**Project Report**

**Group 4**

Version 1.01

(01/06/2015)

**Contents**

**1. Introductory Summary** (page 6)

**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. Meeting Minutes #4 (page 135)

10.2. Memorandum #5 (page 137)

10.3. Meeting Minutes #5 (page 138)

10.4. Memorandum #6 (page 140)

10.5. Meeting Minutes #6 (page 141)

10.6. Memorandum #7 (page 143)

10.7. Meeting Minutes #7 (page 135)

10.8. Memorandum #8 (page 137)

10.9. Meeting Minutes #8 (page 138)

10.10. Memorandum #9 (page 140)

10.11. Meeting Minutes #9 (page 141)

**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. Introductory Summary**

The purpose of this report is to document any changes and/or additions made to the project since the first report, which was delivered on Tuesday, the 22nd of March. This report is a continuation to the previous report, so everything in the previous report still stands, unless this report specifies otherwise.

**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 SystemSoftware 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 that 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 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 bases 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’sflight 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 teir 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 successfullycloses 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 customers 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 consists 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 sholdbe 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** |

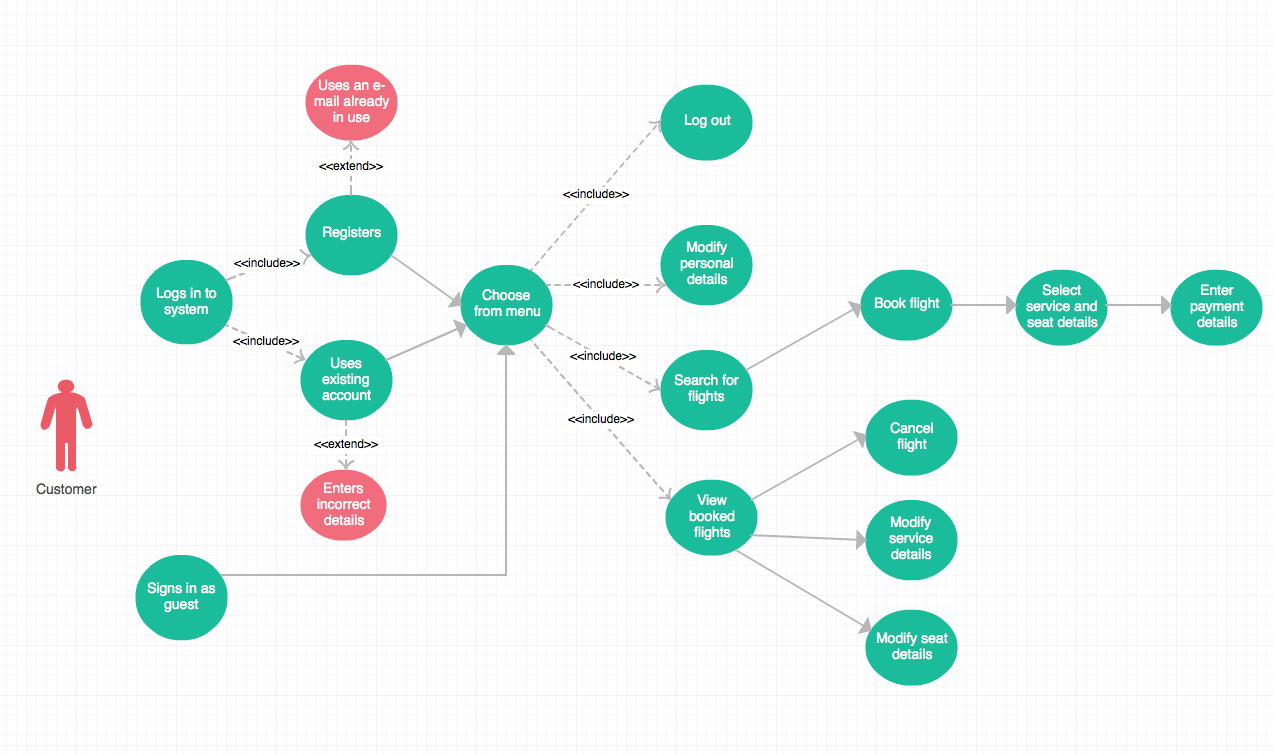
**5.3.3. Non-Functional Requirements**

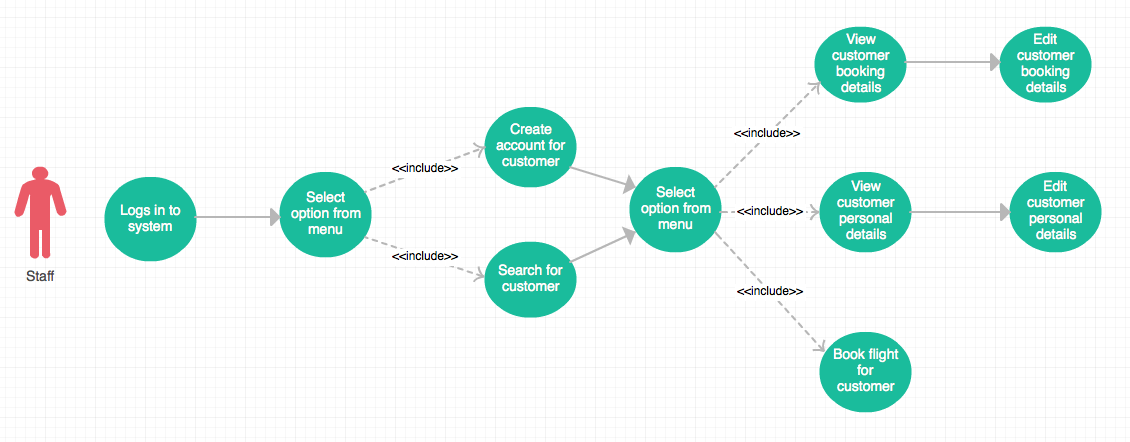
This section includes all functions that are defined as non-functional requirements.

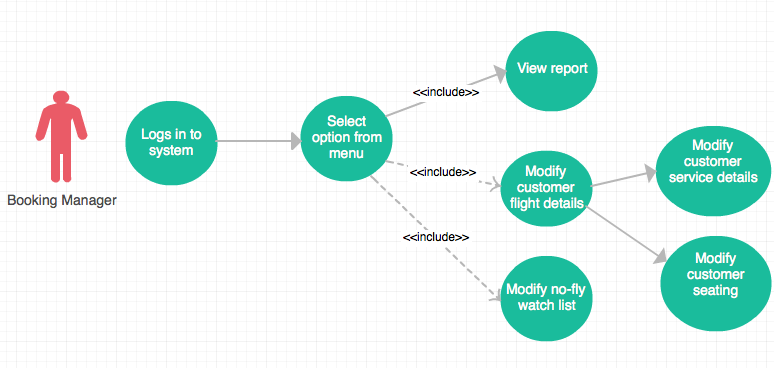
|  |  |  |
| --- | --- | --- |
| Requirement# NF\_1 | Requirement Type: NonFunctional | Event/Use Case: N/A |
| Description: The system should operate on both Windows and Linux systems. | | |
| Rationale: Users may use system on either linux or windows architectures. | | |
|
| Originator: none | | |
| 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** |

