

Project Report

Group 4

Version 1.00
(21/04/2015)

Contents

1. Business Case (page 6)

1.1. Business Plan Summary (page 6)

1.1.1. The Business (page 6)

1.1.2. The Market (page 6)

1.1.3. The Future (page 6)

1.2. The Business (page 7)

1.2.1. Organisation Chart (page 7)

1.2.2. Key Personnel (page 7)

1.2.3. Operations (page 8)

1.3. The Market (page 8)

1.3.1. Market Targets (page 8)

1.3.2. Customer Demographics (page 8)

1.4. The Future (page 8)

1.4.1. Vision Statement (page 8)

1.4.2. Mission Statement (page 9)

1.4.3. Goals/Objectives (page 9)

1.4.4. Action Plan (page 9)

2. Project Plan (page 10)

2.1. Project Estimation (page 10)

2.2. Project Schedule (page 10)

2.3. Rational Unified Process Model (page 11)

3. Feasibility Study (page 13)

3.1. Technical Feasibility (page 13)

3.2. Economical Feasibility (page 13)

3.3. Organisational Feasibility (page 14)

3.4. References (page 14)

4. Risk Analysis and Management Plan (page 15)

4.1. Executive Summary (page 15)

4.2. Introduction (page 15)

4.3. Risks (page 16)

4.3.1. Loss of Team Members (page 16)

4.3.2. Loss of Access to Technology (page 17)

- 4.3.3. Loss of Documentation/Code (page 17)
- 4.3.4. Loss of Client (page 17)
- 4.3.5. Loss of Funding (page 18)
- 4.3.6. Suspension of the Project (page 18)
- 4.3.7. Changing of Requirements (page 19)
- 4.3.8. Changing of Clients (page 19)
- 4.3.9. Maintenance (page 20)
- 4.3.10. Over-Evolution of the System (page 21)
- 4.4. References (page 21)**

5. System Requirements Specifications (page 22)

5.1. Introduction (page 22)

- 5.1.1. Purpose (page 22)
- 5.1.2. Scope (page 22)
- 5.1.3. References (page 23)

5.2. Overview/Description (page 23)

- 5.2.1. Product Perspective (page 23)
- 5.2.2. System Interfaces (page 23)
 - 5.2.2.1. *User Interfaces* (page 24)
 - 5.2.2.2. *Hardware Interfaces* (page 24)
 - 5.2.2.3. *Software Interfaces* (page 24)
 - 5.2.2.4. *Communication Interfaces* (page 24)
 - 5.2.2.5. *Memory Constraints* (page 24)
 - 5.2.2.6. *Operations* (page 25)
- 5.2.3. Product Functions (page 25)
- 5.2.4. User Characteristics (page 26)
- 5.2.5. Constraints (page 27)
- 5.2.6. Assumptions and Dependencies (page 27)

5.3. Specific Requirements (page 27)

- 5.3.1. User Management Functions (page 27)
 - 5.3.1.1. *Customer Functions* (page 28)
 - 5.3.1.2. *Travel Agent Functions* (page 47)
 - 5.3.1.3. *Staff Functions* (page 53)
 - 5.3.1.4. *Booking Manager Functions* (page 62)
 - 5.3.1.5. *Service Manage Functions* (page 66)
 - 5.3.1.6. *Profile Manage Functions* (page 72)
 - 5.3.1.7. *Flight Manage Functions* (page 80)
 - 5.3.1.8. *Administrator Functions* (page 90)
- 5.3.2. System Requirements (page 92)
- 5.3.3. Non-Functional Requirements (page 95)

6. Use Case Diagrams (page 97)

7. Architectural Design (page 101)

7.1. Logical View (page 101)

7.2. Doman Model (page 102)

7.2.1. Sub-System and Database View (page 103)

7.2.2. User and Sub-System View (page 104)

7.3 Process View (page 105)

7.3.1. Staff Member Creates Account for Customer – Sequence Diagram (page 105)

7.3.2. Customer Books Flight – Sequence Diagram (page 106)

7.3.3. Travel Agent Books Flight for Customer – Sequence Diagram (page 107)

7.3.4. Service Manage Modifies Service Items for a Flight – Sequence Diagram (page 108)

7.4. Implementation View (page 109)

8. Data Dictionary (page 110)

8.1. Class Name: Airport (page 110)

8.2. Class Name: Route (page 111)

8.3. Class Name: Aircraft (page 112)

8.4. Class Name: Schedule (page 113)

8.5. Class Name: Flight (page 114)

8.6. Class Name: Customer (page 116)

8.7. Class Name: TravelAgent (page 118)

8.8. Class Name: ServiceItem (page 119)

8.9. Class Name: FlightService (page 120)

8.10. Class Name: Booking (page 120)

8.11. Class Name: Staff (page 121)

8.12. Class Name: ProfileManager (page 122)

8.13. Class Name: FlightManager (page 123)

8.14. Class Name: BookingManager (page 123)

8.15. Class Name: ServiceManager (page 124)

8.16. Class Name: Databse (page 124)

8.17. Class Name: ProfileSystem (page 129)

8.18. Class Name: ReservationSystem (page 130)

8.19. Class Name: ReportSystem (page 131)

8.20. Class Name: ServiceSystem (page 131)

8.21. Class Name: FlightSystem (page 132)

9. Group Summary (page 133)

10. Team Meetings (page 134)

- 10.1.1. Memorandum #1 (page 134)
- 10.1.2. Meeting Minutes #1 (page 135)
- 10.1.3. Memorandum #2 (page 137)
- 10.1.4. Meeting Minutes #2 (page 138)
- 10.1.5. Memorandum #3 (page 140)
- 10.1.6. Meeting Minutes #3 (page 141)
- 10.1.7. Memorandum #4 (page 143)

11. Work Diaries (page 144)

- 11.1. Diary – Kresimir Bukovac (page 144)
- 11.2. Diary – Ali Sayed (page 153)
- 11.3. Diary – Peter Mavridis (page 155)
- 11.4. Diary – Darryl Murphy (page 158)

12. Versioning Evidence (page 165)

1. Business Case

1.1. Business Plan Summary

1.1.1. The Business

Business name: Group 4

Business structure: Students

Business location: University of Wollongong

Date established: March 4th, 2015

Relevant owner experience: We are 2nd and 3rd year students all completing a Computer Science degree.

Products/services: Flight Booking System

1.1.2. The Market

Target market:

The targeted market are stakeholders wishing to book flights either for holidays or business trips. Staff will also be a target market allowing them to perform various roles through the system.

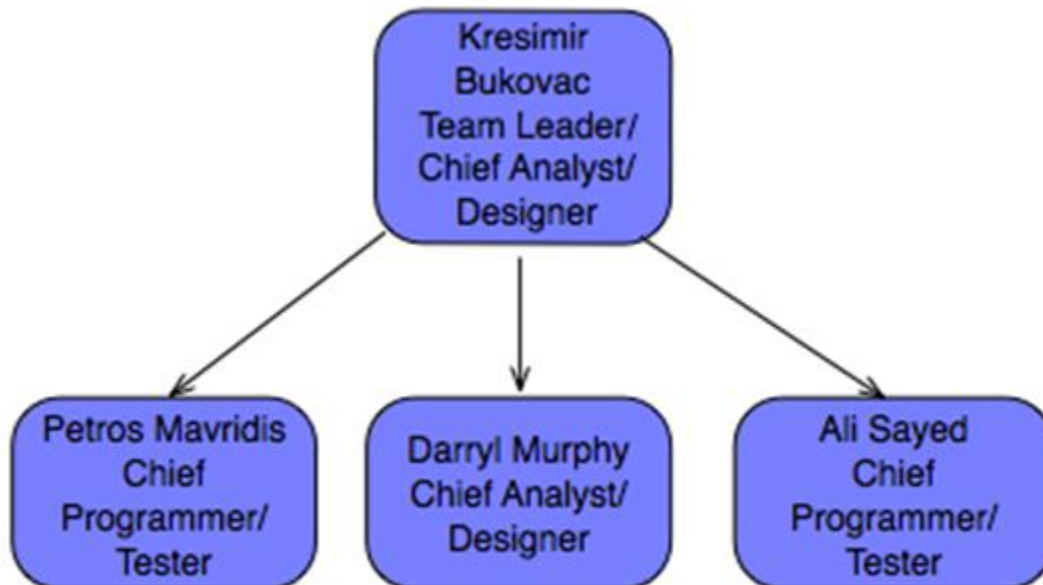
1.1.3. The Future

Goals/objectives:

The team's goal is to provide a stable and fully functional flight booking system that will meet our client's needs and its customers.

1.2. The Business

1.2.1. Organisation chart



1.2.2. Key personnel

Current staff

Job Title	Name	Skills or strengths
Team leader Chief Analyst/Designer	Kresimir Bukovac	Knowledge of C++ programming language and an understanding of team management.
Chief Programmer/Tester	Petros Mavridis	Knowledge of the C++ programming language and some GUI programming.
Chief Analyst/Designer Programmer	Darryl Murphy	Knowledge of the C++ programming language. Skills include designing the system for the team to code.
Chief Programmer/Tester	Ali Sayed	Knowledge of the C++ programming language and some GUI programming.

1.2.3. Operations

Production process

The process used to develop the flight booking system will be a Rational Unified Process along with the Prototyping method of showing our clients a progress of the developed product for feedback.

Technology (Software):

The technology we will be using will be C++ language with the initial prototyping being a command line package. Our stretch goal will be to implement a GUI using Qt.

Communication channels:

Our team has been communicating through face to face meetings and over Skype®. Our meeting with our client has been face to face during allocated times in tutorials.

Quality control:

Quality control will be measured through testing of our code, each function or portion of code will have corresponding test code to test its correctness.

1.3. The Market

1.3.1. Market targets

We will be targeting people that want to book flights for themselves or others. Our customers will typically be business people needing to travel for work and individuals who will be booking flights for themselves or their families for holidays.

1.3.2. Customer demographics

Typically users of the system will have to be over the age of 18 to make bookings and be in possession of a credit card. People under the age of 18 can fly but will need to have the booking made by an adult for them to fly.

1.4. The Future

1.4.1. Vision statement

Our future plan for the flight booking system is to iteratively and incrementally develop the system over a period of 5 weeks with prototyping to show the client and get feedback from them. This will allow the team to develop a system that best meets the needs of our client.

1.4.2. Mission statement

Our team will be working on the code in a modular fashion, where each person will be able to work on small functions and test them, then bring all the functions together under the main program. The final system will be white box tested with test cases to ensure the correctness of the system.

1.4.3. Goals/objectives

The teams goal is to be able to achieve a stable and fully functional system that will meet our clients needs. The team would like to achieve a high standard in producing the system. One of the stretch goals the team would like to achieve is to have a GUI instead of a command line tool, this will give the system a more presentable look and smoother feel to the system.

1.4.4. Action plan

Milestone	Date of expected completion	Person responsible
Software requirement specification	21-04-2015	Kresimir Bukovac
Complete gantt chart	19-04-2015	Ali Sayed
Complete business case	21-04-2015	Petros Mavridis
Use cases	21-04-2015	Darryl Murphy
Domain Model	21-04-2015	Darryl Murphy
First prototype to show	06-05-2015	All team members

2. Project Plan

2.1. Project Estimation

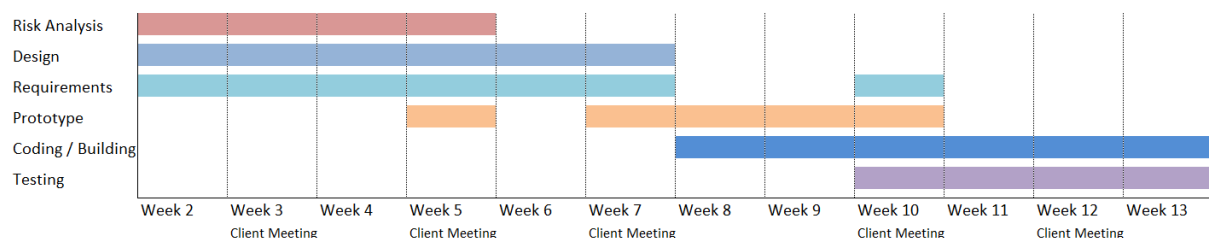
Using the Constructive Cost Model (COCOMO), the project is estimated to take 1 month and 17 days to complete between four people. Planning is estimated to take 7 days (15%), analysis is estimated to take 9 days (20%), design is estimated to take 17 days (35%) and implementation is estimated to take 14 days (30%). The calculations are shown below:

$$\text{Effort} = c * \text{KLOC}^k$$
$$6.28 = 2.4 * 2.5^{1.05}$$

The project is estimated to take 6 months and 8 days (6.28 person-months) of work split between four people becomes 1 month and 17 days-worth of work.

2.2. Project Schedule

A gantt chart was created to give a base schedule plan for the life of the project. It shows at what periods throughout the life of the project and for how long the team will be working on specific phases of the project.

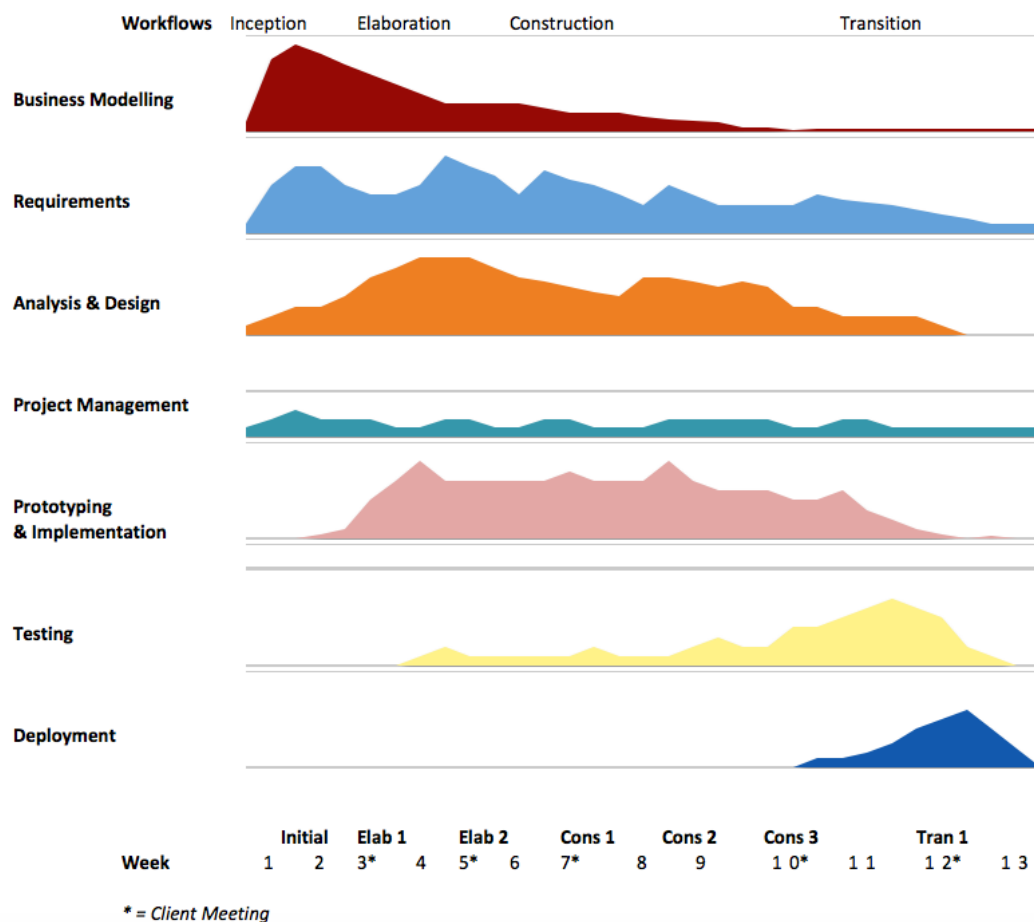


Week 2	Risk analysis, design and gathering general requirements
Week 3	Risk analysis, design and gathering general requirements
Week 4	Risk analysis, design and constructing prototypes for the next client meeting
Week 5	Risk analysis, design, gather general requirements and present rough prototype
Week 6	Design, gathering more requirements
Week 7	Design, gathering more requirements and present prototypes in the client meeting
Week 8	Create more detailed prototypes
Week 9	Create more detailed prototypes
Week 10	Present detailed prototypes, confirm final requirements, begin coding/testing 'skeleton'
Week 11	Building and testing
Week 12	Building and testing
Week 13	Building and testing

2.3. Rational Unified Process Model

For this project, we will be using the Rational Unified Process (RUP) model. We plan to develop the project iteratively through the four phases; Inception, Elaboration, Construction and Transition. Our project plan is presented visually below.

Flight Management - Project Plan



In the inception phase (weeks 1-3), our team objective is to obtain a scope and business case for the project and expand on the general requirements from the client. In this phase, we will also develop use cases. After these two weeks, we will have our first meeting with the client in week 3, which will begin the Elaboration Phase.

In the first block of this phase, we will seek to define specific requirements. We will also expand our understanding of our initial use cases, and add more if need be after the first client meeting. In addition to the use cases, we will present a prototype walkthrough to present to the client in the second client meeting in week 5. In the second elaboration block, we will further refine the requirements, as well as begin creating class diagrams to plan which classes are needed, as well as their association and inheritance.

At this point, we plan for most of the requirements and planning to be done, but do expect for them to be changed and some requirements added by the client in the future meetings. We will now begin constructing the program and start our first construction block. Using our Use Case diagrams and our UML Class diagrams, we will begin by constructing 'skeleton' files. Our main objective here is to get all of the classes and menu systems working, interacting, and functioning properly, before coding algorithms. To finish off this block, we will complete thorough testing on the program, and complete any bug reports or system failures.

Using our collected error reports from the previous construction block, we will begin our second construction block by fixing these errors and continuing to build the program. Our main objective here is to expand on the program's functionality and key features. After most of the coding is done in this block, we will begin thorough testing. We will then create a detailed prototype to present to the client in week 10.

After we present the detailed prototype to the client in week 10, we will finalize the bulk of the construction phase. We anticipate that the client will want to add a few more system requirements and less important features, but most of the coding will be done by this point.

In the transition phase, we will prioritize testing and overall refinement of the program's functionality. Our team plans to develop a beta system, which will allow thorough testing over a short period of time if given to many users. The bulk of the transition phase will be to deploy the project to the client including a final presentation of how to use it. This phase will finalize the project.

3. Feasibility Study

3.1. Technical Feasibility

The requirement of the project is to create a system for a flight booking system. The team consists of lead analysts and designers to conduct research and deduct all and the best possible paths and designs which can be taken to achieve the desired end result.

The lead programmers will then be able to implement the system by following the planned-out design of the analysts and designers. As they implement the system, testers will be trying to break the system to ensure that at the time of release, the system is secure, bug-free and not crashable. This will be done simultaneously with each version of the project that gets given to the testers after specific functions of the system have been implemented to ensure the real-time access to feedback. With this, the chance of errors delaying the project will greatly be reduced, because the problem will immediately be put to attention and fixed before the system develops any further and possibly to a stage where changes become much more costly to go back and repair.

The project requires the access of a database through a text-based c++ program, which will be run by the employees of the flight booking agency. The system must be able to handle vast amounts of data and be able to store them into a database. This requires knowledge in both c++ programming and SQL/database management. Our team consists of members who are able to work such a system.

3.2. Economic Feasibility

Before anything can be started, the project requires time and effort to analyse proficiently. Once that is done, a full-scale design must be implemented before the programmers get to work. Any mistakes or set-backs in both schedule and/or progress of the project will increase the cost of the project.

If at any time throughout the implementation, evolution or maintenance stages of the project it is found that the design had been poorly or incorrectly done, the analysts and designers must sit back down and either redesign broken aspects or recreate an entirely new system.

The reason for implementing this new system is that it will be a much faster and secure system. It will allow for easier evolution and maintenance of the system in the future. This reduces any future potential expenses, because only the portion that requires change in an already well-developed system needs to be analysed, designed and implemented, as opposed to starting a new project from scratch.

3.3. Organisational Feasibility

Organisational feasibility studies are important in identifying the primary stakeholders of a project. The stakeholders of a project each have a different level of influence on how the project proceeds. Thus we need to identify who will have the most significant influence/importance on the project and find the best way to communicate and satisfy their needs.

The provided stakeholder matrix identifies the primary stakeholders of the project, and categorises them based on their level of 'importance' and level of 'influence'. Importance is identified as the level of 'stake' a person has in the success of the system. 'Influence' is identified as how much their opinion matters in the development of the system.

	Unknown	Little Importance	Some Importance	Significant importance
Significant Influence		Analyst/Designer	Project Manager	Company Owner
Some Influence		Customer	Staff and Management	
little influence	Tester	Programmer		
Unknown				

[1]

This matrix identifies the company owner to be of the most importance and influence to the project's success. We have established constant communications with the company owner via meeting and email correspondence in the hopes of gaining his approval of the design and functionality of the system. Approval of the Company owner is most important as it will ensure that the software we create will be integrated and used successfully by the organisation, the staff employed by the company, and the customers the company is hoping to engage.

3.4. References

- Stakeholder Analysis (Stakeholder Matrix), 26/04/2013. Available from: <<http://www.dse.vic.gov.au/effective-engagement/toolkit/tool-stakeholder-analysis-stakeholder-matrix>>.

4. Risk Analysis and Management Plan

4.1. Executive Summary

The purpose of this document is to provide a management framework to ensure that levels of risk and uncertainty are properly managed for the remainder of the project. As risk management is an ongoing process over the life of a project, the Risk Register must be considered a 'snap shot' of relevant risks at one point in time.

This document will achieve this by defining the following:

- The process that will be/has been adopted by the project to identify, analyse and evaluate risks during the remainder of the project.
- How risk mitigation strategies will be developed and deployed to reduce the likelihood and/or impact of risks.
- How often risks will be reviewed, the process for review and who will be involved.
- Roles and responsibilities for risk management.
- How reporting on risk status and changes to risk status will be undertaken within the project and to the Steering Committee.
- A complete risk register, containing all risks identified for the project, their current gradings and the identified risk mitigation strategies to reduce the likelihood and seriousness of each risk.

4.2. Introduction

The purpose of risk management is to ensure levels of risk and uncertainty are identified and then properly managed in a structured way, so that any potential threat to the delivery of outputs (level of resourcing, time, cost and quality) and the realisation of outcomes/benefits by the business owner is appropriately managed to ensure the project is completed successfully.

The objectives of the risk management approach in the Flight Booking Management System Project are to identify, assess and mitigate risks where possible and to continually monitor risks throughout the remainder of the project as other risks or threats emerge or a risk's impact or likelihood changes.

As risk management is an ongoing process over the life of a project, this Risk Management Plan and Risk Register must be considered a 'snap shot' of relevant risks at one point in time.

