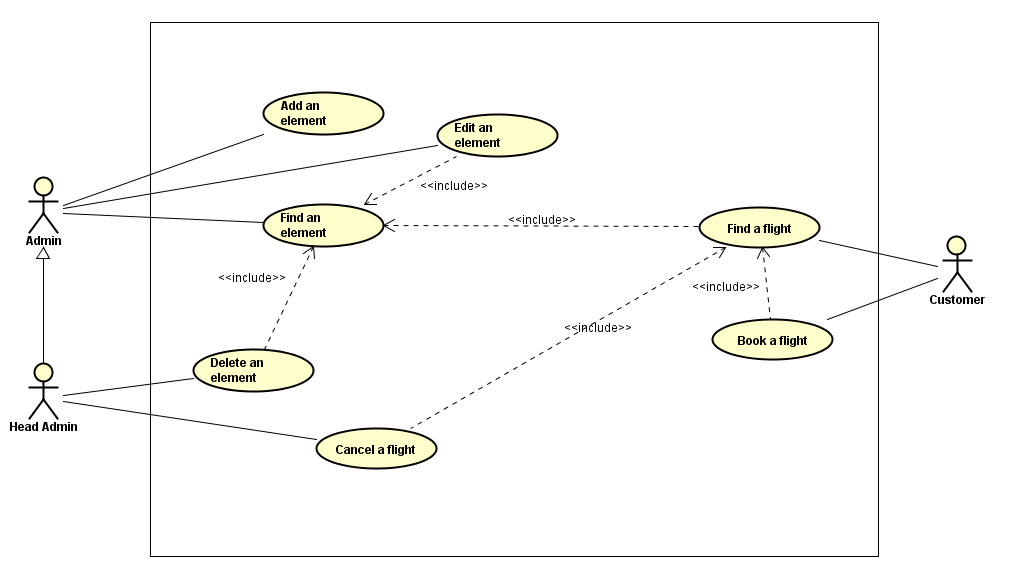


Figure 1 - Use case diagram

The use case shown above (Figure 1) presents all functional feature that every user of  
the FlyHigh application can perform. The use cases are following:

* **Add an element** – The administrator or the head administrator can add an airplane, airport, crew member or flight into 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.
* **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.

### Use case description

|  |  |
| --- | --- |
| UseCase | Book a flight |
| Summary | A customer books a flight |
| Actor | Customer |
| Precondition | None. |
| Postcondition | The flight becomes booked, 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, person is redirected to another site in order to make a 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

### Activity diagram

Figure 3 - Booking a flight activity diagram

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

* First the customer initiates an option to search for a flight.
* Then all the 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.
* In the last step 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.

### 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.

## Design

### Design class diagram

### 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 is very easy to describe, first a proper request is being sent. Then the server replies and asks about the sender’s details. When it receives back the info, the client is registered.

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

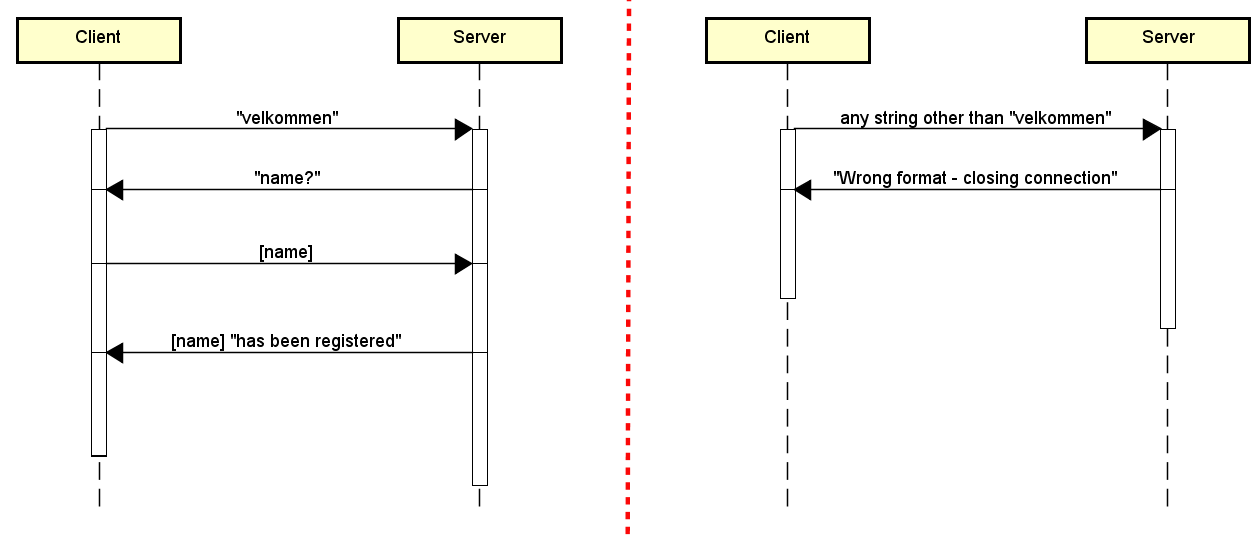


Figure 4 - TCP connection diagram

### 

### Sequence diagram

For sequence diagram the project team decided to show a method filterAirportsFrom() from class E\_Flight in package Controller.Edit. First the administrator enters manage a flight data section and decides to edit a flight data. The diagram explains what happens when departure country is being changed.

First, when the country is being changed, the system assures that the field is not empty and clears the city of the airport that has been set there. In the for loop the system checks every airport if its city is located in just entered new 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, city field is also empty (null is being returned).