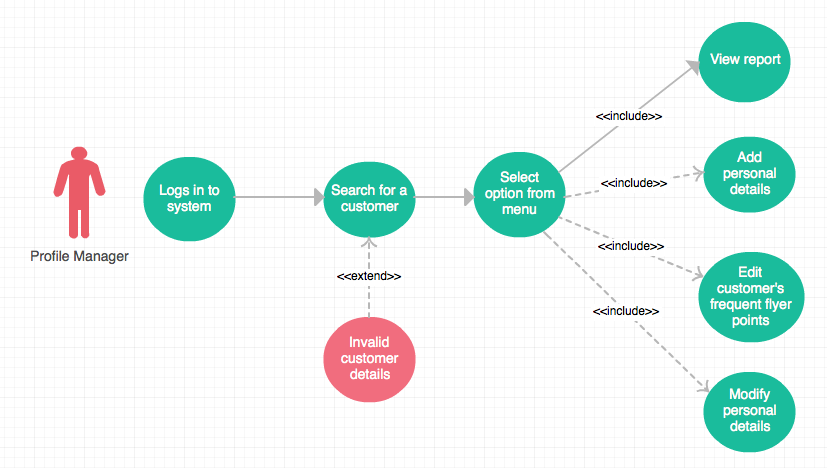
**6. Use Case Diagrams**

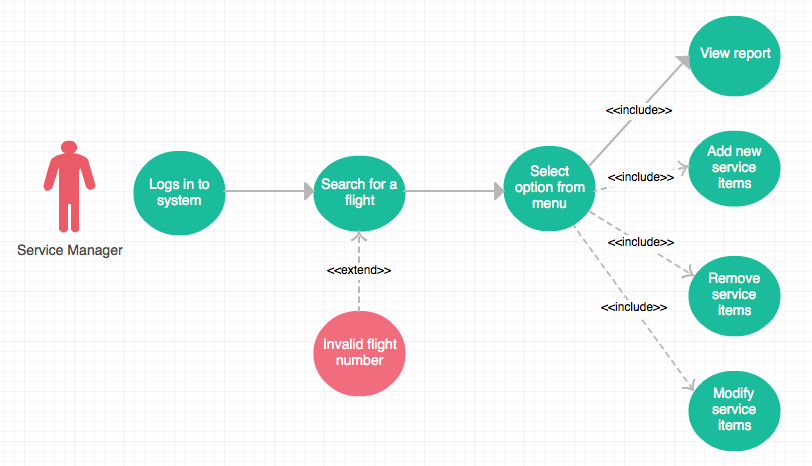


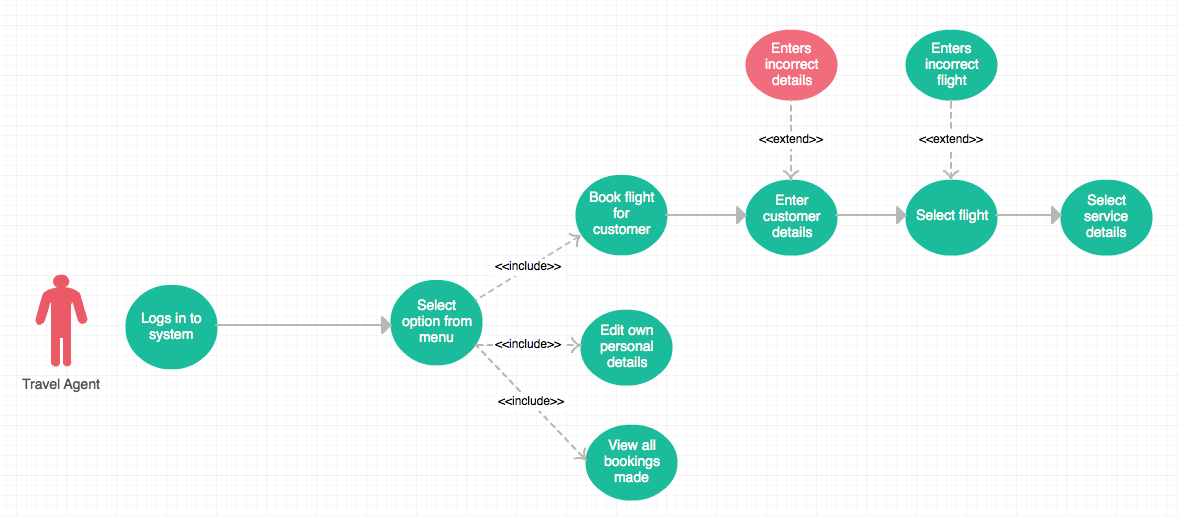












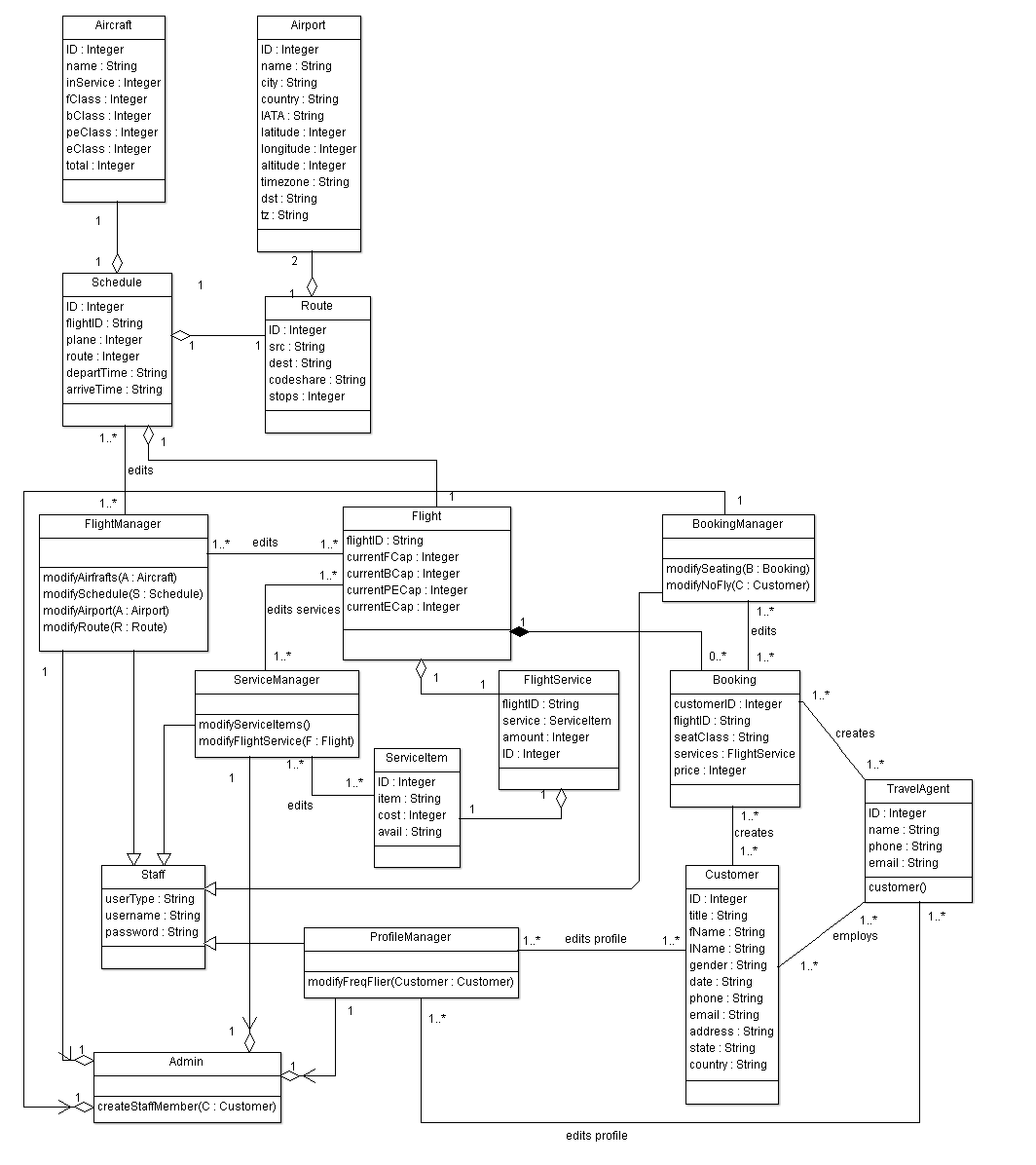
**7. Architectural Design Document**

**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).