Where required, the process of risk identification, assessment and the development of countermeasures will involve consultation with the Steering Committee members, the Flight Booking System Reference Group, other relevant stakeholders and the Project Team members.

4.3. Risks

4.3.1. Loss of Team Members

At any moment throughout the project, the team could lose members due to many possible reasons, such as dropping the subject, leaving university, leaving the country, death, world war 3, etc.

In such a case where one member of the team leaves, the rest of the team will have to do a bit more work to cover for the lost member. If 2 or more were to leave the team, that's where it would become a serious problem in regards to brain power and working power.

We have one Manager, one Chief Programmer, one Chief GUI Designer/Tester and two Analysts/Designers. We have already lost one member of the team, but everyone agreed to do a bit more work and shift around to the areas that needed more work. If we were to lose one Analyst/Designer, it wouldn't be critical, but developing a perfect system will become much harder as the workload will be doubled for the Analyst/Designer. If the GUI Designer were to leave, since it is a specialised field, another member would need to spend time and effort to learn it. If that is the case, we would most probably not develop a GUI unless the development was well ahead and we had the time to do so.

If the Tester were to leave, the Programmer could make notes of all the aspects in the system that need to be tested and hand them out to the rest of the team to work on. If the Programmer were to leave, since they are our strongest programmer, it will hurt our development quite a lot, but everyone else would have to work together in the programming to get the job done. If the manager were to leave, the documentation, organising and scheduling has already been done, so for another member to pick up that role wouldn't be too much of a challenge.

In order to try and minimise any reason people may have for leaving the team, weekly meetings are held, the team is managed as organised as possible. Each team member must feel comfortable and motivated to work on this project.

We cannot prevent a member leaving the project, but we can prevent the loss of the work they contribute to the project with information sharing. During each meeting all information created in the previous week is shared with the group. This way in the event a member leaves, all his/her work with the project is not lost as we have a record of their work.

4.3.2. Loss of Access to Technology

It is possible that there are technical problems that occur, whether it be the hardware dies, a fire destroys the entire office or something else. In those situations, development will be required to halt until the necessary funding and/or replacement technology/equipment is obtained. This may set the project back quite a long time depending on the severity of the incident. If all of our access to the required technology/equipment for this project was to disappear, we would not be able to do anything until the client supplies us with new technology/equipment. This will also bring in additional project costs, which if get too big for the budget or scope of the project, will have the project terminated.

4.3.3. Loss of Documentation/Code

Just like with the loss of access to technology, if documentation and/or code were to be lost, the project would most likely be set back much further since it cannot be replaced as easily with a sum of money. The corruption of files and or human error which results in the misplacement/deletion of files must be re-done either from scratch or the latest available checkpoint/version.

To help minimise these risks, all of the files that are worked on are uploaded to a github server which each team member has access to. Each member also has a copy of the server downloaded on their home desktops/laptops. There are even more copies of files stored on other data storage devices such as USBs. Therefore, in the case of the loss of documentation and/or code, the time and work lost will be marginally minimised and easily re-done.

4.3.4. Loss of Client

In the case that the client disappears due to family issues, health issues or other, there will no longer be a project to do. The worst-case scenario would be that no one a part of the project will receive anything. The entire project will have been a waste of time, money and effort.

In order to help minimise this, a weekly meeting is scheduled with the client with well-planned questions and deliverables to present to them. If the client feels confident in the project team, their will to work with us will increase. The weekly meetings also give us a better sense on whether the client is serious about this project and whether they will go through with it all the way with us or not. Ensuring that we fulfil all of the client's requirements and goals is our most important task, as in the end, the project is alive and we are on the project team because of the client.

4.3.5. Loss of Funding

Similar to the loss of a client, without any funding, the project will have nowhere further to go and the team will halt the project. The loss of funding could happen for one of many reasons. Some of those reasons may be that the client becomes bankrupt or decides to abandon important portions if not the entire project due to budget and financial instability.

If this were to happen, all the time and effort spend on the project would be null and void. Although this situation is not entirely under our control, there are factors which could help minimise the chances even a little bit.

It is up to the Analysts and Designers to understand what is required, plan and design the most efficient possible way to advance the project. A thorough and optimised design/plan will bring a higher success rate to the project and minimise the chances of financial failure.

4.3.6. Suspension of the Project

It is possible that the project gets suspended if not cancelled due to one of many reasons as mentioned in the loss of the client or the loss of funding. Ultimately, this is not something anyone a part of the team wants to occur at any moment throughout the project. It could happen at the beginning or towards the end, in some cases being worse than others. The goal is to see the project through to the end.

The suspension of the project will make all the time and effort spent up until that point completely wasted. The team will have nothing further to do and depending on the reason and the way the project got suspended or cancelled, the team may never receive their pay or work certificate as opposed to just losing time and effort.

The only way the project team could minimise the chances of the project being suspended or terminated is to make everyone involved confident and keep their morale high. This is done with a well-done analysis, which leads to the planning, design and then implementation of the project. Weekly meetings are conducted between both the client and project team, as well as a meeting for the project team themselves.

The client meeting acts as a way for the project team to suggest ideas and show prototypes while receiving the requirements of the client to make sure both parties agree to what is being worked on. The project team meetings are held to discuss any issues and to set tasks on what has to be done as well as how. The meetings put everyone involved in the project at the same level when it comes to the progress and status of the project.

4.3.7. Changing of Requirements

The project is susceptible to constant change, and that is expected to happen throughout the life of this project. The flow of the project greatly depends on how these changes come into play. If the base requirements were established at the beginning and the project was planned out well, then small changes and added specifications to the requirements will not be as great of an addition as a complete overhaul or many conflicting requirements.

The changing of requirements means that analysis, design, implementation and testing must be done constantly over the duration of the entire project. This brings the project to the issue of having more obstacles to be handled and work to be done, resulting in a higher cost and longer schedule of the project. If the requirements are too far-fetched because the client is inexperienced or lacks knowledge of how project development works, the project will be hard to work on in an efficient manner.

Therefore, frequent meetings are conducted between both the client and the project team, as well as frequent meetings between the project team themselves. The requirements can be discussed by both parties real-time, utilising good communication to clarify any options and/or issues with the requirements as soon as possible. After the client meetings conclude with a successful understanding in feasible changes, then project team goes through their own meeting to discuss and plan the technical side of the changes. Constant planning and re-design will be required to effectively handle this project.

4.3.8. Changing of Clients

Just like in the changing of the requirements, the changing of clients also leads to re-evaluation and re-designing to be needed. The changing of clients contains all the element in the changing of requirements. Although, the changing of clients usually leads to a bit more technical changes.

The changing of a client is usually done from company to company as opposed to a single, small-scale client. If the current client were to pass on or sell the project to another person or company, they may have a different view and goal to the previous client. This leads to further changes in the requirements of the project. Although, with the changing of clients, it may not always be the same case that they have the same budget and/or technology as the previous client. This could be a good or bad aspect of client change.

If the new client has broader and greater views of the future of the implemented project, that may be the reason why they have taken over the project and are able to fund it with a greater budget, greater technology and crew. This would then require the system to be re-designed based off of new requirements, whether it be technical or the client's.

However, if it happens to be the opposite and instead, the new client is has a smaller view and goal for this project, negative changes could be put into place. Also, with the changing of clients, the project team does not have as easy a choice of deciding to work on the project with the client or not as it's already underway. Since the new client may desire a complete overhaul of the system to be designed, this could lead to the changing of the team as well. Team members may be removed from the project team and this would go back to the problems of the loss of team members.

In the chance that the client were to change, the best approach would be to immediately establish a client meeting and brief them on the entire project. The point is to show them what has been done and that the analysis of and planning done on the project are effective and do not need to be changed. As it is a new client, they may be inexperienced and unknowledgeable about what they may want, so they must be reassured with the future of the system we have designed.

4.3.9. Maintenance

After the project has been completed as per the requirements of the client, they may still desire future upgrades to the system. In the case where maintenance is required, whether it be from the project team or a future team, the design of the system must be developer-friendly.

Maintenance could involve the fixes of problems which are encountered while the system is implemented and live, but assuming our design was done properly, this will not be a problem except for in specific cases which were purposely disregarded based on the feasibility studies of the project. For example, in the future, all computers may run a 128-bit operating system or all executables get a complete structure change. These are not factors that will be an issue in the near future, so designing the system to fit such different requirements is not the most efficient design available.

In terms of the maintenance of desired upgrades in the near future, which could come from the client's new visions or feedback of customers, the system must be easy to understand and work on down the track. This requires all documentation, plans, sketches, user manuals and code to be well-done. Well-done documentation is easy to understand, consistent and well-formatted.

As for the code to allow for easy amendment in the future, the code is detailed and legible. This is achieved through the documentation of the system, such as diagrams and a data dictionary. The code itself also allows for commenting, which allows for complete English to be used to describe what parts of the system do exactly what. It eases the work required of any programming in the future, especially if done by another person who has never seen the system before.

4.3.10. Over-Evolution of the System

Similar to maintenance, the issue of over-evolution or lacking capabilities of the system can be answered by a different design, but it would have been less feasible. However, this issue could also come from straight-out poor planning and design. An example issue is that in 5 years time, the company using the system may have been so successful, that their infrastructure and/or the system is unable to cope with the amount of data being processed.

The data requirements or executable structure may change in the future, causing a lack of processing ability by the system to operate with such types or amounts of data. There may be a new language required or the database may run out of space, or the system may not be able to handle that much memory.

The measures taken against this are done with feasible and near-future ideals set in mind. The system is tested throughout its development lifecycle and is ensured to be stable and efficient for the requirements at the time of implementation. Once again, the code is designed in such a way, that should this issue ever need to be solved in the future, it can be done so as easily as possible.

4. References

http://www.egovernment.tas.gov.au/_data/assets/pdf_file/0004/78367/Realising_Project_Benefits_Project_Risk_Management_Plan.pdf

(Accessed: Thursday, 26/03/2015, 3:20pm)

5. System Requirements Specification

5.1. Introduction

5.1.1. Purpose

The purpose of this document is to specify the functional and non-functional requirements of an electronic flight management system. This document aims to be a comprehensive overview of the system that illustrates requirements, constraints and the external factors that could affect the system. This document will be useful for the system development team and potential end users.

5.1.2. Scope

The system to be constructed will be a 'Flight Management System'. Three categories of users will utilise this system: Customers, staff members and travel agents.

Staff members are further broken down into 6 categories:

- **Staff:** A generic category that will handle many customer related tasks.
- **Booking manager:** A staff member who handles several more complex tasks relating to the flight bookings of customers.
- **Flight Manager:** A staff member who handles tasks relating to the management of airport information, aircraft information, route information, scheduling information and flight information.
- **Profile manager:** A staff member who handles tasks relating to the creation and modification of customer and staff profiles.
- **Service Manager:** A staff member who handles tasks relating to the available flight service menus and service item information (e.g. food and drink services.).
- **Administration:** A staff member who has permission to perform all staff and management tasks. Also performs tasks relating to staff member creation and permissions.

In order to properly manage the full functionality, flight management system is comprised of 5 sub-systems:

- **Reservation system:** This system handles all functionality relating to the booking process. All flight searching and flight selection options are handled here.

- **Profile System:** This system handles all customer profile creation and modifications as well as travel agent profile creation and modifications.
- **Flight System:** This system handles all tasks relating to the creation and modifications of airport information, aircraft information, route information, scheduling information and flight information.
- **Report system:** This system handles all report creation tasks. Information is obtained from all other sub-systems and compiled to form reports for management level staff members.
- **Service System:** This system handles the creation and modification of flight services information and service item information (e.g. food and drink services.).

5.1.3. References

- Volere Requirements template. Available from <http://www.volere.co.uk/template.htm>
- Sample SRS - E-Library System Software Requirements Specifications, Provided by Hoa Dam. Available from: <https://moodle.uowplatform.edu.au/mod/folder/view.php?id=218934>

5.2. Overview/Description

5.2.1. Product Perspective

Currently all booking and management activities occur manually. This product aims to create a more efficient method of performing these actions, and in turn saving the company time, money and effort.

5.2.2. System interfaces

The system is self-contained. No external systems are required for operation. The database system has been custom built specifically for this program based on text file access, and is contained within the system.

Deployment will require the program to be loaded on a windows or linux based machine within the company before operation.

5.2.2.1. User Interfaces

The user interfaces to be provided to customers must be a text based console interface.

User interfaces given to customers provide different options than other user interfaces.

Customers have the lowest system access rights in the system.

The user interface provided to staff members must be a text based console interface. The UI here provides many of the same options provided to the customer. However, some additional options must be provided for the staff member to effectively perform their jobs (e.g. search for customers, edit bookings).

The user interfaces provided to management level staff members must be a text based console interface. The UI here will give the user options specific to the tasks each manager needs to perform. Primarily it will allow the manager access to their respective sub-system.

The user interface provided to the Administration staff must be a text based console interface. The UI here will provide full access to all staff and management functionalities, as well as some extra privileges such as the ability to grant access to staff members.

The user interface provided to the travel agents must be a text based console interface. The UI here will provide access to functionality travel agents need to perform their role.

5.2.2.2. Hardware Interfaces

The program must be able to execute on any windows or linux based pc within the company.

5.2.2.3. Software Interfaces

The system is self-contained. No external software is required for operation.

5.2.2.4. Communication Interfaces

The system is not web enabled, nor does it include client/server architecture. No Communication is required outside of the system.

5.2.2.5. Memory constraints

No memory constraints are strictly defined. However, the program needs to run as efficiently as possible, and perform tasks in a prompt manner.

5.2.2.6. Operations

The Flight Management System user interfaces must be easy to use. Only a minimum amount of training should be needed to perform actions with the system. No specific skills should be needed to utilise the system.

Installation should be simple enough for the system administrator to handle. No additional training should be required to set up the system.

The database should import and store information from text files. Also it should be robust enough to handle improper shut down cases (such as power loss) as effectively as possible.

5.2.3. Product Functions

One of the main functions of the Flight Management system is to manage the flight company's flight information and provide a means for users to utilise, create, delete and modify the information.

A customer should be able to use these primary functions:

- Book flights for themselves or others. A booking consists of a Customer, flight, route, services, seating, price, and arrival/departure times.
- View booked flights
- Manage their own personal details
- Manage their own payment details

A staff member should be able to use these primary functions:

- Book flights for a customer. A booking consists of a Customer, flight, route, services, seating, price, and arrival/departure times.
- View customers booked flights.
- View a customer's personal details.
- Modify a customer's personal details
- Modify a customer's booking details
- Search for a customer.

A Booking manager should be able to use these functions:

- Modify a customer's flight details.
- Modify customers seating if destination seat is occupied.
- Set customers no-fly/watch status.
- View system reports relating to the reservation system.

A Service Manager should be able to use the following functions:

- Modify service items details.
- Add new service items to the database.
- Add new flight services to the database.
- Remove service items from the database.
- Remove flight services from a flight.

- View reports relating to the service system.

A Profile Manager should be able to use the following functions:

- Modify personal details of a customer.
- Modify a customer's frequent flier points.
- View reports relating to the profile system.
- Modify travel agents details.
- Modify staff details.
- Give discounts to travel agents.
- Give discounts to customers.

A Flight Manager should be able to use these functions:

- Add/remove aircrafts, schedules, airports, routes, and flights.
- Modify aircrafts, schedules, airports, routes, and flights.
- View system reports relating to the flight system.

An Administrator should be able to perform these functions:

- Access all staff functionalities.
- Access all management level functionality.
- Grant access to staff members.

A Travel agent should be able to perform these functions:

- Must be able to book flights for customers.
- Must be able to edit own details.
- View all bookings made for customers.
- Modify existing bookings made for customers.

All user functionalities have been defined from client meeting requirements elicitation so that the Flight Management System can perform all tasks required by the client.

5.2.4. User Characteristics

- Staff members have good knowledge about the tasks and processes of customer activities
- Booking managers have a high level knowledge in the tasks and processes of customer booking flights.
- Profile managers have high level knowledge of the tasks and processes of system profiles and their management.
- Service Managers have high level knowledge of the tasks and processes of the service information system.
- Flight Managers have a high level knowledge of the tasks and processes of the flight management system.
- Administrators have good knowledge of all tasks and processes of all system users and sub systems. Also has knowledge of system installations.
- Travel agents have good knowledge of the tasks and processes of booking customers.

5.2.5. Constraints

The system should obey the following constraints:

- User Authentication: For a user to book a flight or for a staff member to use the system, The system should require a log In and authentication process.
- All users should have appropriate permissions when accessing the system.
- Database must be robust. It should be easy for users to save/backup the database to prevent corruption/data loss.
- Database integrity. The Flight management system involves several sub-systems that access the database. It is imperative that these sub-systems interface properly to maintain database integrity.
- System must be documented, developed and released by the end of may 2015.

5.2.6. Assumptions and Dependencies

The following assumptions and dependencies are defined:

- All customers must have an email address.
- All Travel agents must have an email address.

5.3. Specific Requirements

Each requirement has a priority level associated with them. Priority levels are defined as:

- Critical – Highest level. Defines core functionality.
- Essential – Secondary to critical. Should be implemented immediately after core functionality has been implemented.
- Low – Non essential functions. System can operate without these functions. Should be implemented after
- Optional – lowest importance priority. It would be desirable but not necessary to implement.

5.3.1. User Management Functions

This section defines functionalities provided for customers, staff members, travel agents and Administrators.

5.3.1.1. Customer Functions

This section includes all functions that a customer can use to perform required tasks.

Requirement# F_1 **Requirement Type:** Functional **Event/Use Case:** N/A

Description: The system should provide a text based user interface for an unregistered user to search for flights. Search criteria consists of a date and a route.

Rationale: A guest user wants to search for flights.

Originator: Guest

Fit Criterion: Guest searches for flights

Dependencies: none

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_2 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for an unregistered user to register to the system. Registration consists of Personal details and login details. Personal details consist of title, first name, last name, gender, date of birth, phone number, email, address, state, country, credit card details, and passport details. Login details consist of a username and password.

Rationale: A guest user wants to register to the system

Originator: Guest

Fit Criterion: Guest registers to the system

Dependencies: none

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_3 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a guest user to log into the system. Log in details consist of a username and password.

Rationale: A guest user wants to log into the system

Originator: Guest

Fit Criterion: Guest logs in to the system

Dependencies: The log in details of the user must exist

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_4 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a customer to search for flights. Search criteria consists of a date and a route. The system should display a list of available flights from the given start date to 7 days after the start date

Rationale: A customer wants to view available flights

Originator: Customer

Fit Criterion: Customer views available flights from a given date to 7 days past the given date

Dependencies: none

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_5 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a customer to book a flight with the system. A booking consists of a flight search,route, services, and seat selection.

Rationale: A customer wants to book a flight

Originator: Customer

Fit Criterion: Customer successfully books a flight.

Dependencies: The flight to be booked must exist

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_6 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a customer to view and select service options for a flight. A flight service consists of a one or more items, prices and quantity.

Rationale: A customer wants to choose services for a flight.

Originator: Customer

Fit Criterion: Customer successfully chooses flight services

Dependencies: The service to be selected must exist

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_7 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a customer to view and select seating options for a flight. A seating choice consists of a seating class and price.

Rationale: A customer wants to choose a seat on a flight

Originator: Customer

Fit Criterion: Customer successfully chooses seating.

Dependencies: Seating to be chosen must be available

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_8 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a customer to allow the system to automatically allocate seating for a flight. A seating choice consists of a seating class and price.

Rationale: A customer wants to the system to automatically allocate seating.

Originator: Customer

Fit Criterion: System allocates seating for customer.

Dependencies: The seat to be selected must exist

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_9 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a customer to allow the customer to add their credit card details. Credit card details consists of a full name, card number, and expiry date.

Rationale: A customer wants to add credit card details to their profile.

Originator: Customer

Fit Criterion: Customer successfully adds credit card details.

Dependencies: The customer profile to be edited must exist

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_10 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a customer to revise and modify booking details before confirming the booking. A booking consists of a flight search, route, services, seat selection and payment details.

Rationale: A customer wants revise or modify booking details before confirmation.

Originator: Customer

Fit Criterion: Customer successfully revises and modifies booking details.

Dependencies: The customer profile to be edited must exist

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_11 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface with the option for a customer to book a returning flight. A returning flight option will add extra flight selections, service selections and seating selections during booking.

Rationale: A customer wants to book a returning flight

Originator: Customer

Fit Criterion: Customer successfully books a returning flight.

Dependencies: A departing flight must already exist

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_12 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a customer to view all bookings they have made. A booking consists of a flight, route, services, seating, price, and arrival/departure times.

Rationale: A customer wants to view all booking they have made.

Originator: Customer

Fit Criterion: Customer successfully views previous bookings.

Dependencies: Customer bookings must already exist

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_13 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a customer to set and modify their own personal details. Personal details consist of title,first name, last name, gender,date of birth, phone number, email, address,state,country, credit card details, and passport details.

Rationale: A customer wants to set or modify personal details.

Originator: Customer

Fit Criterion: Customer successfully sets or modifies personal details.

Dependencies: The customer profile to be edited must exist

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_14 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a customer to view their own personal details. Personal details consist of title,first name, last name, gender,date of birth, phone number, email, address,state,country, credit card details, and passport details.

Rationale: A customer wants to view personal details.

Originator: Customer

Fit Criterion: Customer successfully views personal details.

Dependencies: The customer profile to be viewed must exist

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_15 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a customer to cancel any bookings they have made. A cancelled flight incurs a cancellation fee to the customer.

Rationale: A customer wants to cancel a booked flight.

Originator: Customer

Fit Criterion: Customer successfully cancels a booking.

Dependencies: the booking to be deleted must exist

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_16 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a customer to close their account. A close of account will remove their login privileges. Customer details such as no-fly status must still be kept.

Rationale: A customer wants to close their account.

Originator: Customer

Fit Criterion: Customer successfully closes their account.

Dependencies: The account to close must exist.

Priority: Essential.

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_17 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a customer to book a flight for another person. All detail about the customer the booking is for must be entered. Customer details consist of title,first name, last name, gender,date of birth, phone number, email, address,state,country, credit card details, and passport details.

Rationale: A customer wants to book a flight for another person

Originator: Customer

Fit Criterion: Customer successfully books a flight for another person.

Dependencies: none.

Priority: Essential.

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_60 Requirement Type: Functional Event/Use Case: N/A

Description: The system should alert a booking manager if a 'watch' no fly status customer books a flight.

Rationale: The booking manager can take appropriate action after being alerted to the customers booking.

Originator: Customer

Fit Criterion: Booking manager is successfully alerted when 'watch' status customer books a flight.

Dependencies: Customer with 'watch' status must exist, booking manager must exist.

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_66 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface that allows the customer to choose a customer age category when booking. Customers should need to specify either 'Adult' or 'Child' when navigating the booking procedure.

Rationale: Ticketing requires a specification of customer age.

