Project Report

SOFT20091: Software Design & Implementation

****

**Project Title:** Bluebrook Airport

**Members:**

Samuel Crane – N0626631

Paul Havelin – N0623631

Adil Rajal Hussain – N0626369

Joe Zalewski – N0623403

Contents

[1. Overview 4](#_Toc481527302)

[1.1. Functional Requirements 4](#_Toc481527303)

[2. Requirements In-depth 6](#_Toc481527304)

[2.1. UML Use Case Diagram 6](#_Toc481527305)

[2.2. UML Activity/Sequence Diagrams 7](#_Toc481527306)

[2.3. UML Class Diagrams 8](#_Toc481527307)

[3. Approach 9](#_Toc481527308)

[3.1. Chosen Methodology 9](#_Toc481527309)

[3.2. Description of Agile (Scrum) 10](#_Toc481527310)

[3.2.1. Planning/Analysis Phase 10](#_Toc481527311)

[3.2.2. Sprint Planning/Meeting: 10](#_Toc481527312)

[3.2.3. Sprint 10](#_Toc481527313)

[3.2.4. System Release 10](#_Toc481527314)

[3.3. MosCow Anlaysis 11](#_Toc481527315)

[3.4. Object Orientated Programming 12](#_Toc481527316)

[3.5. SDLC 13](#_Toc481527317)

[3.6. Data Structure 14](#_Toc481527318)

[3.6.1. Advantages: 14](#_Toc481527319)

[3.6.2. Disadvantages 14](#_Toc481527320)

[3.6.3. Decision 14](#_Toc481527321)

[3.7. Algorithms 15](#_Toc481527322)

[3.7.1. Searching Algorithm 15](#_Toc481527323)

[3.7.2. Sorting Algorithm 16](#_Toc481527324)

[4. Implementation 17](#_Toc481527325)

[4.1. Software Testing Process 17](#_Toc481527326)

[4.1.1. Test plan cases 17](#_Toc481527327)

[4.1.2. Alpha testing 17](#_Toc481527328)

[4.2. Patterns 18](#_Toc481527329)

[4.2.1. Separating Methods 18](#_Toc481527330)

[4.2.2. Variable Names 18](#_Toc481527331)

[4.2.3. Function Names/Declaration 18](#_Toc481527332)

[5. Results 20](#_Toc481527333)

[5.1. Reflection on Performance 20](#_Toc481527334)

[5.1.1. Reliability 20](#_Toc481527335)

[5.1.2. Security 20](#_Toc481527336)

[5.1.3. Portability 20](#_Toc481527337)

[5.1.4. Maintainability 20](#_Toc481527338)

[5.1.5. Scalability 20](#_Toc481527339)

[6. Conclusion 22](#_Toc481527340)

[6.1. Professional, Social, Ethical and Legal Aspects 22](#_Toc481527341)

[6.2. Reflection of Methods 23](#_Toc481527342)

[6.3. Reflection of Final Version 24](#_Toc481527343)

[6.4. Project Evaluation 25](#_Toc481527344)

[6.4.1. Positives 25](#_Toc481527345)

[6.4.2. Negatives 25](#_Toc481527346)

[6.4.3. Future Additions 25](#_Toc481527347)

[7. Appendices 1](#_Toc481527348)

[7.A. Use Cases 1](#_Toc481527349)

[7.B. Flowchart 6](#_Toc481527350)

[7.C. Menu Areas 7](#_Toc481527351)

[7.D. Gantt Chart 8](#_Toc481527352)

[7.E. Test Cases 8](#_Toc481527353)

[8. References 9](#_Toc481527354)

# 1. Overview

For this software implementation project, we decided to look at a system used by a small domestic airport that deals with arrivals and departures, seat allocations, emergency flight plans and other various features. The aim of the app will be to streamline and provide accurate, important information for flight administrators to ensure that emergencies are dealt with efficiently and safely and the day-to-day running of the airport runs smoothly.

## 1.1. Functional Requirements

* Manage all flights
* Store airplane related data including passenger’s data, plane capacity, which seats are booked.
* Monitor departure and arrivals. These two types of information should be displayed in different monitors across the screens available in the airport.
* Provide search options: Allow users to search for flight or passenger info, depending of their level of access.
* Display seat allocations (if GUI display booked seats in red – user can’t click on them and display non-booked seats in green?)
* Handle security: check passengers identify, if ok let them pass through the airport
* Passengers/Security and admin have different level of access to the system

See Table 1.

|  |  |  |  |
| --- | --- | --- | --- |
| *FR* | *Requirement* | *Implications* | *Type of Requirement* |
| FR1 | Manage Flight | Data Structure required for handling:  -Schedule  Live Departure/Arrival  -Flight information (number, no of seats, airline etc.) | Functional |
| FR2 | Manage airport spaces & staff | Data structures required for handling:  -airport runways  -airport bays  ground force  -assign staff to bay for landing airplane | Functional |
| FR3 | Manage passenger info | Edit passenger personal details (name, address, booking history etc.) | Functional |
| FR4 | Manage Staff info | Edit staff personal information (name, address etc.)  Assign staff to terminal | Functional |
| FR5 | Assign physical spaces to flights | Correlate requirements 1 and 2. Generate error messages if there is an attempt to assign a runway/airport bay being used in a time slot.  Prioritise landings and departures. | Functional |
| FR6 | Login system | Create account for new user or staff member  Log in to system using stored credentials | Functional |
| FR7 | Book/Cancel/Edit flight | Staff and passenger can create, confirm, edit, and cancel booking. They can change seat and class. Passengers can also make payment | Functional |

Table 1

# 2. Requirements In-depth

## 2.1. UML Use Case Diagram

By designing a use case diagram the system can be visualised and to see how interactions will be taking place between different elements. For example, you can see how the Administrator has access to very high level functions that the Customer will also be able to see once completing steps beforehand. See Figure 1.

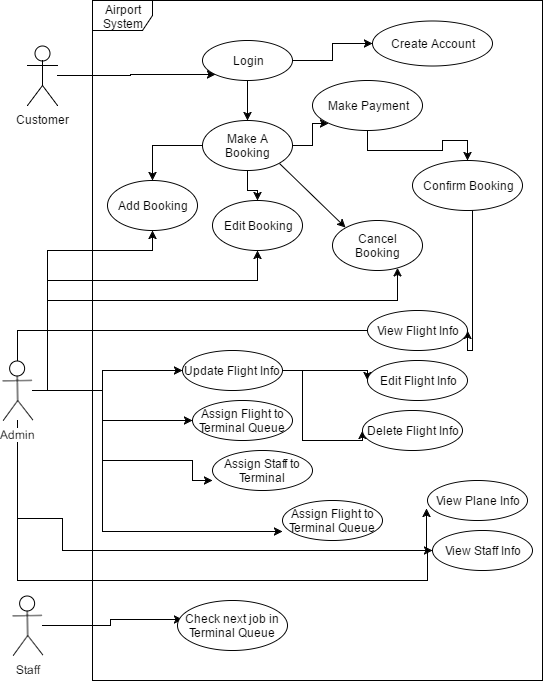


Figure 1: Use Case

More in-depth, text-based use cases to describe this diagram can be found in appendices.