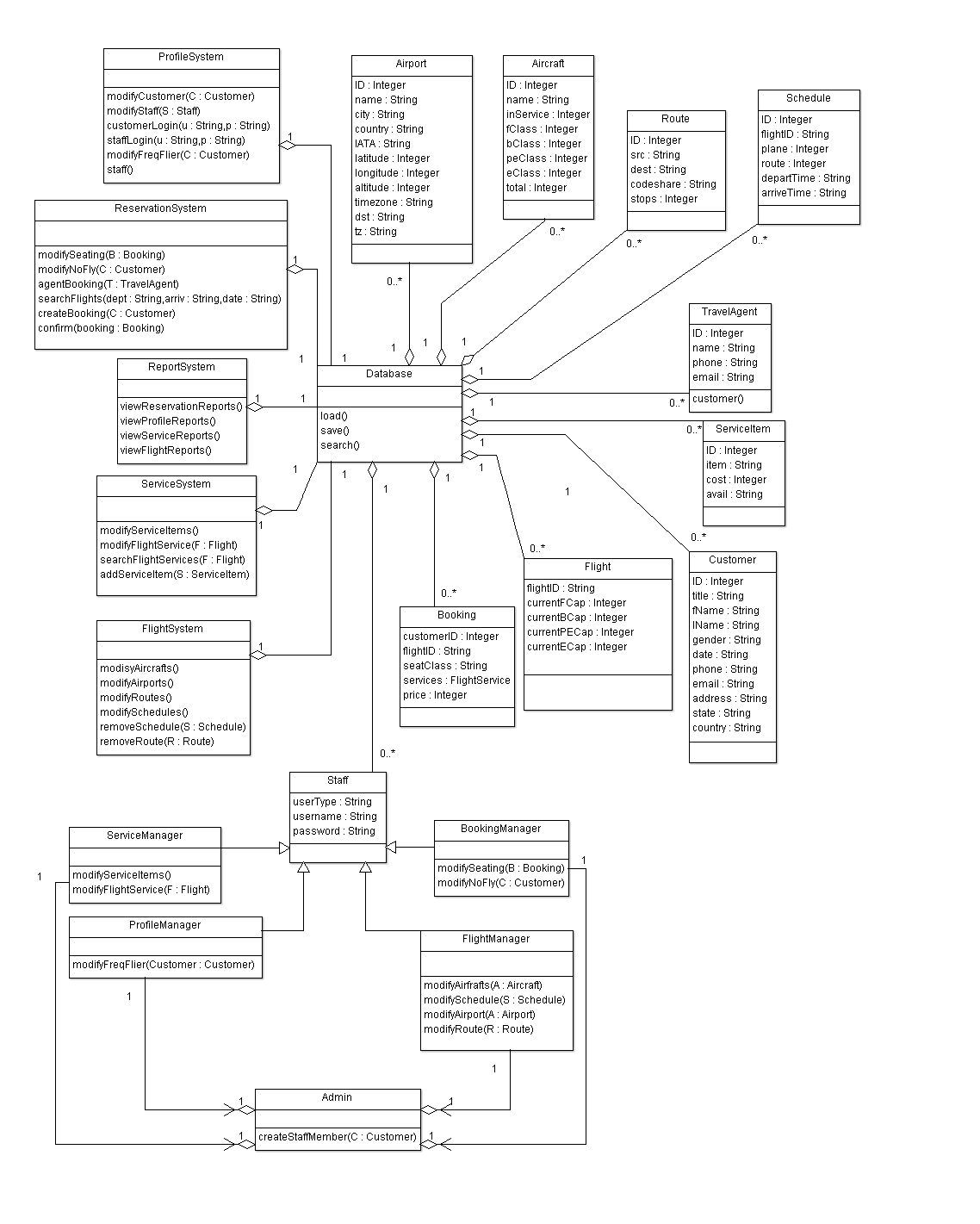
**7.2. Domain Model**

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 be seen below.



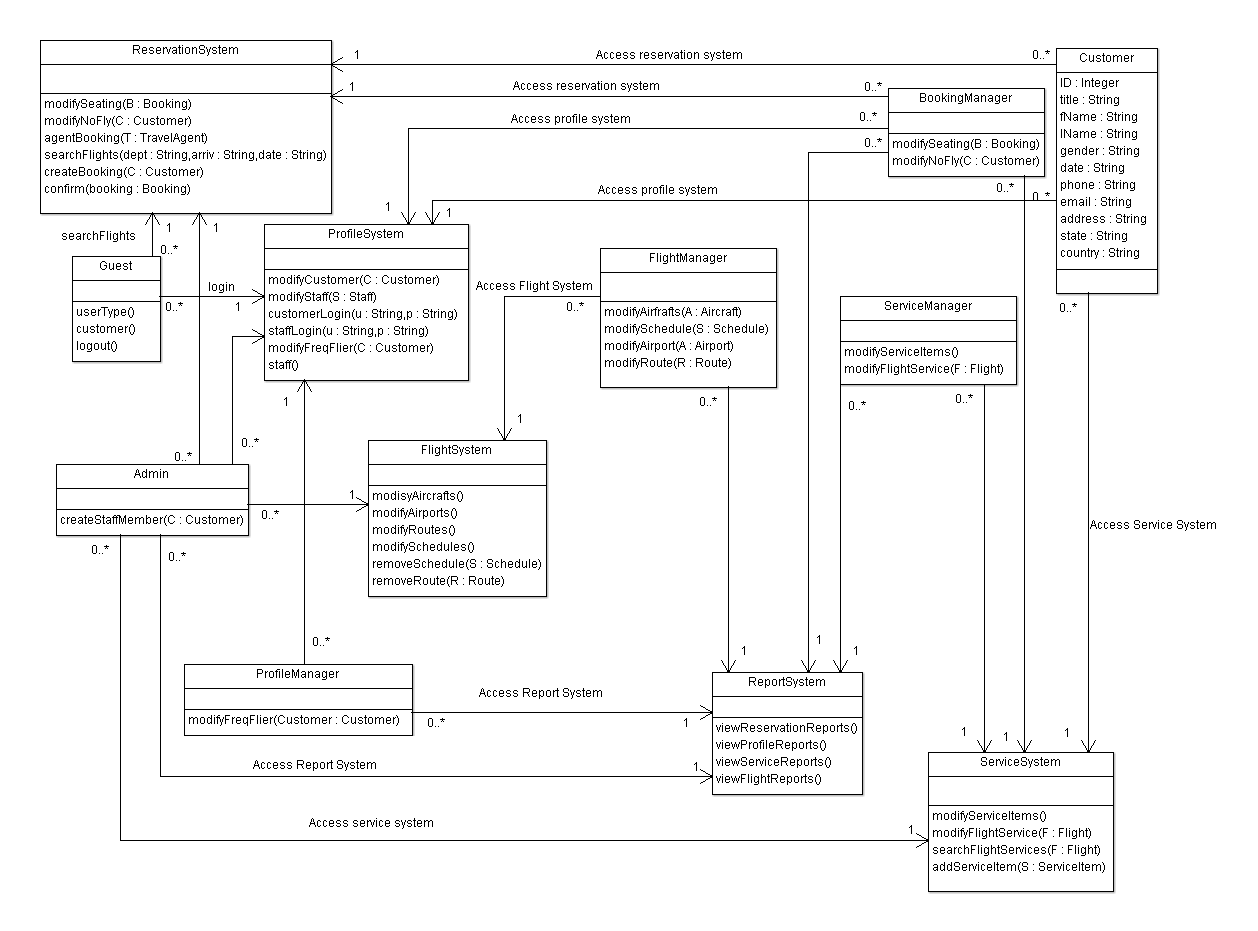
**7.2.1. Sub-System and Database View**

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.