Originator: Customer

Fit Criterion: Customer successfully specifies customer age while booking.

Dependencies: Customer who is booking must exist.

Priority: essential

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

5.3.1.2. Travel Agent Functions

This section includes all functions that a Travel Agent can use to perform required tasks.

Requirement# F_18 **Requirement Type:** Functional **Event/Use Case:** N/A

Description: The system should provide a text based user interface for a Travel agent to book a flight for a customer. A booking consists of a flight search,route, services, and seat selection. Customer personal details must also be obtained during booking, Personal details consist of title,first name, last name, gender,date of birth, phone number, email, address,state,country, credit card details, and passport details.

Rationale: A travel agent wants to book a flight for a customer.

Originator: Travel Agent

Fit Criterion: Travel agent successfully books a flight for a customer.

Dependencies: Flight to be booked must exist

Priority: Critical.

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_19 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Travel agent to modify their profile details. Travel agent details consist of name, phone number and email.

Rationale: A travel agent wants to edit profile details.

Originator: Travel Agent

Fit Criterion: Travel agent successfully modifies profile details.

Dependencies: Profile to be modified must exist

Priority: Critical.

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_20 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Travel agent to log into the system. Log in details consist of a username and password.

Rationale: A travel agent wants to log into system.

Originator: Travel Agent

Fit Criterion: Travel agent successfully logs into system

Dependencies: Travel agent login details must exist in system.

Priority: Critical.

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_21 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Travel agent to view all bookings made from its profile. A booking consists of a Customer, flight, route, services, seating, price, and arrival/departure times.

Rationale: A travel agent wants view bookings made from its profile.

Originator: Travel Agent

Fit Criterion: Travel agent successfully views bookings

Dependencies: Bookings to be viewed must exist.

Priority: Critical.

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_22 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Travel agent to modify bookings made from its profile. A booking consists of a Customer, flight, route, services, seating, price, and arrival/departure times.

Rationale: A travel agent wants modify bookings made from its profile.

Originator: Travel Agent

Fit Criterion: Travel agent successfully modifies bookings

Dependencies: Bookings to be modified must exist.

Priority: Critical.

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_23 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Travel agent to modify customer's personal details that have registered to the system through a travel agent. Personal details consist of title, first name, last name, gender, date of birth, phone number, email, address, state, country, credit card details, and passport details.

Rationale: A travel agent wants modify a customers personal details.

Originator: Travel Agent

Fit Criterion: Travel agent successfully modifies customers personal details.

Dependencies: Customers profile to be modified must exist.

Priority: low

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

5.3.1.3. Staff Functions

This section includes all functions that a staff member can use to perform required tasks.

Requirement# F_24 **Requirement Type:** Functional **Event/Use Case:** N/A

Description: The system should provide a text based user interface for a staff member to log into the system. Log in details consist of a username and password. Staff members also have an associated role.

Rationale: A staff member wants to log into the system.

Originator: Staff

Fit Criterion: Staff member successfully logs into the system.

Dependencies: Staff member log in details must exist in the system.

Priority: Critical.

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_25 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a staff member to create a booking for a customer. A booking consists of a Customer, flight, route, services, seating, price, and arrival/departure times.

Rationale: A staff member wants to create a booking for a customer.

Originator: Staff

Fit Criterion: Staff member successfully creates a booking for a customer.

Dependencies: Customer profile to be booked for must exist. Flight to be booked for must exist.

Priority: Critical.

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_26 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a staff member to create an account for a customer. A customer account personal details consist of title,first name, last name, gender,date of birth, phone number, email, address,state,country, credit card details, and passport details. Log in details consist of username and password.

Rationale: A staff member wants to create an accountfor a customer.

Originator: Staff

Fit Criterion: Staff member successfully creates an account for a customer.

Dependencies: none.

Priority: Critical.

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_27 **Requirement Type:** Functional **Event/Use Case:** N/A

Description: The system should provide a text based user interface for a staff member to view a customer's personal details and login details. Personal details consist of title,first name,last name, gender,date of birth, phone number,email, address,state,country, credit card details,and passport details. Log in details consist of username and password.

Rationale: A staff member wants to view a customers personal details.

Originator: Staff

Fit Criterion: Staff member successfully views a customers personal details.

Dependencies: Customer to be viewed profile must exist.

Priority: Critical.

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_28 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a staff member to view a customers Booking details. A booking consists of a Customer, flight, route, services, seating, price, arrival/departure times and agency name.

Rationale: A staff member wants to view a customer's booking details.

Originator: Staff

Fit Criterion: Staff member successfully views a customer's booking details.

Dependencies: Customer to be viewed profile must exist. Customer to be viewed bookings must exist.

Priority: Critical.

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_29 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a staff member to modify a customer's personal details. Personal details consist of title, first name, last name, gender, date of birth, phone number, email, address, state, country, credit card details, and passport details. Log in details consist of username and password.

Rationale: A staff member wants to modify a customer's personal details.

Originator: Staff

Fit Criterion: Staff member successfully modifies a customer's personal details.

Dependencies: Customer to be modified profile must exist.

Priority: Critical.

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_30 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a staff member to modify a customer's Booking details. A booking consists of a Customer, flight, route, services, seating, price, arrival/departure times and agency name.

Rationale: A staff member wants to modify a customer's booking details.

Originator: Staff

Fit Criterion: Staff member successfully modifies a customer's booking details.

Dependencies: Customer to be accessed profile must exist. Customer to be modified bookings must exist.

Priority: Critical.

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_31 **Requirement Type:** Functional **Event/Use Case:** N/A

Description: The system should provide a text based user interface for a staff member to search for a customer's profile. Search criteria consist of a customer's email address. A customer's profile consists of personal details and booking details. Personal details consist of title, first name, last name, gender, date of birth, phone number, email, address, state, country, credit card details, and passport details. Booking details consist of a Customer, flight, route, services, seating, price, and arrival/departure times.

Rationale: A staff member wants to search for a customer's profile.

Originator: Staff

Fit Criterion: Staff member successfully searches for a customer's profile.

Dependencies: Customer to be searched profile must exist.

Priority: Critical.

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_32 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a staff member to modify a customers seating details in a booking. Seating details consist of a class type and seat number. Seating can only be modified if destination seat is unoccupied.

Rationale: A staff member wants to modify a customer's seating details in a booking.

Originator: Staff

Fit Criterion: Destination seat is unoccupied. Staff member successfully modifies seating details of a booking.

Dependencies: Booking to be modified must exist. Destination seat must be unoccupied.

Priority: Essential

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

5.3.1.4. Booking Manager Functions

This section includes all functions that a staff member can use to perform required tasks.

Requirement# F_33 **Requirement Type:** Functional **Event/Use Case:** N/A

Description: The system should provide a text based user interface for a booking manager to modify a customer's booking details. A booking consists of a Customer, flight, route, services, seating, price, and arrival/departure times.

Rationale: A Booking Manager wants to modify a customer's booking details.

Originator: Booking Manager

Fit Criterion: Booking Manager successfully modifies a customer's booking details.

Dependencies: Booking to be modified must exist.

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_34 **Requirement Type:** Functional **Event/Use Case:** N/A

Description: The system should provide a text based user interface for a booking manager to modify a customer's seat allocation for a booking. The booking manager has the permissions to modify seating even if the destination seat is occupied. Seating consists of a flight, seat number and seat class.

Rationale: A Booking Manager wants to modify a customer's seating details for a booking.

Originator: Booking Manager

Fit Criterion: Booking Manager successfully modifies a customer's seating details for a booking.

Dependencies: Booking to be modified must exist.

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_35 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a booking manager to modify a customer's no-fly status. A not fly status can be either null, 'watch' or 'no-fly'.

Rationale: A Booking Manager wants to modify a customers no fly status.

Originator: Booking Manager

Fit Criterion: Booking Manager successfully modifies a customers no fly status.

Dependencies: Customer to be modified must exist.

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_36 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a booking manager to access the report system to view reports relating to the reservation system. A report is a computation or compilation of informative data produced from the system.

Rationale: A Booking Manager wants to view reservation reports.

Originator: Booking Manager

Fit Criterion: Booking Manager successfully views reservation reports.

Dependencies: none

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

5.3.1.5. Service Manager Functions

This section includes all functions that a staff member can use to perform required tasks.

Requirement# F_37 **Requirement Type:** Functional **Event/Use Case:** N/A

Description: The system should provide a text based user interface for a Service Manager to modify service items details. A service item consists of an item, cost and availability.

Rationale: A Service Manager wants to modify service item details.

Originator: Service Manager

Fit Criterion: Service Manager successfully modifies a service item.

Dependencies:Service item to be modified must exist.

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_38 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Service Manager to add new service items. A service item consists of an item, cost and availability.

Rationale: A Service Manager wants to add a new service item.

Originator: Service Manager

Fit Criterion: Service Manager successfully adds a new service item.

Dependencies: none.

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_39 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Service Manager to add new flight services to a flight. A flight service is a list of service items that are required for a flight. A flight service consists of a flight, service item and amount.

Rationale: A Service Manager wants to add a new flight service.

Originator: Service Manager

Fit Criterion: Service Manager successfully adds a new flight service.

Dependencies: Flight to be added to must exist. Item to be added must exist.

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_40 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Service Manager to remove a service item. A service item consists of an item, cost and availability. Removal of service item will cascade removal of references to this service item.

Rationale: A Service Manager wants to remove a service item.

Originator: Service Manager

Fit Criterion: Service Manager successfully removes a service item.

Dependencies:Service item to be modified must exist.

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_41 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Service Manager to remove a flight service from a flight. A flight service consists of a flight,item and amount.

Rationale: A Service Manager wants to remove a flight service.

Originator: Service Manager

Fit Criterion: Service Manager successfully removes a flight service.

Dependencies:Flight to be modified must exist. Flight service to be removed must exist.

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_42 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Service Manager to view reports from the report system that relate to the service system. A report is a computation or compilation of informative data produced from the system.

Rationale: A Service Manager wants to view service system reports

Originator: Service Manager

Fit Criterion: Service Manager successfully views service system reports.

Dependencies:none.

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

5.3.1.6. Profile Manager Functions

This section includes all functions that a staff member can use to perform required tasks.

Requirement# F_43 **Requirement Type:** Functional **Event/Use Case:** N/A

Description: The system should provide a text based user interface for a Profile Manager to modify personal details of a customer. Personal details consist of title,first name,last name, gender,date of birth, phone number,email, address,state,country, credit card details,and passport details.

Rationale: A Profile manager wants to modify a customers profile.

Originator: Profile Manager

Fit Criterion: Profile manager successfully modifies a customers profile.

Dependencies:Customers profile to be modified must exist.

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_44 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Profile Manager to modify a customer's frequent flier points. Frequent flier points are an integer that accumulates when a customer books a flight.

Rationale: A Profile manager wants to modify a customers frequent flier points.

Originator: Profile Manager

Fit Criterion: Profile manager successfully modifies a customers frequent flier points.

Dependencies: Customers profile to be modified must exist.

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_45 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Profile Manager to modify a travel agents profile. A travel agents profile consists of a name, phone number and email address.

Rationale: A Profile manager wants to modify a travel agents profile.

Originator: Profile Manager

Fit Criterion: Profile manager successfully modifies a travel agents profile.

Dependencies: Travel agents profile to be modified must exist.

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_46 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Profile Manager to modify a staff members details. Staff details consist of a username and a password.

Rationale: A Profile manager wants to modify a staff member's profile.

Originator: Profile Manager

Fit Criterion: Profile manager successfully modifies a staff member's profile.

Dependencies:Staff members profile to be modified must exist.

Priority: Essential

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_47 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Profile Manager to give discounts to customers. This discount could be due to sufficient frequent flier points, or due to purchasing large amount of flights. Discount consists of a percentage value.

Rationale: A Profile manager wants to discount a customer.

Originator: Profile Manager

Fit Criterion: Profile manager successfully discounts a customer.

Dependencies: Customers profile to be modified must exist.

Priority: Essential

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_48 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Profile Manager to give discounts to Travel agents. This discount could be due to sufficient bookings. Discount consists of a percentage value.

Rationale: A Profile manager wants to discount a travel agent.

Originator: Profile Manager

Fit Criterion: Profile manager successfully discounts a travel agent.

Dependencies:travel agents profile to be modified must exist.

Priority: Essential

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_49 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Profile Manager to view system reports relating to the profile system. A report is a computation or compilation of informative data produced from the system.

Rationale: A Profile manager wants to view profile reports.

Originator: Profile Manager

Fit Criterion: Profile manager successfully views profile reports.

Dependencies: none

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement#

F_65

Requirement Type: Functional Event/Use Case: N/A

Description: The system should assign frequent flier points automatically to customers who book flights. Frequent flier points are determined as a percentage of the price of a flight.

Rationale: Profile manager wants frequent flier points to be assigned automatically.

Originator: Profile Manager

Fit Criterion: Frequent flier points are automatically assign when a customer books a flight.

Dependencies:none.

Priority: low

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

5.3.1.7. Flight Manager Functions

This section includes all functions that a staff member can use to perform required tasks.

Requirement# F_50 **Requirement Type:** Functional **Event/Use Case:** N/A

Description: The system should provide a text based user interface for a Flight manager to add a new aircraft to the system. An aircraft consists of a name, amount in service, first class seats, business class seats, premium economy seats, economy seats and total seats.

Rationale: A Flight manager wants to add an aircraft to the system.

Originator: Flight Manager

Fit Criterion: Flight manager successfully adds a new aircraft to the system.

Dependencies:none

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_51 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Flight manager to add a new schedule to the system. A schedule consists of a flight ID, plane, route, depart time and arrival time.

Rationale: A Flight manager wants to add a schedule to the system.

Originator: Flight Manager

Fit Criterion: Flight manager successfully adds a new schedule to the system.

Dependencies: Flight ID to be added must exist, plane to add must exist, route to add must exist.

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_52 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Flight manager to add a new airport to the system. An airport consists of an id, name, city, country, IATA, Latitude, longitude, altitude, timezone, DST and tz.

Rationale: A Flight manager wants to add an airport to the system.

Originator: Flight Manager

Fit Criterion: Flight manager successfully adds a new airport to the system.

Dependencies: none

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_53 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Flight manager to add a new route to the system. A route consists of an ID, source airport, destination airport, codeshare and stops.

Rationale: A Flight manager wants to add a route to the system.

Originator: Flight Manager

Fit Criterion: Flight manager successfully adds a new route to the system.

Dependencies: Source airport must exist, destination airport must exist.

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_54 **Requirement Type:** Functional **Event/Use Case:** N/A

Description: The system should provide a text based user interface for a Flight manager to modify or remove an aircraft. An aircraft consists of a name, amount in service, first class seats, business class seats, premium economy seats, economy seats and total seats. Removal of this aircraft will cascade removal to all references in the system.

Rationale: A Flight manager wants to modify an aircraft.

Originator: Flight Manager

Fit Criterion: Flight manager successfully modifies an aircraft.

Dependencies: aircraft to be modified/removed must exist .

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_55 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Flight manager to modify a schedule. A schedule consists of a flight ID, plane, route, depart time and arrival time. Removal of this schedule will cascade removal to references in the system.

Rationale: A Flight manager wants to modify/remove a schedule

Originator: Flight Manager

Fit Criterion: Flight manager successfully modify/remove a schedule.

Dependencies: Schedule to be modified must exist.

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_56 **Requirement Type:** Functional **Event/Use Case:** N/A

Description: The system should provide a text based user interface for a Flight manager to modify or remove an airport. An airport consists of an id, name,city,country,IATA,Latitude, longitude,altitude,timezone,DST and tz. Removal of this airport will cascade removal to all references in the system.

Rationale: A Flight manager wants to modify/remove an airport.

Originator: Flight Manager

Fit Criterion: Flight manager successfully modify/remove an airport.

Dependencies:airport to modify exists

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_57 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Flight manager to modify or remove a route. A route consists of an ID, source airport, destination airport, codeshare and stops. Removal will cascade removal to all references in the system.

Rationale: A Flight manager wants to modify/remove a route

Originator: Flight Manager

Fit Criterion: Flight manager successfully modified/removed a route.

Dependencies: Route to be modified must exist.

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_58 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based user interface for a Flight manager to view system reports that relate to the flight system. A report is a computation or compilation of informative data produced from the system.

Rationale: A Flight manager wants to view flight reports.

Originator: Flight Manager

Fit Criterion: Flight manager successfully views flight reports.

Dependencies:none.

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_64 Requirement Type: Functional Event/Use Case: N/A

Description: The system should alert all customers that are booked on flights that have been cancelled. The alert should appear the first time a customer logs in after a flight has been cancelled.

Rationale: Customers need to know that their flight is no longer valid.

Originator: Flight Manager

Fit Criterion: Customers are alerted when they are booked on a cancelled flight.

Dependencies: none.

Priority: essential

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

5.3.1.8. Administrator Functions

This section includes all functions that an administrator can use to perform required tasks

Requirement# F_67 **Requirement Type:** Functional **Event/Use Case:** N/A

Description: The system should provide a text based console user interface to allow an administrator to add new staff members to the system.

Rationale: Admin wants to grant an access to a staff member.

Originator: Administrator

Fit Criterion: Administrator adds new staff member.

Dependencies:Staff member to be granted access must exist

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_68 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a text based console user interface to allow an administrator to access all staff member functionality

Rationale: Admin wants to access any functionality from staff members

Originator: Administrator

Fit Criterion: Administrator access staff functionality

Dependencies:none

Priority: Critical

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

5.3.2. System Requirements

This section includes all functions that a staff member can use to perform required tasks.

Requirement# F_61 **Requirement Type:** Functional **Event/Use Case:** N/A

Description: The system should provide a text based interface for a customer to reschedule cancelled flights. A list of closest upcoming flights on the cancelled flights route should be displayed to the customer.

Rationale: A customer wants to reschedule a cancelled flight.

Originator: Customer

Fit Criterion: Customer is books a flight to replace a cancelled flight.

Dependencies: Customer with previous cancelled flight must exist. Upcoming flight on correct route must exist.

Priority: optional

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_62 Requirement Type: Functional Event/Use Case: N/A

Description: The system should provide a GUI for all user interactions.

Rationale: A user wants to use a GUI to operate the system.

Originator: User

Fit Criterion: Customer uses a GUI to operate the system.

Dependencies:none.

Priority: optional

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# F_63 Requirement Type: Functional Event/Use Case: N/A

Description: The system shouldallow multiple users to access the system simultaneously.

Rationale: Many users want to access the system at the same time.

Originator:Un Registered User

Fit Criterion: Many customers access the system concurrently.

Dependencies:none.

Priority: optional

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

5.3.3. Non-Functional Requirements

This section includes all functions that are defined as non-functional requirements.

Requirement# NF_1 **Requirement Type:** NonFunctional **Event/Use Case:** N/A

Description: The system should operate on both Windows and Linux systems.

Rationale: Users may use system on either linux or windows architectures.

Originator: none

Fit Criterion: System operates on either windows or linux systems.

Dependencies:none.

Priority: essential

Supporting Materials: none

Volere

Copyright © Atlantic Systems Guild

Requirement# NF_2 Requirement Type: NonFunc Event/Use Case: N/A

Description: A customers login password must be displayed as astrixes during login.

Rationale: Displaying astrixes instead of text increases system security

Originator: Customer

Fit Criterion: Customers login password is displayed with astrixes.

Dependencies:none.

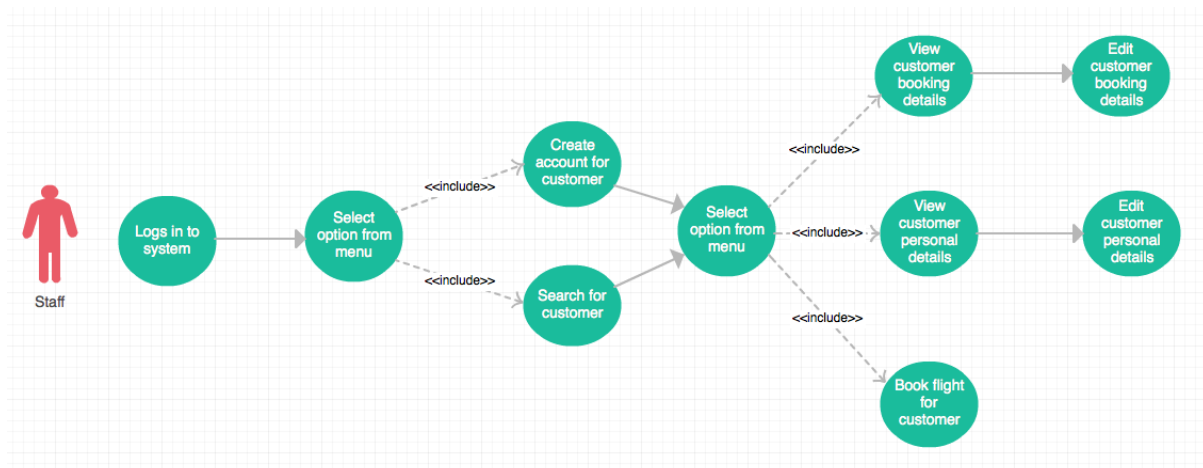
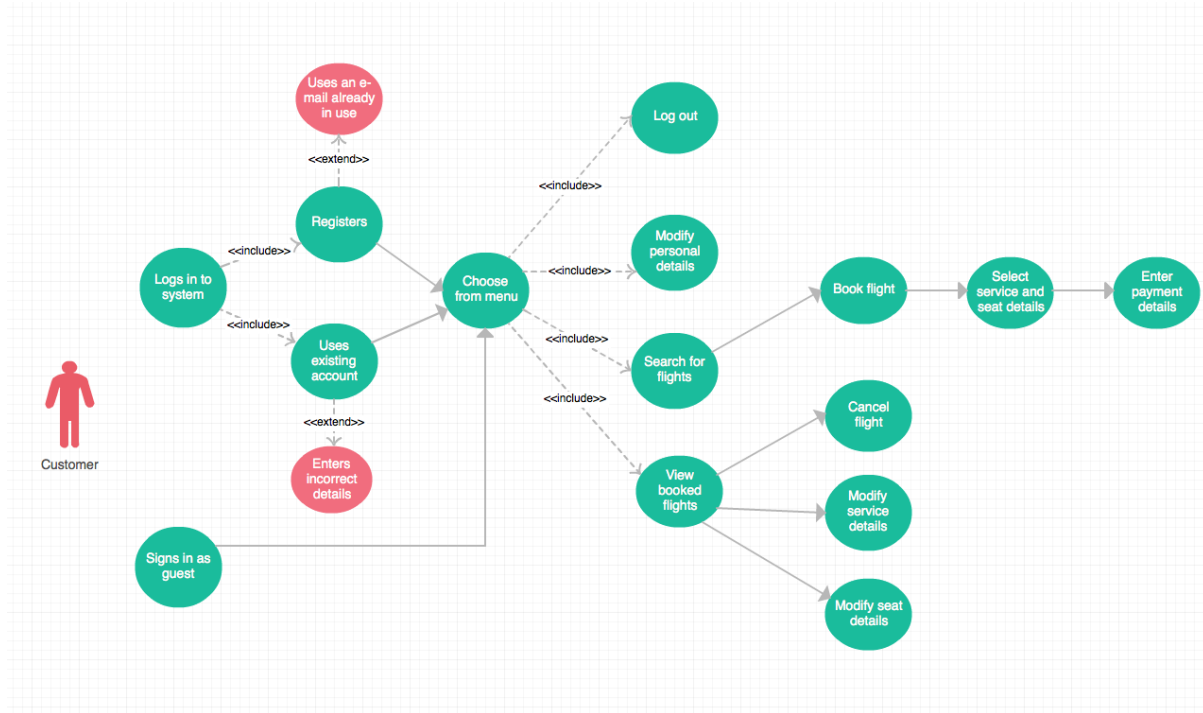
Priority: Critical