## 2.2. UML Activity/Sequence Diagrams

A sequence diagram is useful for seeing how different functions will be passing through specific elements and visualising what kind of structure functions will have to be as we will see what kind of variables will be passing through and how each element will have to communicate (Visual-paradigm.com, 2017). This is essential during the designing code process simply because of the fact code can be structured by looking at this diagram and provides a significant starting point.

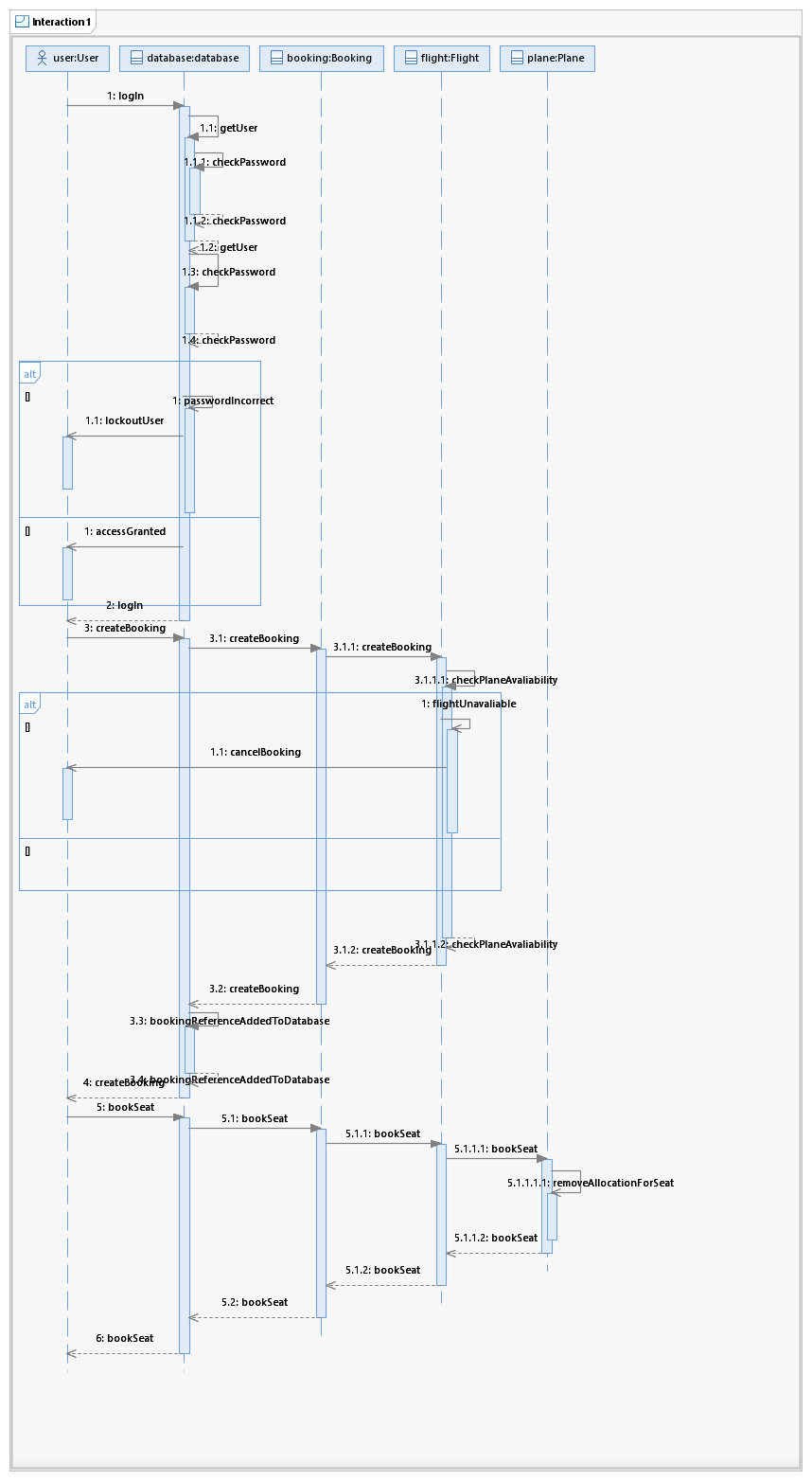


Figure 2: Sequence Diagram

## 2.3. UML Class Diagrams

We have created a class diagram for our project as this gave us a clear idea of how the system will work and how each class will be implemented and used for. In our class diagram we have included 8 diagrams. Each diagram includes the name of the class, the variables of the class and the functions in the class. This gives us a good idea of what the system is composed of and how the system is aimed to function.

The class diagram is essential for planning a project, without one it can be very difficult to understand how the functions of the system will be implemented; it does not require technical knowledge and can be understood by everyone in the project and not just the members who are working on the programming.

Class diagrams are also useful for determining the structure of the system, even after starting the development we can refer to the class diagram to clear any confusion while the system is in development.

Each class is linked to other classes using associations, this shows us what classes are connected and rely on another to a certain extent. Without class diagrams, a lot of confusion can be created during the development of the project, members of the project may be on different pages as to how the classes will be implemented which can lead to a less efficient development and a program that may not function as originally discussed.

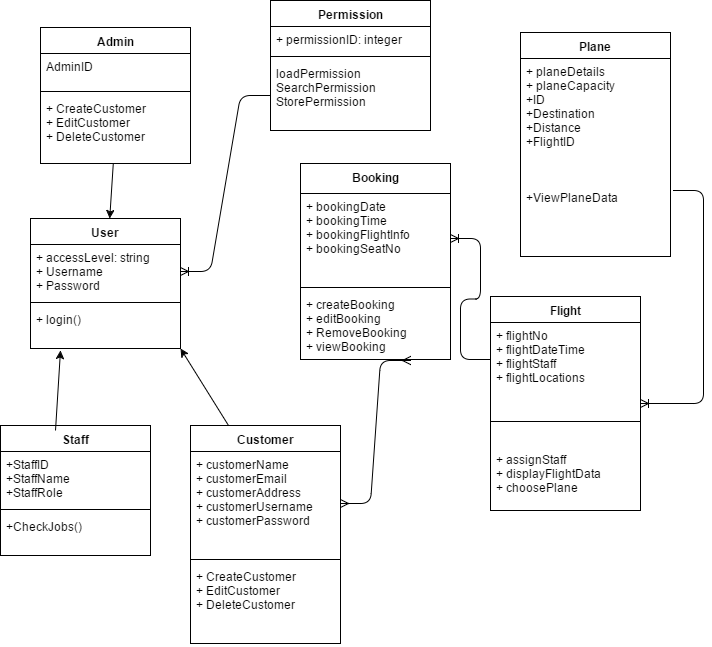


Figure 3: Class Diagram

# 3. Approach

## 3.1. Chosen Methodology

Methodologies are methods which are used to carry out projects and manage development teams in the IT industry. Many different mythologies fall under the agile methodology category (Agilemethodology.org, 2017). The main difference between agile methodologies and more traditional mythologies (Waterfall for example). Unlike waterfall model, agile mythologies follow a sequential approach (see Figure 4) which gives us room to change the requirements of the projects if we were to decide to aim for higher grades during the project.