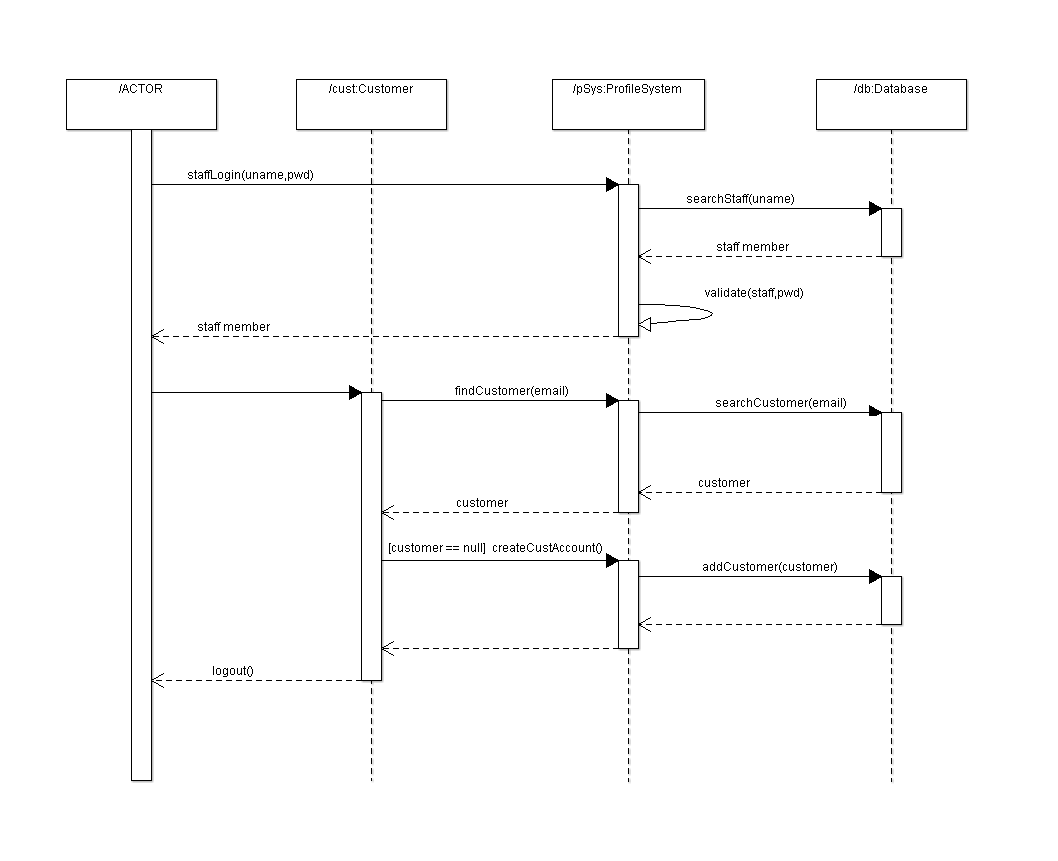
**7.3. Process View**

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.

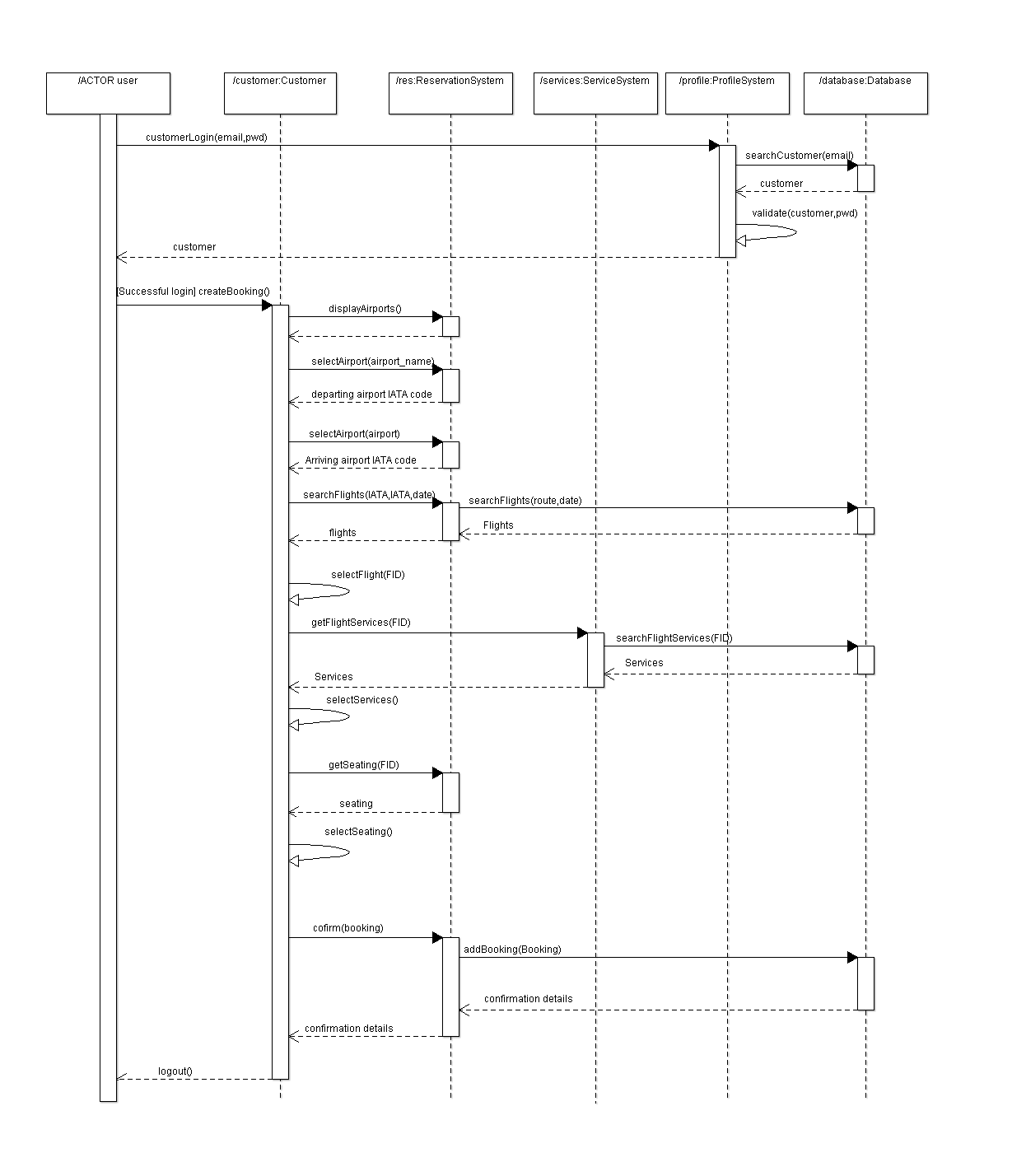
**Note 1:** ArgoUML does not support the ‘Actor’ stick figure in sequence diagrams. Thus the actor will be defined with ‘ACTOR’ displayed at the top of an object instance.

**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.

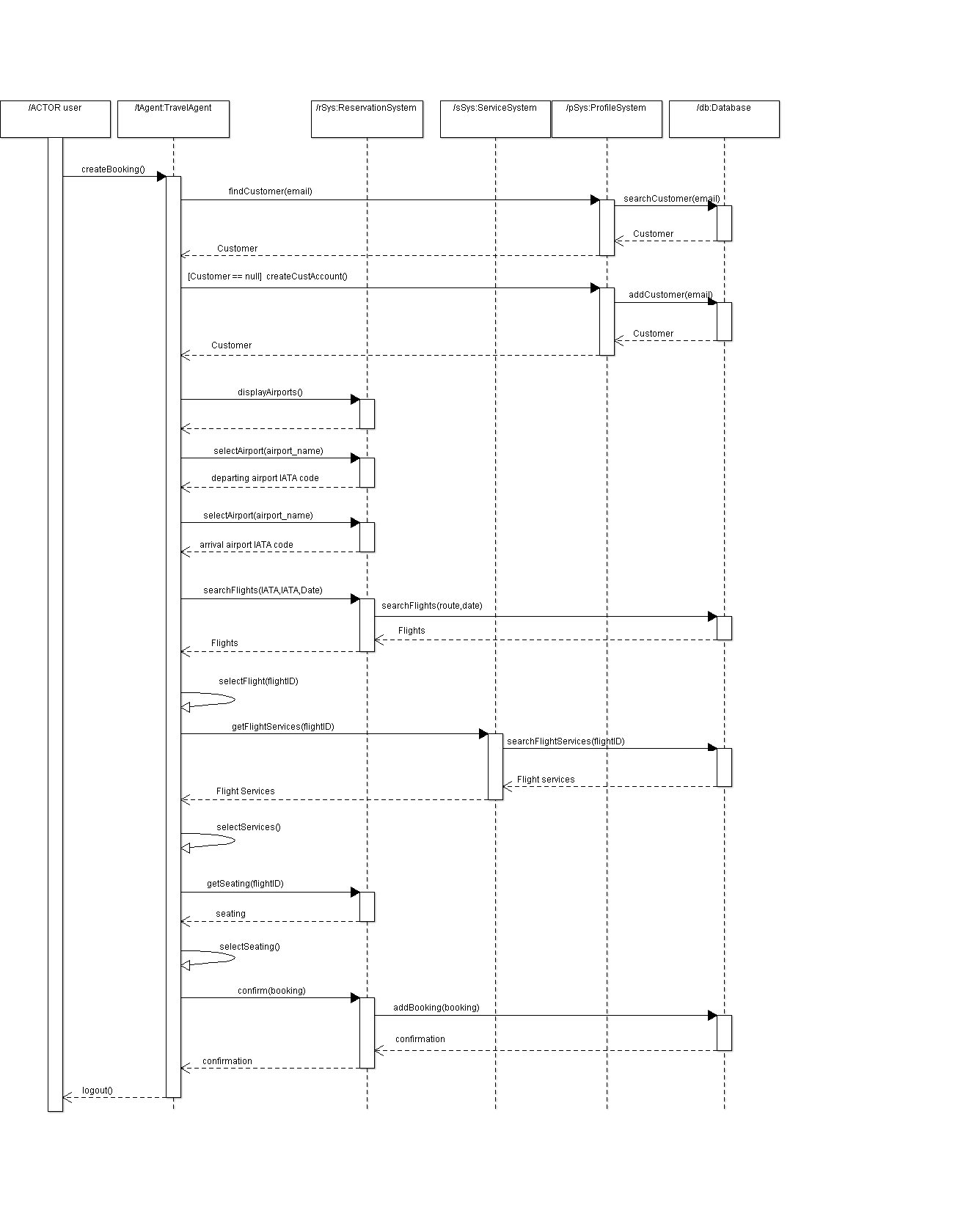
**7.3.1. Staff Member Creates Account for Customer - Sequence Diagram**