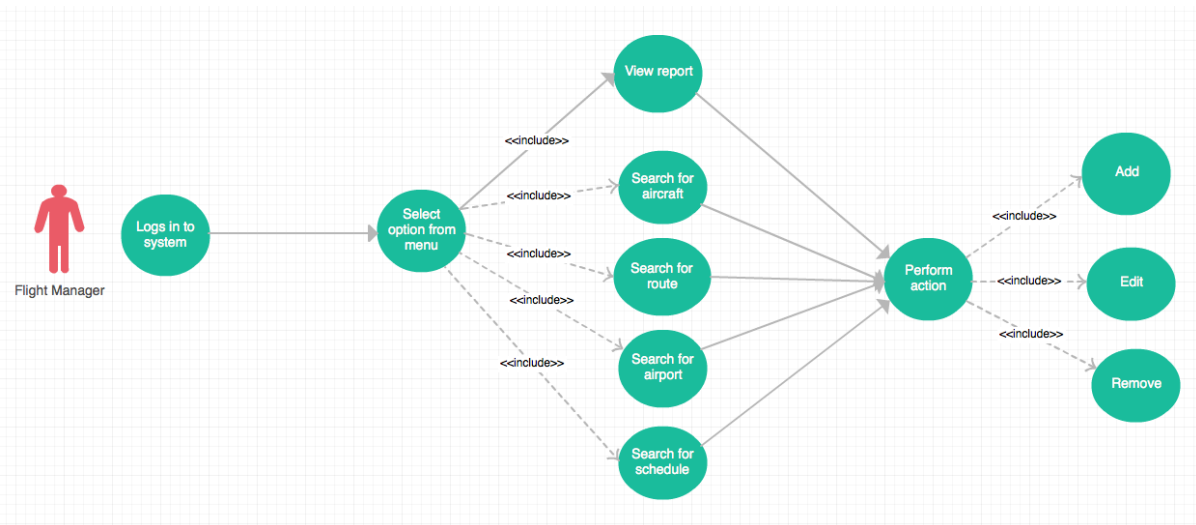
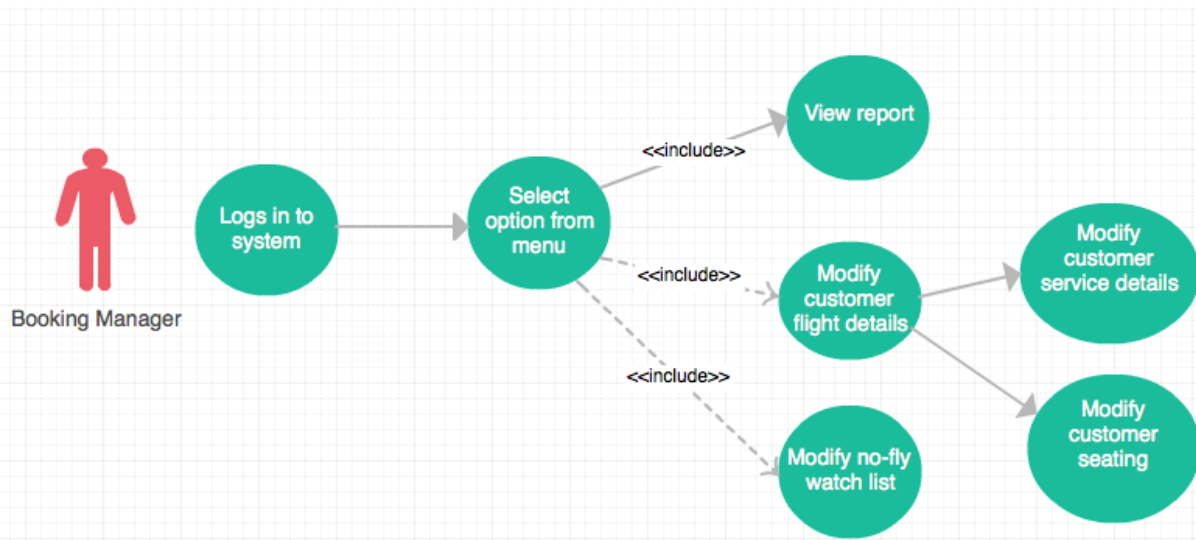
Supporting Materials: none

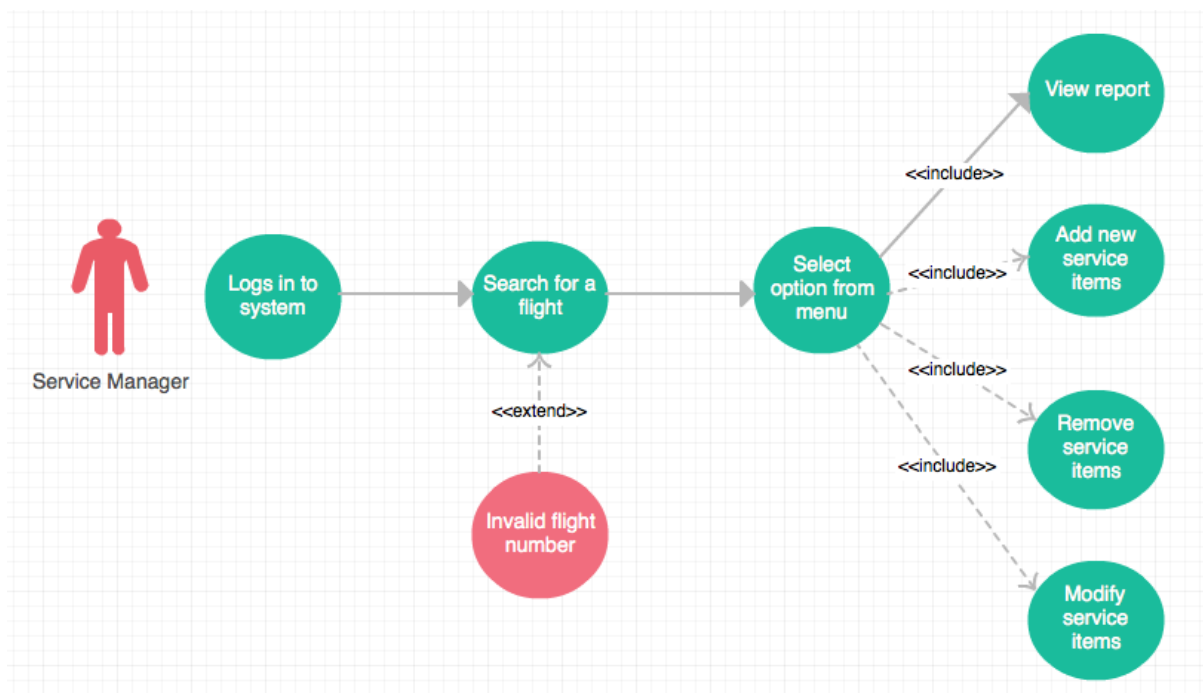
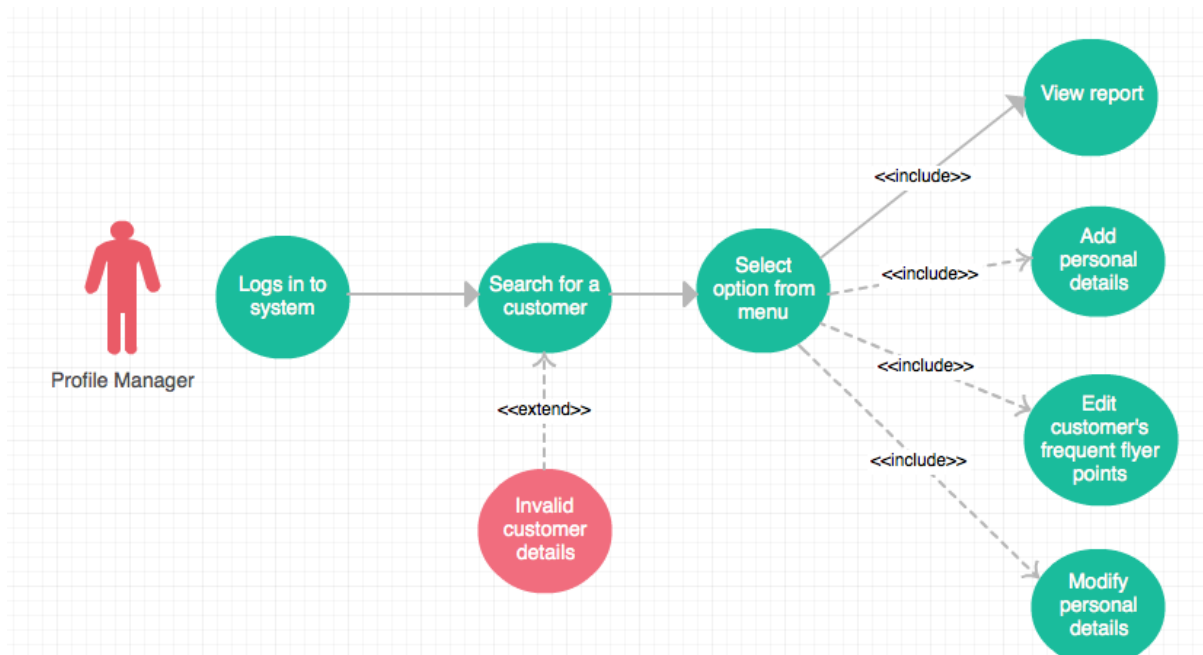
Volere

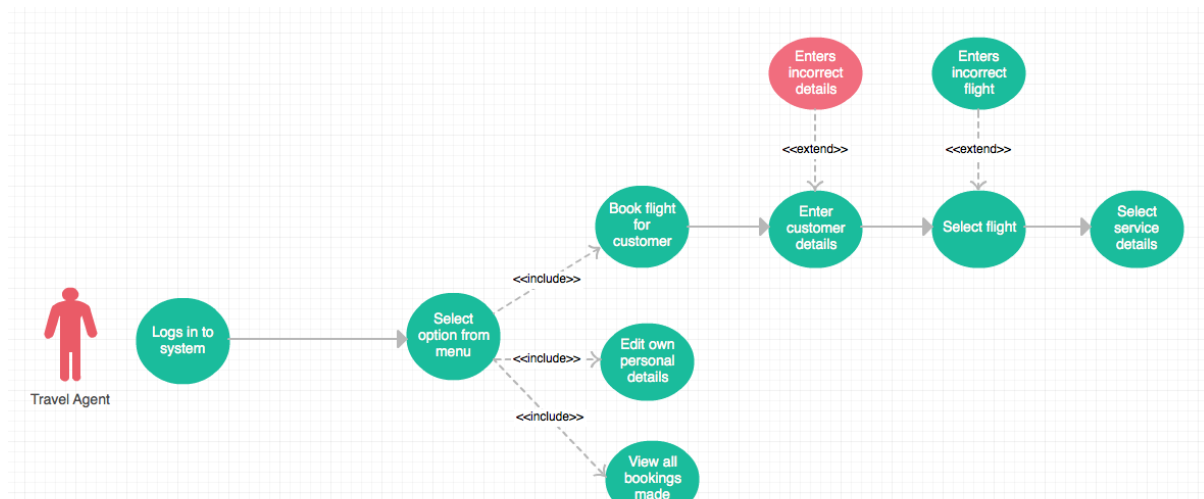
Copyright © Atlantic Systems Guild

6. Use Case Diagrams









7. Architectural Design

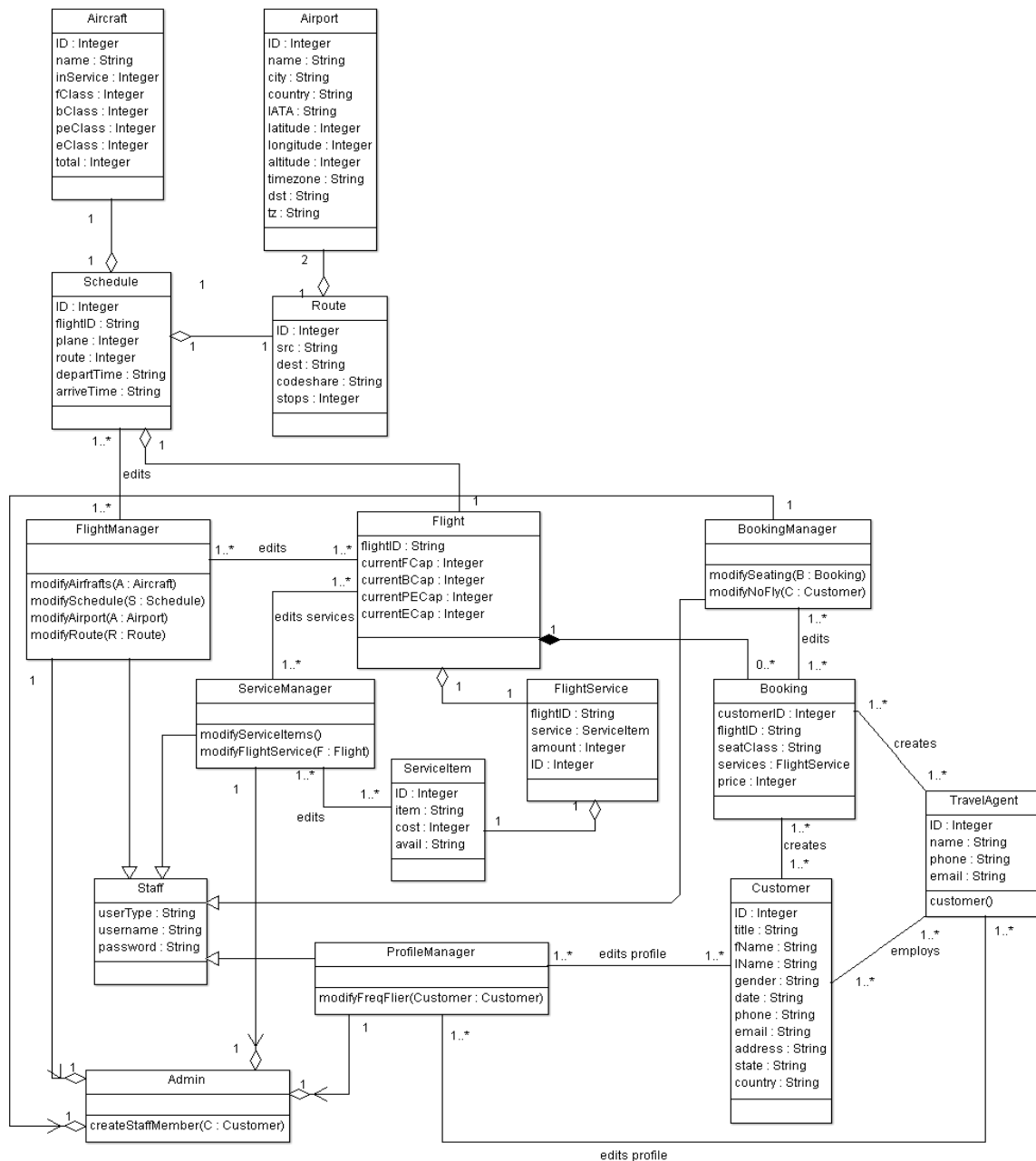
Document

7.1. Logical View

The purpose of the logical view is to describe the static structure of the system and the functionality that the system will provide to the end user. Our model is split into 3 different class diagrams (or “views”). These views were created in order to illustrate parts of the system in a context that is easier to understand (compared to one large diagram).

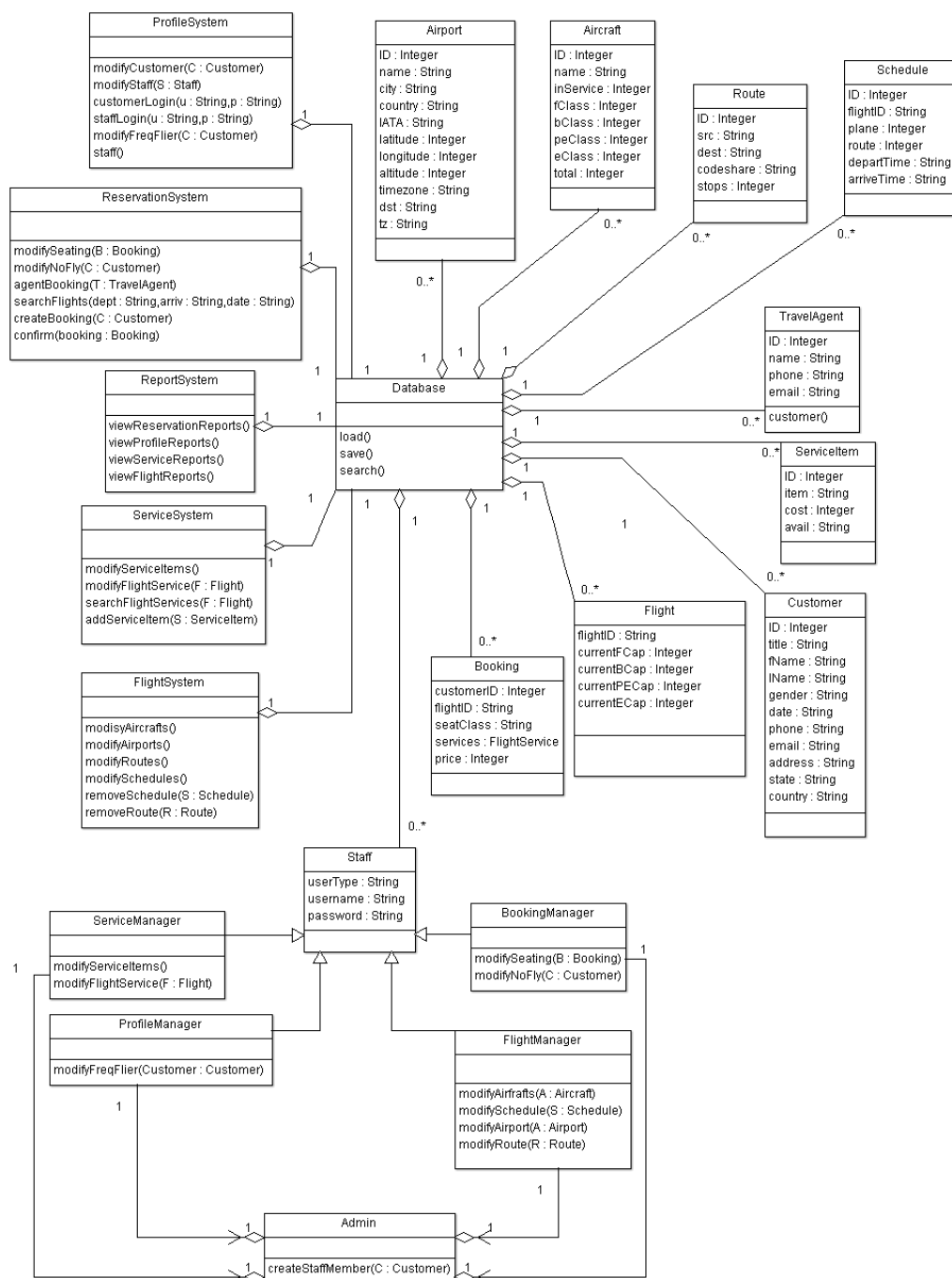
7.2. Domain Model

The domain model depicts the key data elements from the system requirements and how they are associated to the users of the system. More complex views of the system seen below.



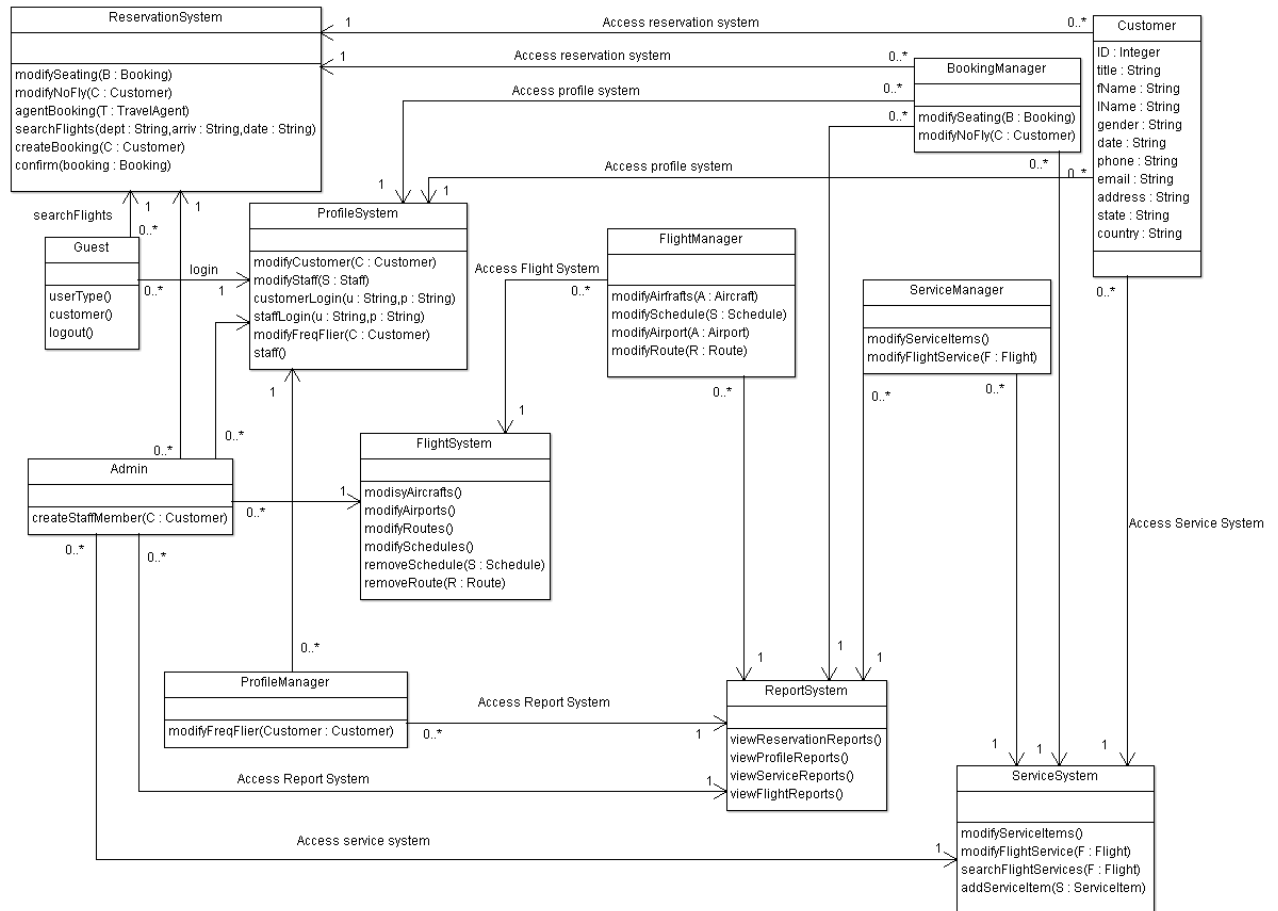
7.2.1. Sub-System and Database View

The goal of this class diagram is to illustrate how data storage classes (e.g. 'Airport', 'Route') are related to the database system, and how the several sub-systems relate to the database system. The most important concept shown here is that no sub-system can directly access a base class. All access to these classes must be done via database functionality. This will help maintain database integrity.