This also gives us the chance to work on our project and start the very basic features without spending too much of our given time doing extensive planning, instead we can start building on the early features of the project.

This also allows us to start testing the project as we work on it, as opposed to only leaving the testing as one of the last phases, so we can adjust the early features before we start working on the more advanced features as opposed to testing the whole project at once, which may lead to us fixing a large amount of errors during the final weeks of the deadline.

One of the disadvantages of agile methodology is that if it not followed consistently, the project can become stages of code sprints, which could lead us to having an incomplete final project; but the reason we are choosing an agile methodology over waterfall is, if we do decide to go for the higher grades; we can adjust the requirements and we can add in new features once we have the basic features complete.

More specifically, we are following the Scrum agile mythology. This method is more beneficial to us than other methods is because of the constant meetings we have, so we can stay up to date on what has been complete, and what still needs to be worked on. This ensures we stay on top of the project work as opposed to delaying the work and completing other modules, because we won’t have to present our work at weekly meetings if we were using waterfall methodology for example.

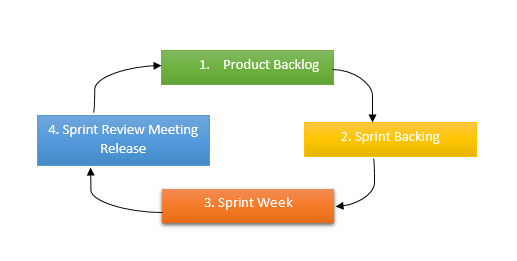


Figure 4: Agile Scrum Diagram

## 3.2. Description of Agile (Scrum)

### 3.2.1. Planning/Analysis Phase

In the very first phase of the lifecycle, our team is going to be planning and analysing extensively so we can start to work on the basics of the program as soon as the very first sprint. In this phase we plan, what our software will do, what features it should or shouldn’t have.

### 3.2.2. Sprint Planning/Meeting:

In this stage of the lifecycle, we assign work to the members of the team; determining what work which will be completed by each member. This will determine what work they will cover in the sprint week and present during the next meeting. Every meeting apart from the very first, we will discuss what we have completed since the last meeting, the problems we encountered and if we fixed them. Work is then assigned for the sprint.

### 3.2.3. Sprint

During this phase, which is the key phase of the lifecycle, during this week; work that was assigned to the members is completed. In this phase, we plan on creating our software, creating the actual functionalities, working on the documentation and testing the program. Each sprint will be one week, we will then return to the last phase of our lifecycle, iterating the meeting and sprint phase until the project is complete.

### 3.2.4. System Release

We will have multiple releases for our software, the later versions will have more features and will hopefully work as intended and early features may have bugs that will need to be fixed. The final release of the software is the one we will upload as the project submission and demo to our lab tutor. As it will be the most refined and polished version of the project.

## 3.3. MosCow Anlaysis

|  |  |  |  |
| --- | --- | --- | --- |
| **Must Haves** | **Should Haves** | **Could Haves** | **Won’t Haves** |
| Login System | Book specific Seats | Landing areas available for planes | Change plane route |
| Book Flights | View plane information (Engine size etc.) | Staff information on plane. | Delete/edit plane information (Engine Size) |
| Edit Bookings | Automatic updating flight departure board | Payment system for customer | 3D visualisation |
| View Bookings | Arrival times of planes | Admins change flight information |  |
| Edit Plane information | Two different type of users (Customer and admin) |  |  |

Table 2

## 3.4. Object Orientated Programming

The Bluebrook Airport system utilises an object-oriented approach in its code design. The entire concept of the program revolves around objects interacting with one another, such as users, flights and planes, making object-orientated programming ideal and efficient. Additionally, in a system like an airport booking system, maintainability is something that should always be thought about as the travel industry evolves and object-oriented design lends itself very well to this – adding functionality to objects is a much easier process than going through a procedural based program.

One useful feature of object-oriented programming is inheritance, which is something that has been utilised with the User class. There are several methods that all users should be able to perform, such as update their details, but additionally each different type of user will have different methods that they want to invoke. For example, regular accounts exclusively would want to view their flight details and their frequent flier points whereas staff accounts would want to be able to print off flight chart details and manually override flight details if required. None of these things should be accessed by the other class, which means we can make use of inheritance to make two subclasses, UserRegular and UserStaff. This is flexible and helps with maintenance, as we can make specific user subclasses for each staff role to allow them to do perform their own methods without having to rewrite all the shared methods inherited from the User class individually.

Another feature of an object-oriented system is that allows for easy maintainability is that it is much easier to model the system through UML (Unified Modelling Language) (www.tutorialspoint.com, 2017). This not only has benefits in the design process and initial building of a system, but a well-designed and maintained model makes it a lot easier to sustain and further develop already existing systems because of how easily read these models are and how clearly, they show all the relations between classes.

## 3.5. SDLC

The software development life cycle was created using Microsoft Project in the form of a Gantt chart. The team decided that this was the best option as the software not only allows the visualisation of how the project should be coming along at all stages but also has resource allocation and management that gives an accurate and realistic view of how the project will use assets.