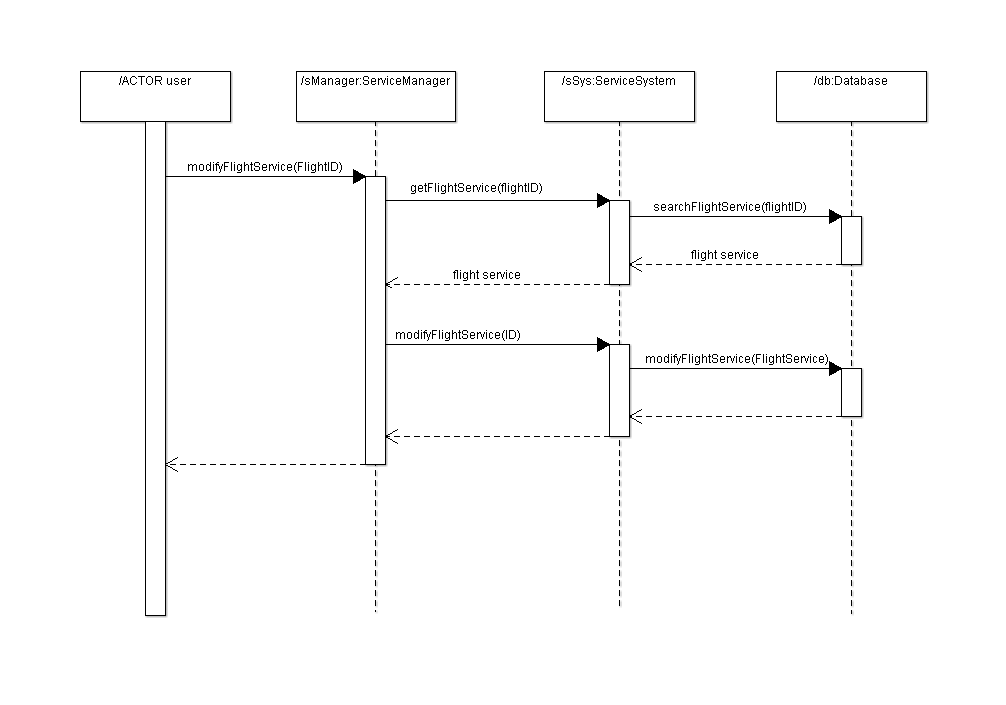
**7.3.2. Customer Books Flight – Sequence Diagram**



**7.3.3. Travel Agent Books Flight for Customer – Sequence Diagram**

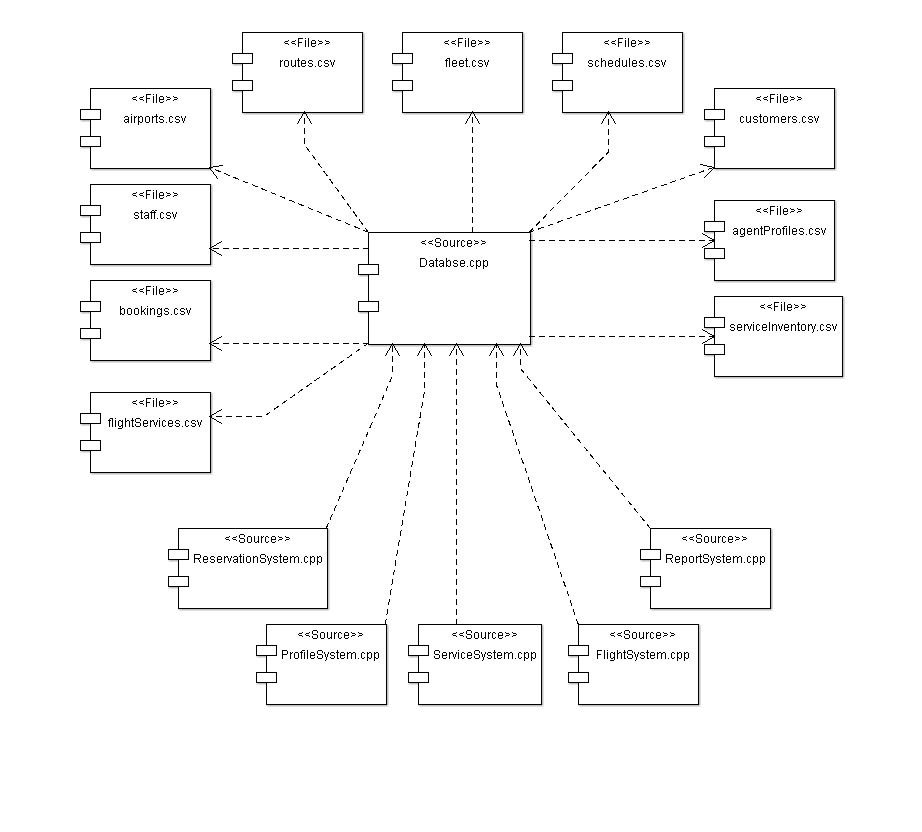


**7.3.4. Service Manager Modifies Service Items for a Flight – Sequence Diagram**



**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 methos 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 that aircrafts fClass value.

currentBCap: A number representing how many people are currently booked as Business class on a flight. Must be less that aircrafts BClass value.

currentPECap: A number representing how many people are currently booked as premium economy class on a flight. Must be less that aircrafts PEClass value.

currentECap: A number representing how many people are currently booked as economy class on a flight. Must be less that aircrafts EClass value.

serviceOnFlight: 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 databse.

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(): 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(): 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 agentss 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 the 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 toadd a airport to the airports list.

