Flight Management System

Project Report

Version 1.7

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 19/May/15 | 1.0 | Initial Report Compilation | Michael Kong  Matthew King  Jicheng Li  Siyuan Chen |
| 20/May/15 | 1.1 | Data Persistence Table Added | Matthew King |
| 25/May/15 | 1.3 | User Interface Design Added | Matthew King |
| 29/May/15 | 1.4 | Architectural Design Added | Matthew King |
| 30/May/15 | 1.5 | Further Additions and Editing | Matthew King |
| 31/May/15 | 1.6 | Constructions Phase and Implemented Functionality Added | Matthew King |
| 01/Jun/15 | 1.7 | Final Review and Editing for Print | Matthew King |

Executive Summary

The following report provides the reader with a detailed description, analysis and evidence of a competently handled Construction phase of the Rational Unified Process for a Flight Management System undertaken by group members, Michael Kong, Matthew King, Jicheng Li and Siyuan Chen.

This report includes a detailed architectural design description of the proposed system, summary of the requirements implemented, summary of the construction phase activities and outcomes, source code listing, test cases and outcomes and meta report detailing work completed for the phase.

The architectural design description contains the 4 + 1 views of the proposed system including, Use case view, Logical view, Process view, Implementation view and Deployment view. It also describes the type of architectural pattern used and justifies the patterns use.

The summary of requirements implemented is a tabular representation of the functional requirements implemented from the Software Requirements Specification.

The construction phase summary briefly details extensions to the existing implementation, describes informal unit testing and the outcomes of such testing, and summarizes integration of code during each iteration of the phase.

The source code listing includes all code from the final product. It also incorporates unit testing code used to verify project code correctness.

Lastly, a meta-report summarizes work completed, group member contributions, meeting outcomes and agendas. Github commit logs for the construction phase of the project are also included.

The outcome of this report shows that the software development team have competently planned and executed the construction phase of the Rational Unified Process for the development of a Flight Management System.

Table of Contents

1. Architectural Design 6

1.1 Top Level Architecture Design Diagram 6

1.2 Architectural Representation 6

1.3 Design Patterns 8

1.4 Subsystems 10

1.5 Use Case View 12

1.5.1 Overview 12

1.6 Logical View 12

1.6.1 Overview 12

1.7 Process View 12

1.7.1 Overview 12

1.8 Implementation View 12

1.8.1 Overview 12

1.8.2 Component Diagram 12

1.9 Deployment View 13

1.9.1 Overview 13

1.9.2 Deployment 13

1.9.3 Deployment Diagram 14

1.10 Data Persistence 15

1.10.1 Overview 15

1.11 User Interface Design 22

1.11.1 Overview 22

1.11.2 Usability and Design Goals for the Interface 22

2. Summary of Requirements Implemented 22

2.1 Overview 22

2.2 Implemented Functionality Table 23

3. Summary of Construction Phase 25

3.1 Overview 25

3.2 Iteration 1 of Construction Phase 25

3.2.1 Plan 25

3.2.2 Summary of Iteration 25

3.3 Iteration 2 of Construction Phase 26

3.3.1 Plan 26

3.3.2 Summary of Iteration 27

3.4 Iteration 3 of Construction Phase 29

3.4.1 Plan 29

3.4.2 Summary of Iteration 30

4. Code Listing 33

4.1 Overview 33

5. Testing 33

5.1 Overview 33

5.2 Black Box Test Cases 33

5.3 White Box Test Cases 39

6. Installation Instructions 43

7. Meta-Report 45

7.1 Overview 45

7.2 Work Completion Summary 45

7.3 Group Meeting Summary 46

7.4 Team Member Weekly Diaries 47

7.4.1 Work Diary – Matthew King (Team Manager) 47

7.4.2 Work Diary – Siyuan Chen (System Developer and Tester) 49

7.4.3 Work Diary – Jicheng Li (Lead System Developer and Tester) 50

7.4.4 Work Diary – Michael Kong (System Architect) 51

7.5 Member Contribution Summary 52

7.6 Github Commit Logs 53

**Appendix A - Use Case View**

**Appendix B - Logical View**

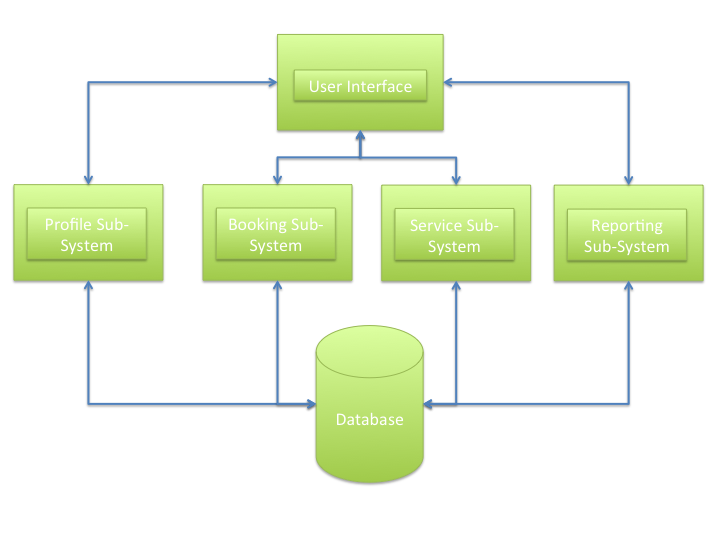
**Appendix C - Process View**

# Architectural Design

The following section is designed to provide an overall view and justification of the architectural design for the proposed Flight Management System. The architectural design artifacts included in this section are use-case diagrams, sequence diagrams, class diagrams, state diagrams, data persistence tables and user interface design summary. These artifacts form the 4 + 1 view of the system making up the Process view, Logical view, Implementation view, Deployment view and Use case view.

## Top Level Architecture Design Diagram

This diagram gives a high level view of the system and it’s sub systems including some general grouping of like functionality and components within the proposed system.



## Architectural Representation

The architectural design for the proposed system is largely based upon the Layered architectural pattern. This pattern is defined as organizing the system in such a way that it forms layers. Each layer relies only upon the layer beneath it. This architectural pattern was chosen for the following reasons –

* Layered architecture facilitates redundancies
* Layered architecture allows sub-systems to be added incrementally
* Layered architecture benefits building software in a team environment
* Layered architecture provides the ability to reuse layers in other systems that function similarly
* Layered architecture provides replaceable layers; a layer can be rewritten with out having to rewrite or break other layers of the software

The facilitation of redundancy has been an important consideration regarding the design of the software. This is due to the non-functional requirement surrounding availability of the system (#NF\_01 from SRS). Redundancy provides increased dependability of a system therefore justifying this design decision.

Another consideration is the ability to be able to add sub-systems in an incremental way. This project has been undertaken using the Rational Unified Process (RUP), which is an iterative process therefore being able to work iteratively and incrementally is of major importance. A layered architecture provides this benefit and is another reason for the selection of this pattern.

Using a layered architecture also provides a team oriented development benefit. Each team of developers or member of a development team can work on an individual layer of the system. This provides developers with a clear understanding and goal for the development of the layer. Each developer or team knows what their layer is responsible for and which other layers it will interact with. This is further justification of the layered architecture design decision.

The last two points are less important but still provide reasonable justification. Layered architecture means layers can be reused in similar types of systems therefore increasing the reusability of code. The systems layers can also be updated or changed in future versions with minimal effect on the rest of the software. This results in a more maintainable system, which will benefit the customer in the long term.

**High Level System Layer Diagram**

|  |
| --- |
| **User Interface** |

|  |  |  |
| --- | --- | --- |
| **Interface Management** | **Authentication** | **Report Generation** |

|  |
| --- |
| **Core Logic and Functionality (Controllers)** |

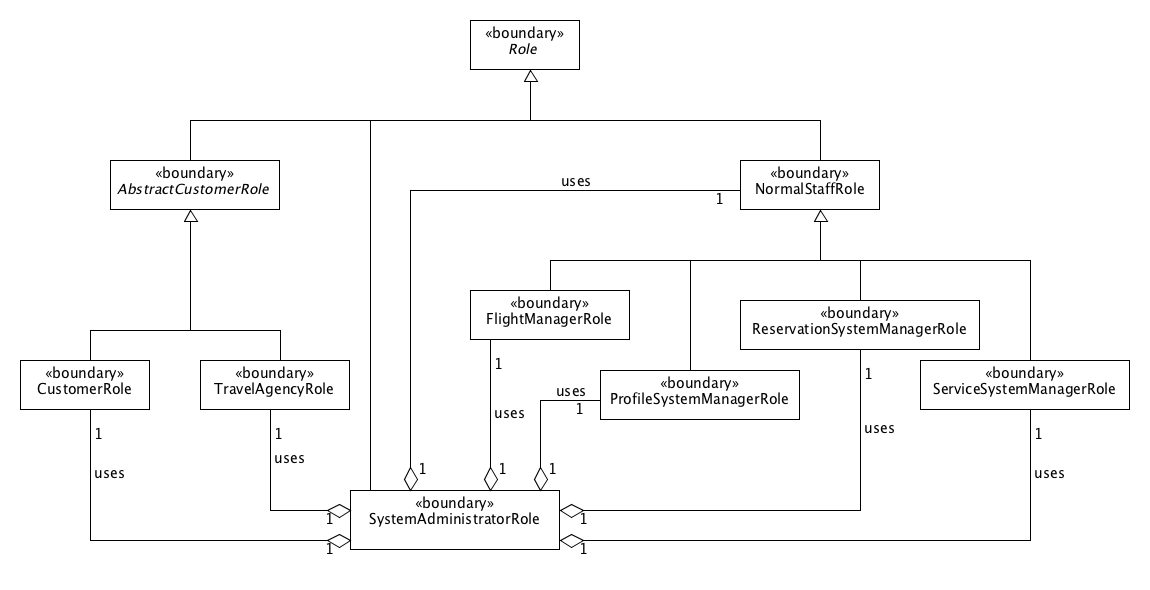
|  |
| --- |
| **Database and Entities (Persistent and Transient Data)** |

## Design Patterns

In the Flight Management System we incorporated a few design patterns into our source code. During the design phase, we realized some problems with our design that could potentially cause issues of code being rewritten in many places. So after consulting some experienced software engineers and developers, they recommended design patterns that could improve our system design. The three design patterns that we incorporated in our implementation are:

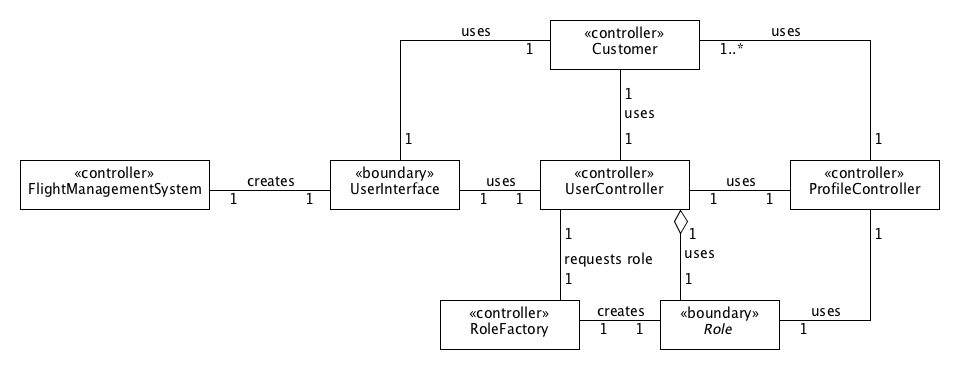
1. Delegation pattern
2. Factory pattern
3. Builder pattern

1. Delegation Pattern

The delegation pattern is where an object uses another object (helper object) to perform a task. The reason we used this pattern is to solve our problem of having classes that require multiple inheritance to work. Therefore, the non-abstract classes in the “role” package are all helper objects. They represent the possible tasks that a particular user of the system is able perform in the system. The SystemAdministratorRole class is the class that uses this pattern. As shown in the class diagram below, all non-abstract classes have an aggregation relation with the SystemAdministratorRole class. 