7.2.2. User and Sub-System View

The goal of this class diagram is to illustrate how user classes relate to the sub-systems. The most important concept shown here is that the user classes can only access the sub-systems that they require to perform their respective jobs. Restricting access to sub-systems based on the user class will help ensure a user cannot access functionality they are not authorized to use.



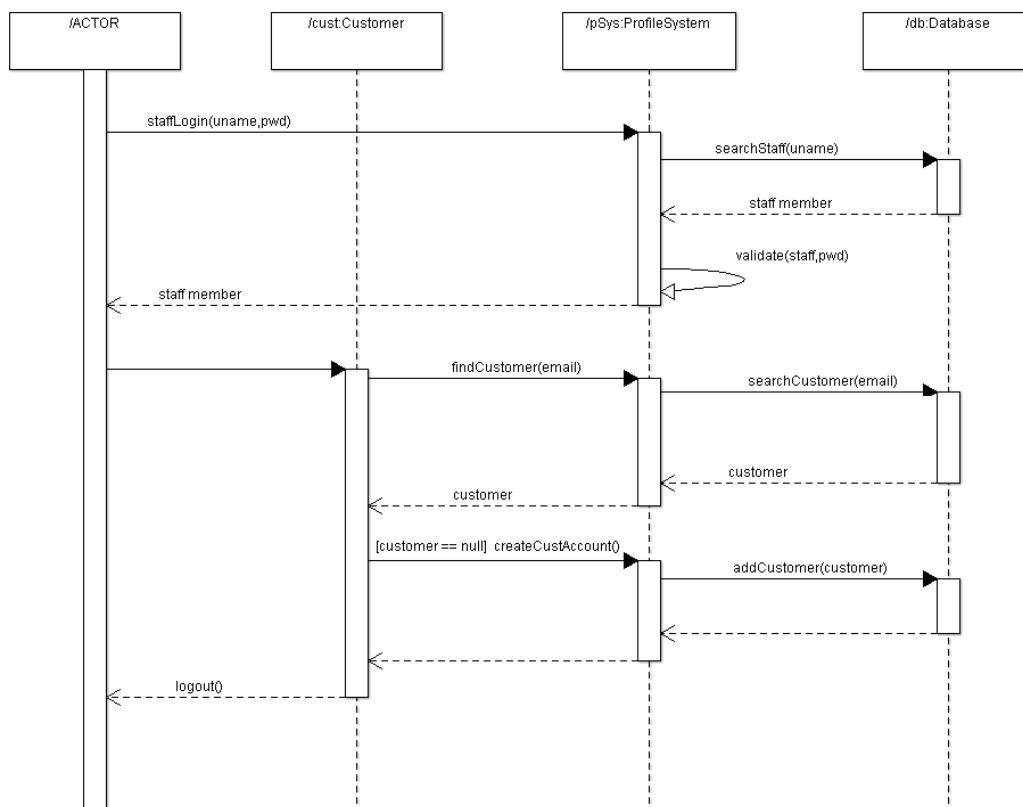
7.3. Process View

The purpose of the process view is to describe the dynamic nature of the system. We have 4 sequence diagrams that realise selected major use-cases, and depict the run-time behaviour of the system.

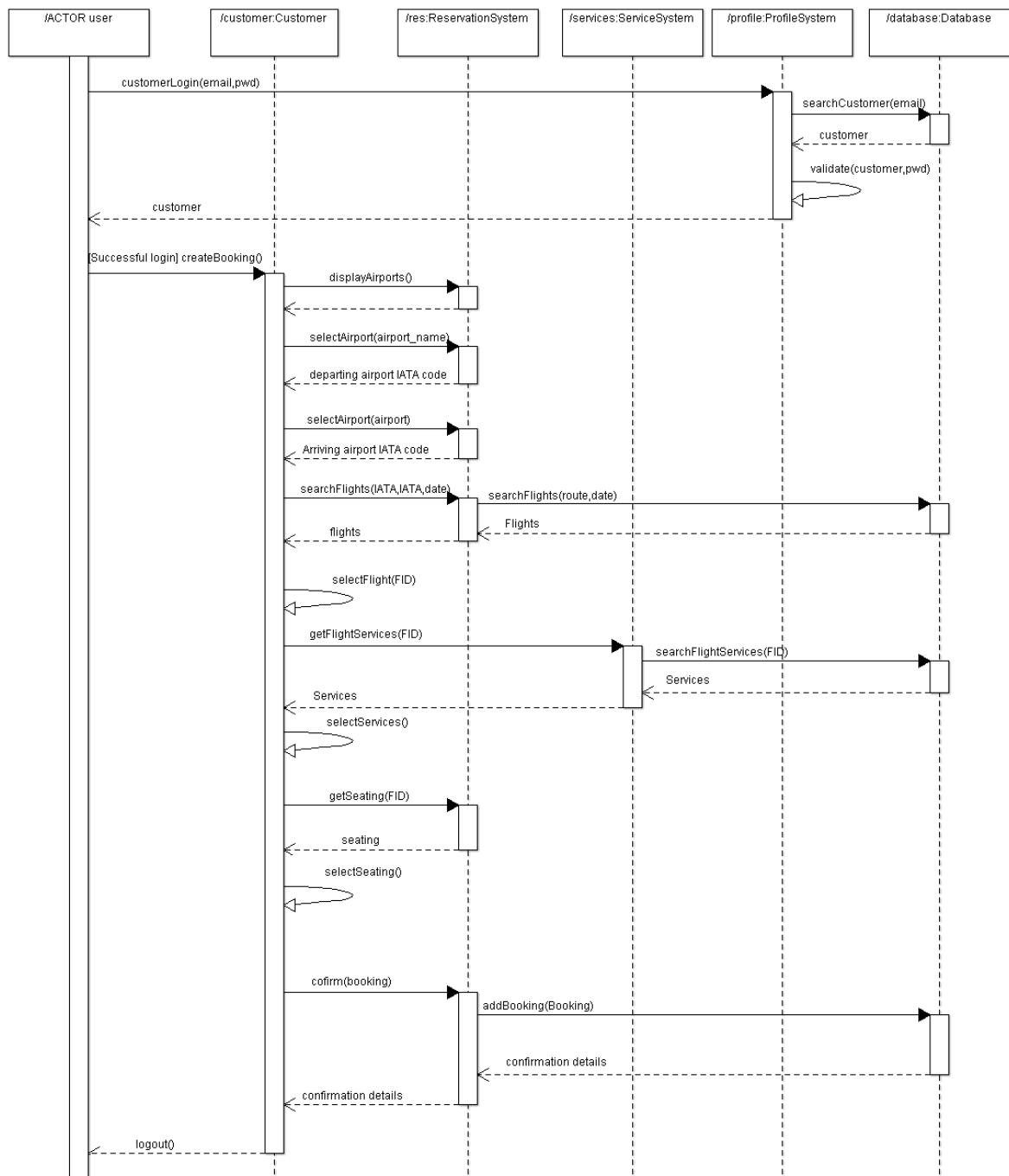
Note 1: ArgoUML does not support the 'Actor' stick figure in sequence diagrams. Thus the actor will be defined with 'ACTOR' displayed at the top of an object instance.

Note 2: Not all functions in the sequence diagrams are displayed in the class diagram (too many functions created clutter and adversely affect the clarity of the diagram). Full functionality of a class is defined in the Data Dictionary.

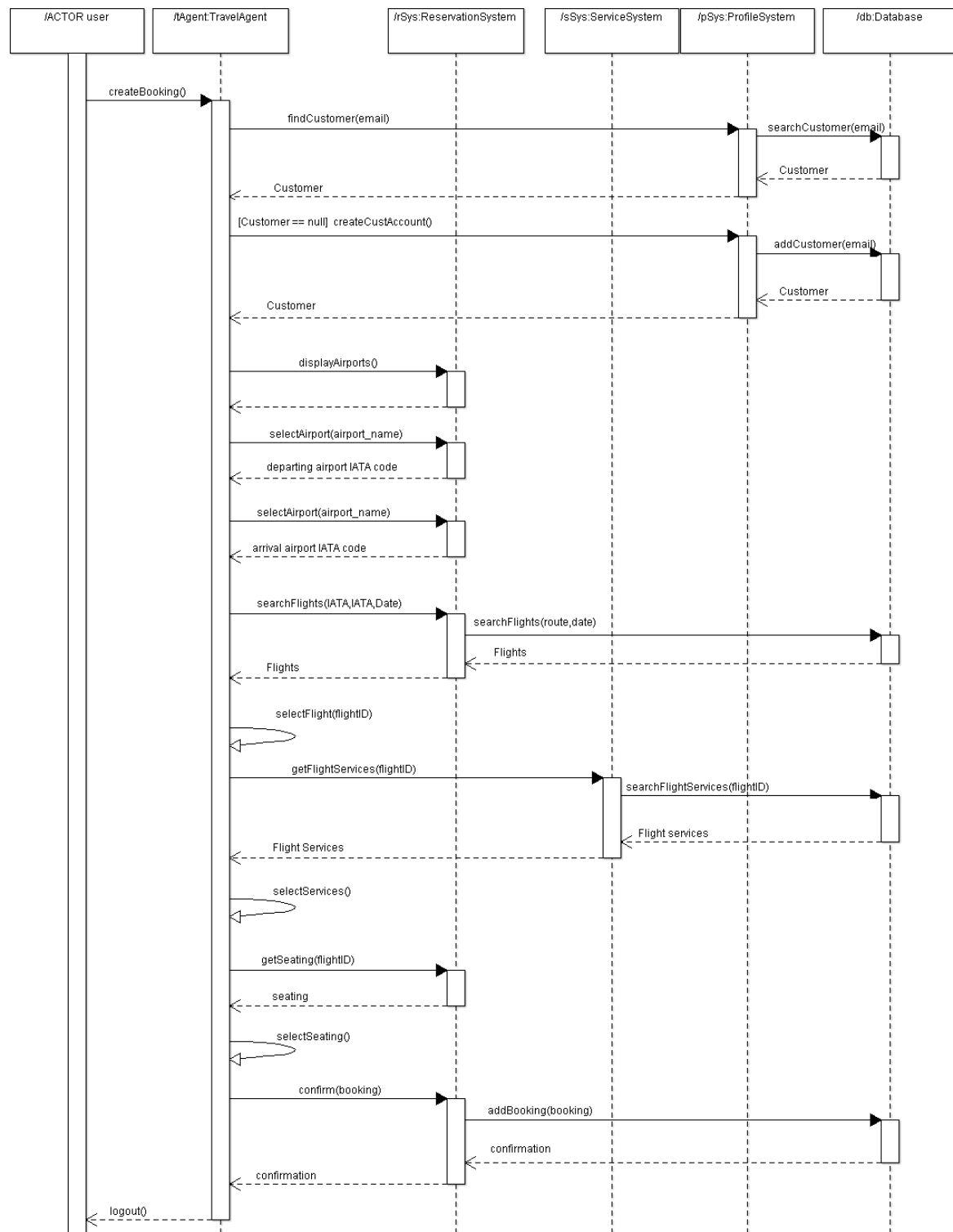
7.3.1. Staff Member Creates Account for Customer - Sequence Diagram



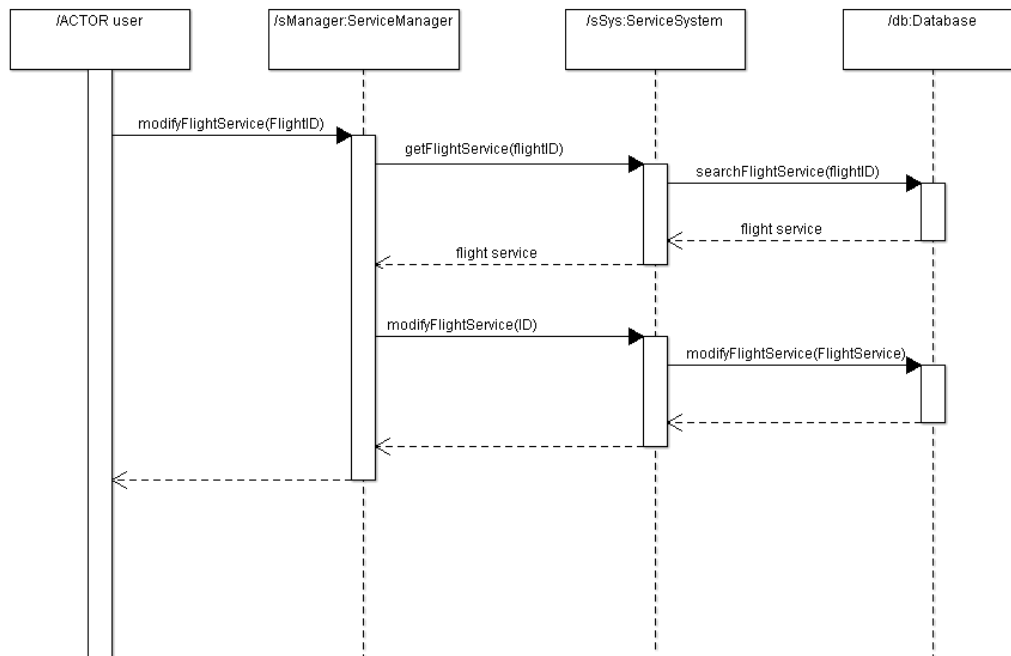
7.3.2. Customer Books Flight – Sequence Diagram



7.3.3. Travel Agent Books Flight for Customer – Sequence Diagram

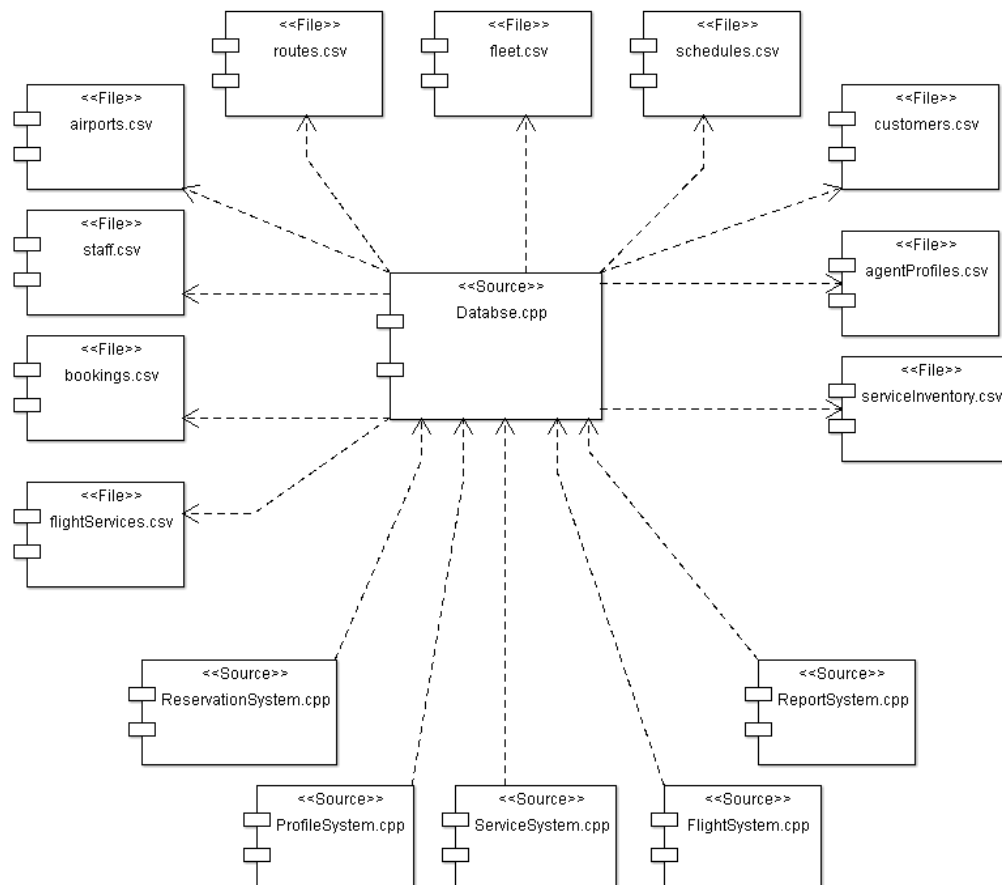


7.3.4. Service Manager Modifies Service Items for a Flight – Sequence Diagram



7.4. Implementation View

The implementation views purpose is to illustrate the systems organisations of static modules. A component diagram is provided below that depicts the dependencies between modules, and some modules dependency on text files.



8. Data Dictionary

8.1. Class Name: Airport

Superclass: None

Attributes:

Id :	A number that Uniquely identifies an airport
Name :	Alphanumeric
City :	Alphanumeric
Country :	Alphanumeric
IATA/FAA :	Alphanumeric, an abbreviation of name.
Latitude :	A Decimal number representing latitude
Longitude:	A decimal number representing longitude.
Altitude:	A number representing altitude in meters.
Timezone:	A number representing timezone related to GMT time. Example this value is 12 for NZ. This represents GMT+12.
DST:	A character that represents daylight saving time
Tz:	Alphanumeric, represents timezone from the tz database.

Methods:

getID():	This method allows access to an airports ID.
getName():	This method allows access to an airports name.
getCity():	This method allows access to an airports City.
getIATA():	This method allow access to an airports IATA code.
getLat():	This method allows access to an airports Latitude.
getLong():	This method allows access to an airports Longitude.
getAlt():	This method allows access to an airports Altitude.
getTimeZone():	This Method allows access to an airports timezone(GMT).

getDST(): This method allows access to an airports Daylight saving code.

getTZ(): This method allows access to an airports TZ database time zone.

setID(id): This method allows us to set an airports ID.

setName(name): This method allows us to set an airports Name.

setCity(city): This method allows us to set an airports City.

setIATA(IATA code): This method allows us to set an airports IATA/FAA abbreviation code.

setLat(latitude): This method allows us to set an airports latitude.

setLong(longitude): This method allows us to set an airports Longitude.

setAlt(altitude): This method allows us to set an airports altitude.

setTimeZone(amt_hours): This method allows us to set an airports timezone(GMT)

setDST(DST_code): This method allows us to set an airports daylight saving code.

setTZ(timezone_name): This method allows us to set an airports TZdatabase time zone.

8.2. Class Name: Route

Superclass: None

Attributes:

ID: A number that uniquely identifies a route.

SourceAirport: An IATA/FAA code that corresponds to an airport

DestinationAirport: An IATA/FAA code that corresponds to an airport

codeshare: A character that can be either 'NULL' or 'Y'. Defines whether codesharing occurs on this route.

stops: A number representing the number of stops on this route.

Methods:

getID(): This method allows access to a routes ID.

getSrc(): This method allows access to a routes source airport IATA/FAA code.

getDest():	This method allows access to a routes destination airport IATA/FAA code.
getCodeShare():	This method allows access to a routes codeshare status.
getStops():	This method allows access to the amount of stops on a route.
setID(id):	This method allows us to set the ID of a route.
setSrc(IATA_code):	This method allows us to set the source airport IATA/FAA code.
setDest(IATA_code):	This method allows us to set the destination airport IATA/FAA code.
setCodeShare(codeshare_status):	This method allows us to set the codeshare status for a route.
setStops():	This method allows us to set the amount of stops on a route.

8.3. Class Name: Aircraft

Superclass: None

Attributes:

ID:	A number used to uniquely identify an aircraft.
name:	Alphanumeric. The name used for the type of aircraft.
inService:	A number representing how many of this type of aircraft are in service.
fClass:	A number representing how many first class seats are on this aircraft.
bClass:	A number representing how many business class seats are on this aircraft.
peClass:	A number representing how many premium economy seats are on this aircraft.
eClass:	A number representing how many economy seats are on this aircraft.
total:	A number representing the total number of seats available on the aircraft.

Methods:

getID():	This method allows access to an aircrafts ID.
----------	---

getName(): This method allows access to an aircrafts type.

getInService(): This method allows access to the amount of aircrafts of this type that are in service

getFClass(): This method allows access to the amount of first class seats on this aircraft.

getBClass(): This method allows access to the amount of business class seats on this aircraft.

getPEClass(): This method allows access to the amount of premium economy class seats on this aircraft.

getEClass(): This method allows access to the amount of economy class seats on this aircraft.

getTotal(): This allows access to the total amount of seats on this aircraft.

setID(id): This method allows us to set an aircrafts ID.

setName(name): This method allows us to set an aircrafts name.

setInService(numOfAircraft): This method allows us to set the amount of aircraft of this type that are in service.

setFClass(fName): This method allows us to set the amount of first class seats on this aircraft

setBClass(lName): This method allows us to set the amount of business class seats on this aircraft.

setPEClass(numOfseats): This method allows us to set the amount of premium economy class seats on this aircraft.

setEClass(numOfSeats): This method allows us to set the amount of economy class seats that are on this aircraft.

setTotal(): This method calculates and sets the total number of seats on this aircraft by adding together the values in fClass,bClass,peClass,andeClass.

8.4. Class Name: Schedule

Superclass: None

Attributes:

ID: A number used to uniquely identify a schedule in the database.

flightID: Alphanumeric. A string used to uniquely identify a Flight.

Plane: A number that identifies the type of plane to be used for this schedule. This number refers to a plane in the Aircraft database.