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

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

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

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

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

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

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

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

addBooking(Booking): This method allows us toadd 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 an 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 an 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 databse 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*** | | |
| ***Role*** | ***Name*** | ***Documents Contributed to*** |
| Manager/  Chief Analyst/  Chief Designer | Kresimir Bukovac | Memorandums, Meeting Minutes, Coding Style Specification, Requirements List, Group Summary, Code Documentation, Versioning Evidence, Work Diary, Report |
| Chief Analyst/  Chief Designer | Darryl Murphy | Architectural Design, , Work Diary |
| Chief Programmer/  Chief Tester | Ali Sayed | , Work Diary |
| Chief Programmer/  Chief Tester | Peter Mavridis | Business Case, Program Prototypes, SRS Template, Work Diary |

**10. Team Meetings**

**10.1. Meeting Minutes #4**

**Name of Organisation:** CSCI222 - 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 22, 2015 – 9:30am

**Chairperson:** Kresimir Bukovac

**Members Present:** Peter Mavridis

Darryl Murphy

Ali Sayed

|  |  |  |  |
| --- | --- | --- | --- |
| **Agenda/Topic** | **Discussion** | **Action** | **Person Responsible** |
| Team Meeting #3 – Assigned tasks | All of the tasks assigned were completed on time and the report had been submitted on time. | NONE | NONE |
| Graphical User Interface (GUI) | Qt is being used to design a GUI. The structure of the system was discussed and it has been decided that the code will be done as normal command line, then Peter will link it with Qt’s GUI aspect at the end. | Learn and Implement a Qt GUI. | Peter Mavridis |
| Programming | After discussing how the user interface was to look and work in the system, they need to have their implementation started. | Implement the command line interface for different users. | Kresimir Bukovac  Darryl Murphy  Ali Sayed |

|  |  |  |  |
| --- | --- | --- | --- |
| Journals | Journals are to be updated whenever work is done. | Update Journal. | Kresimir Bukovac  Peter Mavridis  Darryl Murphy  Ali Sayed |

**Documents/Deliverables:**

- Client Meeting #3 - Meeting Minutes

- Report

- Code

**Closing Time:** 10:30am

**10. 2. Memorandum #5**

**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 28, 2015

**Subject:** Confirm the work allocated from Team Meeting #4 was done, set new tasks to be completed and prepare for Client Meeting #4.

On Wednesday, the 22nd of April, 2015, the team had Team Meeting #4. 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 #4, to assign new tasks to be completed by the next Team Meeting and to prepare for 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](mailto:kb100@uowmail.edu.au)

Mobile: 0410 410 411

**10.3. Meeting Minutes #5**

**Name of Organisation:** CSCI222 - 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 29, 2015 – 9:30am

**Chairperson:** Kresimir Bukovac

**Members Present:** Peter Mavridis

Darryl Murphy

Ali Sayed

|  |  |  |  |
| --- | --- | --- | --- |
| **Agenda/Topic** | **Discussion** | **Action** | **Person Responsible** |
| Team Meeting #4 – Assigned tasks | The base interface has been implemented. The GUI is still being worked on. | NONE | NONE |
| Graphical User Interface (GUI) | When Qt is used, the code connected to Qt must be slightly altered. The re-naming of certain variables and functions along with some other small syntax fixes must be done. | Ensure that Qt is able to work with the command line code. | Peter Mavridis |
| System Structure | After some ideas of how Qt will be integrated with the system, the system’s design was slightly tweaked. | Enforce the new design. | Kresimir Bukovac  Peter Mavridis  Darryl Murphy  Ali Sayed |
| SQL Database | The idea of an SQL database being used was brought up. The database would make the requirement of the system much easier to implement and allows us to work towards stretch goals. | Research and design an SQL framework for the system. | Darryl Murphy |
| Programming | After discussing how the user interface was to look and work in the system, they need to have their implementation started. | Implement the command line interface for different users. | Kresimir Bukovac  Ali Sayed |
| Journals | Journals are to be updated whenever work is done. | Update Journal. | Kresimir Bukovac  Peter Mavridis  Darryl Murphy  Ali Sayed |

**Documents/Deliverables:**

- Client Meeting #4 - Meeting Minutes

- Code

- Qt prototype

**Closing Time:** 10:30am

**10.4. Memorandum #6**

**To:** Peter Mavridis – GUI Programmer/Chief Tester

Darryl Murphy – Chief Analyst/Designer

Ali Sayed – Chief Programmer

**From:** Kresimir Bukovac – Manager + Chief Analyst/Designer

**Date:** May 5, 2015

**Subject:** Confirm the work allocated from Team Meeting #5 was done and set new tasks to be completed.

On Wednesday, the 29th of April, 2015, the team had Team Meeting #5. 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 #5 and to assign new tasks to be completed by the next Team Meeting.

You must bring a copy of Team Meeting #5’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](mailto:kb100@uowmail.edu.au)

Mobile: 0410 410 411

**10.5. Meeting Minutes #6**

**Name of Organisation:** CSCI222 - 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:** May 6, 2015 – 9:30am

**Chairperson:** Kresimir Bukovac

**Members Present:** Peter Mavridis

Darryl Murphy

Ali Sayed

|  |  |  |  |
| --- | --- | --- | --- |
| **Agenda/Topic** | **Discussion** | **Action** | **Person Responsible** |
| Team Meeting #5 – Assigned tasks | The base interface has been implemented. The SQL database has been implemented and is working. | NONE | NONE |
| Graphical User Interface (GUI) | Huge complexities were found when using both Qt and SQL together, and after speaking with the client, it has been decided that SQL will be used and work on Qt’s GUI will be halted. | NONE | NONE |
| SQL Database | An example and basic SQL database has been made and shown to the rest of the team members. This will now be developed further to encompass more features of the system. | Improve the SQL database and its functionality. | Darryl Murphy |
| Programming | The base user interface has been created, so now functionality has to be implemented. The high-priority files to be implemented are to do with registration and login. Discussion is still ongoing. | Discuss and finalise the method and order of the system and its implementation. | Kresimir Bukovac  Peter Mavridis  Darryl Murphy  Ali Sayed |

|  |  |  |  |
| --- | --- | --- | --- |
| Journals | Journals are to be updated whenever work is done. | Update Journal. | Kresimir Bukovac  Peter Mavridis  Darryl Murphy  Ali Sayed |

**Documents/Deliverables:**

- Client Meeting #5 - Meeting Minutes

- Code

- SQL Database

- Qt prototype

**Closing Time:** 10:30am

**10.6. Memorandum #7**

**To:** Peter Mavridis – GUI Programmer/Chief Tester

Darryl Murphy – Chief Analyst/Designer

Ali Sayed – Chief Programmer

**From:** Kresimir Bukovac – Manager + Chief Analyst/Designer

**Date:** May 12, 2015

**Subject:** Confirm the work allocated from Team Meeting #6 was done, set new tasks to be completed and prepare for Client Meeting #5.

On Wednesday, the 6th of May, 2015, the team had Team Meeting #6. 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 #6 and to assign new tasks to be completed by the next Team Meeting and discuss any issues or queries we must present in the Client Meeting.

You must bring a copy of Team Meeting #6’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](mailto:kb100@uowmail.edu.au)

Mobile: 0410 410 411

**10.7. Meeting Minutes #7**

**Name of Organisation:** CSCI222 - 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:** May 13, 2015 – 9:30am

**Chairperson:** Kresimir Bukovac

**Members Present:** Peter Mavridis

Darryl Murphy

Ali Sayed

|  |  |  |  |
| --- | --- | --- | --- |
| **Agenda/Topic** | **Discussion** | **Action** | **Person Responsible** |
| Team Meeting #6 – Assigned tasks | The SQL database has been implemented and is working with login for different users. Some features of SQL must still be evolved. | NONE | NONE |
| SQL Database | The SQL database is now functional and has been integrated into the system. The other requirements which are yet to be implemented are the creation and deletion of data as opposed to simply reading and modifying data. | Improve the SQL database and its functionality. | Darryl Murphy |
| Implement the create and delete features for managers. | Kresimir Bukovac |
| Programming | Registration and login both work with the SQL database now. From here on, programming can progress at a greater speed as these cores have been successfully implemented. | Code the manager UIs and their controllers. | Kresimir Bukovac |

|  |  |  |  |
| --- | --- | --- | --- |
| Programming | The feature of hiding passwords with asterisks will be done with ncursors. | Learn how ncursors work and implement them. | Peter Mavridis  Ali Sayed |
| A Date class will allow the system to efficiently query the database and show the user what they want. (e.g. flights in a date range) | Implement a Date class | Darryl Murphy |
| Other Features | A suggestion of a weather application which retrieves actual weather information in real time to be used as an added feature in the system is being looked into. | Research how to and implement a weather application. | Darryl Murphy |
| Client Meeting | No major issues are currently present, so there is nothing to discuss at the Client Meeting apart from the use of the SQL database and its implementation. | Present queries to the client. | Kresimir Bukovac  Peter Mavridis  Darryl Murphy  Ali Sayed |
| Journals | Journals are to be updated whenever work is done. | Update Journal. | Kresimir Bukovac  Peter Mavridis  Darryl Murphy  Ali Sayed |

**Documents/Deliverables:**

- Client Meeting #6 - Meeting Minutes

- Code

- SQL Database

**Closing Time:** 10:30am

**10.8. Memorandum #8**

**To:** Peter Mavridis – GUI Programmer/Chief Tester

Darryl Murphy – Chief Analyst/Designer

Ali Sayed – Chief Programmer

**From:** Kresimir Bukovac – Manager + Chief Analyst/Designer

**Date:** May 19, 2015

**Subject:** Confirm the work allocated from Team Meeting #7 was done and set new tasks to be completed.

On Wednesday, the 13th of May, 2015, the team had Team Meeting #7. 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 #7 and to assign new tasks to be completed by the next Team Meeting.