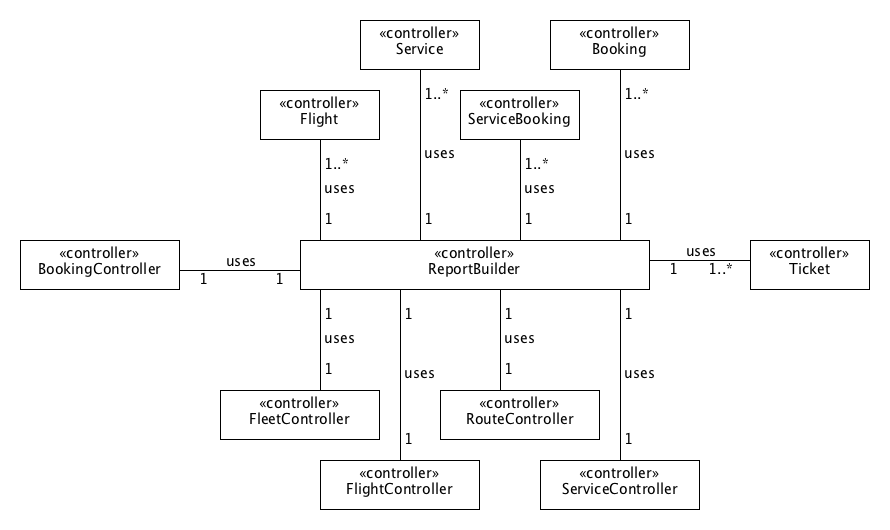
With this design pattern in place, the System Administrator actor can act as any of the roles that the SystemAdministratorRole class has. Furthermore, this also solved the problem of multiple inheritance in our implementation language, Java.

2. Factory Pattern

The factory pattern is a creational pattern where a class method is called to produce an object. In our implementation, we used this design pattern to create a role based on the classes in the “role” package shown in the class diagram above. Based on the role of the user that was stored in the database, we passed a String representation of the user’s role to the RoleFactory’s class method “getRole” to get an object from the “role” package. Naturally, the code does not want to be concerned about the object it got back from the factory class. So we the “getRole” method always returned a Role reference to the object. So to use the returned object, the overridden methods in the Role class are called to take advantage of the polymorphism that the classes in the “role” package implement. The class diagram below shows how the UserController class requests a Role from the RoleFactory class and gets a Role reference to an object of the “role” package.

3. Builder Pattern

The builder pattern is where a class in instantiated by a client, its methods are called to “build” the desired object piece-by-piece by the client, and finally a “build” method is called to return an object that has been initialized to the building methods that the client called. In our implementation, we used a pseudo-builder pattern. The ReportBuilder class does not return an object but rather builds a report based on the given parameters and displays it. In the class diagram below, the ReportBuilder class uses several controller classes to build a report. The client will call a class method “displayReport” with arguments and based on the given arguments, the ReportBuilder class will generate the appropriate report. The reason why we used this design pattern is to make sure our system has high maintainability and is able to easily evolve. With this in mind, this design allows the flexibility in adding more report types as the Flight Management System evolves into the next version.



## Subsystems

The Flight Management System is composed of 4 subsystems:

1. Reservation System that has the ability to manipulate all flight reservations, seat selection, ticketing, flight availability, flight details, rates and conditions.
2. Profile Subsystem that manages all profiles within the system.
3. Service Subsystem that manages in-flight services such as food and drinks.
4. Reporting Subsystem that can generate various summery reports.

Each of the subsystems are further decomposed into smaller areas of concern:

1. Airports (represented in the system by Airport.java)
2. Bookings (represented in the system by Booking.java, ServiceBooking.java and Ticket.java)
3. Fleet (represented in the system by Plane.java)
4. Flights (represented in the system by Flight.java)
5. Routes (represented in the system by Route.java)
6. Services (represented in the system by Service.java)
7. Profiles (represented in the system by Customer.java, Person.java and Staff.java)
8. Reports (produced by ReportBuilder.java)

We shall now describe each subsystem in more detail and outline the classes that enable each subsystem to work together as a single, cohesive system.

1. Reservation System

The reservation system, which is the core of the system, allows users to perform all operations on bookings, seating, ticketing, flights, rates and conditions. The classes that support this system include:

* AirportController.java
* AirportEntity.java
* BookingController.java
* BookingEntity.java
* FleetController.java
* FleetEntity.java
* FlightController.java
* FlightEntity.java
* ProfileController.java
* ProfileEntity.java
* RouteController.java
* RouteEntity.java
* ServiceController.java
* ServiceEntity.java

In short, these are all the controller and entity classes that exist in the system, making it the most important and complex part of the system. When users make bookings, airport names and codes need to be known for the system to identify the routes that the user is interested in. In turn, we need to know those routes to identify which flights plough those routes. Furthermore, we need to know the services that are available to let the user book them. Lastly we need to know the user’s profile so that the system can charge the booking fee to the correct user.

2. Profile Subsystem

The profile subsystem is able to handle all aspects of profiles including user login details, user personal details and their fly status. This system is necessary for users to perform actions such as logging in, signing up and making bookings. The classes that support this system include:

* ProfileController.java
* ProfileEntity.java

These two classes contain the necessary methods to allow the client to perform actions such as updating profiles, changing passwords, and managing the watch and no fly list, in addition to the abovesaid functions.

3. Service Subsystem

The service subsystem is a less important part of the system that deals solely with services that can be booked for a particular user on a particular flight. This system would allow clients to perform actions such as add and delete services. The classes that support this system include:

* ServiceController.java
* ServiceEntity.java

These classes have methods to allow clients to manipulate and manage services in terms of pricing and avalability.

4. Reporting Subsystem

This subsystem allows clients to be able to view reports generated from the data that exist in the database. Reports such as flight reports and revenue reports are generated by the system for the various users of the system. The classes that support this system include:

* BookingController.java
* BookingEntity.java
* FleetController.java
* FleetEntity.java
* FlightController.java
* FlightEntity.java
* ReportBuilder.java
* RouteController.java
* RouteEntity.java
* ServiceController.java
* ServiceEntity.java

The reason why this system requires this many classes is because it requires the data from these various classes in order to generate useful reports. For example, we cannot know what routes are popular during a particular month if we did not link the routes and flights together.

## Use Case View

### Overview

This section is made up of the Use-Case diagrams and their accompanying descriptions to form the Use Case View of the system. Each Use-Case Diagram depicts interactions of users with the proposed Flight Management System. Each Use-Case has an associated description which details the actors involved, action invoked and flow of the case.

The complete document can be found in Appendix A.

## Logical View

### Overview

The logical view for the Flight Management System is made of three diagrams. Firstly the Package diagram shows packages of the system and their dependencies. The next set of diagrams is the Class diagrams. These are a refined version of the original domain model and have been refined using the BCE (Boundary Control Entity) framework model. Lastly, the State Machine diagrams depict the states and transitions of different object lifecycles throughout the system.

The complete document can be found in Appendix B.

## Process View

### Overview

The process view for the Flight Management System is made up of sequence, communication and activity. The sequence diagrams which show a sequence of internal system interactions between objects forming a timeline. The communication diagrams show a better representation of object relationships. The activity diagrams shows what actions are required to take place and when they should happen. Together all of these diagrams form a comprehensive process view of the proposed system.

The complete document can be found in Appendix C.

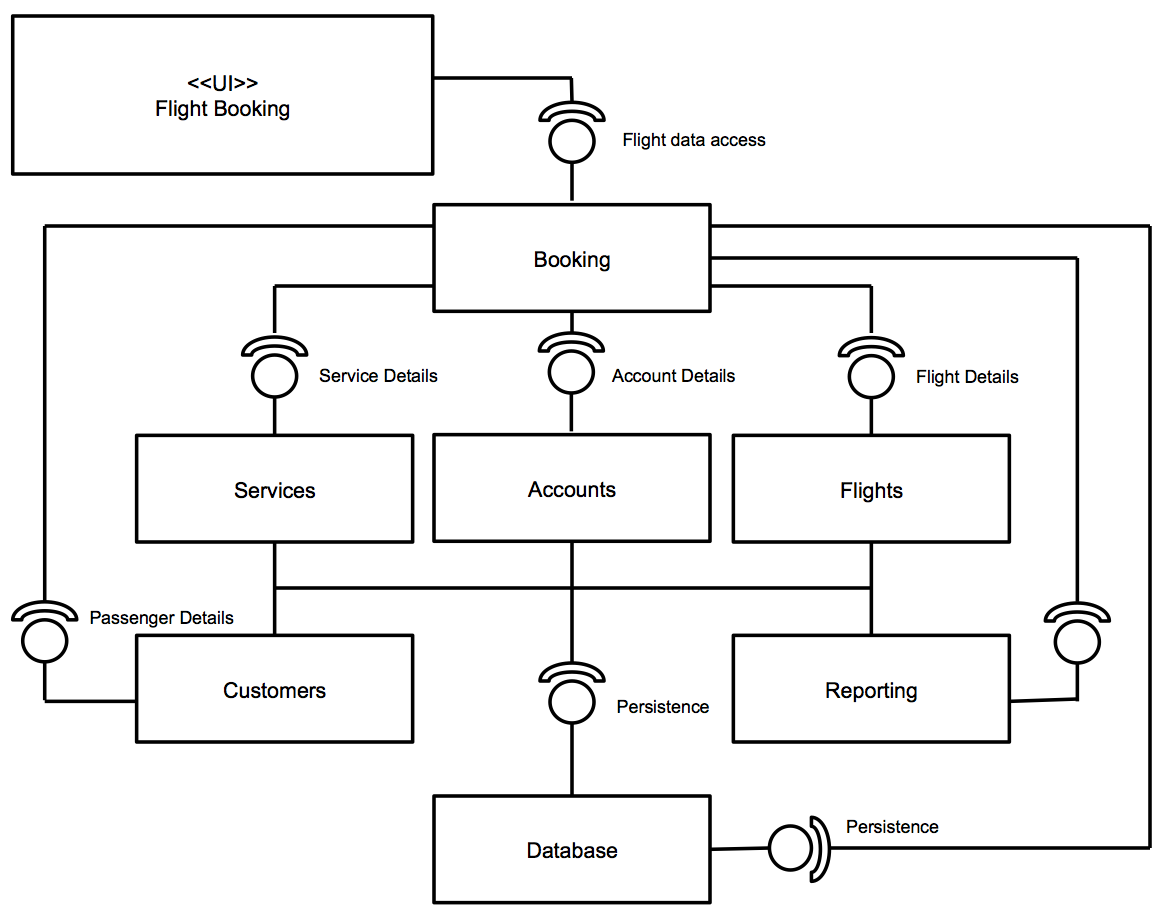
## Implementation View

### Overview

The implementation view of the Flight Management System is made up of a component diagram. This diagram helps to model the physical aspects of the system and the dependencies they have on other components.