Route: A number that identifies the route to be used in this schedule. This number refers to a route in the route database.

departTime: A string detailing the date,day,time,year and timezone of the departure.

arriveTime: A string detailing the date,day,time,year and timezone of the arrival

Methods:

getID(): This method allows us access to a schedules database ID.

getFlightID(): This method allows us access to the Flight ID.

getPlane(): This method allows us to access the plane type used in a schedule.

getRoute(): This method allows us to access the route used in a schedule.

getDeparture(): This method allows us to access the departure date/time of the schedule.

getArrival(): This method allows us access the arrival date/time of the schedule.

setID(): This method allows us to set a schedules database ID.

setFlightID(): This method allows us to set a schedules flight ID.

setPlane(): This method allows us to set a schedules plane type.

setRoute(): This method allows us to set a schedules route.

setDeparture(): This method allows us to set the schedules departure date/time.

setArrival(): This method allows us to set the schedules arrival date/time

8.5. Class Name: Flight

Superclass: None

Attributes:

flightID: Alphanumeric. A string that uniquely identifies a flight. Refers to a schedules flightID.

currentFCap: A number representing how many people are currently booked as first class on a flight. Must be less than aircrafts fClass value.

currentBCap:	A number representing how many people are currently booked as Business class on a flight. Must be less than aircrafts BClass value.
currentPECap:	A number representing how many people are currently booked as premium economy class on a flight. Must be less than aircrafts PEClass value.
currentECap:	A number representing how many people are currently booked as economy class on a flight. Must be less than aircrafts EClass value.
serviceOnFlight: on this flight.	A list of FlightService objects that are allocated to be on this flight.

Methods:

getFlightID():	This method allows us access to a schedules database ID.
getFCap():	This method allows us to access the current amount of first class customers booked on this flight.
getBCap():	This method allows us to access the current amount of business class customers booked on this flight.
getPECap():	This method allows us to access the current amount of premium economy customers booked on this flight.
getECap():	This method allows us to access the current amount of economy customers booked on this flight.
setFlightID():	This method allows us to set the FlightID of a flight.
setFCap():	This method allows us to set the current first class capacity of a flight.
setBCap():	This method allows us to set the current business class capacity of a flight
setPECap():	This method allows us to set the current premium economy capacity of a flight.
setECap():	This method allows us to set the current economy capacity of a flight.

8.6. Class Name: Customer

Superclass: None

Attributes:

ID:	A number used to uniquely define a person
Title:	Alphanumeric.
fName:	Alphanumeric. Represents a persons first name.
lName:	Alphanumeric. Represents a persons last name.
gender:	A character that represents a persons gender. Can be either 'm' or 'f'.
DOB:	Instantiation of a 'Date' class. Represents a persons date of birth.
Phone:	Alphanumeric. Represents a persons contact phone number.
Email:	String of characters that represent a persons email address.
Address:	Alphanumeric. Represents a persons street address.
State:	Alphabetic. Represents which state a person resides in.
Country:	Alphabetic. Represents which country a person resides.
CardType:	Alphanumeric. Represents what type of card the customer has.
CardNum:	Numeric. A Number that represents the customers card number.
freqFlierPoints:	Numeric. Represents the number of frequent flier points a customer has accredited to them.
Passport:	Either 'TRUE' or 'FALSE'. Represents whether a customer has a passport.
Status:	Either 'NULL', 'no fly', or 'watch'. Represents whether a customer is allowed to book a flight based on previous flight behaviour.
TravelAgent:	Alphanumeric. A String that corresponds to the name of a travel agent in the agents database.

Methods:

getID():	This method allows access to a customers ID.
getTitle():	This method allows access to a customers title.
getFName():	This method allows access to a customers first name.
getLName():	This method allows access to a customers last name.

getGender(): This method allows access to a customers gender.

getDOB(): This method allows access to a customers date of birth.

getPhone(): This method allows access to a customers phone number.

getEmail(): This method allows access to a customers email Address.

getAddress(): This method allows access to a customers street address.

getState(): This method allows access to a customers state of residence.

getCountry(): This method allows access to a customers country of residence.

getCardType(): This method allows access to a customers credit card type.

getCardNum(): This method allows access to a customers credit card number.

getfreqFly(): This method allows access to the amount of frequent flier points accredited to the customer.

getPassport(): This method allows access to a customers passport status.

getStatus(): This method allows access to a customers fly status.

getAgent(): This method allows access to the name of a travel agent that the customer used to book a flight.

setID(id): This method allows us to set the ID of a customer.

setTitle(title): This method allows us to set the title of a customer.

setFName(fName): This method allows us to set the first name of a customer.

setLname(lName): This method allows us to set the last name of a customer.

setGender(gender): This method allows us to set the gender of a customer.

setDOB(date): This method allow us to set the date of birth of a customer.

setPhone(ph#): This method allows us to set the phone number of a customer.

setEmail(emailAddress): This method allows us to set the email address of a customer.

setAddress(stAddress): This method allows us to set the street address of a customer.

setState(state): This method allows us to set the state of residence of the customer.

setCountry(country): This method allows us to set the country of residence of the customer.

setCardType(cardType): This method allows us to set the credit card type of the customer.

setCardNum(cardNum): This method allows us to set the credit card number of the customer.

setFreqFly(freqFlyPts): This method allows us to set the frequent flier points for a customer.

setPassport(passportState): This method allows us to set the passport status of the customer.

setStatus(status): This method allows us to set the no fly status of the customer.

setAgent(name): This method allows us to set the agency name that the customer used to book a flight.

8.7. Class Name: Travel Agent

Superclass: None

Attributes:

ID: A number used to uniquely identifies a Travel Agent

Name: Alphanumeric. Represents the name of the agency.

Phone: Numeric. Represents the travel agents contact phone number.

Email: Alphanumeric. Represents the travel agents contact email address.

Methods:

getID(): This method allows access to a travel agencies ID.

getName(): This method allows access to a travel agencies name.

getPhone(): This method allows access to a travel agencies phone number.

getEmail(): This method allows access to a travel agencies email address.

setID(id): This method allows us to set the ID of an agency.

setName(name): This method allows us to set the name of an agency.

setPhone(phone#): This method allows us to set the phone number of an agency.

setEmail(emailaddress): This method allows us to set the email address of an agency.

createBooking(): This method allows a travel agent to create a booking for a customer.

8.8. Class Name: ServiceItem

Superclass: None

Attributes:

ID: A number used to uniquely identify a service item

Item: Alphabetic. Represents a description/name of a service item.

Cost: Numeric. Represents the price of an item.

Availability: Either 'All' or 'International'. Represents what type of flights this item is available on.

Methods:

getID(): This method allows access to a Service Items ID.

getItem(): This method allows access to a service items description.

getCost(): This method allows access to a service items price.

getAvail(): This method allows access to a service items availability.

setID(id): This method allows us to set the ID of a service Item.

setItem(desc): This method allows us to set the description of a service item.

setCost(price): This method allows us to set the price of a service item.

setAvail(availability): This method allows us to set the availability of a service item.

8.9. Class Name: FlightService

Superclass: none.

Attributes:

- ID: A number that uniquely identifies a FlightService in the database.
- FlightID: A string that identifies the flight this service belongs to. Refers to a Flight class's FlightID.
- ServiceItem: A number that identifies a ServiceItem. This refers to a ServiceItem class's ID.
- itemAmount: A number that identifies how many of the ServiceItem will be on the flight.

Method:

- getID(): This method allows us to access a FlightServices ID.
- getFlightID(): This method allows us access to a FlightService's Flight ID.
- getServiceItem(): This method allows us access to a FlightService's service item.
- getItemAmount(): This method allows us to access to the amount of serviceItem's that will be on a flight.
- setID(): This methos allows us to set the ID of a flight service object.
- setFlightID(): This method allows us to set the flightID of the flight this FlightService belongs to.
- setServiceItem(): This method allows us to set which item this flightService is for.
- setItemAmt(): This method allows us to set the amount of ServiceItems that will be on the flight.

8.10. Class Name: Booking

Superclass: none

Attributes:

- CustomerID: A number that uniquely identifies the customer that has booked a flight.
- FlightID: A string that identifies the flight this Booking belongs to
- seatNumber: A String that identifies which seat on the plane this booking is for.

seatClass:	A string that identifies what class of seating this booking is for.
Services:	A list of FlightService objects that represent what services the customer has requested for this flight.
Price:	A number that represents the price of this booking.

Methods:

getCustomer():	This method allows us access to the Customers ID.
getFlight():	This method allows us access to the Flight's ID.
getSeatNum():	This method allows us access to the seat number this booking is for.
getSeatClass():	This method allows us access to the seat class this booking is for.
getServices():	This method allows us access to the ServiceItems that the customer has requested for this flight.
getPrice():	This method allows us access to the price of this booking.
setCustomer():	This method allows us to set the customer ID for this booking,
setFlight():	This method allows us to set the FlightID for this booking.
setSeatNum():	This method allows us to set the seat number for this booking.
setSeatClass():	This method allows us to set the seat class for this booking.
setServices():	This method allows us to set the customers requested services for this booking.
setPrice():	This method allows us to set the price of this booking.

8.11. Class Name: Staff

Superclass: None

Attributes:

ID:	A number used to uniquely identify a staff member.
Password:	A sting of characters representing a staffs login password.
userType:	A string that identifies the type of staff member the object represents. This can be either "Staff", "ProfileManager", "ServiceManager", "FlightManager", "BookingManager" or "Admin".

Methods:

getID():	This method allows us to access a staff members ID.
getPassword(): password	This method allows us to access a staff members password
setID(id):	This method allows us to set the id of a staff member.
setPassword(password):	This method allows us to set the password of a staff member.
findCustomer():	This method allows the user to retrieve an existing customers details.
createCustAccount():	This method allows the user to create an account for a new customer.
viewCustBookings():	This allows the user to view all the bookings made by an existing customer including service details.
modifyCustDetails():	This allows the user to modify an existing customers personal details.
modifyCustBooking():	This allows the user to modify an existing customers booking details.

8.12. Class Name: ProfileManager

Superclass: Staff

Attributes:

ID:	A number used to uniquely identify a staff member.
Password:	A sting of characters representing a staffs login password.

Methods:

getID():	This method allows us to access a staff members ID.
getPassword(): password	This method allows us to access a staff members password
setID(id):	This method allows us to set the id of a staff member.
setPassword(password):	This method allows us to set the password of a staff member.
findCustomer():	This method allows the user to retrieve an existing customers details.

createCustAccount(): This method allows the user to create an account for a new customer.

viewCustBookings(): This allows the user to view all the bookings made by an existing customer including service details.

modifyCustDetails(): This allows the user to modify an existing customers personal details.

modifyFreqFlier(customerID): This method allows the user to modify a customers frequent flier points.

8.13. Class Name: Flight manager

Superclass: Staff

Attributes:

ID: A number used to uniquely identify a staff member.

Password: A sting of characters representing a staffs login password.

Methods:

modifyAircrafts(): This method allows the user to add new/edit existing information about aircrafts in the database.

modifySchedules(): This method allows the user to add new/edit existing information about schedules in the database.

modifyAirports(): This method allows the user to add new/edit existing information about Airports in the database.

modifyRoutes(): This method allows the user to add new/edit existing information about Routes in the database.

viewFlightReports(): This method allows the user to see various reports about flights created by the system.

8.14. Class Name: BookingManager

Superclass: Staff

Attributes:

ID: A number used to uniquely identify a staff member.

Password: A sting of characters representing a staffs login password.

Methods:

- modifyCustBooking(): This allows the user to modify an existing customers booking details.
- modifySeating(): This allows the user to modify customer seating even if seats are occupied.
- modifyNoFly(): This allows the user to modify the no-fly list in the database.
- viewBookingReports(): This method allows the user to see various reports about Bookings created by the system.

8.15. Class Name: ServiceManager

Superclass: Staff

Attributes:

- ID: A number used to uniquely identify a staff member.
- Password: A sting of characters representing a staffs login password.

Methods:

- modifyServiceItems(): This method allows the user to modify existing, or add new items to the ServiceItems database.
- modifyFlightService(flightID): This method allows the user to modify the service details of a flight.
- viewServiceReports(): This method allows the user to view various reports about services generated by the system.

8.16. Class Name: Database

Superclass: none

Attributes:

- Employees: A list of staff objects.
- Airports: A list of Airport objects.
- Routes: A list of Route objects.
- Aircrafts: A list of Aircraft objects.
- Customers: A list of Customer objects.
- Agents: A list of TravelAgent objects.

ServiceItems: A list of ServiceItem objects.

FlightServices: A list of FlightService objects.

Flights: A list of Flight objects.

Bookings: A list of Booking objects.

Method:

loadEmployees(): This method loads information from a file into staff objects which are places in the employees list.

loadAirports(): This method loads information from a file into Airport objects which are places in the airports list.

loadRoutes(): This method loads information from a file into Route objects which are places in the routes list.

loadAircrafts (): This method loads information from a file into Aircraft objects which are places in the aircrafts list.

loadCustomers(): This method loads information from a file into Customer objects which are places in the customers list.

loadAgents(): This method loads information from a file into TravelAgent objects which are places in the agents list.

loadServiceItems(): This method loads information from a file into ServiceItem objects which are places in the serviceItems list.

loadFlightServices(): This method loads information from a file(If it already exists) into FlightService objects which are placed in the flightServices list.

loadFlights(): This method loads information from a file into Flightobjects which are placed in the flights list.

loadBookings(): This method loads information from a file into Booking objects which are placed in the Bookings list.

Load(): This method loads all data for the database. Utilised functions defined above.

searchStaff(username): This method allows us to search the employees list and access an employee based on their username.

searchAirports(ID): This method allows us to search the airports list and access an airport based on its ID.

searchRoutes(ID): This method allows us to search the routes list and access a route based on its ID.

`searchRoutes(src,dest)`: This method allows us to search the routes list and access a route based on a source airport and a destination airport.

`searchAircrafts(ID)`: This method allows us to search the aircrafts list and access an aircraft based on its ID.

`searchCustomers(customer_email)`: This method allows us to search the Customers list and access a customer based on their email address.

`searchAgents(agent_name)`: This method allows us to search the agents list and access a TravelAgent based on their name.

`searchServiceItems(ID)`: This method allows us to search the serviceItems list and access a ServiceItem based on its ID.

`searchFlightServices(flightID)`: This method allows us to search the flightServices list and access a FlightService based on its flightID.

`searchFlightsByID(flightID)`: This method allows us to search the flights list and access a flight based on its flightID.

`searchFlightsByDate(route,date)`: This method allows us to search the flights list and access flights based on their date and route.

`searchSchedule(flightID)`: This method allows us to search the schedules list and access a schedule based on its flightID.

`searchBookings(customerID)`: This method allows us to search the bookings list and access a booking based on its customerID.

`addEmployees(staff)`: This method allows us to add a staff member to the employees list.

`addAirports(airport)`: This method allows us to add a airport to the airports list.

`addRoutes(route)`: This method allows us to add a route to the routes list.

`addAircrafts(aircraft)`: This method allows us to add a aircraft to the aircrafts list.

`addCustomers(customer)`: This method allows us to add a customer to the customers list.

`addAgents(TravelAgent)`: This method allows us to add a TravelAgent to the agents list

`addServiceItems(ServiceItem)`: This method allows us to add a ServiceItem to the serviceItems list

`addSchedule(Schedule)`: This method allows us to add a Schedule to the schedules list.

`addFlightServices(FlightService):` This method allows us to add a `FlightService` to the `flightServices` list.

`addFlights(Flight):` This method allows us to add a `Flight` to the `flights` list

`addBooking(Booking):` This method allows us to add a booking to the bookings list. returns booking number. All updates to the database also occur (Example add seating details etc).

`removeEmployee(Employee):` This method allows us to remove an employee from the `employees` list.

`removeAirport(Airport):` This method allows us to remove an `Airport` from the `airports` list.

`removeRoute(Route):` This method allows us to remove a route from the `routes` list.

`removeAircraft(Aircraft):` This method allows us to remove an aircraft from the `aircrafts` list.

`removeCustomer(Customer):` This method allows us to remove a customer from the `customers` list.

`removeAgent(TravelAgent):` This method allows us to remove a `Travel agent` from the `agents` list.

`removeServiceItem(ServiceItem):` This method allows us to remove a `ServiceItem` from the `serviceItems` list.

`removeSchedule(Schedule):` This method allows us to remove a schedule from the `schedules` list.

`removeFlightService(FlightService):` This method allows us to remove a `FlightService` from the `flightServices` list.

`removeFlight(Flight):` This method allows us to remove a `Flight` from the `flights` list.

`removeBooking(Booking):` This method allows us to remove a booking from the `bookings` list.

`modifyEmployee(username):` This method allows us to modify an employee in the `employees` list.

`modifyAirport(ID):` This method allows us to modify an airport in the `airports` list.

`modifyRoute(ID):` This method allows us to modify a route in the `routes` list.

`modifyAircraft(ID):` This method allows us to modify an aircraft in the `aircrafts` list.

`modifyCustomer(customer_email):` This method allows us to modify a customer in the `customers` list.

`modifyAgent(agent_name)`: This method allows us to modify a travel agent in the agents list.

`modifyServiceItem(ID)`: This method allows us to modify a serviceItem in the serviceItems list.

`modifyFlightService(FlightService)`: This method allows us to modify a flightService in the flightServices list.

`modifyFlight(flightID)`: This method allows us to modify a flight in the flights list.

`modifySchedule(flightServiceID)`: This method allows us to modify an schedule in the schedules list.

`modifyBooking(customerID)`: This method allows us to modify a booking in the bookings list.

`saveEmployee()`: This method saves the contents of the employees list to a file.

`saveAirport()`: This method saves the contents of the airports list to a file.

`saveRoute()`: This method saves the contents of the routes list to a file.

`saveAircraft()`: This method saves the contents of the aircrafts list to a file.

`saveCustomer()`: This method saves the contents of the customers list to a file.

`saveAgent()`: This method saves the contents of the agents list to a file.

`saveServiceItem()`: This method saves the contents of the serviceItems list to a file.

`saveFlightService()`: This method saves the contents of the flightServices list to a file.

`saveSchedule()`: This method saves the contents of the schedules list to a file.

`saveFlight()`: This method saves the contents of the flights list to a file.

`saveBooking()`: This method saves the contents of the bookings list to a file.

`Save()`: This method saves all data from the database. Utilises methods defined above.

8.17. Class Name: ProfileSystem

Superclass: none

Attributes:

Database: A reference to the database.

Methods:

findCustomer(email): This method allows the user to retrieve an existing customers details.

modifyNoFly(): This allows the user to modify the no-fly list in the database.

modifyCustomer(email): This method allows the user to modify any customer information.

findStaff(username): This method will find a staff member in the database via their username

modifyStaff(username): This method will allow the user to modify any staff information.

findTravelAgent(agentName): This method will find a TravelAgent in the database via their agency name.

modifyTravelAgent(agentName): This method allows the user to modify any TravelAgent information.

customerLogin(email,password): This method allows a customer to log into the system.

staffLogin(username,password): This method allows a staff member to log into the system.

Validate(customer,pwd): This method validates the login details of a customer against a password. Method used in customerLogin() function.

Validate(staff,pwd): This method validates the login details of a staff object against a password. Method used in staffLogin() function.

createCustAccount(): This method allows the user to create an account for a new customer.

viewCustBookings(): This allows the user to view all the bookings made by an existing customer including service details.

modifyCustDetails(email): This allows the user to modify an existing customers personal details.

removeCustomer(email):	This method allows the user to remove a customers details from the system.
removeStaff(username):	This method allows the user to remove a staff member from the system.

8.18. Class Name: ReservationSystem

Superclass: none

Attributes:

Database: A reference to the database.

Methods:

createBooking(customerID):	This method allows the user to create a new booking.
modifyCustBooking():	This allows the user to modify an existing customers booking details.
modifySeating():	This allows the user to modify customer seating even if seats are occupied.
viewBookingReports():	This method allows the user to see various reports about Bookings created by the system.
getFlightSeating(flightID):	This method returns information about seating occupancy for a flight.
AddCustomerToFlight(booking,flightID):	This method modifies a flight objects information when a customer has booked seats on a flight
addBooking(flightID):	This method adds a new booking into the bookings list
getTicketPrice(flightID):	This method returns the ticket price for a flight.
searchFlights(IATA,IATA,date):	This method returns a list of flights.Obtained by searching the databasewith a route object(created with 2 IATA codes), and a date.
selectAirport(airport name):	This method allows the user to select an airport during the booking process.
custCancelBooking(customerID,booking):	This method allows a customers booking to be removed from the system.
agentBooking(agentID):	This method allows a booking agent to book a flight for a customer.

displayAirports():	This method returns a list of airports sorted alphabetically and displays them to the user.
Confirm(booking):	This method confirms booking details for a customers booking. addBooking(booking) is called to update the database.
getSeating(FlightID):	This method retrieves seating details about a flight

8.19. Class Name: ReportSystem

Superclass: none

Attributes:

Database: A reference to the database.

Methods:

generateReservationReport():	This method allows the user to see all reports about reservations from the system.
generateServiceReport():	This method allows the user to see all reports about services from the system.
generateFlightReport():	This method allows the user to see all reports about flights from the system.
generateProfileReport():	This method allows the user to see all reports about profiles from the system.

8.20. Class Name: ServiceSystem

Superclass: none

Attributes:

Database: A reference to the database.

Methods:

getFlightService(FlightID):	This method returns a list of available service items on a particular flight.
modifyServiceItem(ID):	This method allows the user to modify any information about a serviceItem.
modifyFlightService(ID):	This method allows the user to modify any information about a FlightService.

`addServiceItem(serviceItem):` This method allows the user to add a `serviceItem` to the database.

`removeServiceItem(ID):` This method allows the user to remove a `serviceItem` from the database.

`addFlightService(ID):` This method allows the user to add a `flightService` to the database.

`removeFlightService(ID):` This method allows the user to remove a `flightService` from the database.

`searchFlightServices(flightID):` This method returns a list of `flightService` information about a flight.

8.21. Class Name: FlightSystem

Attributes:

`Database:` A reference to the database.

Methods:

`modifyAircrafts():` This method allows the user to add new/edit existing information about aircrafts in the database.

`modifySchedules():` This method allows the user to add new/edit existing information about schedules in the database.

`modifyAirports():` This method allows the user to add new/edit existing information about Airports in the database.

`modifyRoutes():` This method allows the user to add new/edit existing information about Routes in the database.

`removeAircraft(aircraft):` This method allows us to remove an aircraft from the database. This will also alert the user if the airport to be deleted is referenced elsewhere in the system.

`removeRoute(route):` This method allows us to remove a route from the database. This will also alert the user if the route or its attributes are referenced elsewhere in the system.

`removeSchedule(schedule):` This method allows us to remove a schedule from the database. This will also alert the user if the schedule or its attributes are referenced elsewhere in the system.

`removeAirport(airport):` This method allows us to remove a airport from the database. This will also alert the user if the airport is referenced elsewhere in the system.