You must bring a copy of Team Meeting #7’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](mailto:kb100@uowmail.edu.au)

Mobile: 0410 410 411

**10.9. Meeting Minutes #8**

**Name of Organisation:** CSCI222 - 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:** May 20, 2015 – 9:30am

**Chairperson:** Kresimir Bukovac

**Members Present:** Peter Mavridis

Darryl Murphy

Ali Sayed

|  |  |  |  |
| --- | --- | --- | --- |
| **Agenda/Topic** | **Discussion** | **Action** | **Person Responsible** |
| Team Meeting #7 – Assigned tasks | The SQL database structure has been modified to better fit the requirements and ease implementation. The date class and weather application have been made, but fixes are needed. Ncurses are still being learnt. | NONE | NONE |
| SQL Database | The SQL database has been modified slightly, so some code rework will be done to improve the quality of the way the system works and the code in general. | Find out how to auto increment unique IDs. | Darryl Murphy |
| Modify the create and delete features for managers. | Kresimir Bukovac |
| Programming | Ncurses need to be implemented to be able to hide users’ passwords with asterisks. | Implement password hiding for login. | Ali Sayed |

|  |  |  |  |
| --- | --- | --- | --- |
| Programming | The manager reports have been discussed and now just need to be implemented to show the user important related information. Different users have had their classes revamped. | Implement the booking and flight manager reports. | Peter Mavridis |
| Implement the service and profile manager reports. | Ali Sayed |
| Fix the travel agent functions. | Peter Mavridis |
| A Date class has been made, but some small ideas have been pointed out and will be changed. | Upgrade the date class’ functions. | Darryl Murphy |
| The weather application has limitations due to it being free, so it must be modified to be better used in the system. | Fix the issues of the weather application. | Darryl Murphy |
| Code Style | A neat a consistent code style must be specified and followed by all members of the team. Style has been chosen by the majority of existing code. | Create and upload the code style document. | Kresimir Bukovac |
| Formalise all existing code. | Kresimir Bukovac |
| Journals | Journals are to be updated whenever work is done. | Update Journal. | Kresimir Bukovac  Peter Mavridis  Darryl Murphy  Ali Sayed |

**Documents/Deliverables:**

- Client Meeting #7 - Meeting Minutes

- Code

- SQL Database

- Weather application

- Code Style Specification

**Closing Time:** 10:30am

**10.10. Memorandum #9**

**To:** Peter Mavridis – GUI Programmer/Chief Tester

Darryl Murphy – Chief Analyst/Designer

Ali Sayed – Chief Programmer

**From:** Kresimir Bukovac – Manager + Chief Analyst/Designer

**Date:** May 26, 2015

**Subject:** Confirm the work allocated from Team Meeting #8 was done and set new tasks to be completed.

On Wednesday, the 20th of May, 2015, the team had Team Meeting #8. 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 #8 and to assign new tasks to be completed by the project’s deadline next Tuesday.

You must bring a copy of Team Meeting #8’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](mailto:kb100@uowmail.edu.au)

Mobile: 0410 410 411

**10.11. Meeting Minutes #9**

**Name of Organisation:** CSCI222 - 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:** May 27, 2015 – 9:30am

**Chairperson:** Kresimir Bukovac

**Members Present:** Peter Mavridis

Darryl Murphy

Ali Sayed

|  |  |  |  |
| --- | --- | --- | --- |
| **Agenda/Topic** | **Discussion** | **Action** | **Person Responsible** |
| Team Meeting #8 – Assigned tasks | All of the code is in a consistent style now. Apart from the reports, all other programming tasks have been completed. | NONE | NONE |
| Programming | Streams seem to be an easier way than ncurses to implement password hiding with asterisks. | Implement cross-platform password hiding for login. | Ali Sayed |
| The manager reports have been discussed and now just need to be implemented to show the user important related information. All of the reports will now be done by the same person. | Implement the report functions for all four managers. | Ali Sayed |
| Manager functions need to be updated with auto incrementing IDs and format consistency. | Update the manager functions. | Kresimir Bukovac |
| Error checking and final testing must be conducted on the entire system to decrease the risk of errors during runtime. | Perform error checking and testing. | Peter Mavridis |

|  |  |  |  |
| --- | --- | --- | --- |
| Programming | Different operating systems require different system calls to clear the screen for the UI. | Implement cross-platform screen clearing. | Peter Mavridis |
| The weather application has been designed and fits with the program nicely. | Finalise the weather application. | Darryl Murphy |
| The booking system has a few functions which need updating. | Update the booking system. | Darryl Murphy |
| Users must be notified of any status changes or specific issues related to them when they log in. | Implement the notification system. | Ali Sayed |
| Code Style | A neat a consistent code style must be specified and followed by all members of the team. | Formalise all existing code. | Kresimir Bukovac |
| Documentation | The report section in A2.pdf has been split up between all four members mainly based on which sections they have focused on in the implementation. | Report pt.1. | Darryl Murphy |
| Report pt.2. | Peter Mavridis |
| Report pt.3. | Kresimir Bukovac |
| Report pt.4. | Ali Sayed |
| All of the documentation including work diaries must be done in a consistent format and then compiled into a final report. | Document all of the code files. | Kresimir Bukovac |
| Document all testing done. | Peter Mavridis |
| Develop and finalise all aspects of the meta report. | Kresimir Bukovac |
| Compile the final report. | Kresimir Bukovac |
| Journals | Journals are to be updated whenever work is done. | Update Journal. | Kresimir Bukovac  Peter Mavridis  Darryl Murphy  Ali Sayed |

**Documents/Deliverables:**

- Client Meeting #8 - Meeting Minutes

- Code

- SQL Database

- Weather application

- A2.pdf

**Closing Time:** 10:30am

**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. |

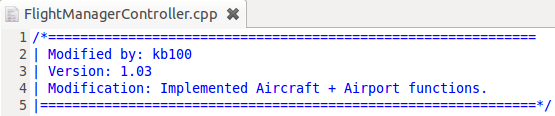
|  |  |
| --- | --- |
| Date | 4/03/2015 |
| Action | I emailed out the contact details and method to all of the group members. |
| Time Expected | 2 minutes. |
| Time Taken | 2 minutes. |
| Problems | None. |
| Solutions | None. |

|  |  |
| --- | --- |
| 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. |

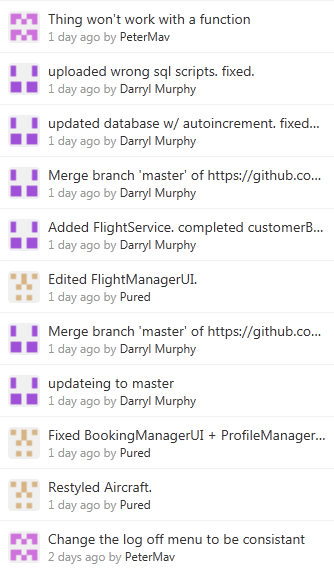
|  |  |
| --- | --- |
| Date | 11/03/2015 |
| Action | I found a fifth person to join the team as a chief programmer. |
| Time Expected | 2 weeks. |
| Time Taken | 3 days. |
| Problems | None. |
| Solutions | None. |