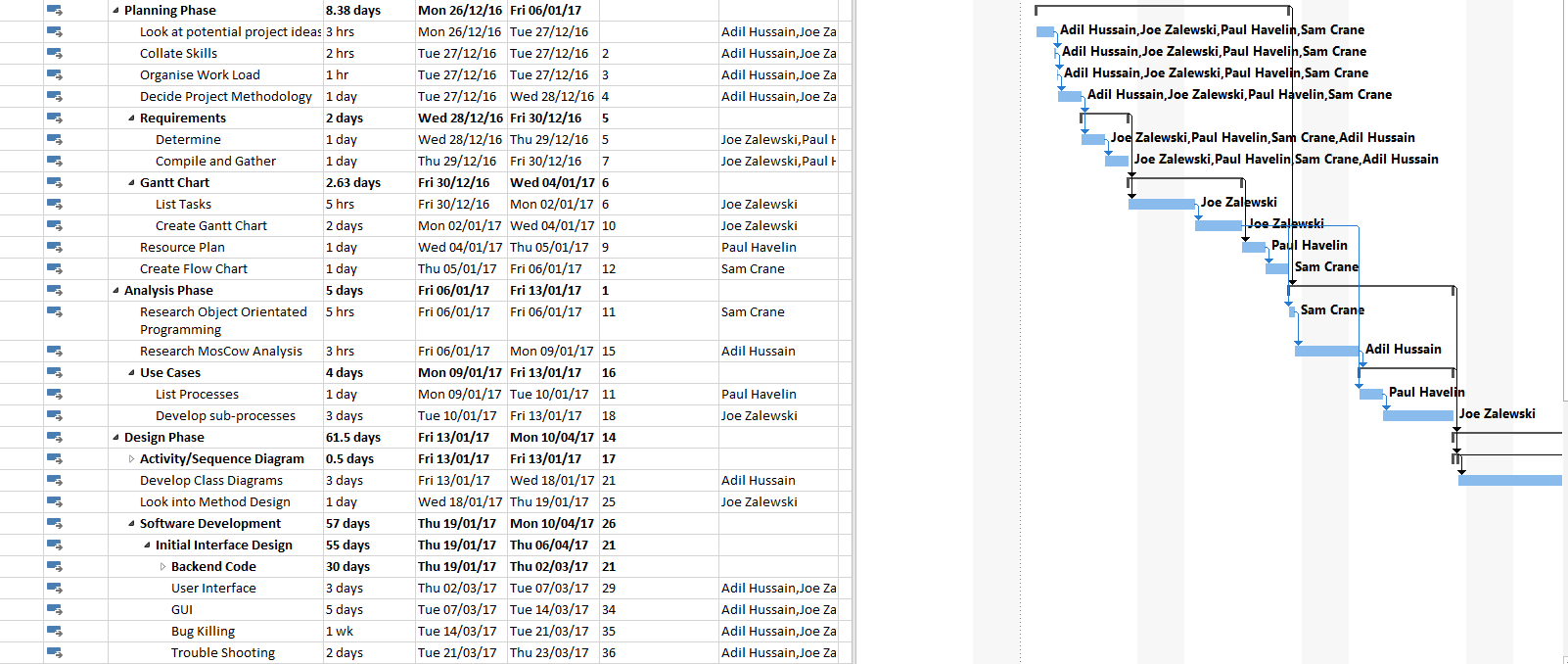


Figure 5 - Gantt Chart

The complete version of this Gantt chart can be found in the appendices.

## 3.6. Data Structure

The chosen data structure for the Bluebrook Airport System is a linear structure; a queue. The purpose of a queue is to organise data is a First In, First Out (FIFO) structure (see Table 3). This is implemented in the flight time board, which will display the current system time, the flight ID, flight locations, terminal number, departure time and current system time. There is a fixed amount of flights displayed on screen at one time, previously sorted in the system by departure time. The system/current time exceeds the time on the lowest sorted time, the time dequeues from the time board and the row is hidden from the time board, as this happens a later time, previously not displayed will be enqueued on the time board. The process loops throughout the day until all flights for that day have departed the airport.

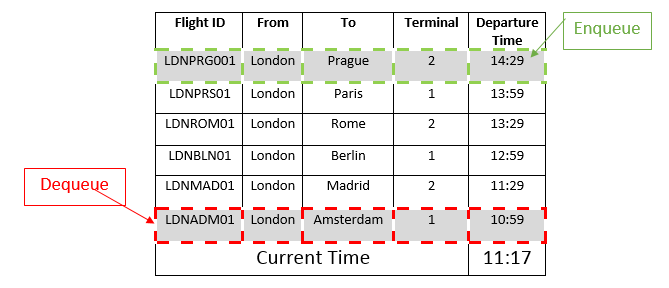


Table 3

### 3.6.1. Advantages:

* **Speed**: queue is quick to implement, use and maintain as there is only two main stages to organising the data (enqueuer/dequeuer).
* **Memory**: as a queue data structure is linear, it will take up more memory than a non-linear structure.

### 3.6.2. Disadvantages

* **Sorting**: Before the data can be put in the queue, it must be sorted first, in the case of the Bluebrook airport system, data must be sorted by time closest to system time

3.6.3. Decision

The decision to use a queue data structure was an obvious choice, as is used in real time board systems as it is the most accurate, easy to understand method of displaying sorted time data closest to a certain time (current system time).

## 3.7. Algorithms

For the implementation side of this module, the task was to implement two different types of algorithm that are efficiently and appropriately used to make the project more streamlined and relevant to normal coding standards, especially when discussing the size of the system.

### 3.7.1. Searching Algorithm

For the implementation of a searching algorithm into the Bluebrook Airport system, the chosen algorithm is a Binary Search Tree (figure 6). The purpose of using a searching algorithm in the system is to search for data within the system. Data can include flight data, plane data and customer data, staff rotas.  
  
Each vertex has each 0, 1 or 2 branches, which holds a record/key. At each vertex, smaller keys than the vertex are added to a subtree on the left, larger or equal keys are added to the subtree on the right.

#### 3.7.1.1. Main Operations

* Search
* Insert
* Pre-order Traversal
* In-order Traversal
* Post-order Traversal

#### 3.7.1.2. Steps Included

* User enters a flight time
* System uses the first flight time element in the xml file, this time becomes the root node
* The system cycles through the next flight time, if it is less than the root, the time is added to the left side of the tree
* If the time is greater or equal to the previous flight time, add to the right side of the node.

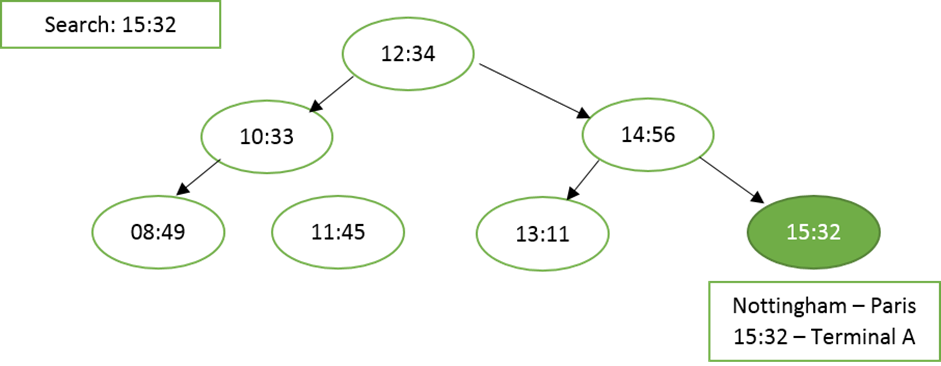


Figure 6

Advantages of this is that it is simple to implement and can be used for a variety of different data searches, for example flight, customers and staff. Despite this, data must be sorted first for this to work effectively.