9. Group Summary

Group Summary		
<i>Role</i>	<i>Name</i>	<i>Documents Contributed to</i>
Manager/ Chief Analyst/ Chief Designer	Kresimir Bukovac	Feasibility study, Risk Analysis, Domain Model, Project Plan, Risk Analysis and Management Plan, Memorandums, Meeting Minutes, Client Questionnaires, Client Requirements, Versioning Evidence, Work Diary, Report
Chief Analyst/ Chief Designer	Darryl Murphy	Domain Model, Architectural Design, Data Dictionary, SRS, Feasibility study, Work Diary
Chief Programmer/ Chief Tester	Ali Sayed	Use Case Diagrams, Program Prototypes, Project Plan, Work Diary
Chief Programmer/ Chief Tester/ GUI Designer	Peter Mavridis	Business Case, Program Prototypes, SRS Template, Work Diary

10. Team Meetings

10.1.1. Memorandum #1

To: Peter Mavridis – GUI Programmer/Chief Tester
Thomas Baillie – Chief Programmer
Darryl Murphy – Chief Analyst/Designer
Ali Sayed – Chief Programmer

From: Kresimir Bukovac – Manager + Chief Analyst/Designer

Date: March 23, 2015

Subject: Review of Client Meeting #1 and planning for Client Meeting #2

On Wednesday, the 18th of March, 2015, the team had their first meeting with the client. A summary of the meeting has been compiled and put onto github. Tomorrow's Team Meeting will be to review the contents of the summary and plan for the next Client Meeting.

You must bring a copy of the compiled meeting summary with you to tomorrow's Team Meeting in order for everyone to go through it together. If you have any notes or changes which need to be amended to the summary, bring them to the Team Meeting as well. Also, bring any plans or documentations you have made in regards to the project or for the next Client Meeting.

If there are any questions as to this memo, the Team Meeting, or you are unable to attend the Team Meeting, you can contact me on the following:

Email: kb100@uowmail.edu.au
Mobile: 0410 410 411

10.1.2. Meeting Minutes #1

Name of Organisation: Group 4

Purpose of Meeting: To review Client Meeting #1 and plan for Client Meeting #2

Date and Time: March 24, 2015 – 3:30pm

Chairperson: Kresimir Bukovac

Members Present: Darryl Murphy
Ali Sayed

Agenda/Topic	Discussion	Action	Person Responsible
Client Meeting #1 - Requirements	Discussed everything we knew from Client Meeting #1 of the project's requirements and began to plan solutions.	Upload the compiled requirements to github.	Kresimir Bukovac
Client Meeting #2 - Questionnaire	From the planning done off the compiled requirements from Client Meeting #1, questions were formed to be asked in the next Client Meeting.	Upload the compiled Questions to github.	Kresimir Bukovac
Storyboard #1 – Customer Bookings	From the requirements from Client Meeting #1, a general idea was created for how the customer will register a booking.	Create and upload a storyboard to github.	Darryl Murphy
Prototype of the Report Menu	From the requirements from Client Meeting #1, ideas were thrown together as to what managers should be able to do in regards to reports in the system.	Create and upload a prototype to github.	Ali Sayed
Class Diagram	Went through the requirements step by step to plan the hierarchy of all the classes that will be needed in the system based off the requirements from Client Meeting #1.	Create and upload a Class Diagram to github.	Kresimir Bukovac

The team members who were not present	Discussed what jobs we would assign to the two members who didn't attend the meeting.	Tell them to read all products of this meeting and develop further questions.	Kresimir Bukovac
---------------------------------------	---	---	------------------

Documents/Deliverables:

- Client Meeting #1 - Requirements
- Client Meeting #2 - Questionnaire
- Storyboard #1 - Customer Bookings
- Prototype of the Report Menu
- Class Diagram

Closing Time: 4:30pm

10.1.3. Memorandum #2

To: Peter Mavridis – GUI Programmer/Chief Tester
Darryl Murphy – Chief Analyst/Designer
Ali Sayed – Chief Programmer

From: Kresimir Bukovac – Manager + Chief Analyst/Designer

Date: March 31, 2015

Subject: Confirm the work allocated from Team Meeting #1 was done and prepare for Client Meeting #2.

On Tuesday, the 24th of March, 2015, the team had their first Team Meeting. The Meeting Minutes has been put onto github. Tomorrow's Team Meeting will be to make sure everyone has done their assigned tasks from Team Meeting #1 and to prepare our presentations and questions for the next Client Meeting.

You must bring a copy of Team Meeting #1's Meeting Minutes with you to tomorrow's Team Meeting in order for everyone to go through and confirm that all the assigned tasks have been completed. You must also bring any deliverables you have made or edited since the last Team Meeting to show the rest of the team to confirm and agree on. Also, bring any plans or documentations you have made in regards to the project or for the next Client Meeting.

If there are any questions as to this memo, the Team Meeting, or you are unable to attend the Team Meeting, you can contact me on the following:

Email: kb100@uowmail.edu.au
Mobile: 0410 410 411

10.1.4. Meeting Minutes #2

Name of Organisation: Group 4

Purpose of Meeting: To make sure everything from last week's Team Meeting has been done and prepare for the Client Meeting.

Date and Time: April 1, 2015 – 9:30am

Chairperson: Kresimir Bukovac

Members Present: Peter Mavridis
Darryl Murphy
Ali Sayed

Agenda/Topic	Discussion	Action	Person Responsible
Client Meeting #2 – Questionnaire	Discussed what we wanted to ask in the next Client Meeting and whether there were any more questions to add to the list.	Added more questions.	Kresimir Bukovac Peter Mavridis Darryl Murphy
Storyboard #1 – Customer Bookings	Had a look at and agreed that the storyboard made after last week's Team Meeting showed what we wanted it to.	Present it to the client in the next Client Meeting.	Darryl Murphy
Prototype of the Report Menu	Had a look at and agreed that the prototype made after last week's Team Meeting showed what we wanted it to.	Present it to the client in the next Client Meeting.	Ali Sayed
Class Diagram	Had a look at and agreed that the Class Diagram made after last week's Team Meeting showed what we wanted it to.	Present it to the client in the next Client Meeting.	Kresimir Bukovac
Journals	Everyone agreed to compile their journals together into the group Journal whenever they had done any work on the project.	Update Journal.	Kresimir Bukovac Peter Mavridis Darryl Murphy Ali Sayed
Graphical User Interface (GUI)	Discussed the pros and cons of implementing a GUI.	Learn to do Qt in Visual Studio.	Peter Mavridis

Code	Generally discussed how the system will be designed and what files/procedures will be needed.	Create skeleton code files.	Kresimir Bukovac
		Design prototypes in the program.	Ali Sayed
Documentation	Talked about all the documentation we currently have and what still needs to be done or expanded on.	Work on documentation.	Kresimir Bukovac Darryl Murphy

Documents/Deliverables:

- Client Meeting #2 - Questionnaire
- Storyboard #1 - Customer Bookings
- Prototype of the Report Menu
- Class Diagram

Closing Time: 10:30am

10.1.5. Memorandum #3

To: Peter Mavridis – GUI Programmer/Chief Tester
Darryl Murphy – Chief Analyst/Designer
Ali Sayed – Chief Programmer

From: Kresimir Bukovac – Manager + Chief Analyst/Designer

Date: April 14, 2015

Subject: Confirm the work allocated from Team Meeting #2 was done, set new tasks to be completed and prepare for Client Meeting #3.

On Wednesday, the 1st of April, 2015, the team had Team Meeting #2. The Meeting Minutes has been put onto github. Tomorrow's Team Meeting will be to make sure everyone has done their assigned tasks from Team Meeting #1, to assign new tasks to be completed by the next Team Meeting and to prepare our presentations and questions for the next Client Meeting.

You must bring a copy of Team Meeting #2's Meeting Minutes with you to tomorrow's Team Meeting in order for everyone to go through and confirm that all the assigned tasks have been completed. You must also bring any deliverables you have made or edited since the last Team Meeting to show the rest of the team to confirm and agree on. Also, bring any plans or documentations you have made in regards to the project or for the next Client Meeting.

If there are any questions as to this memo, the Team Meeting, or you are unable to attend the Team Meeting, you can contact me on the following:

Email: kb100@uowmail.edu.au
Mobile: 0410 410 411

10.1.6. Meeting Minutes #3

Name of Organisation: Group 4

Purpose of Meeting: To make sure everything from the previous team meeting has been completed and to assign new tasks as well as discuss our design.

Date and Time: April 15, 2015 – 9:30am

Chairperson: Kresimir Bukovac

Members Present: Peter Mavridis
Darryl Murphy
Ali Sayed

Agenda/Topic	Discussion	Action	Person Responsible
Team Meeting #2 – Assigned tasks	All of the tasks were prepared on time and the deliverables were shown to the client in Client Meeting #2. Journals are being updated whenever work is done. The skeleton code files and program prototypes were created. The documentation is still being worked on.	NONE	NONE
Class Diagram and skeleton code files.	After meeting with the client in Client Meeting #2, the requirements were more clearly established, and with a few fixes to our initial design, we will edit our design to fit the new requirements.	Revise and re-design the Class Diagram and skeleton code files.	Kresimir Bukovac Darryl Murphy
Journals	Everyone agreed to compile their journals together into the group Journal whenever they had done any work on the project.	Update Journal.	Kresimir Bukovac Peter Mavridis Darryl Murphy Ali Sayed

Graphical User Interface (GUI)	With the complications of training and time needed to learn and implement a GUI with Qt, the team has decided to work strictly on a command-line-based system. However, if the project goes smoothly, a GUI may be adapted at a later date when the core requirements have been implemented.	Implement a Command-line system.	Kresimir Bukovac Peter Mavridis Darryl Murphy Ali Sayed
Code	Generally discussed how the system will be designed and what we will need to implement.	NONE	NONE
Documentation	Talked about all the documentation we currently have and what still needs to be done or expanded on.	Compile Requirements and work on Risk Management.	Kresimir Bukovac
		Develop a Business Case.	Peter Mavridis
		Work on System Requirements Specification and architectural design.	Darryl Murphy
		Develop use cases and detailed plans.	Ali Sayed

Documents/Deliverables:

- Client Meeting #2 – Meeting Minutes
- Class Diagram
- Skeleton code files
- Program prototypes
- Data Dictionary

Closing Time: 10:30am

10.1.7. Memorandum #4

To: Peter Mavridis – GUI Programmer/Chief Tester
Darryl Murphy – Chief Analyst/Designer
Ali Sayed – Chief Programmer

From: Kresimir Bukovac – Manager + Chief Analyst/Designer

Date: April 21, 2015

Subject: Confirm the work allocated from Team Meeting #3 was done, set new tasks to be completed and prepare for Client Meeting #3.

On Wednesday, the 15th of April, 2015, the team had Team Meeting #3. The Meeting Minutes has been put onto github. Tomorrow's Team Meeting will be to make sure everyone has done their assigned tasks from Team Meeting #2, to assign new tasks to be completed by the next Team Meeting and to finalise the documentation we currently have to present at the Client Meeting.

You must bring a copy of Team Meeting #3's Meeting Minutes with you to tomorrow's Team Meeting in order for everyone to go through and confirm that all the assigned tasks have been completed. You must also bring any deliverables you have made or edited since the last Team Meeting to show the rest of the team to confirm and agree on. Also, bring any plans or documentations you have made in regards to the project or for the next Client Meeting.

If there are any questions as to this memo, the Team Meeting, or you are unable to attend the Team Meeting, you can contact me on the following:

Email: kb100@uowmail.edu.au
Mobile: 0410 410 411

11. Work Diaries

11.1. Diary - Kresimir Bukovac

Date	4/03/2015
Action	Gathered together a group of four skilled individuals.
Time Expected	2 weeks.
Time Taken	3 days.
Problems	I needed to find people with specific skills to make a well-rounded team. I would like to have one more member on the team.
Solutions	I analysed the project beforehand and declared how many members we needed to fill each role. I will look for a fifth person as soon as possible.

Date	4/03/2015
Action	I emailed out the contact details and method to all of the group members.
Time Expected	2 minutes.
Time Taken	2 minutes.
Problems	None.
Solutions	None.

Date	4/03/2015
Action	Briefly discussed possible paths and options for the project based off of what we currently know is required of us with the other team members.
Time Expected	10 minutes.
Time Taken	5 minutes.
Problems	We don't know much about the project at the current date.
Solutions	We will speak with the client next week.

Date	11/03/2015
Action	I found a fifth person to join the team as a chief programmer.
Time Expected	2 weeks.
Time Taken	3 days.
Problems	None.
Solutions	None.

Date	11/03/2015
Action	I updated the group member list and distributed it to all of the members in our group chat on Skype.
Time Expected	1 minute.
Time Taken	1 minute.
Problems	None.
Solutions	None.

Date	12/03/2015
Action	I organised with the group for our weekly team meetings to be at 3:30pm on Tuesdays after the first meeting with the client.
Time Expected	10 minutes.
Time Taken	5 minutes.
Problems	There were many timetable clashes between the team members.
Solutions	We had to check all of our available times and Timothy had to change their plans to fit with everyone else's availability.

Date	13/03/2015
Action	I designed a questionnaire and gave it to the rest of the team to add their questions to for the client meeting.
Time Expected	10 minutes.
Time Taken	20 minutes.
Problems	I didn't know much about the project at all.
Solutions	I designed the questionnaire to have general questions on which we could expand on during the client meeting.

Date	14/03/2015
Action	I began working on the project documentation for analysis and feasibility studies.
Time Expected	2 hours.
Time Taken	1 hour 30 minutes.
Problems	There was not much information to work off.
Solutions	I wrote simple documents on what I know the project was about.

Date	17/03/2015
Action	Due to time clashes with one of the members' other business and the time of the meeting, I had to change the team meeting time to 12:30pm every Tuesday.
Time Expected	10 minutes.
Time Taken	5 minutes.
Problems	I had to re-evaluate everyone's schedules once again and this issue may occur again.
Solutions	I picked a time that seemed to fit everyone's timetable the best.

Date	18/03/2015
Action	I established a Github repository and told the other team members to create accounts for me to invite them into the repository.
Time Expected	5 minutes.
Time Taken	15 minutes.
Problems	It was the first time I had ever used Github.
Solutions	I learnt how to work Github and invite people into the team repository.

Date	18/03/2015
Action	Due to new clashes with the time of the meeting, I requested everyone to send me an updated timetable, so that I could re-evaluate the team meeting time.
Time Expected	1 day.
Time Taken	3 days.
Problems	There was some delay in getting a response from all of the team members online.
Solutions	I had to bring the topic up in person with the team members.

Date	18/03/2015
Action	We had a client meeting as a team to gather the requirements for the project.
Time Expected	10 minutes.
Time Taken	25 minutes.
Problems	Our questions were too general.
Solutions	We expanded on our questions to gather more specific requirements as the meeting progressed and we learnt more about the project.

Date	18/03/2015
Action	We took notes on the requirements and project specifications during the client meeting.
Time Expected	10 minutes.
Time Taken	25 minutes.
Problems	The meeting was fast-paced.
Solutions	We all took notes to be compiled together at the end.

Date	19/03/2015
Action	With the information we got from the client, I estimated how long the project would take to complete in person-months.
Time Expected	5 minutes.
Time Taken	5 minutes.
Problems	There is no accurate way to guess the lines of code that will be produced.
Solutions	With my experience in programming, I thought of what functions would be required in the system and how big they would be to get a rough estimate.

Date	23/03/2015
Action	I wrote up a memo for tomorrow's team meeting and sent it out to all of the team members.
Time Expected	25 minutes
Time Taken	15 minutes.
Problems	None.
Solutions	None.

Date	24/03/2015
Action	I compiled all of the team members' notes on the client meeting and uploaded it to Github.
Time Expected	1 hour.
Time Taken	35 minutes.
Problems	Github uses text files.
Solutions	I had to re-format the document to look nice.

Date	24/03/2015
Action	I set up all of the documentation files that will most likely be needed throughout the project's life on Github for the entire team to have access to and be able to add to.
Time Expected	5 minutes.
Time Taken	3 minutes.
Problems	None.
Solutions	None.

Date	24/03/2015
Action	We had a 3-man (Kresimir/Darryl/Ali) team meeting to discuss what we know of our requirements so far and begin planning for design possibilities and our next client meeting.
Time Expected	30 minutes.
Time Taken	45 minutes.
Problems	We were still in shallow waters in terms of what was required.
Solutions	We made templates of what could be required of us for us to use later on.

Date	24/03/2015
Action	After discussing with Darryl and Ali what tasks we would assign the other two members would do, I sent them an email, telling them to read all the products of this meeting and develop further questions.
Time Expected	5 minutes.
Time Taken	5 minutes.
Problems	None.
Solutions	None.

Date	29/03/2015
Action	I had to slightly re-assign team roles because Timothy left the team.
Time Expected	10 minutes.
Time Taken	2 minutes.
Problems	None.
Solutions	None.

Date	31/03/2015
Action	I made small changes to the layout of the files on Github.
Time Expected	10 minutes.
Time Taken	30 minutes.
Problems	I am still unfamiliar with Github and the functionality of Github seems to be really restricted.
Solutions	I learnt more about Github and then made changes to make working with the files easier for the team members.

Date	31/03/2015
Action	The team members all had a brief chat and we mutually agreed to have our weekly meetings at 9:30am on Wednesdays.
Time Expected	2 minutes.
Time Taken	2 minutes.
Problems	None.
Solutions	None.

Date	31/03/2015
Action	I thought of questions to ask the client in tomorrow's client meeting and put them on Github for others to add on to.
Time Expected	10 minutes.
Time Taken	5 minutes.
Problems	We only had general requirements specified to us.
Solutions	I based my questions on the requirements we received in the previous client meeting to ask for more specific requirements.

Date	31/03/2015
Action	I wrote up a memo for tomorrow's team meeting and sent it out to all of the team members.
Time Expected	25 minutes
Time Taken	15 minutes.
Problems	None.
Solutions	None.

Date & Time	01/04/2015
Action	We conducted our second team meeting, showing the other members what work we had done since the last team meeting in order to gain suggestions and/or acceptance of the quality of our work.
Time Expected	1 hour.
Time Taken	1 hour.
Problems	None.
Solutions	None.

Date	01/04/2015
Action	I created and uploaded a class diagram of the system to Github.
Time Expected	30 minutes.
Time Taken	1 hour 10 minutes.
Problems	It was my first time using ArgoUML, so I had difficulties with specific aspects of the design of the class diagram.
Solutions	I eventually figured out the program and was able to slowly design the class diagram.

Date & Time	01/04/2015
Action	I reminded everyone to keep a constant diary of all their project-related work whenever they do anything.
Time Expected	20 seconds.
Time Taken	20 seconds.
Problems	None.
Solutions	None.

Date & Time	01/04/2015
Action	We conducted our second client meeting and showcased all of our plans to receive feedback and gather more specific requirements.
Time Expected	30 minutes.
Time Taken	35 minutes.
Problems	None.
Solutions	None.

Date	02/04/2015
Action	I compiled the notes the team had taken during the second team meeting and uploaded it to Github.
Time Expected	30 minutes.
Time Taken	35 minutes.
Problems	There were a few discrepancies in our notes.
Solutions	I spoke with the team members in order to clarify what the requirements were.

Date	04/04/2015
Action	I created skeleton files with all of the classes and/or procedures for all of our code and uploaded them to Github.
Time Expected	1 hour.
Time Taken	35 minutes.
Problems	The skeleton files are just a start to the implementation of the project and are subject to change.
Solutions	I made the skeleton to the best of my ability so that in the future when the requirements change, the files can be edited as easily as possible.

Date	09/04/2015
Action	I revised and re-designed the class diagram to better suit the requirements.
Time Expected	45 minutes.
Time Taken	30 minutes.
Problems	The class diagram I made encompassed the diamond of doom.
Solutions	I looked at ways to solve the problem but could not find anything apart from removing some generalisations or using virtual functions in c++.

Date	09/04/2015
Action	I re-designed the skeleton code files to feature the changes I had made in the class diagram.
Time Expected	45 minutes.
Time Taken	30 minutes.
Problems	The class diagram I made encompassed the diamond of doom.
Solutions	I implemented the use of virtual functions into the class skeletons.

Date	13/04/2015
Action	I added more details to the feasibility study based off any new information I had become aware of.
Time Expected	30 minutes.
Time Taken	40 minutes.
Problems	None.
Solutions	None.

Date	14/04/2015
Action	I wrote up a memo for tomorrow's team meeting and sent it out to all of the team members.
Time Expected	25 minutes
Time Taken	15 minutes.
Problems	None.
Solutions	None.

Date	15/04/2015
Action	We conducted our third team meeting, showing the other members what work we had done since the last team meeting in order to gain suggestions and/or acceptance of the quality of our work.
Time Expected	1 hour.
Time Taken	1 hour.
Problems	None.
Solutions	None.

Date	15/04/2015
Action	I assigned roles to each of the other members, detailing what needed to be done by next week's lecture.
Time Expected	10 minutes.
Time Taken	7 minutes.
Problems	None.
Solutions	None.

Date	18/04/2015
Action	I finalised the risk analysis and management plans for the project and the format of the document.
Time Expected	2 hours.
Time Taken	1 hour 15 minutes.
Problems	None.
Solutions	None.

Date	19/04/2015
Action	I reminded each member of their job and checked their progress on what they were assigned to do.
Time Expected	10 minutes.
Time Taken	10 minutes.
Problems	None.
Solutions	None.

Date	20/04/2015
Action	I gathered everyone's finished documentation.
Time Expected	5 minutes.
Time Taken	3 minutes.
Problems	None.
Solutions	None.

Date	21/04/2015
Action	I wrote up a memo for tomorrow's team meeting and sent it out to all of the team members.
Time Expected	25 minutes
Time Taken	15 minutes.
Problems	None.
Solutions	None.

Date	21/04/2015
Action	I compiled everyone's documentation into one report.
Time Expected	2 hours 30 minutes.
Time Taken	5 hours.
Problems	Everyone's document had a completely different format. Reformatting and making a contents page.
Solutions	I had to go through each document and make it as consistent as possible. This was an extremely lengthy and delicate process.

11.2. Diary – Ali Sayed

Date	16/3/2015
Action	Group meeting after lecture. We all collaborated our gathered system requirements.
TimeExpected	30mins
TimeTaken	45mins
Problems	/
Solutions	/

Date	25/3/2015
Action	Group meeting before client meeting. We decided what questions we needed to ask the clients. I proposed we get the reporting function sorted, and that I would create a prototype to the client and see what their input was.
TimeExpected	1hr
TimeTaken	1hr
Problems	/
Solutions	/

Date	25/3/2015
Action	Created a reporting prototype (reporting.cpp) - quick program which generates static reports. Presented this to the clients. They said we need more 'interesting facts' in the report, not just things that they can find by looking at the database.
TimeExpected	20mins
TimeTaken	40mins
Problems	/
Solutions	/

Date	1/4/2015
Action	Group meeting. Kresimir assigned jobs to each member, mine was to get started on the project plan and create a gantt chart to illustrate the plan.
TimeExpected	1hr
TimeTaken	1hr
Problems	/
Solutions	/

Date	9/4/2015
Action	Evaluated different project planning schemes, created the gantt chart to display the project plan.
TimeExpected	1hr
TimeTaken	2hr
Problems	Experienced a few problems. Most of the trouble was how to illustrate each iteration in the project plan; I think this is pretty ineffective with a gantt chart.
Solutions	Tell the team that an RUP approach illustrated through area charts is going to be a better approach.