**12. Versioning Evidence**

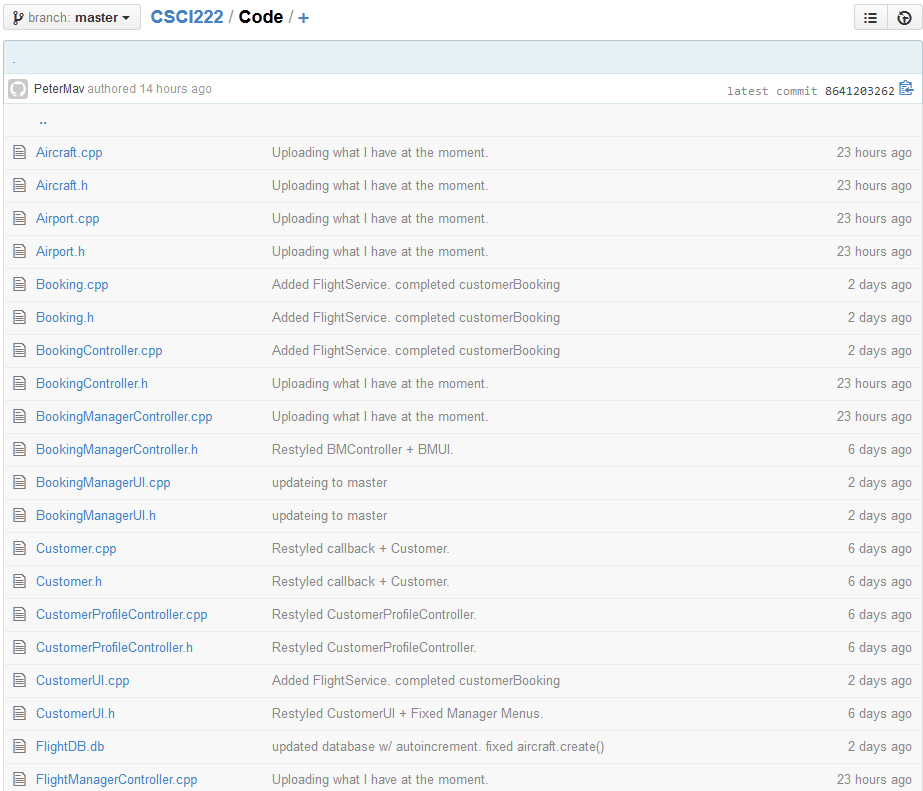
The following image is an example of the header used in every code file. Its job is to detail who the file was modified by, what the version number is and what the modification was.



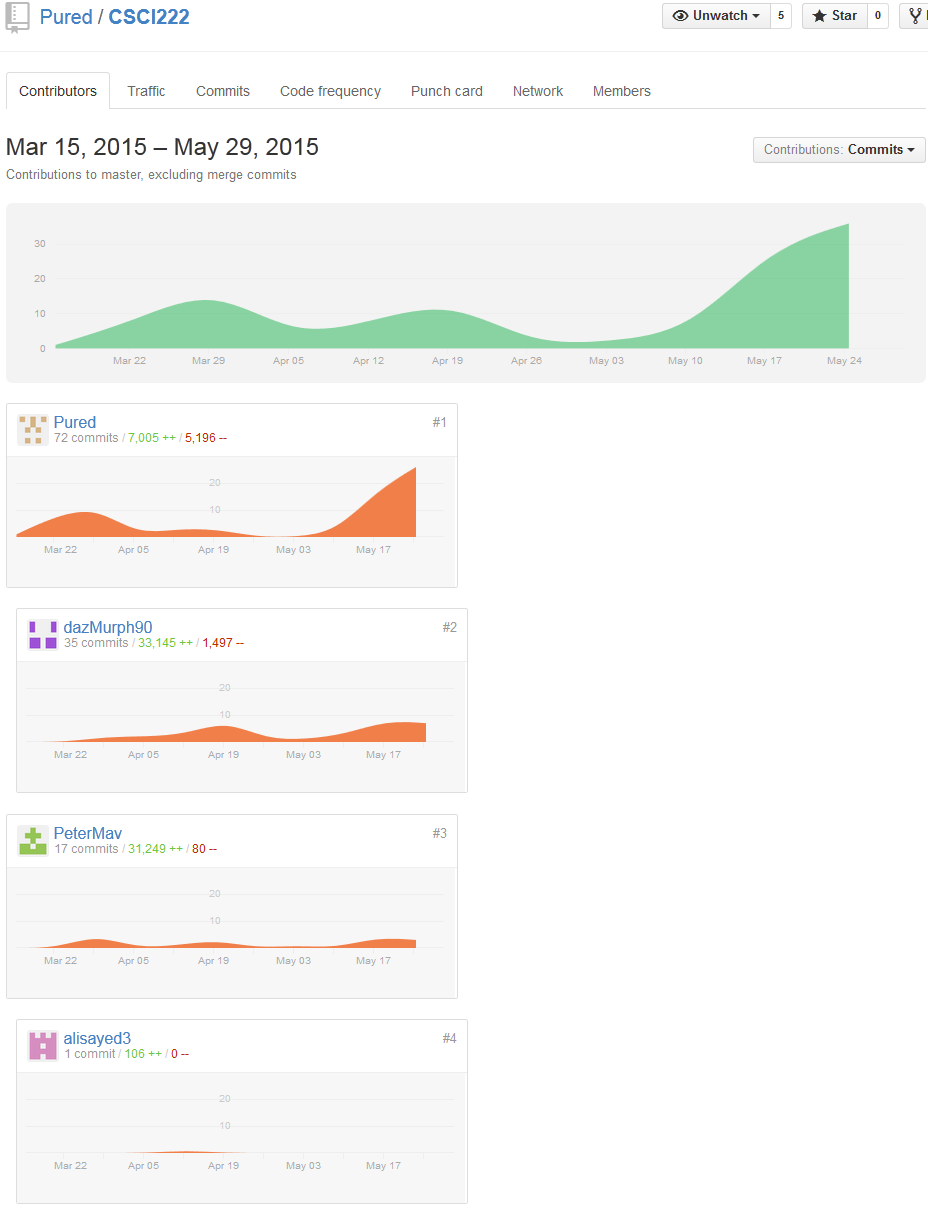
The following is a screenshot of a section of commits sent to the Master GitHub branch.



The following image is an example of the code files on GitHub with varying commit dates.



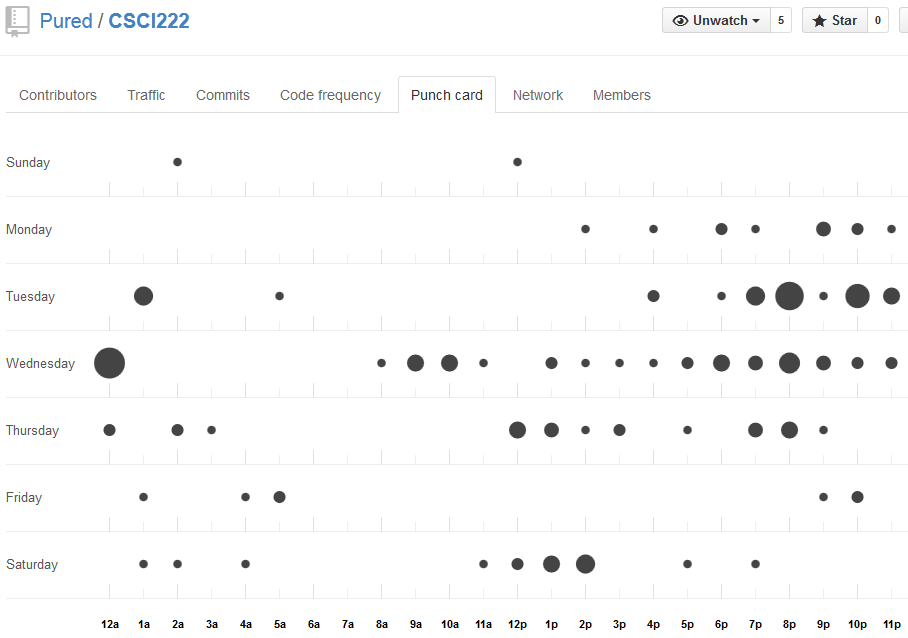
The following set of graphs are the statistics for the commits sent by each member.



The following graph represents the amount of code added and removed.



The following graph shows statistics of when and how much GitHub was being worked on.



The following image shows how many commits were sent, how many files were changes and how many additions and deletions have been made. It also shows each member’s commit contribution at the end.