### Component Diagram

The following diagram illustrates the components and their dependencies within the Flight Management System.



## Deployment View

### Overview

This section provides details of the planned deployment accompanied by a Deployment diagram of the installed system.

### Deployment

The Flight Management System will be deployed as a client server application. As the deployment diagram demonstrates, different brands of client may be developed to suit the specific needs of the individual users. For example, customers and travel agents do not need functionality of admin, managers and staff included in their client software and therefore it should not be unnecessarily included.

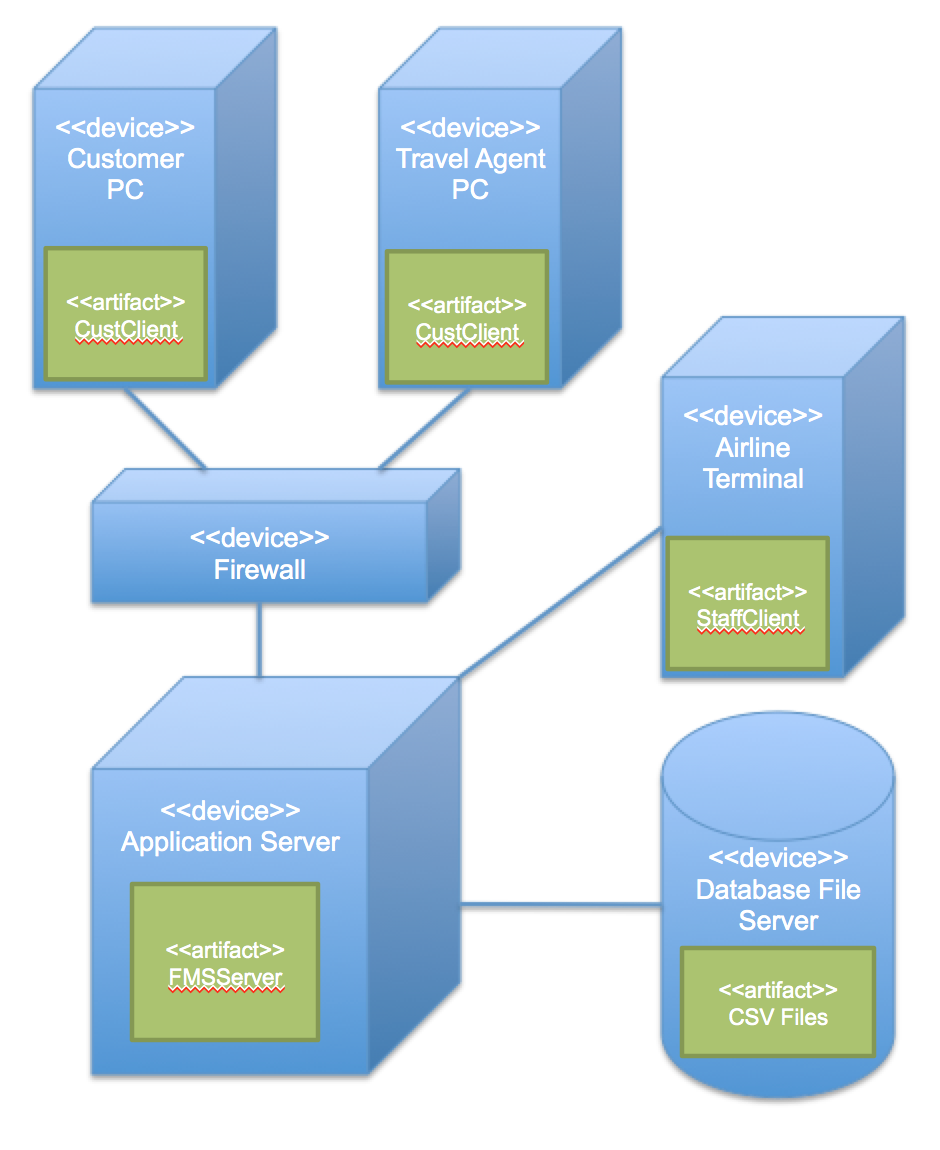
Customers and travel agents will connect to the application server through a firewall to carry out flight booking related activities.

Admin and staff will connect directly to the application server to carry out their flight booking and management related activities.

This deployment model also be provides the ability to include a web server if the airline decides to change out the customer client for a more modern and aesthetically pleasing website.

### Deployment Diagram

Below is the proposed deployment diagram for the Flight Management System.



## Data Persistence

### Overview

The following table illustrates all data persistence within the Flight Management System’s database tables.

**flight\_schedule.csv**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Description | Persistent | Store |
| ID | integer | Integer representing the flight record number in the file | Yes | flight\_schedule.csv:ID |
| Flight ID | string | Unique code representing a flight | Yes | flight\_schedule.csv: FlightID |
| Plane ID | integer | Unique integer used to identify a plane the airlines fleet | Yes | flight\_schedule.csv:  PlaneID |
| Route ID | integer | Unique integer used to identify a route | Yes | flight\_schedule.csv:  RouteID |
| Depart Time | string | Departure time of the flight | Yes | flight\_schedule.csv:  DepartTime |
| Arrival Time | string | Arrival time of the flight | Yes | flight\_schedule.csv:  ArrivalTime |
| First class available | integer | Number of first class seats available for this flight | Yes | flight\_schedule.csv:  fc\_available |
| Business class available | integer | Number of business class seats available for this flight | Yes | flight\_schedule.csv:  bc\_available |
| Premium economy class available | integer | Number of premium economy class seats available for this flight | Yes | flight\_schedule.csv:  pc\_available |
| Economy class available | integer | Number of economy class seats available for this flight | Yes | flight\_schedule.csv: ec\_available |
| First class price | double | Price of first class seats for this flight | Yes | flight\_schedule.csv: fc\_price |
| Business class price | double | Price of business class seats for this flight | Yes | flight\_schedule.csv: bc\_price |
| Premium economy class price | double | Price of premium economy class seats for this flight | Yes | flight\_schedule.csv: pc\_price |
| Economy class price | double | Price of economy class seats for this flight | Yes | flight\_schedule.csv: ec\_price |

**user\_account.csv**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Description | Persistent | Store |
| User name | string | User name of the FMS user. Required to login to the FMS. | Yes | user\_account.csv:  UserName |
| Password | string | A users password. Required to login to the FMS. | Yes | user\_account.csv:  Password |
| User Role | string | A string that represents the user role | Yes | user\_account.csv:  UserRole |

**user\_details.csv**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Description | Persistent | Store |
| User Name | string | User name of the FMS user. | Yes | user\_details.csv:  UserName |
| Title | string | Title of the person | Yes | user\_details.csv:  Title |
| First Name | string | First name of the user | Yes | user\_details.csv:  FirstName |
| Last Name | string | Last name of the user | Yes | user\_details.csv:  LastName |
| Gender | string | Character representing the gender of the user | Yes | user\_details.csv:  Gender |
| Date of Birth | string | String representing the date of birth of the user | Yes | user\_details.csv:  DateOfBirth |
| Phone Number | string | String representing the contact phone number of the user | Yes | user\_details.csv:  PhoneNumber |
| Email | string | String representing the email address of the user | Yes | user\_details.csv:  Email |
| Street Address | string | String representing the street address of the user | Yes | user\_details.csv:  StreetAddress |
| State | string | String representing the state the user resides in | Yes | user\_details.csv:  State |
| City | string | String represents the city the user resides in | Yes | user\_details.csv:  City |
| Country | string | String representing the country the user resides in | Yes | user\_details.csv:  Country |
| Card Type | string | String representing the users card type | Yes | user\_details.csv:  CardType |
| Card Number | string | String representing the users credit card number | Yes | user\_details.csv:  CardNumber |
| Frequent Flyer Points | integer | Frequent flyer points balance of the user | Yes | user\_details.csv:  FrequentFlyerPoints |
| Passport Holder | string | String representing the passport holder status of the user. Typically true or false. | Yes | user\_details.csv:  PassportHolder |
| No-Fly Status | string | String representing the No-Fly status typically blank, watch or no-fly. | Yes | user\_details.csv:  NoFlyStatus |
| Amount due | double | The amount owed to the airline | Yes | user\_ details.csv: AmountDue |

**airports.csv**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Description | Persistent | Store |
| Airport ID | integer | Unique integer used to identify an airport. | Yes | airports.csv:  AirportID |
| Name | string | String representing the name of the airport | Yes | airports.csv:Name |
| City | string | String representing the city where the airport is located | Yes | airports.csv:City |
| Country | string | String representing the country where the airport is located | Yes | airports.csv:Country |
| IATA Code | string | String representing an International Airport Transport Association Code | Yes | airports.csv:  IATACode |
| Latitude | string | Representing the latitudinal location of the airport | Yes | airports.csv:  Latitude |
| Longitude | string | String representing the longitudinal location of the airport | Yes | airports.csv:  Longitude |
| Altitude | string | String representation of the altitude of the airport | Yes | airports.csv:  Altitude |
| Timezone | string | String representing the timezone | Yes | airports.csv:  Timezone |
| DST | String | String representing the daylight savings time | Yes | airports.csv:  DST |
| TZ Database Timezone | string | String representing the TZ database timezone | Yes | airports.csv:  TZDatabaseTZ |

**routes.csv**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Description | Persistent | Store |
| ID | integer | Integer representing the record number in the file | Yes | routes.csv:  ID |
| Origin Airport  IATA Code | string | String representing the origin IATA code for the route | Yes | routes.csv:  OriginAirportIATA |
| Destination Airport  IATA Code | string | String representing the destination IATA code for the route | Yes | routes.csv:  DestAirportIATA |
| Code Share | string | String representing whether or not the route is shared by different airplines. Typically ‘y’ or blank | Yes | routes.csv:  CodeShare |
| Stops | integer | Integer representing the number of stops for the route | Yes | routes.csv:  Stops |