Date	15/4/2015
Action	Meeting with all group members. We evaluated what work was left remaining and assigned jobs suitably to each member. I was assigned to finish the use case diagrams and also publish the project plan.
TimeExpected	1hr
TimeTaken	1hr
Problems	/
Solutions	/

Date	16/4/2015
Action	Go back over lecture notes of Rational Unified Process (RUP)
TimeExpected	1hr
TimeTaken	45mins
Problems	/
Solutions	/

Date	17/4/2015
Action	Created the RUP project plan.
TimeExpected	2hr
TimeTaken	3hr
Problems	Had to use separate graphs for each phase
Solutions	Used Excel's area graphing function

Date	19/4/2015
Action	Create use case diagrams. I already had some of these drafted from earlier, but a lot of them weren't finished.
TimeExpected	3hr
TimeTaken	2hr 30mins
Problems	Struggled to find an effective way to do this on PowerPoint or Word. Used Creately but couldn't save unless I created an account.
Solutions	Used Creately.com (web), and used screen capture function.

11.3. Diary – Peter Mavridis

Date	04-03-2015
Action	Was asked to join the group by Kresimir as a chief programmer
Time Expected	N/A
Time Taken	N/A
Problems	N/A
Solution	N/A

Date	04-03-2015
Action	Read through the assignment 1 pdf Started to take my own notes on what is needed.
Time Expected	30 minutes
Time Taken	30 minutes
Problems	None
Solution	N/A

Date	05-03-2015
Action	Started looking in to using a GUI for the assignment first thought is to use Qt, will bring it up in our next team meeting.
Time Expected	2 hours
Time Taken	2 hours
Problems	Installing Qt on a Mac is a little troublesome.
Solution	Ended up downloading from Qt a package that comes with an IDE called Qt Creator.

Date	20-03-2015
Action	Thought of some questions to ask about if a GUI was to be used or if command line tool is sufficient.
Time Expected	30 minutes
Time Taken	1 hour
Problems	Distracted by other subjects work.
Solution	Better manage my time with calendar entries.

Date	24-03-2015
Action	Had a group meeting where I brought up about using a GUI in particular Qt. Helped with the setup of Github during tutorial.
Time Expected	15 minutes
Time Taken	15 minutes
Problems	Working out how to invite others to the repository.
Solution	Read the help documentation and online tutorials.

Date	02-04-2015
Action	Started going through Qt tutorials on creating applications.
Time Expected	2 hours
Time Taken	4 hours
Problems	No real problem I allocated only 2 hours but got involved with some of the tutorials and watching videos on Qt
Solution	N/A

Date	05-04-2015
Action	Want to create a basic application in Qt where a user can register and login to an account.
Time Expected	2 hours
Time Taken	1 hour.
Problems	I needed a database for Qt to play nice.
Solution	I created a free account with Enigio which is a cloud service and acts as a backend that keeps a database for users login accounts.

Date	10-04-2015
Action	Coding for registering and logins of users
Time Expected	1 hour
Time Taken	30 minutes
Problems	Internet issues
Solution	Had to wait for my network to return at home

Date	15-04-2015
Action	Create a business case
Time Expected	4 hours
Time Taken	6 hours
Problems	Took a little longer than expected trying to find exactly what is needed and what is relevant for our assignment.
Solution	Read lecture notes and Google.

Date	18-04-2015
Action	More coding in the GUI environment
Time Expected	4 hours
Time Taken	4 hours
Problems	Still some trouble with the backend for user accounts
Solution	More research on Qt documentation to make it work.

Date	20-04-2015
Action	Have finally successfully created the backend database for Qt GUI to have users register and login
Time Expected	30 minutes
Time Taken	1 hour
Problems	I was using the wrong ID associated with the database.
Solution	Use the right ID.

11.4. Diary – Darryl Murphy

Planned work schedule

The work schedule I intend to implement will be determined in an 'hours per week' unit of measure. Simultaneously, my schedule will be broken up into 3 phases each requiring a different amount of hours to complete.

Phase 1: Requirements elicitation.

Phase 2: Design and analysis.

Phase 3: Implementation and testing.

Phase 1 work schedule:

Requirements elicitation is comprised mainly of analysing some preliminary descriptions of the system, and generating questions to ask the client in order to get specific requirements for the system. I anticipate this will require up to 5 hours a week to complete.

Planned work times:

Monday 12:00pm – 2:00pm

Tuesday 4:30pm – 6:30pm

Thursday 8:00pm – 9:00pm

Phase 2 work schedule:

Given the role I am assigned, the design and analysis phase will be the most time consuming phase of the project. Design and analysis will see me creating Architectural Designs and the System requirements specifications. I anticipate I will require at least 6 hours a week to complete this phase.

Planned work times:

Tuesday 4:30pm – 6:30pm

Wednesday 1:00pm – 3:00pm

Thursday 7:00pm – 9:00pm

Phase 3 work schedule:

My role during the implementation phase will be to oversee the programming of the project, and ensuring it meets the design specifications. Given the size of the project this phase will also see me contributing a decent amount of coding time in the process. I anticipate this phase will require at least 6 hours work per week.

Planned work times:

Monday 12:00pm – 2:00pm

Tuesday 4:30pm – 6:30pm

Thursday 7:00pm – 9:00pm

Work Diary Phase 1

5/3

Joined group and assigned the role of analyst and designer.

Work time: Thursday 12:30pm – 1:30pm

Work done: Read through assignment specifications.

Expected Duration: 30mins.

Difficulties: The description of the project to be built is very limited. Understanding exactly what work I will need to do to fulfil my role other than use cases and a domain model (e.g. the assignment states it requires “Detailed plans for the whole project”, this seems ambiguous to me at this point in time).

Solution: Studying the roles of analyst/designer in software engineering.

10/3

Work time: Tuesday 5:30pm -6:30pm

Work Done: Compiled questions about project for client.

Expected Duration:1 hour.

Difficulties: Having never done requirements elicitation before, it has been difficult to define what an effective question would be. The questions established today relate mainly to user interface, persistent data, sub systems and actors within the system.

Solution: Decided to stick to more general questions at this point in time. For example whether a gui is needed, etc.

13/3

Work time: Friday 10:30am – 11:00am

Work Done: Compiled questions about project for client. Further added to question list for client meeting. I looked at the ‘Flight DB’ excel file that was provided.

Expected Duration:1 hour.

Difficulties: This resulted in some questions about what some attributes are used for, some are not labelled. Need client clarification.

Solution: get client to clarify ambiguous parts of database.

18/3

Work time: Wednesday 10:30am – 12:30pm

Work Done: Client meeting. Project Clarifications.

Expected Duration:2 hours.

Difficulties: Note taking. Questions were answered very quickly by client which made it hard to keep up. Answers often lead to new questions which made it harder to keep on track with the questionnaire.

Solution: Make sure everyone in group is taking notes during client meetings to take pressure off the person asking questions.

23/3

Work time: Monday 11:30pm – 12:30pm

Work Done: Used provided 'Flight DB' database file to draw up preliminary class structures. The connections between classes in the database with the database were identified. Some rough sketches of the domain model were created to better understand what this system may require.

Expected Duration: 30 mins.

Difficulties: Identifying what needs to be shown in the domain model. For example does the database class or sub system classes need to be identified here.

Solution: Talk to tutor/lecturer for guidance.

24/3

Work time: Tuesday 3:30pm – 4:30pm

Work Done: group meeting

Expected Duration: 1 hour.

Difficulties: Explaining ideas to group members. I feel the rough sketches I have were not enough to fully explain my thoughts on the structure of the program.

Solution: I will need more preparation before presenting ideas so to maximise understanding in the group. Draw clearer diagrams.

30/3

Work time: Monday 12:30pm – 1:30pm

Work Done: Compiled list of actor responsibilities for clarification with client. We need to know exactly what each actor can or cannot do.

Expected Duration: 30mins.

Difficulties: none.

Solution: none.

31/3

Work time: Tuesday 8:30pm – 10:30pm

Work Done: Created use case scenario of a customer booking a flight. The use case consists of a text file that shows the information/screens a user will see during a booking.

Expected Duration: 1 hour.

Difficulties: A few assumptions had to be made about the system during creation. For example the need to book return flights, the need to book for adult/child/infant specifications, display of calendars during booking. Needs further clarification with the client.

Solution: Needs further clarification with the client. They will be able to identify what they need/want during this process.

1/4

Work time: Wednesday 10:30am – 1:00pm

Work Done: Client meeting. Requirements elicitation. Walk through of use case scenario.

Expected Duration: 2 hours.

Difficulties: Fire alarm triggered. Took some extra time to talk to client.

Solution: Client took extra time out of lab to address our questions.

Phase 1 Summary

Phase one consisted of 9 work periods. 2 of which were on schedule or very close to schedule. Clearly, being on schedule only 22% of the time is a poor statistic. One of 2 things needs to happen to rectify this situation: Change the schedule, or be more rigorous in sticking to it. The work times suggest that my most consistent work times are Tuesday afternoons and Wednesday mornings/afternoons.

On average, I completed 2.2 hours of work per week. My estimates were up to 5 hours per week. This is a reasonably consistent estimate as at the most I worked 4.5 hours in a week. However, some weeks could of definitely used more hours of work.

My time estimates were often inaccurate. Sessions that ran over the estimate occurred 4 times. Sessions that were under the estimate occurred twice, and sessions that were correct to the estimate occurred twice. This reveals that 50% of the time, I underestimate the task at hand. I will need to allocate more time to tasks in the future.

Work Diary Phase 2

3/4

Work time: Friday 9:30am – 12:00pm

Work Done: Study of what is involved in 'Design'. Decided I will need to create an 'Architectural design document' to fully convey my design thought processes. The document will be split into 3 sections: Logical view, process view and implementation view.

Expected Duration: 1 hour.

Difficulties: Lack of understanding of architectural design. Intuitively I look for what diagrams are required for each view, however design requires using many of the same diagrams to portray different contexts of the system.

Solution: I need to identify what Information I want to convey, and what diagram will best aid me to do so.

8/4

Work time: Wednesday 1:00pm-2:00pm

Work Done: Defined work to be done in logical, process and implementation views of architectural design.

Expected Duration: 1 hour.

Difficulties: How to display some concepts with diagrams. For example, how to show how a user interacts with a subsystem which interfaces with the database etc. Will need to revise sequence diagrams, collaboration diagrams, activity diagrams, and component diagrams.

Solution: possibly break up class diagrams to portray the system in a clearer way. re-learning how to create the many diagrams needed for this document.

9/4

Work time:Monday 12:30pm – 2:30pm

Work Done:Created Data Dictionary

Expected Duration:2 hours.

Difficulties:Defining all the functions for each class is quite difficult as there are many interactions I may have not covered.

Solution: Continuously update the data dictionary as more of the system is designed/developed. Diagrams of the dynamic nature of the system will aid in identifying some needed functions.

10/4

Work time:Tuesday6:30pm – 7:30pm

Work Done:Created Domain model.

Expected Duration:1.5 hours.

Difficulties:Displaying All aspects of the system in one diagram(sub systems, databases etc).

Solution: I eventually decided to split the diagram into 3. Now it exists as a domain model, user – sub system view and sub system – database view. The 2 new views will be used to form the logical view.

11/4

Work time:Saturday 9:00pm – 11:00pm

Work Done:Created prototype user interface program. Purpose is to map out a rough implementation of the system to ensure ease of implementation before development phase.

Expected Duration:1.5 hours.

Difficulties:Vector implementation. In particular iterators and operator overloading.While this held up the implantation process, this was a minor difficulty.

Solution: Some extra study of vector iterator and operator overloading was required to implement parts of the database.

12/4

Work time:Sunday 3:30pm – 5:00pm

Work Done:Sketched sequence diagrams.

Expected Duration:2 hours.

Difficulties:Defining which use cases should be modelled. The models aim to portray how the system operates dynamically. So the use cases need to show as many components of the system as possible.

Solution: Focus mainly on booking functionality as they require a few sub systems to complete (reservation, profile, and service). Some less complicated use cases were chosen to display other sub system interactions.

13/4

Work time: Monday 9:00am – 10:00am

Work done: sketched dependency diagram

Expected Duration: 30mins.

Difficulties: Having never done a component diagram before, I had to do some research before sketching. Defining what parts of the system were important to be in the diagram.

Solution: Research and practice with component diagrams in argoUML. The primary dependencies will occur with the database system and its file imports. Also dependencies will occur with the sub-systems and the database.

15/4

Work time: Wednesday 1:00pm – 4:00pm

Work Done: Created Architectural Design Diagrams.

Expected Duration: 2 hours.

Difficulties: Had to install and use ArgoUML software. ArgoUML crashed and erased 2 diagrams during operation. ArgoUML has some ineffective functionality (creating self calls in sequence diagrams is frustrating) and missing functionalities (cannot display stick figure as actor in sequence diagrams.)

Solution: Re-create lost diagrams. Save work consistently during creation. Clearly labelling actors in sequence diagrams.

17/4

Work time: Friday 7:30pm – 10:30pm

Work Done: Completed Architectural Design, added explanations to the diagrams and sorted diagrams into views.

Expected Duration: 2 hours.

Difficulties: Finding the best way to explain the views. Ideally these diagrams will be used by my group members to implement the system. Therefore these explanations need to be as coherent as possible.

Solution: Constant re-evaluation of diagrams and explanations. Some explanations highlighted inconsistencies and missing components in diagrams.

18/4

Work time: Saturday 3:00pm – 4:00pm

Work Done: Compiled list of requirements in preparation of SRS.

Expected Duration: 1 hour.

Difficulties: none.

Solution: none.

18/4 – 19/4

Work time: Saturday/Sunday 9:30pm – 4:00 am

Work Done: Created SRS

Expected Duration: 4 hours.

Difficulties: Converting the summarized requirements I have in the list into one or more requirement tables. Some requirements that are not in the list needed to be added to the list.

Solution: Constant re-evaluation of requirements. One requirement in the list may need to be split into several distinct requirements in the SRS






















Phase 2 Summary

Phase two consisted of 11 work periods. 2 of which were on schedule or very close to schedule. This means I only worked on schedule 18% of the time, Even worse than phase 1. This may have occurred due to more documentation being required during this phase. Often I had to learn new documentation methods or re-study familiar methods. I have noted that dealing with unfamiliar methodologies can prolong my work times. The work times suggest that my most consistent work times during phase 2 were Friday mornings and Saturday afternoons/nights. This could be a 'last minute rush' mentality causing the bulk of work to be committed towards the end of the phase. I believe being proactive in overcoming the 'unfamiliar methodology' issue highlighted above will allow me to complete work in a more timely fashion.

On average, I completed 6 hours of work per week. My estimates were at least 6 hours per week. While this statistic is accurate to my estimate, My work times were heavily skewed to the last week of phase 2 (14.5 hours work). Again, this highlights the poor work practices in phase 2.

My time estimates in phase 2 were more accurate than phase 1. Sessions that ran over the estimate occurred 5 times. Sessions that were under the estimate occurred twice, and sessions that were correct to the estimate occurred 5 times. This reveals that 45% of the time, I underestimated the task at hand. However, some tasks in phase 2 were drastically underestimated. For example I estimated the SRS at 4 hours work, whereas it took me 6.5 hours to complete. Again, this issue boils down to underestimating tasks that I have little experience in performing. Identifying future tasks that I have not performed before will hopefully rectify many of the issues encountered in phase 2.

12. Versioning Evidence

 Client Meeting #1 - Questionnaire....	Updated and added new documentation	13 days ago
 Client Meeting #1 - Requirements....	Updated and added new documentation	13 days ago
 Client Meeting #2 - Questionnaire....	Updated and added new documentation	13 days ago
 Client Meeting #2 - Requirements....	Updated and added new documentation	13 days ago
 CompiledRequirementsList.docx	Added SRS, Updated requirementsList	2 days ago
 DataDictionary.docx	Added Architectural Design Document. Modified DataDictionary.	4 days ago
 Feasibility Study.docx	Changed the files to word documents.	20 days ago
 Journal	Update Journal	20 days ago
 Project Analysis.docx	Changed the files to word documents.	20 days ago
 Project Design.docx	Changed the files to word documents.	20 days ago
 Project Plan.docx	Project Plan	4 days ago
 README.md	Initial commit	a month ago
 Rational Unified Process - Picture...	Added Team Meeting Documentation	3 days ago
 Rational Unified Process.xlsx	Added Team Meeting Documentation	3 days ago
 Risk Analysis and Management P...	Updated the Risk Analysis and Management Plan	2 days ago
 Storyboard #1 - Customer Bookin...	Updated and added new documentation	13 days ago
 SystemRequirementsSpecificatio...	Added personal journal, modified SRS and arch design	4 hours ago
 Team Meeting #1 - Meeting Minut...	Updated and added new documentation	13 days ago
 Team Meeting #1 - Memo.docx	Updated and added new documentation	13 days ago
 Team Meeting #2 - Meeting Minut...	Added Team Meeting Documentation	3 days ago
 Team Meeting #2 - Memo.docx	Updated and added new documentation	13 days ago

Contributors

Traffic

Commits

Code frequency

Punch card

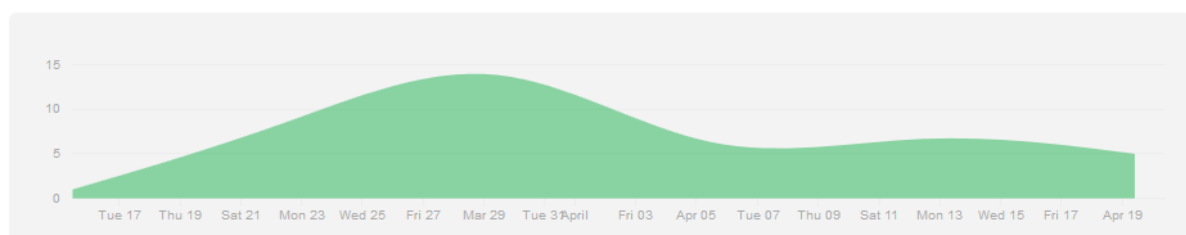
Network

Members

Mar 15, 2015 – Apr 20, 2015

Contributions to master, excluding merge commits

Contributions: **Commits** ▼

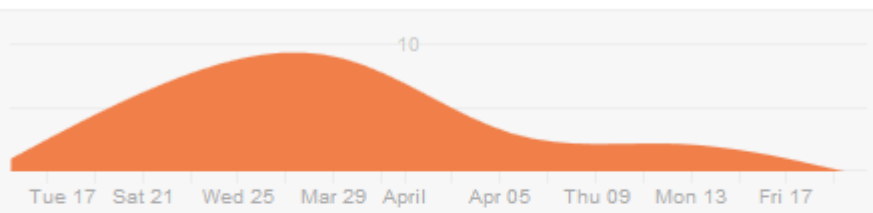




Pured

24 commits / 649 ++ / 294 --

#1



dazMurph90

9 commits / 0 ++ / 0 --

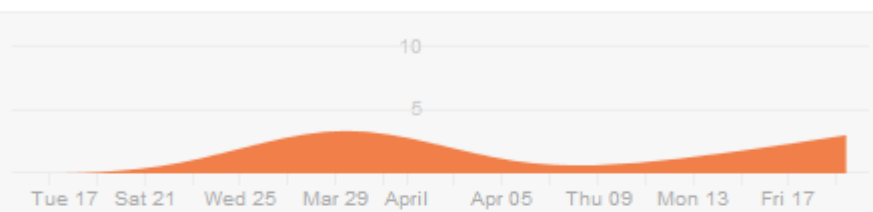
#2



PeterMav

9 commits / 13 ++ / 0 --

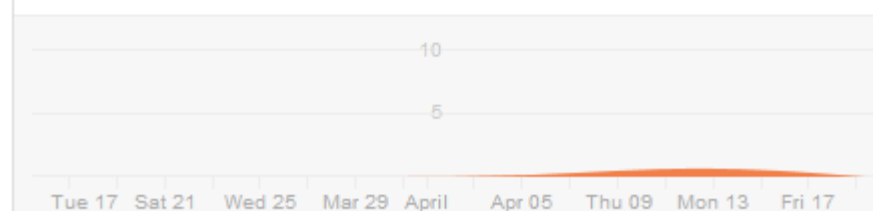
#3

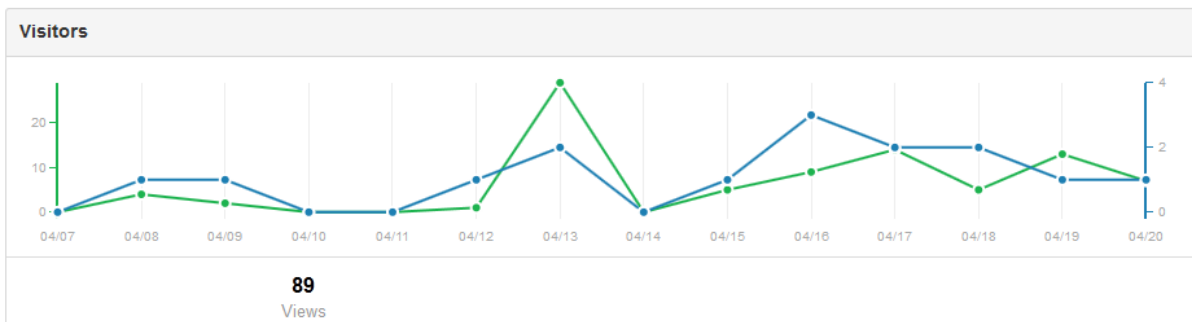


alisayed3

1 commit / 106 ++ / 0 --

#4





History



Added journal
1 hour ago by alisayed3



F
2 hours ago by PeterMav



Diary
2 hours ago by PeterMav



Merge branch 'master' of [https://github.co...](https://github.com)
3 hours ago by Darryl Murphy



Added personal journal, modified SRS and...
3 hours ago by Darryl Murphy



Updated
1 day ago by PeterMav



Merge branch 'master' of [https://github.co...](https://github.com)
1 day ago by Darryl Murphy



Added SRS, Updated requirementsList
1 day ago by Darryl Murphy



Merge branch 'master' of [https://github.co...](https://github.com)
2 days ago by Pured



Updated the Risk Analysis and Managemen...
2 days ago by Pured