### 3.7.2. Sorting Algorithm

For the implementation of a sorting algorithm into the Bluebrook Airport system, the chosen algorithm is Quicksort (Khan Academy, 2017). The purpose of using a sorting algorithm in the system is to sort all flight times in order of earliest, this is essential for the time board as all flight times are added to the queue data structure for the time board.

#### 3.7.2.1. Process of Quicksort in Airport System

1. The system will select a pivot point from the unsorted list of flight times from the XML file.
2. Place all flight times earlier/later than the pivot flight time.
3. Sort sub list of flight times less than/greater than the pivot flight time.
4. Ignore sub lists if flight times is empty.
5. Place all sorted flight times in the time board queue

Amount of comparisons: n²/2

Amount of flight time swaps: 2 N log N

Big O’ Notation: O(N log N) – Linear logarithmic (Increasing growth with N for times)

An advantage of using Quicksort is it has the fastest computational speeds of the other sorting algorithms, however a downside would be that it is difficult to implement in comparison to other algorithms.

# 4. Implementation

## 4.1. Software Testing Process

Throughout the process of the assignment, it was very important for us to test each feature of the program to ensure it works appropriately and fix any code which causes any strange behaviour with the program. To do this we used multiple testing methods. We used two different methods to test our program. One of them was the final testing method before uploading the project, this was the test plan cases. While the other testing method we used was Alpha testing, this was used from the very start of our assignment as it helped us identify problems with the feature we were currently working on at that time and fix the issue.

### 4.1.1. Test plan cases

Once we are ready to upload our assignment, we will use test cases to go through the functions of our system and see if they work as they are intended; if they are not we will attempt to fix them before the final upload. Test plan cases will consist of 5 columns (Test No, Test Name, how we tested the feature, did it work? And if it did not work, how we fixed it) this testing process will help us determine if certain features work as intended.

This process is very thorough in finding errors as we test every possible feature of the program with multiple inputs, with this testing we also check if error messages are thrown up when we enter invalid information on purpose to ensure the system is completely validated for any user errors.

### 4.1.2. Alpha testing

Throughout the course of our project we have and will continue doing beta testing on our program, using all features of the program as if we were the staff, passenger or admin. This method is used so we can look for possible bugs that users would have come across while the system. We used this method because we found it to be effective to find bugs that would not have been tested using the test plan cases, as that tests specific features.

This method of testing is used throughout the course of our assignment as opposed to the end like test plan cases, it shows us bugs with features that we are currently working on and helps us fix those errors.

## 4.2. Patterns

The purpose of patterns in the Bluebrook Airport system is to make the programming easier to understand, fix bugs, ensure future errors.

4.2.1. Separating Methods   
--------------------------------------------------------------------------------------------------------------------------  
Name: Make Booking  
Description: The user will view empty seats, choose a seat then save their booking in XML  
Functions used: AddBooking / GetCustomerInfo  
--------------------------------------------------------------------------------------------------------------------------

To separate the various areas of the system, commented sections are used. Commented sections are useful as they help identify the purpose of each area to ensure the user can easily find and understand each section in a simple to understand manner before viewing the functions and variables. An example is displayed above for making a booking, giving the section a name and brief description of what occurs in this section to help other users easily understand, where appropriate, the commented section includes which functions are used in this section, this can help identify bugs easily and prevent future errors.

### 4.2.2. Variable Names

|  |  |  |
| --- | --- | --- |
| **Customer** | **Booking** | **Plane** |
| cFirstName cLastName cPhoneNo cEmail | bBookingNo bSeats bFlightNo bFlightName | pCapacity pStatus pMileage pTerminal |

Table 4

For the variable names, the declarations are split into sections using a lowercase identifier such as in the example above such as a lower case ‘c’ for customer variables, a lower case ‘b’ for booking variables. The purpose of this is to be able to identify where variables come from when they are called in later function to reduce ambiguity such as if the customer variables are used in the booking section to link a customer to a booking. These variable names ensure any user can understand the program easily and reduce errors in future as the user can identify the origin of a bug easier.

4.2.3. Function Names/Declaration

For the naming of functions, nouns are used to clearly distinguish a function from variables or class names. Comments are not needed, as the functions should be descriptive of the purpose of function linked with the commented section, in which the function is defined and used.  
  
**Public:** The purpose of using functions within public classes is to ensure any class has access to the function such as viewing flight info, which can be done by all types of users e.g. customer, staff or admin.

**Private:** The purpose of using functions within private classes is to add security to the program, in which only the current private class will have access to a specific function such as admin functions cannot be used by staff or regular customers.

**Protected:** The purpose of using functions within protected classes is to ensure only the current class and subclasses have access to function, such as different levels of staff.  
  
**void Functions – Verbs (an action):** void functions are used throughout the Bluebrook Airport, they are unique functions which do not return a value, they are defined as camelCase using verbs as they are performing an action within the system such as the ‘createXML’ function, which is used create an xml file for the customer data to be stored in.

**Pre-Processor directives – ALL CAPS:** Pre-processor directives are either defined in the C++ properties for the project or using a #define at the top of the program using #define followed by the directives in capitals. An example of this is the /////// directive in the /////// section of the system.

**Booleans:** Boolean functions within the system are given correct Boolean names to return either a true or false output, an example for a function with the purpose of searching an xml file for a flight within the system is named ‘doesFlightExist’ as the function will return a value of True if the flight is found or a value of False if the flight cannot be found in the system. The purpose of this name is to include

# 5. Results

## 5.1. Reflection on Performance

The evaluation of the performance of the project is paramount due to being able to look at how the project can be added to, maintained and how it runs if it were an actual program being used. This is done by looking how fast it compiles and how easy it is to be taken elsewhere and still ran and how much memory it takes up when loading.

### 5.1.1. Reliability

The fact that the system will always complete its required job regardless proves that it is reliable, however, as stated later in this evaluation, it does require the specified files for the program to work properly as it doesn’t create its own XML files. However, as the code is modular in its programming style, it can be edited and still work in full, for when a part of the program is under construction.

### 5.1.2. Security

The security of the program is easily compromised. It holds user data as static strings in XML files that can be easily opened and viewed using file viewers, including passwords and email combinations. If this was to become a release it would have to be hashed and then decrypted when passwords are entered. This is also an issue due to the fact sensitive plane data is also set up as static information so can also be opened, which could compromise the planes and be a threat to the passenger’s safety.

### 5.1.3. Portability

The way the program is set up is it users various XML files that are scattered in the source directory, meaning that if this was moved around different machines, it would have to have all the files added to the same directory otherwise it wouldn’t input correctly. It wouldn’t let the user know, either, when a file cannot be found and would just stop working, which would take away from the user experience.