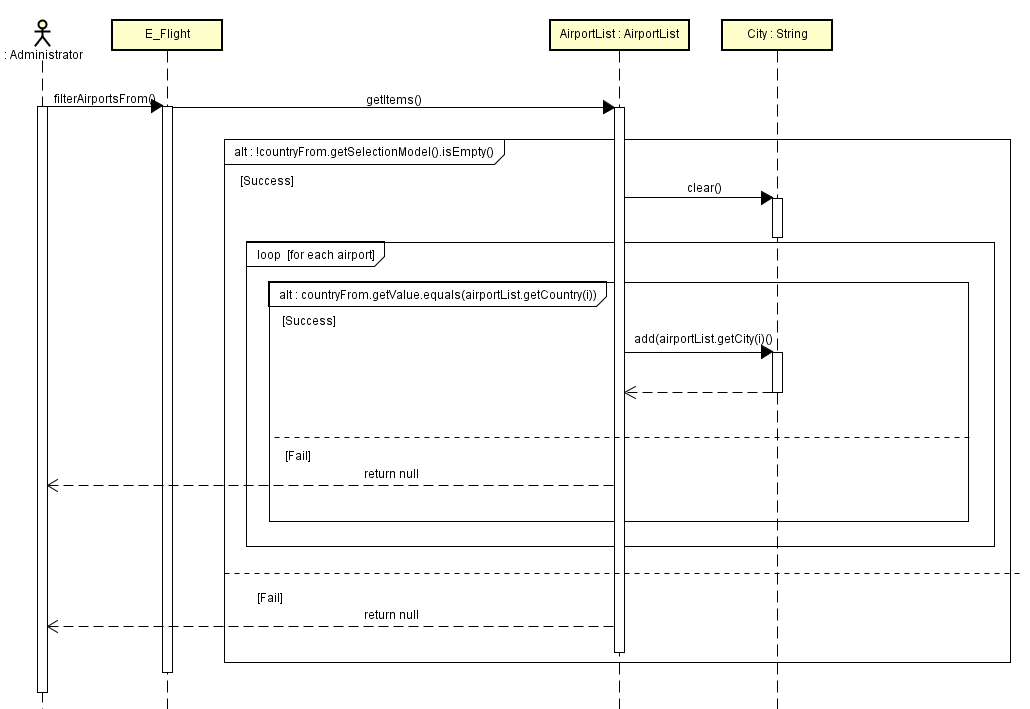
In the end it results in a drop down list of possible cities to choose when the country is already entered.

Figure 5 - Sequence diagram - filter airports from method

### GUI design

## Implementation

## Testing

### Test cases

In order to get a complete list of all functionalities of the system together with all possible combinations of events that can go in a different way than they were meant, the project team developed a list of test cases that show it. They are based on the requirements and use cases of the system and their main purpose is to make sure that any possible scenario will not be forgotten. Owing to it the system should not have any situations it would not know how to handle.



Figure 6 - Book a flight test case

### JUnit testing

# Results/findings and Discussion

## Results

Majority of the requirements are fulfilled and the tasks they represent are achieved. When launching the application, at first there appears the log in screen where the user has three possible options: log in, become a club member (registration) or skip.

When choosing a log in option, the user is redirected to one of the following:

* Administrator profile: with options to manage flights, airports, airplanes, crew members or club members. In all cases manage means to add, search for or edit an element, with an exception for club members, where it is not possible to add new ones.
* Head administrator profile: with options to manage flights, airports, airplanes, crew members or club members and to cancel a flight. In all cases manage means to add, search for, edit or remove an element, with two exceptions: it is not possible to add a new club member or remove a flight.
* Club member profile: with options to search for a flight from and to given airport in given period of time and possibly book it and with an option to edit the account info.

When choosing a become a club member (registration) option, the user is redirected to the form where they can fill the form in order to become a club member and gain discount for flights.

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

**Fulfilled requirements:**

* x
* x
* x

**Unfulfilled requirements:**

* x
* x
* x

## Discussion

Design of the application is very simple, therefore it should be clear and easy for the user to find any functionality they want. Every feature is accessible basically in only one step (after logging in) and everything that is required to go through it is listed. If it comes to searching for an element, the system refreshes automatically, giving results after every entered character, which saves the time.

However, the project group have not implemented every functionality that was planned at the beginning. The biggest missing part is <xxxxxxxxxx>. Major reason why it is missing is because <xxxxxxxxxxx>. <xxxxxxxxxx> and <xxxxxxxxxxx> do not work fully proper and <xxxxxxx>.

In the end, the application that the group developed, works without any problems and covers the most important majority of the requirements.

# Conclusions

The project group main 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 it will fit the Fly High company’s needs: storing and managing all the necessary data as well as let the customers purchase the flights.

Because of the complexity of the system, it required to plan and forecast many different scenarios in advance in order to achieve the goal. That is why the analysis part was the one which was changed many times, after it was clear that some parts cannot be done like the group had thought before. In the end, all the documents and diagrams fit the implementation almost 1:1 so any person that wants to understand any part of the system, can use them in order to do so. All the files are categorized and organized in a logical way so it should be very easy to find the desired one.

To encapsulate, the group achieved their goal and carried out a development process that resulted in a functional system.

# 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

Appendix A – Use Case Diagram

Appendix B – Activity Diagrams

Appendix C – Analysis Class Diagram

Appendix D – Design Class Diagram

Appendix E – Sequence Diagram

Appendix F – TCP Connection Diagram

Appendix G – Test Cases

Appendix H – Java Files