**persons.csv**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Description | Persistent | Store |
| ID | integer | Integer representing the record number in the file | Yes | persons.csv:  ID |
| Title | string | String representing title of an additional person associated with a booking | Yes | persons.csv:  Title |
| First Name | string | First name of an additional person associated with a booking | Yes | persons.csv:  FirstName |
| Last Name | string | Last name of an additional person associated with a booking | Yes | persons.csv:  LastName |
| Gender | string | String representing the gender of an additional person associated with a booking | Yes | persons.csv:  Gender |
| Date of Birth | string | String representing the date of birth of an additional person associated with a booking | Yes | persons.csv:  DateOfBirth |
| Phone Number | string | String representing the contact phone number of an additional person associated with a booking | Yes | persons.csv:  PhoneNumber |
| Email | string | String representing the email address of an additional person associated with a booking | Yes | persons.csv:  Email |
| Street Address | string | String representing the street address of additional person associated with a booking | Yes | persons.csv  StreetAddress |
| State | string | String representing the state of an additional person associated with a booking | Yes | persons.csv:  State |
| City | string | String representing the city of an additional person associated with a booking | Yes | persons.csv:  City |
| Country | string | String representing the country of an additional person associated with a booking | Yes | persons.csv:  Country |
| Card Type | string | String representing the card type of an additional person associated with a booking | Yes | persons.csv:  CardType |
| Card Number | string | String representing the card number of an additional person associated with a booking | Yes | persons.csv:  CardNumber |
| Passport Holder | string | String representing the passport holder status. Typically true or false | Yes | persons.csv:  PassportHolder |
| Frequent Flyer Points | integer | Integer value representing the frequent flyer point balance of an additional person associated with a booking | Yes | persons.csv:  FrequentFlyerPoints |
| No-Fly Status | string | String representation of the no-fly status of an additional person associated with a booking. Typically no-fly, watch or blank | Yes | persons.csv  NoFlyStatus |

**booking\_constants.csv**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Description | Persistent | Store |
| Discount ratio | double | The discount ratio of frequent flier points to a dollar. | Yes | booking\_constants.csv:  DiscountRatio |
| Cancellation fee | double | The cancellation fee for a booking. | Yes | booking\_ constants.csv:  CancellationFee |

**booking.csv**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Type | Description | Persistent | Store | |
| Booking ID | integer | Integer representing the record number in the file | Yes | booking.csv:  BookingID | |
| Flight ID | string | Unique code representing a flight | Yes | booking.csv: FlightID |
| Booking total | double | The total cost of this booking. | Yes | booking.csv:  BookingTotal | |
| Booking time | time | The time this booking was made | Yes | booking.csv:  BookingTime | |

**fleet.csv**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Type | Description | Persistent | Store | |
| Plane ID | integer | Integer representing the record number in the file | Yes | fleet.csv:  PlaneID | |
| Plane model | string | Model of the plane. | Yes | fleet.csv: PlaneModel |
| Total number | integer | Total number of planes of this model in the fleet | Yes | fleet.csv: TotalNumber |
| First class available | integer | Number of first class seats available on this plane | Yes | fleet.csv:  fc\_available |
| Business class available | integer | Number of business class seats available on this plane | Yes | fleet.csv:  bc\_available |
| Premium economy class available | integer | Number of premium economy class seats available on this plane | Yes | fleet.csv:  pc\_available |
| Economy class available | integer | Number of economy class seats available on this plane | Yes | fleet.csv: ec\_available |
| Total seats available | integer | Total number of seats on this plane model | Yes | fleet.csv: TotalAvailable |

**services\_booked.csv**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Description | Persistent | Store |
| Booking ID | integer | Booking that this service booking is associated with. | Yes | services\_booked.csv:  BookingID |
| Service ID | integer | The ID of the service booked. | Yes | services\_booked.csv:  ServiceID |
| Ticket ID | integer | Ticket that this service booking is associated with. | Yes | services\_booked.csv:  TicketID |

**tickets.csv**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Description | Persistent | Store |
| Ticket ID | integer | The ID of this ticket. | Yes | tickets.csv:  TicketID |
| Username | string | The username associated with this ticket. May be null if person ID is not –1. | Yes | tickets.csv:  Username |
| Person ID | integer | The person ID associated with this ticket. May be –1 if username is not null. | Yes | tickets.csv:  PersonID |
| Seat number | string | The seat number associated with this ticket. | Yes | tickets.csv:  SeatNumber |
| Ticket price | double | The price of this ticket including the services booked. | Yes | tickets.csv:  TicketPrice |

**services.csv**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Description | Persistent | Store |
| Service ID | integer | The ID of this service. | Yes | services.csv:  ServiceID |
| Service Name | string | The name of this service. | Yes | tickets.csv:  ServiceName |
| Service Price | double | The price of this service | Yes | tickets.csv:  ServicePrice |
| Service availability | string | The availability of this service. | Yes | tickets.csv:  ServiceAvailability |

## User Interface Design

### Overview

The following section provides details of the User Interface design for the proposed Flight Management System.

### Usability and Design Goals for the Interface

The following usability goals have been identified and categorized into key areas of concern for the design of the user interface. The interface type to be designed will be a command line interface.

#### Effectiveness

The interface should provide all users with an effective way to carry out all flight booking related activities. This will be achieved by –

* Providing relevant selection menus for each user type
* Providing error checking relating to input data
* Providing format indicators for expected input from the user

#### Efficiency

The interface should provide users with an efficient way to carry out all flight booking related activities. This will be achieved by –

* Implementing a shallow depth menu system at the cost of more general menu options and larger menus

#### Learnability

The interface should provide users with an intuitive and familiar way to carry out all flight booking related activities. This will be achieved by –

- Content windows not too densely populated with information

#### Memorability

The interface should provide users with a memorable way to navigate and carry out all flight booking related activities. This will be achieved by –

- Providing relevantly titled menu options

#### Constraints

The interface should provide a good way of notifying users of input errors and giving them an appropriate way to rectify them. This will be achieved by –

* Providing confirmation of user input for booking process
* Error messages for invalid input with prompting for re-entry of input data

#### Consistency

The interface should provide users with a consistent way to carry out all flight booking related activities. This will be achieved by –

- Providing a consistent menu layout across the entire system via the use of numbered menu options

# Summary of Requirements Implemented

## Overview

This section provides a summary of all functionality implemented within the construction phase of the Flight Management System. The functionality implemented is presented in a table listing all functionality identified in the SRS and listing the requirement number, brief description of the requirement and whether or not it was implemented. Note that the requirement ID’s are references to the requirements within the SRS document.

## Implemented Functionality Table

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement ID | Description | Iteration | Implemented |
| F\_3.1.1.1\_01 | The system should allow the administrator to create staff profiles. | Iter. 3 | YES |
| F\_3.1.1.1\_02 | The system should allow the administrator to modify staff profiles. | Iter. 3 | YES |
| F\_3.1.1.1\_03 | The system should allow the administrator to remove staff profiles. | Iter. 3 | YES |
| F\_3.1.1.1\_04 | The system should allow the administrator access to all functionality afforded to all staff. | Iter. 3 | YES |
| F\_3.1.1.2\_01 | The system should allow users (customer/travel agent) to create profiles. | Iter. 2 | YES |
| F\_3.1.1.2\_02 | The system should allow users (customer/travel agent) to modify their profiles. | Iter. 3 | YES |
| F\_3.1.1.2\_03 | The system should allow all users to change their password. | Iter. 2 | YES |
| F\_3.1.1.2\_04 | The system should allow users (customer/travel agent) to remove an existing profile. |  | YES |
| F\_3.1.1.2\_05 | The system should allow all users to login using a user name and password. | Iter. 1 | YES |
| F\_3.1.1.2\_06 | The system should provide the users (customer/travel agent) with domain specific options at the command line upon login to the system. | Iter. 1 | YES |
| F\_3.1.1.3\_01 | The system should allow a Profile Manager to remove user profiles (customer/travel agent) from the system. | Iter. 3 | YES |
| F\_3.1.1.3\_02 | The system should allow a Profile Manager/Reservation Manager to modify a customer profile’s no fly status. | Iter. 3 | YES |
| F\_3.1.1.3\_03 | The system should provide the Profile Manager with domain specific options at the command line upon login to the system | Iter. 3 | YES |
| F\_3.1.2.1\_01 | The system should allow a user (Reservation Manager/normal staff) to transfer a customer from one flight to another. |  | NO |
| F\_3.1.2.1\_02 | The system should allow a user (Reservation Manager/normal staff) modify booking seat selection. | Iter. 3 | YES |
| F\_3.1.2.1\_03 | The system should provide the users (reservation manager/staff) with domain specific options at the command line upon login to the system. | Iter. 3 | YES |
| F\_3.1.2.1\_04 | The system should allow the reservation manager to set ticket pricing of all flights. | Iter. 3 | YES |
| F\_3.1.2.2\_01 | The system should allow a user (customer/travel agent) to make a flight booking. A booking can be made for multiple people. | Iter. 2 | YES |
| F\_3.1.2.2\_02 | The system should allow a user (customer/travel agent) to view and select a flight. Flights with departure dates and times in the past should not be listed. | Iter. 2 | YES |
| F\_3.1.2.2\_03 | The system should allow a user (customer/travel agent) to select a travel class. | Iter. 2 | YES |
| F\_3.1.2.2\_04 | The system should allow a user (customer/travel agent) to view and select a seat. | Iter. 2 | YES |
| F\_3.1.2.2\_05 | The system should display a booking summary to the user (customer/travel agent). | Iter. 2 | YES |
| F\_3.1.2.2\_06 | The system should allow user (customer/travel agent) to view and select in-flight services. Any services selected should be added to booking. | Iter. 2 | YES |
| F\_3.1.2.2\_07 | The system should allow user (customer/travel agent) to modify in-flight services for an existing booking. Any service changes should be made to booking. | Iter. 3 | YES |
| F\_3.1.2.2\_08 | The system should alert a user (customer/travel agent) when no seats are available in selected travel class. | Iter. 3 | YES |
| F\_3.1.2.2\_09 | The system should **not** allow a user (customer/travel agent) to book a flight when customer no fly status is set to “No Fly”. | Iter. 3 | YES |
| F\_3.1.2.2\_10 | The system should warn a user (customer/travel agent) that wishes to book an international flight when customer passport holder status is set to false. The user can still continue with the booking process. | Iter. 3 | YES |
| F\_3.1.2.2\_11 | The system should allow a user (customer/travel agent) to cancel an existing booking. Booking cancellation fee applies. | Iter. 3 | YES |
| F\_3.1.2.3\_01 | The system should provide the Flight Manager with domain specific options at the command line upon login to the system. | Iter. 3 | YES |
| F\_3.1.2.3\_02 | The system should allow the Flight Manager to modify routes for the airline. | Iter. 3 | YES |
| F\_3.1.2.3\_03 | The system should allow the Flight Manager to modify domestic and international airport records. | Iter. 3 | YES |
| F\_3.1.2.3\_04 | The system should allow the Flight Manager to modify the airlines fleet of aircraft. | Iter. 3 | YES |
| F\_3.1.2.3\_05 | The system should allow the Flight Manager to modify the flight schedule. | Iter. 3 | YES |
| F\_3.1.3.1\_01 | The system should provide the Services Manager with domain specific options at the command line upon login to the system. | Iter. 3 | YES |
| F\_3.1.3.1\_02 | The system should allow the Services Manager to modify the in-flight services available for purchase from the airline. | Iter. 3 | YES |
| F\_3.1.4.1\_01 | The system should allow all users to request statistical reports from the reporting subsystem. Access to reports should be context specific. | Iter. 3 | YES |
| F\_3.1.4.1\_02 | The system should provide the Reporting Manager with domain specific options at the command line upon login to the system. | Role Removed Incorrect requirement | REMOVED |
| F\_3.1.4.1\_03 | The system should display reports relating to (daily/monthly/annual) airline revenue. | Iter. 3 | YES (Partial) |
| F\_3.1.4.1\_04 | The system should display reports relating to (daily/monthly/annual) service popularity. | Iter. 3 | YES(Partial) |
| F\_3.1.4.1\_05 | The system should display reports relating to (daily/monthly/annual) route popularity. | Iter. 3 | YES(Partial) |
| F\_3.1.4.1\_06 | The system should display reports relating to user (customer/travel agent) statistics and historical data. | Iter. 3 | YES |

# Summary of Construction Phase

## Overview

This section provides details of the Construction Phase using the Rational Unified Process. Included is the original project plan for the iterations of this phase, as well as a defect and integration summary. This phase is comprised of five iterations targeting incremental implementation of core, essential and optional requirements from the Software Requirement Specification.