To improve this, we would have to add a directory that holds all the user information and the XML files, then this can be moved around with the release rather than having to add everything later. Or, another way of getting around this issue, is to be able to create the XML file if it doesn’t exist previously.

### 5.1.4. Maintainability

The program can be looked after constantly using the very easily viewable, split up functions and then code can be added and changed. Due to the way, the menu is set up, when a menu item is being worked on it can simply be taken out of the menu and not change how the program functions. Obviously, it wouldn’t be able to be inputted either. This is a slow way of maintaining a program but it is effective, although basic.

One of the only big issues with maintaining the program is the large size of it, as it would need someone who knows what they’re doing to change the program, as someone who didn’t and didn’t research the program enough could cause fatal errors to the system.

### 5.1.5. Scalability

The way the code is set up allows the program to be easily added to later in development, as each function is called from the menu and then the functions are kept separately. This also means that it can be synchronised when multiple people are adding to the code and Github commits can be completed more successfully. Also, due to the fact it is a command line based program, it can be added to slightly easier as new graphical interfaces don’t have to be made and instead one can just create a new function and add a text based number code that locates it in the menu.

Another aspect of how scalable the program is being the fact it can work with only a few flights and plane information and millions of the same data. How efficient the program is could be tester however due to the data structure of SQL it would allow millions of data entries and still be as easy to read from, although it might take a little longer to search through the file. This would be necessary for the system due to the fact it’s an airport system that would contain lots of flight codes and passenger information.

# 6. Conclusion

## 6.1. Professional, Social, Ethical and Legal Aspects

Due to the information that is collected by the system, one must be able to protect their sensitive information efficiently. As of current, the files are simply string and static, which would have to be strengthened and encrypted if this were to be a real system being used for an airport. It would also have to be hosted on a server, which would allow it to be used even more inappropriately, requiring even stronger efforts to maintain the network.

Due to the collection of user information, we would have to look at the maintenance of privacy for the users of the system, whether they be customers, staff members or administrators. A program of this scale should comply with the Data Protection Act 1998 (Legislation.gov.uk, 2017) as personal information is being processed constantly to ensure that user information isn’t lost and isn’t misused by hackers and third parties who have gained unauthorised access.

## 6.2. Reflection of Methods

As a group, we decided upon using Agile (Scrum) as our chosen methodology. After doing a lot of research as a group and discussing what we wanted to do as a project and how we code as a group, we found out this was the correct thing to do. By allowing us to release separate working products each time we had a ‘sprint’, this allowed us to look at how much work we had to do for the next release as defined by our project milestones which were tracked separately. By using this method, we could bug fix on each release and have pre-defined releases, which gave us a good scope for the project and allowed us to work on priorities for the next release. Also, by using Agile as a general methodology, we could take a step back at various points throughout the lifecycle and change how we were working and what we wanted to change.

One of the drawbacks of using Scrum is the fact that it is constant code development with not much time in-between. As soon as one release was done, another release was being worked on with no much time to evaluate what we did well, what we didn’t do well and what we need to improve, which lead to confusion after releases on what has been achieved and who has contributed effectively. Due to this, a different methodology might’ve been the most appropriate for this kind of activity, one that allowed more reflection and evaluation.

By developing a Gantt chart that mapped out our software development life cycle alongside our chosen methodology we could see how timings were working and how far along the cycle we are and where we should be aiming for next. It also allowed us to simulate a real-life project and resource map to look at how different people will be working on different things. It proved that we were terrifically over subscribed for the workload we were taking on and if this were a real-life company there would be a team of at least 10 working on this at any one time (when considering working days, weekends, overtime and payment).

Developing a detailed flowchart was a good move as we could look at how different elements communicated with each other, in the same way we used the use cases. The flowchart accurately matches the final version product which is a huge benefit as it shows we had a structured development process for our project and shows our chosen methodology, although being impractical at times, worked. The flowchart could’ve included slightly more detail to allow alternative flows, as set out in the use cases in appendices.

Deadlines and milestones were predefined at a very early point in the life cycle to give us an idea of how far we were getting, how much work we had left to do and where we were falling short. This allowed us to chase up issues that were coming up ASAP so that we didn’t fall behind on the deadlines and by having separate releases, in line with our methodology, allowed bug fixes and evaluation. Despite the positives, it was difficult to keep on track of the milestones at certain points and when members were falling behind on documentation work.

Github turned out to be the fifth group member during this process. Although there were issues with branches early in the process (especially when working on the same file which really isn’t easy), it allowed remote working and helped us to track the progress the group was making. Using Github instead of Dropbox was a lot better as this ensured that no one could affect the code itself and then it being lost forever as Github allows you to go back to a previous commit that basically backed up each implementation made.

## 6.3. Reflection of Final Version

Firstly, the user can login to the system by accessing the XML files. This creates a stable way to accessing files in a structured way that can be viewed outside of the system for exporting purposes and diagnostic. Another advantage of the way log in is used is the fact it searches through a very easy to read file, which makes the efficiency of the program very fast and uses very little memory reading through the XML file. It iterates through as if it is a big array which allows it to run in the background and memory to be targeted where it needs to be.

A user can also be registered which makes the program into a slightly more user friendly design, as it allows direct input from the user into the system. This adds to the XML file directly which the login function reads from which shows the fact that the system is well integrated with each other. It stores static information and an admin can go through and change the permissions of a user which will open more options in the system’s menu and allow higher privileges.

A user can book a flight that is valid at any time. This is done by reading a separate XML file with seat allocations into the system and allowing the customer to pick a seat depending on that. This then adds the user to the flight information XML document, which also adds to how integrated the system is. Bookings can be altered by the admins. This was a good example of implementation as it takes different types of information and displays it into the same file without causing issues and altering efficiency.

By having a time board in our software, it allows the implementation of our Quicksort and Queue algorithms that import times of flights, sorts them appropriately and then places them onto a Queue which would, in the real world, be outputted to a separate TV screen as you see in many airports. This is good as it uses a very complex sorting algorithm efficiency and then the queue would be as you’d see on a real-life situation so shows the scope of our program. This could be better in some ways due to the fact Quicksort is one of the slowest and hardest to implement, so in moving the system around it could cause issues that we hadn’t foreseen.

The plane data that the software receives is also kept inside of an XML file that incorporates the different rows and columns of the plane and has storage capacity information and model types. This is imported when an admin creates a flight using that plane data and it is then loaded in and creates a text file that holds seat information.

## 6.4. Project Evaluation

### 6.4.1. Positives

In this project, we did a lot of things positively and ensured that we worked as a team throughout the process. One of the positives of this task was splitting up the coursework documentation early and looking at our strengths and weaknesses from previous modules and that shaped who did what. For example, Joe was good at Gantt charts and have done them for every single module that required them in the past which led to an easy decision in who was being allocated that task. Also, there was a lack of selfishness in how tasks were allocated and everyone did their fair share throughout the tasks, which helped team dynamics and morale to get things done where required.