## Iteration 1 of Construction Phase

### Plan

This section includes the plan for the first iteration of the Construction phase. The duration of this iteration is 22/04/15 to 05/05/15.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Task # | Artifact | Responsible | Completion Dates | | |
| Planned | Revised | Actual |
| 1.1 | Review iteration plans | Matthew | 23/05/15 |  | 23/05/15 |
| 1.2 | Review design | Michael  Jicheng  Siyuan | 25/05/15 |  | 27/05/15 |
| 1.3 | Implementation and testing |  | 05/05/15 |  | 04/05/15 |
| 1.3.1 | Core functionality –  - Login | Jicheng  Siyuan | 03/05/15 |  | 02/05/15 |
| 1.3.2 | Test core functionality | Jicheng | 04/05/15 |  | 04/05/15 |
| 1.3.3 | Report on functionality added | Jicheng  Siyuan | 05/05/15 |  | 04/05/15 |

### Summary of Iteration

The following is a list of activities undertaken within this iteration.

* Created the basic structure of the system that includes the main boundary class that the user will interact with, the controller class that will coordinate all interactions between the entity classes and the interface class, and the user entity class that enables users to login and signup to the system.
* Login functionality

#### Classes Creation and Modification

The following is a list of classes created or modified within the iteration.

* UserInteraction package:
  + - UserController.java
    - UserEntity.java
    - UserInterface.java
  + role package:
    - Role.java
    - AbstractCustomerRole.java
    - CustomerRole.java
    - TravelAgencyRole.java
    - RoleFactory.java
  + Main file:
    - FlightManagementSystem.java
  + Database files:
    - userAccount.csv

#### Testing Procedures Summary

The following is a list of informal testing procedures carried out by the individual implementers to confirm satisfactory performance of code written.

* We tested the main interface of the system by running the program and comparing the program’s output against the expected output.
* To test the login functionality, we created test login details in the database (because signup functionality is not implemented yet) and we tried logging into the system with the same login details.

#### Integration Summary

All integration went smoothly with nothing of significance to report.

## Iteration 2 of Construction Phase

### Plan

This section includes the plan for the first iteration of the Construction phase. The duration of this iteration is 06/05/15 to 16/05/15.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Task # | Artifact | Responsible | Completion Dates | | |
| Planned | Revised | Actual |
| 2.1 | Review iteration plans using functionality report from previous iteration | Matthew | 07/05/15 |  | 07/05/15 |
| 2.2 | Review design | Michael  Jicheng  Siyuan | 09/05/15 |  | 09/05/15 |
| 2.3 | Implementation and testing |  | 19/05/15 |  |  |
| 2.3.1 | Further core functionality  - Logout  - Signup  - Modify Password  - Booking | Jicheng  Siyuan  Michael | 17/05/15 | Booking functionality still incomplete 21/05 | Booking system more testing 21/05/15 |
| 2.3.2 | Test functionality added | Jicheng | 18/05/15 | 15/05/15 | 15/05/15 |
| 2.3.3 | Report on functionality added | Jicheng  Siyuan  Michael | 19/05/15 | 16/05/15 | 16/05/15 |

### Summary of Iteration

The following is a list of activities undertaken within this iteration.

* Finished the skeleton of all the role classes and the options that will be offered by each role. These options are just text strings that are used to build the main menu for users of that particular role and do not refer to actual system functionality.
* Changed Role.java from an interface to an abstract class to allow subclasses to have a common method to create the main menu.
* Implemented the logout functionality
* Implemented the signup functionality
* Implementing the booking functionality. This includes transactions such as adding user bookings, booked seats, booked services and payment.
* Added change password functionality.

#### Class Creation and Modification

The following is a list of classes created or modified within the iteration.

* role package:
  + FlightManagerRole.java
  + ProfileSystemManagerRole.java
  + ReportingSystemManagerRole.java
  + ReservationSystemManagerRole.java
  + ServiceSystemManagerRole.java
  + SystemAdministratorRole.java
* helpers package:
  + UserLoginDetails.java
  + Booking.java
  + Flight.java
  + Person.java
  + Service.java
  + ServiceBooking.java
  + Ticket.java
* booking package:
  + AirportController.java
  + AirportEntity.java
  + BookingController.java
  + CustomerController.java
  + FlightController.java
  + FlightEntity.java
  + PersonController.java
  + PersonEntity.java
  + PlaneController.java
  + RouteController.java
  + RouteEntity.java
  + SeatController.java
  + ServiceController.java
* profile package:
  + ProfileController.java
  + ProfileEntity.java
* Database files:
  + userDetail.csv
  + airports.csv
  + flight-route-schedule.csv
  + routes.csv
  + discount.csv
  + fleet.csv
  + otherPersons.csv
  + servicesInventory.csv

Modification:

* helpers package:
  + Booking.java
  + Flight.java
  + Person.java
  + UserLoginDetails.java 🡪 Customer.java
* userInteraction package:
  + UserController.java
  + UserEntity.java
  + UserInterface.java
* booking package:
  + AirportController.java
  + AirportEntity.java
  + BookingController.java
  + FlightController.java
  + FlightEntity.java
  + PersonController.java 🡪 moved to profile package
  + PersonEntity.java 🡪 moved to profile package
  + RouteController.java
  + RouteEntity.java
  + ServiceEntity.java
* role package:
  + Role.java
  + AbstractCustomerRole.java
  + CustomerRole.java
  + TravelAgencyRole.java
  + RoleFactory.java
* Main file:
  + FlightManagementSystem.java
* Database files:
  + fleet.csv
  + flight-route-schedule.csv
  + userAccount.csv

Deletion:

* booking package:
  + CustomerEntity.java
  + CustomerController.java
  + PlaneController.java

#### Testing Procedures Summary

The following is a list of informal testing procedures carried out by the individual implementers to confirm satisfactory performance of code written.

* We tested the individual role classes by creating test login details in the database for each role. Then we tried logging into the system with the same login details for each particular role.
* We tested the signup functionality by just performing the actions described in the Signup use case. Then we tried to login to the system using the newly created details.
* We tested the logout functionality by choosing the logout option from the main menu.
* We tested the partial functionality of the booking system by trying both expected input and unexpected input.
* Tested the change password functionality by changing the password for an account, and trying to login to the system using the old password and then trying to login to the system using the new password.

#### Integration Summary

During the integration process for the iteration errors encountered included –

* Method names that were incorrectly spelled
* Methods having the wrong return type
* Missing import statements from programmer not using an IDE

No other significant problems arose during the integration for the iteration.

## Iteration 3 of Construction Phase

### Plan

This section includes the plan for the third and final iteration of the Construction phase. The duration of the iteration is 17/05/15 to 02/06/15.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Task # | Artifact | Responsible | Completion Dates | | |
| Planned | Revised | Actual |
| 3.1 | Review iteration plans using functionality report from previous iteration | Matthew | 17/05/15 | 21/05/15 | 21/05/15 |
| 3.2 | Review design | Michael  Jicheng  Siyuan | 22/05/15 | Team happy with design | N/A |
| 3.3 | Implementation and testing |  | 01/06/15 |  |  |
| 3.3.1 | Implement as much remaining essential functionality within timeframe | Jicheng  Siyuan  Michael | 30/05/15 |  | 30/05/15 |
| 3.3.2 | Test functionality added | Jicheng | 31/05/15 |  | 31/05/15 |
| 3.3.3 | Report on functionality added | Jicheng  Siyuan  Michael | 31/05/15 |  | 30/05/15 |
| 3.4 | Compile and complete required accompanying documentation | Matthew | 01/06/15 |  | TBD |

### Summary of Iteration

The following is a list of activities undertaken within this iteration.

* Extended booking functionality. Customers and travel agencies are able to book flights, seats and services on flights.
* Implemented the edit account details functionality. Customers and travel agencies are able to edit the details of their account.
* Implemented the cancel booking functionality. Customers and travel agencies can cancel bookings they have made. They will be charged a cancellation fee.
* Added the Normal Staff role into the system. Now all manager classes descend from this class, and in turn, this Normal Staff role class descends from the topmost Role class.
* Added cancel customer booking, make booking for customer, and edit services for customer functionality for the Normal Staff. Manager role classes also receive these functionalities.
* Added the add services, remove services, edit service price, and edit service availability functionality for the Service System Manager role. Now the Service System Manager can perform all these services.
* Added the set ticket prices, change frequent flier points discount ratio and set cancellation fee functionality for the Reservation System Manager.
* Added edit travel agency and customer profile, and close user account functionality for the Profile System Manager.
* Added the edit watch and no fly list for the Profile System Manager and Reservation System Manager. Now they can edit the fly status of customers.
* Added the manage routes, manage fleet, manage flight schedule and manage airports functionality for the Flight Manager.
* Added the reporting subsystem. However, not all the roles have reports available yet.
* Removed the Reporting System Manager role because of a mismatch in the requirements. This role was not needed in the first place.

#### Class Creation and Modification

The following is a list of classes created or modified within the iteration.