Another positive was the use of Github for the code development as well as the organisation of coursework files. This acted as a centralised location for all the work to be submitted with the safety of using the backups if things went wrong, as well as having a safe place to keep the files in case of memory loss. By using this we were also able to have issues generated, which could be assigned to different people and then labelled appropriately which helped organisation and led everyone to understand what they had to do.

The code itself mimics a lifelike problem that has been solved, which helps the development as rational scenarios could be thought up and then this changed the code accordingly. This also allowed us to research how airport systems work and what elements have been used before and which ones should be included in our project.

### 6.4.2. Negatives

One of the drawbacks of the program is the fact that it doesn’t have a graphical interface which takes away from the user experience slightly and makes it a little outdated in it’s aesthetic. Despite this, it still has the required functionality for what we set out for, but this would be added later if we had more time. This is discussed below.

Another issue was meeting deadlines that we had pre-defined to try and set up a structure to follow. This became hard when considering different projects and coursework hand-ins, which resulted in a lack of communication at certain points of the module and a sudden kick into action about a week before the final hand in. Although the work was eventually done, more time should’ve been given to allow issues to be sorted in the final week rather than a rush of work that could’ve been a lot better. Despite this, the group worked efficiently when it was required and there was an equal amount of work done by all members and it resulted in a very positive working environment to develop a product in.

### 6.4.3. Future Additions

A GUI that allowed the user to log in and register and have a booking list on the front page. It would also hold information such as flights and online check-in and then would allow the user to book the seat and see a diagram of the plane, rather than 0 and X. This would improve the user experience which is integral for a today’s-world product.

Instead of using a static data file like XML, something like a more significant database could be used, for example SQL, which would allow easier changes within tables and data and would also be easier to write to, as SQL parsing would be a difficult implementation but would then only have to be called remotely from that point onwards.

# 7. Appendices

## 7.A. Use Cases

|  |  |  |
| --- | --- | --- |
| **ID:** 1 | **Name:** Create Account | |
| **Actors Involved:** Customer | | **Preconditions:** User runs system/ selects create from the Main Menu |
| **Main Flow**  The user enters the create account through the selection on the main menu, the user is prompted with each detail, e.g. forename, surname, phone number. Each detail is validated such as length check on phone number and format check on email address. When all valid data is entered, the system asks the user for confirmation, if the user selects yes, the system will save that users valid details to the xml file. | | **Alternative Flow**  User enters invalid data, error message occurs, user cannot continue until valid data is entered  User doesn’t confirm details, system returns to the main menu without saving  Details entered are already in the xml file, error message occurs, create account function begins again from the start |
| **Post conditions:** Customer details are saved to the XML file | | |

|  |  |  |
| --- | --- | --- |
| **ID:** 2 | **Name:** Login | |
| **Actors Involved:** Customer | | **Preconditions:** user has created an account |
| **Main Flow**  The user enters the log-in through the selection on the main menu the user enters their email and password, the system checks the xml file for a matching email and password and returns the user to the main menu, logged in | | **Alternative Flow**  User enters invalid data; an error message occurs until the user enters valid log in detail. The user cannot access features such as ‘Make a Booking’ until they have logged in successfully. |
| **Post conditions:** User returns to the main menu, logged in as an admin or regular user. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** 3 | **Name:** Make A Booking | |
| **Actors Involved:** Customer | | **Preconditions:** User is logged in |
| **Main Flow**  The user enters the ‘Make a Booking’ through the main menu, in which they will be asked to select their flight locations, flight and seat after the system displays all available seats on that flight, prompting the user to enter another seat. When the user has finished booking their seat they will confirm their booking which will be added to the xml file once the user has made their payment. | | **Alternative Flow**  The user selects an unavailable or invalid seat name e.g. ‘A566336’ an error message will occur and the booking will not be saved until a valid seat is entered. |
| **Post conditions:** Make a payment | | |

|  |  |  |
| --- | --- | --- |
| **ID:** 4 | **Name:** Add Booking | |
| **Actors Involved:** Customer, Admin | | **Preconditions:** user has already booked one seat |
| **Main Flow**  After the user, has booked one seat, they will be asked by the system if they wish to book another seat and add this extra seat to their booking, when the user selects ‘yes’ the system will run the ‘Make a Booking loop again until the user no longer wishes to book another seat and continue to the make a payment section. | | **Alternative Flow**  Booking does not save to xml  Available seats are not displayed  Unavailable seats are still displayed as available  Booking isn’t made user specific. |
| **Post conditions:** User makes a payment | | |

|  |  |  |
| --- | --- | --- |
| **ID:** 5 | **Name:** Make Payment | |
| **Actors Involved:** Customer | | **Preconditions:** logged in user has selected their flight and seat. |
| **Main Flow**  After the user, has selected their flight and seat the user will input their payment details which will be validated using format and length checks, a confirmation message will display when the user has successfully entered valid data and will be returned to the main menu. | | **Alternative Flow**  User entered invalid payment information, error message occurs, user prompted to reenter the details until they are correct. |
| **Post conditions:** Payment Confirmation | | |

|  |  |  |
| --- | --- | --- |
| **ID:** 6 | **Name:** Confirm Booking | |
| **Actors Involved:** Customer | | **Preconditions:** payment has been made |
| **Main Flow**  After the payment, has been made, the user will be asked to confirm this booking, if the user confirms the booking, the screen will display their name, flight details and payment information, the data is then saved in an xml file. | | **Alternative Flow**  User doesn’t confirm the booking, error message occurs, returns to the main menu.  Data is not saved to the xml file |
| **Post conditions:** Save XML file, return to main menu | | |

|  |  |  |
| --- | --- | --- |
| **ID:** 7 | **Name:** Edit Booking | |
| **Actors Involved:** Customer, Admin | | **Preconditions:** Booking has been confirmed |
| **Main Flow**  After a user, has made a booking, that is saved in the xml file, the user selects ‘Edit Booking’ in which their data is appended as their current booking is removed and replaced with a new booking as the user selects the flight locations, times and seat allocation, the new booking is then appended in the booking xml file. | | **Alternative Flow**  User doesn’t want to make an edit, wants file booking to remain the same, cancels edit booking midway through, returns to main menu, booking xml file is not appended. |
| **Post conditions:** Add edited flight to time board / Append XML file | | |