* helpers package:
  + Airport.java
  + Plane.java
  + Route.java
  + Staff.java
* main package:
  + FlightManagementSystem.java
* report package:
  + ReportBuilder.java
* role package:
  + NormalStaffRole.java
* Database files:
  + booking.csv
  + services\_booked.csv
  + tickets.csv
  + booking\_constants.csv

Modification:

* booking package:
  + AirportController.java
  + AirportEntity.java
  + BookingController.java
  + BookingEntity.java
  + FleetController.java
  + FleetEntity.java
  + FlightController.java
  + FlightEntity.java
  + RouteController.java
  + RouteEntity.java
  + ServiceController.java
  + ServiceEntity.java
* helpers package:
  + Booking.java
  + Customer.java
  + Flight.java
  + Person.java
  + Service.java
  + ServiceBooking.java
  + Ticket.java
* profile package:
  + ProfileController.java
  + ProfileEntity.java
* role package:
  + AbstractCustomerRole.java
  + CustomerRole.java
  + FlightManagerRole.java
  + ProfileSystemManagerRole.java
  + ReservationSystemManagerRole.java
  + Role.java
  + RoleFactory.java
  + ServiceSystemManagerRole.java
  + SystemAdministratorRole.java
  + TravelAgencyRole.java
* Database files:
  + airports.csv
  + booking.csv
  + fleet.csv
  + flight-route-schedule.csv 🡪 flight\_schedule.csv
  + otherPersons.csv 🡪 persons.csv
  + routes.csv
  + serviceInventory.csv 🡪 services.csv
  + services\_booked.csv
  + tickets.csv
  + userAccount.csv 🡪 user\_account.csv
  + userDetail.csv 🡪 user\_details.csv

Deletion:

* role package:
  + ReportingSystemManager.java
* profile package:
  + PersonController.java
  + PersonEntity.java
* booking package:
  + SeatController.java

#### Testing Procedures Summary

The following is a list of informal testing procedures carried out by the individual implementers to confirm satisfactory performance of code written.

* Tested the booking functionality by running through all possible paths through the booking procedure.
* Tested the cancel booking functionality by running the procedure and checking whether the correct flights associated with the user are shown. Then after user has selected a flight to cancel, the system should change the status of the booking to “Cancelled”.
* Tested the edit account details functionality by editing each possible detail of a user and checking whether it is properly reflected in the database file.
* Tested the view all bookings functionality by running the procedure and checking whether the correct flights associated with the user are shown.
* Tested the adding of the Normal Staff role and its functionalities by trying to access those functionalities from the Normal Staff role and also from its subclasses.
* Tested the functionalities for the Service System Manager role by running through all the functionalities and checking for abnormalities and errors. All functionalities were found to be working well.
* Tested the functionalities for the Reservation System Manager by running through all the functionalities and checking for abnormalities and errors. All functionalities were found to be working well.
* Tested the functionalities for the Profile System Manager by running through all the functionalities and checking for abnormalities and errors. All functionalities were found to be working well.
* Tested the edit watch and no fly list functionality by trying to change the fly status of a customer from the Profile System Manager and Reservation System Manager roles. Found to be working well.
* Tested the functionalities for the Flight Manager by running through all the functionalities and checking for abnormalities and errors. All functionalities were found to be working well.
* Tested the functionalities of the reporting subsystem. The reports that have been implemented were found to be working well.
* Tested the system since the deletion of the Reporting System Manager role. The system is found to be working well.
* Some defects found where the input buffer would contain a newline character and would cause the system to receive invalid input.

#### Integration Summary

No significant problems arose during the integration for the iteration.

# Code Listing

## Overview

This section provides a code listing of all code that is included in the final product at the completion of the construction phase. Also included is the unit testing code created to verify implementation correctness.

These listings can be found on the CD submitted with this report.

# Testing

## Overview

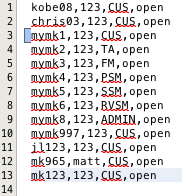
This section contains all details relevant to the formal testing of the code written for the Flight Management System, including black and white box test cases and test reports.

## Black Box Test Cases

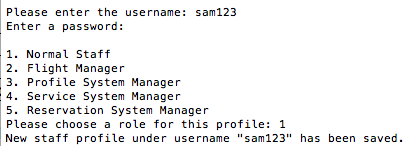
This section contains all black box testing carried out on the Flight Management System.

|  |
| --- |
| **Test Case ID:** 1 |
| **Actors Affected:** System Administrator, Normal Staff |
| **Purpose:** To check whether a staff profile is created correctly. |
| **Expected Results:**   1. The database should contain the details entered into the system. 2. The user should be able to login to the system with the same details. 3. The user should be given the correct role. |
| **Output:**   1. The database contains the same details entered into the system 2. The user is able to login to the system with the same details. 3. The user is given the correct role. |
| **Pass/Fail:** Pass |
| **Resultant Changes:** None |
| **Error Messages:** None |

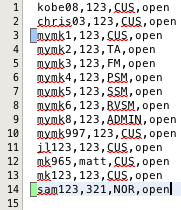
**Database entries before new record:**



**User input into the system:**

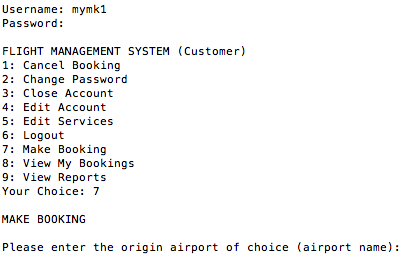


**Database entries after new record:**

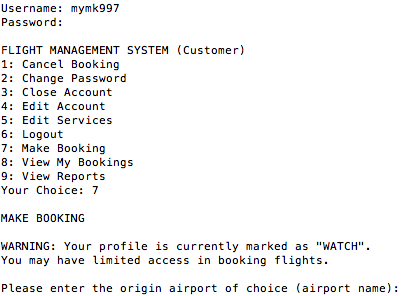


|  |
| --- |
| **Test Case ID:** 2 |
| **Actors Affected:** Customer |
| **Purpose:** To check whether the system responds correctly to the three types of fly status of a customer: “null” (means can fly), “watch” or “no fly”. |
| **Expected Results:**   1. The system should retrieve the correct fly status for the current customer. 2. Based on the retrieved fly status, the system should perform the appropriate actions:    * A customer that is not marked on the no fly list (“null”) is allowed to perform bookings.    * A customer that is marked “watch” is shown a warning message by the system, but is still allowed to perform bookings.    * A customer that is marked “no fly” is not allowed to make a flight booking. |
| **Output:**   1. The system retrieves the correct fly status for the current customer. 2. A customer that is not marked on the no fly list (“null”) was allowed to perform bookings. 3. A customer that is marked “watch” was shown a warning message by the system, but was still allowed to perform bookings 4. A customer that is marked “no fly” was shown a warning message and was not allowed to make a flight booking. |
| **Pass/Fail:** Pass |
| **Resultant Changes:** None |
| **Error Messages:** None |

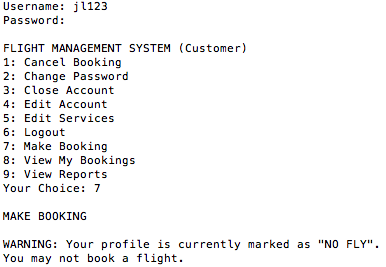
**Customer that can fly:**



**Customer that has “watch” on his record**:



**Customer that has “no fly” on his record:**

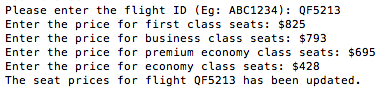


|  |
| --- |
| **Test Case ID:** 3 |
| **Actors Affected:** Reservation System Manager |
| **Purpose:** To check whether the system changes the price of the correct flight selected by the user. |
| **Expected Results:**   1. The system should store the price input from the user in the flight\_schedule.csv file. 2. The system should update the prices of the correct flight. |
| **Output:**   1. The system stored the price input from the user in the flight\_schedule.csv file. 2. The system updated the prices of the correct flight. |
| **Pass/Fail:** Pass |
| **Resultant Changes:** None |
| **Error Messages:** None |

**Database entry before update:**



**User input into the system:**



**Database entry after update:**

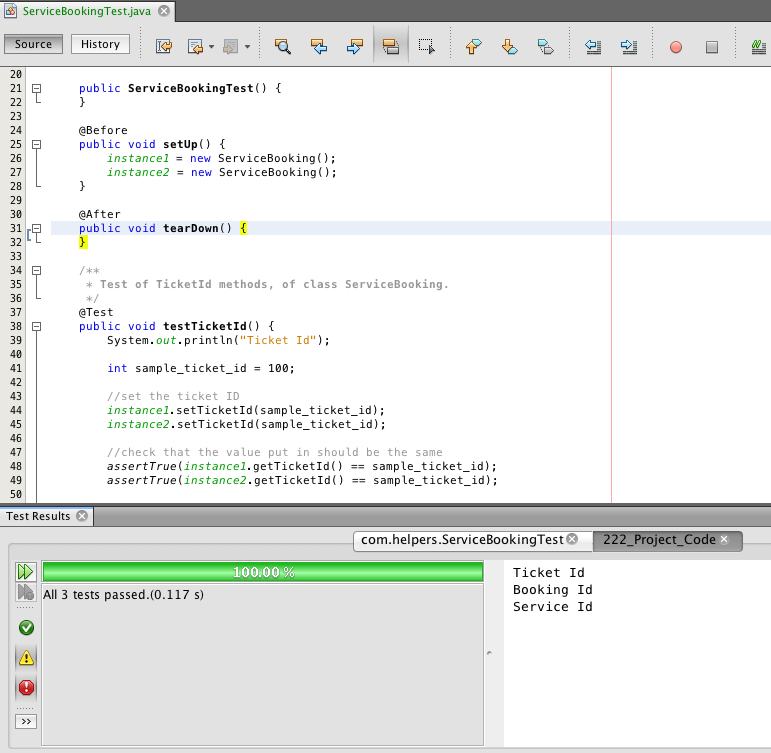


## White Box Test Cases

This section contains all white box testing carried out upon the Flight Management System.

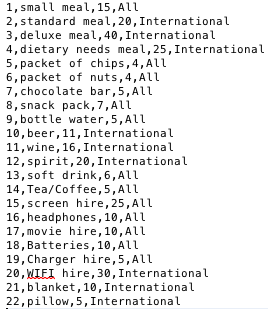
|  |
| --- |
| **Test Case ID:** 4 |
| **Class Tested:** ServiceBooking.java |
| **Tester Class:** ServiceBookingTest.java |
| **Number of Subtests:** 3 |
| **Purpose:** To check whether the get and set methods of ServiceBooking.java are correct and work as expected. |
| **Expected Results:**   1. We are testing the three attributes of the class and the corresponding get and set methods of these attributes:  * bookingId: int * ticketId: int * serviceId: int  1. For each attribute, we will test whether the value stored in the class matches what was passed to it initially. 2. For bookingId:    1. We pass in a value of 5 to the setBookingId method of two instances of the ServiceBooking class.    2. Then we assert that the value returned by the getBookingId method must be the same as the value that was passed in earlier.    3. Lastly we assert that the value returned by the getBookingId method of the first instance must be the same as the value returned by the getBookingId method of the second instance. 3. For ticketId:    1. We pass in a value of 100 to the setTicketId method of two instances of the ServiceBooking class.    2. Then we assert that the value returned by the getTicketId method must be the same as the value that was passed in earlier.    3. Lastly we assert that the value returned by the getTicketId method of the first instance must be the same as the value returned by the getTicketId method of the second instance. 4. For serviceId:    1. We pass in a value of 38 to the setServiceId method of two instances of the ServiceBooking class.    2. Then we assert that the value returned by the getServiceId method must be the same as the value that was passed in earlier.    3. Lastly we assert that the value returned by the getServiceId method of the first instance must be the same as the value returned by the getServiceId method of the second instance. |
| **Output:**   1. The class passed all three subtests. No errors were recorded. |
| **Pass/Fail:** Pass |
| **Resultant Changes:** None |
| **Error Messages:** None |

**Part of the test code with the test results:**

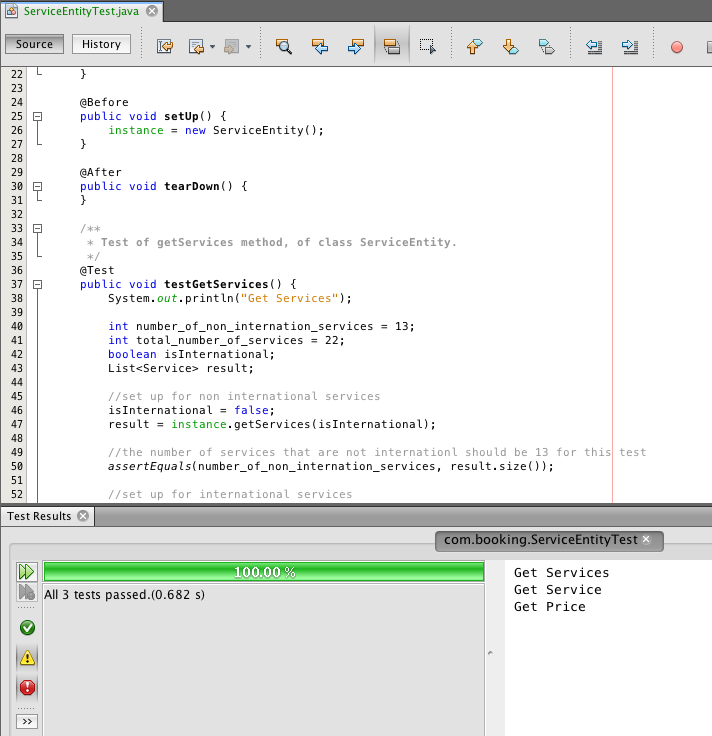


|  |
| --- |
| **Test Case ID:** 5 |
| **Class Tested:** ServiceEntity.java |
| **Tester Class:** ServiceEntity Test.java |
| **Number of Subtests:** 3 |
| **Purpose:** To check whether the database mapping methods of ServiceEntity.java are correct and work as expected. |
| **Expected Results:**   1. We are testing 3 data retrieval methods of the ServiceEntity.java class: 2. getServices(): List<Service> 3. getService(serviceId: int): Service 4. getPrice(serviceId: int): double 5. For item i., we will first get a List of non-international services. Next, we will get a List of all services. Refer to the screenshot of the test data below this table. 6. For item ii., we will create a new Service object with the attributes in the the database entry with service ID of 1. Based on this Service, we will retrieve the same service from the database and compare it for equality against our created Service object. 7. For item iii., we selected item with ID of 12. So the corresponding price for this service is $20. So we assert that this must be true. |
| **Output:**   1. The class passed all three subtests. No errors were recorded. |
| **Pass/Fail:** Pass |
| **Resultant Changes:** None |
| **Error Messages:** None |

**Test data:**



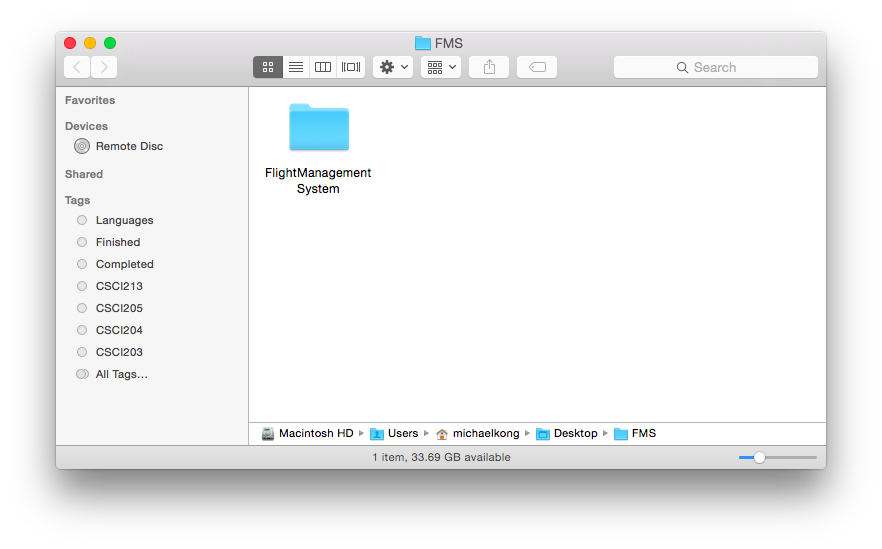
**Part of the test code with the test results:**



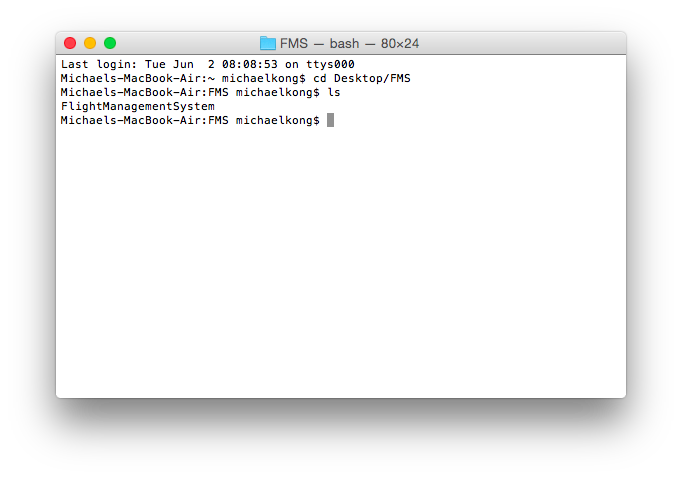
# Installation Instructions

Installation instructions for the Flight Management System have also been included in this document. These instructions are outlined below and include screenshots of the process.

1. Download the “FlightManagementSystem” file and place it in a directory of your choice.

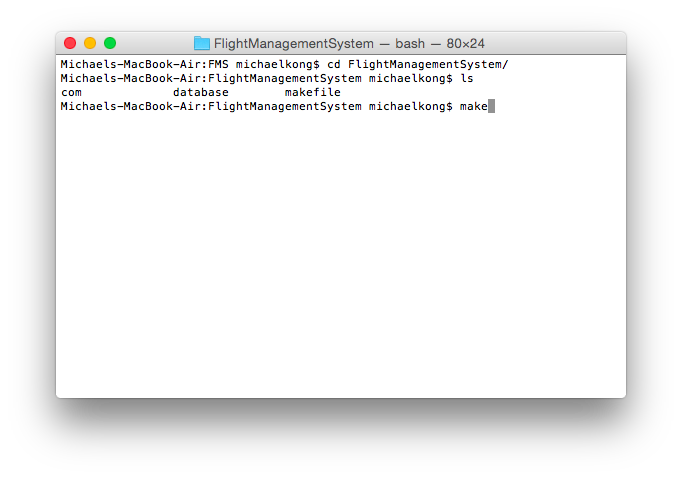


2. Open “Terminal” on UNIX systems or “Command Prompt” in Windows systems. Then navigate to the directory that you placed the “FlightManagementSystem” file in.

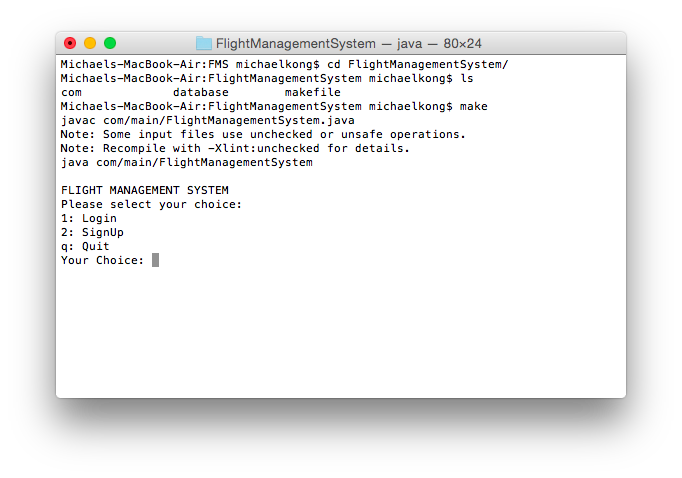


3. Navigate into the FlightManagementSystem folder and execute the following command:

$ make



4. The Flight Management System should start running.



# Meta-Report

## Overview

This section details the group’s work for the Elaboration and Construction phases and includes a summary of work completed by members, member contributions, group meeting records, work diaries for each team member and Github logs of documents and code committed to the online project repository.

## Work Completion Summary

The work completion summary includes a table with each group member, the role they filled and the tasks completed.

|  |  |  |
| --- | --- | --- |
| Group Member | Role | Deliverable |
| Michael Kong | System Architect | 1. Sequence Diagrams 2. State Diagrams 3. Refined Class Diagrams 4. Defect Reports 5. Integration Reports 6. Use-Case diagrams and descriptions 7. Code 8. Unit Testing |
| Matthew King | Team Manager | 1. Project plans 2. Reviewed Iteration Plans 3. Data Persistence Table 4. Package Diagram 5. Final report |
| Jicheng Li | Lead System Developer and Tester | 1. Defect Reports 2. Integration Reports 3. Code |
| Siyuan Chen | System Developer and Tester | 1. Writing of Test Cases 2. Unit Testing |

## Group Meeting Summary

The following table lists meeting number and date, project status review (manager), work completed during meeting, urgent action item identification. It endeavors to summarize work completed throughout the projec

Attendance Status –

Meeting 9 – Matthew Absent

All team members were present at all other meetings

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Meeting | Agenda Items | Status Review | Work Completed | Urgent Action Items |
| 6 – 27/4/15 | Discussed BCE and architecture of system, database type decided, iteration plans discussed | N/A | - Initial Sequence diagrams  - Reviewed iteration plan  - Begun work on prototype  - Decided on csv database  - Action Items identified | - Continue work on prototype, specifically login and logout functionality (Michael, Jicheng)  - Create user database using csv  - Class refinements using BCE |
| 7 – 4/5/15 | Discussed functionality to implemented following the successful implementation of login/logout | Login and logout functionality completed. Prototype working well. Basic but functioning.  - Project meeting goals thus far. Team manager happy with current progress | - Begun signup functionality  - Begun Booking process functionality  - Action items identified | - Implement signup (Jicheng)  - Implement Booking process (Michael, Jason) |
| 8 – 11/5/15 | Current state of implemented functionality, discussion about architectural patterns | Signup functionality completed. Booking is taking more time than expected.  - Project may be little behind schedule. Iteration 2 deadline for booking functionality may need revising | - Layered architectural pattern decided upon as BCE naturally fits this pattern  - Decided to adopt the Delegation pattern for some classes  - Data persistence table started  - Action items identified | - Booking functionality controllers (Michael)  - Booking functionality entities (Jicheng)  - Test cases (Siyuan)  - Data persistence (Matthew) |
| 9 – 18/5/15 | Manager absent. No agenda items recorded | Booking process almost complete. Good progress was made during the past week.  - Project a little behind planned schedule. Due dates will need revising in plan | - Action items identified | - Finalize Booking functionality (Michael, Jicheng)  - Add as much remaining essential functionality as possible for the final iteration (Michael, Jicheng) |
| 10 – 25/4/15 | Architectural elements of system, current functionality implemented, diagram status | Integration reports show substantial amount of implemented functionality.  - Manager optimistic of strong finish | - Report structure decided and fleshed-out  - Action items identified | - Finalize diagrams (Michael)  - Finalize code (Michael, Jicheng)  - Compile final report (Matthew) |

## Team Member Weekly Diaries

Work diaries for all team members can be found in this section.