|  |  |  |
| --- | --- | --- |
| **ID:** 8 | **Name:** Cancel Booking | |
| **Actors Involved:** Admin, Customer | | **Preconditions:** Booking has been made |
| **Main Flow**  After a user, has made a booking, that is saved in the xml file, the user selects ‘Edit Booking’ in which their booking is deleted and appends the booking xml file for that booking from the user, the seat allocation removed from the system will then we available again the Make a Booking section. | | **Alternative Flow**  User no longer wishes to cancel booking, xml file remains the same, returns to main menu.  Booking doesn’t successfully cancel, cancel booking function reruns until booking is successfully canceled. |
| **Post conditions:** Add cancelled flight to time board / Append XML file | | |

|  |  |  |
| --- | --- | --- |
| **ID:** 9 | **Name:** View Flight Info | |
| **Actors Involved:** Admin, Customer, Staff | | **Preconditions:** the user is logged into the system |
| **Main Flow**  When the View Flight Info is selected from the menu regardless of access level, the flight info is displayed for each terminal, the time-board displays all times, previously sorted by time using Quicksort and placed in the queue. | | **Alternative Flow**  Flights are not successfully sorted  Flights are not enqueued or de-queued.  Time board doesn’t successfully display  Flights are delayed or canceled. |
| **Post conditions:** Return to main menu | | |

|  |  |  |
| --- | --- | --- |
| **ID:** 10 | **Name:** Update Flight Info | |
| **Actors Involved:** Admin | | **Preconditions:** User is logged in as Admin |
| **Main Flow**  If a new flight becomes available the admin will add this flight’s locations, and times to the system to become available to book, this flight is sorted using Quicksort and added to the time-board queue. | | **Alternative Flow**  Flight cannot be added to time board queue  Flight gets delays to later than time board times. |
| **Post conditions:** Append XML file, Return to main menu. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** 11 | **Name:** Edit Flight Info | |
| **Actors Involved:** Admin | | **Preconditions:** User is logged in as Admin |
| **Main Flow**  If a flight needs editing, e.g. a flight Is delayed to a later time due to foreseen circumstance, the admin will select edit flight from the admin menu, in which they can edit the flight time, this will edit the xml file, the flight will then have to be passed through the sorting algorithm, in which the system places the flight in the new correct location on the time-board. | | **Alternative Flow**  XML file doesn’t update correctly, reruns editing function  Flight no longer needs to be edited, xml remains the same and Is not appended. |
| **Post conditions:** Flight added to time-board | | |

|  |  |  |
| --- | --- | --- |
| **ID:** 12 | **Name:** Delete Flight Info | |
| **Actors Involved:** Admin | | **Preconditions:** User is logged in as Admin |
| **Main Flow**  The user will select the Delete Booking section from the admin menu, if a flight is cancel | | **Alternative Flow**  XML file doesn’t update correctly, reruns deleting function  Flight no longer needs to be deleted, xml remains the same and Is not appended. |
| **Post conditions:** g | | |

|  |  |  |
| --- | --- | --- |
| **ID:** 13 | **Name:** Assign staff to terminal | |
| **Actors Involved:** Admin | | **Preconditions:** User is signed in as admin |
| **Main Flow**  If the user is logged in as admin, the user selects ‘Assign Staff’ from the admin menu, in which the user can select staff saved in the system and assign them to either Terminal A or B, to each member of unassigned staff. | | **Alternative Flow**  Staff are assigned same job  Staff assigned two jobs at the same time  Staff job not updated after being allocated |
| **Post conditions:** Append staff terminal assignment | | |

|  |  |  |
| --- | --- | --- |
| **ID:** 14 | **Name:** Assign flight to terminal queue | |
| **Actors Involved:** Admin | | **Preconditions:** User is signed in as admin |
| **Main Flow**  When the admin selects assign flight to time board queue, the system will add a flight to either terminal A or B. | | **Alternative Flow**  Terminal currently in use  Two flights using the same terminal |
| **Post conditions:** Flight assigned to Terminal, Returns to main menu. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** 15 | **Name:** View Plane info | |
| **Actors Involved:** Admin | | **Preconditions:** User is logged in |
| **Main Flow**  If the user is logged in as admin, pane information can be viewed such as capacity, size, model and total distance flown across a certain time frame. | | **Alternative Flow**  Plane info has not been updated after flight, e.g. miles not added  Plane info doesn’t load from xml file |
| **Post conditions:** Returns to main menu | | |

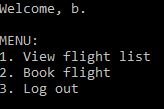
|  |  |  |
| --- | --- | --- |
| **ID:** 16 | **Name:** View Staff info | |
| **Actors Involved:** Admin | | **Preconditions:** The user must be signed in as admin |
| **Main Flow**  When the admin is logged in, from the admin menu the user selects View Staff info in which the system will display staff details such as Name, role in the airport and upcoming jobs. | | **Alternative Flow**  Member of staff has duplicate data  Staff data isn’t correct  No members of staff in the system |
| **Post conditions:** Return to main menu. | | |

|  |  |  |
| --- | --- | --- |
| **ID:** 17 | **Name:** Check next job in terminal queue | |
| **Actors Involved:** Staff | | **Preconditions:** user has logged in as staff |
| **Main Flow**  If the user is logged in as a member as staff, when the user selects ‘Check Job’ from the menu, it will view their upcoming rotas including Terminal assignment and working times. | | **Alternative Flow**  No jobs in the terminal queue  Staff has been assigned two roles at the same time  Multiple members of staff allocated the same job |
| **Post conditions:** Return to main menu. | | |

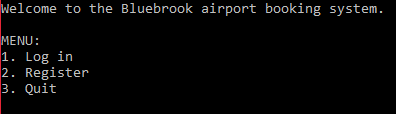
## 7.B. Flowchart

C:\Users\n0623403\Desktop\Software-Design-Imp\doc\7. Appendix\Flowchart UPDATE.png

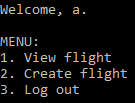
## 7.C. Menu Areas



User Menu



Main Menu



Admin Menu

## 7.D. Gantt Chart

Please find in file as AppendixD-GanttChart.pdf

## 7.E. Test Cases

Please find in file as AppendixE-TestCases.pdf

## 8. References

Agilemethodology.org. (2017). *The Agile Movement*. [online] Available at: http://agilemethodology.org/ [Accessed 2 May 2017].

Khan Academy. (2017). *Khan Academy*. [online] Available at: https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/overview-of-quicksort [Accessed 2 May 2017].

Legislation.gov.uk. (2017). *Data Protection Act 1998*. [online] Available at: http://www.legislation.gov.uk/ukpga/1998/29/contents [Accessed 2 May 2017].

Visual-paradigm.com. (2017). *Software Design Tools for Agile Teams, with UML, BPMN and More*. [online] Available at: https://www.visual-paradigm.com/VPGallery/diagrams/Sequence.htmlntSyqtqXlwOg [Accessed 2 May 2017].

www.tutorialspoint.com. (2017). *UML Tutorial*. [online] Available at: https://www.tutorialspoint.com/uml/ [Accessed 2 May 2017].