### Work Diary – Matthew King (Team Manager)

**Week 6, April 27 – May 03**

**Events of the week**

**Group Meeting 6 -**

This meeting involved planning the iterations for the newly entered construction phase. We also discussed refining the domain model into a class diagram that would fit the BCE framework. This prompted some initial work on sequence diagrams for early functionality. Michael and Jicheng would start work on login and logout functionality for the system in the meeting and continue through the coming week. We also decided to use CSV textfile database for the project to keep things less complicated so we could focus more on the functionality of the system.

Tasks –

1. Review of project plan for construction phase

Time Spent on Tasks –

* Task 1 = 1hrs

Summary of Deliverables –

After some discussion with the team I am able to put together a revised iteration plan with some brief outline of what functionality should be covered during each iteration.

**Week 7, March 04 – 10**

**Events of the week**

**Group Meeting 7 –**

Group meeting 7 involved discussion of current state of functionality implemented. Michael and Jicheng had gotten together on Saturaday (May 02) and finialised the login/logout functionality of the system. This was somewhat exciting as these were the first steps we had taken in moving toward a functioning system, even if it was some simple and basic functionality. The next functionality to implement was decided as being signup and the beginnings of the booking process.

Tasks (completed during meeting as a group)–

N/A

Time Spent on Tasks –

N/A

Summary of Deliverables –

Other subject commitments impeded all progress on the project for many of us this week. Unavoidable. This will likely put us a little behind schedule in the coming weeks.

**Week 8, May 11 – 17**

**Events of the week**

**Group Meeting 8 -**

During group meeting 8 we discussed the current state of the functionality implemented and came to the realization that we were running behind schedule. The booking process was taking more time to implement than first thought. We also discussed other open issues relating to the booking process that were encountered during the implementation. I begun work on putting together some of the documentation required for the final report. Action items for the coming weeks were also identified.

Tasks –

1. Data persistence
2. Formal description of architecture

Time spent on tasks –

* Task 1 = 1.5 hrs
* Task 2 = 1 hr

Summary of Deliverables –

After a few hours work on both data persistence and architecture description, the data persistence table was completed and a good foundation was established for the rest of the report with the architectural description. The description still needs some work but it a good start.

**Week 9, May 18 – 24**

**Events of the week**

**Group Meeting 9 –**

I was absent from this group meeting as a result of other subject commitments. However some good progress was made relating to the booking process during the past week. Michael and Jicheng had 90% implemented the booking process which was a nice surprise. Siyuan had begun work on writing the test cases for the formal testing of the project. There was still much to do but the booking process functionality was a big requirement mostly out of the way.

Tasks –

1. Finalize report structure and begin compiling

Time spent on tasks –

* Task 1 = 5hrs

Summary of deliverables

Some time was spent this week working on the final report. Formalizing our design in the report proved to a little more difficult than I had anticipated. Completed as much as possible pending the rest of the work from other group members in the coming weeks.

**Week 10, May 25 – June 02**

**Events of the week**

**Group Meeting 10 –**

This would be the final group meeting for the entire project apart from the planned presentation run-through before Tuesday (June 02). We all agreed to try and complete as much remaining essential functionality as possible in the coming week. All group members would be very busy in the coming week as functionality needs finalizing, diagrams need revising and the report needs to be compiled.

Tasks –

1. Compile final report including review of diagrams

Time spent on tasks –

* Task 1 = 24hrs

Summary of deliverables

A huge amount of time was spent finalizing documentation this week. At the time of writing this the report was largely completed pending some diagram review from other members.

The project as a whole went quite well. Most functionality was implemented, team dynamic was fairly good, my management was ok but could do with some work. This project has provided an important learning experience moving forward into CSCI321.

### Work Diary – Siyuan Chen (System Developer and Tester)

**Week 6, April 27 – May 03**

**Events of the week**

**Group Meeting 6 –**

We have finished the mid-report. So next step is that need to start the coding. In the meeting, we discuss each use cases and redesign the domain model, in order to make the main function of program more clear and understandable. Such as the user can just exactly do these action, this is helpful for the future design and implement.

**Week 7, March 04 – 10**

**Events of the week**

**Group Meeting 7 –**

In this period, Jason and Michael need to do the designing and coding of the whole system. Besides Matt and me will do some research of the other part. I’m assigned to write the testing part. So I have watched the example of test plan and test case documentation to make sure the format and details.

**Week 8, May 11 – 17**

**Events of the week**

**Group Meeting 8 –**

I have wrote the template of test plan and some test cases based on the lecture notes and SRS. But I think there are few test cases look like a little redundant. So just waiting for the programmer to ensure whether those are needed. And I also need to generate whole test plan and test case as soon as possible.

**Week 9, May 18 – 24**

**Events of the week**

**Group Meeting 9 –**

The test plan and test case have been created. Some parts maybe need to modify until the whole system finished. After all the parts have been prepared, we will begin to test the system. If any part won’t work successfully, the error information will send to the programmer and modify the program.

When all parts done, we will finished the whole final report.

### Work Diary – Jicheng Li (Lead System Developer and Tester)

**Week 6, April 27 – May 03**

**Events of the week**

**Group Meeting 6 –**

This week we discussed mostly the coding structure of the whole system. We decided to use the “user—interface—controller—entity—database” model, encapsulating the information among different parts. For example, the database is only accessed by the entity classes and the user can only operate on the interfaces.

Then we set the plan for the login and sign up/sign out, which will be done by the end of this week.

**Week 7, March 04 – 10**

**Events of the week**

**Code Session –**

During this meeting, we managed to construct the overall coding structure of the system. Specifically, the packages and folder structures were confirmed. Then we finished the login functionality and almost had the sign up functionality done.

We also defined some inside rules, such as the abbreviations of the different roles in the system and how the Role class and its subclasses work with the UserController class. The plan for the next week was to finish the sign up and logout functionality, then move on to the next step. Today’s meeting was from 10:30am to 5:45pm, until the library was closed.

**Group Meeting 7 –**

We completed the logout and sign up functionality. Then we discussed about the roles of different classes in the system. We wrote most of the getter/setter methods for these classes. We defined most of the entity classes, which will directly talk to the database. The code of interaction to the database part is assigned to me, and the roles part is assigned to Michael. The booking functionalities need our cooperation.

**Week 8, May 11 – 17**

**Events of the week**

**Group Meeting 8 –**

This week’s meeting moved from Monday to Tuesday because of the busy week for many other assignments’ submissions. We spent about 4 hours together on coding the booking functionality, on the Boundary-Controller-Entity framework. There are several questions left after our discussion. We need to ask Daniel or Dr.Hoa during the lab about them. The detailed question list is in the meeting summary.

**Week 9, May 18 – 24**

**Events of the week**

**Group Meeting 9 –**

We mainly worked on the booking procedures. My task was to complete the methods defined in BookingEntity class, FlightEntity Class, AirportEntity Class, FleetEntity Class and RouteEntity Class. The controller would pass the arguments and call the functions in these entity classes, then I connect to the database and search/update the records.

**Week 10, May 25 – June 02**

**Events of the week**

**Group Meeting 10 –**

Today we almost finished all the functionalities of the booking procedure and almost finished the service parts. Then we added some supplementary features to most of the roles and entities according to the SRS. The report subsystem is the last task to accomplish. Then during the next several days, we will focus on debugging the system as a whole using the test cases and finished all the remaining diagrams.

The last thing is to prepare for the presentation.

### Work Diary – Michael Kong (System Architect)

**Week 6, April 27 – May 03**

**Events of the week**

**Group Meeting 6 –**

Work to do includes redesigning the domain model so that the program will function as per the SRS. Work also includes minor programming. Estimated time in total to finish tasks is about 24 hours and should finish by Saturday, late afternoon.

**Week 7, March 04 – 10**

**Events of the week**

**Code Session –**

Worked from 10:30am to around 4:30pm with Jason on the system. Managed to finish the overall structure of the system and the login functionality. Also solidified certain constants in the system such as the abbreviations of the different roles in the system and how the Role class and its subclasses work with the UserController class. Partially completed the logout and register functionality. Finishing these functionality should take about two more hours.

**Group Meeting 7 –**

Usual Monday meeting. Finished the logout and register functionality. Also finished the skeleton of the classes for the different roles in the system. Just need to fill those skeleton classes with actual methods that will give the class its particular functionality. Including all the entity classes to talk to the database files, we would need about 2 weeks to finish it. Still having trouble implementing the System Administrator’s role class. Not sure how the delegation pattern would help me in this area, but some more research might prove helpful in solving this problem.

**Personal Note –**

Managed to solve the System Administrator’s role class using the delegation pattern. So the System Administrator can now use the system as an anonymous user under a different role. Best part is that there is no duplication of code, which will be useful should the need arise to update functionalities particular to a role. Now that all the skeleton classes for the roles are complete, we will start on the actual functionality of the system starting with the Customer and Travel Agency actors since they are key users of the system. Implementing the functionalities of these two actors should take us another Saturday.

**Week 8, May 11 – 17**

**Events of the week**

**Group Meeting 8 –**

Work is not going as expected. We are clearly behind schedule. But with other assignments in hand, we have to go for the features that will have the highest returns. So far we have implemented the delegation design pattern and it works well, considering it is the first time that we have ever used it in our programming lives. So far, we have the booking system underway, but I am struggling to implement the Boundary-Controller-Entity framework because it is the first time I am using such a framework. However, I see the framework as a useful tool in separating concerns in the system; hence, finding and isolating buggy code can be quicker and more convenient. We will try to finish up as much as we can of the actual booking functionality this week.

**Week 9, May 18 – 24**

**Events of the week**

**Group Meeting 9 –**

We are still behind schedule, but we have made some good progress. We have implemented, almost completely, the booking procedure. All that is left is the actual saving of the booking to the database files, the displaying of the booking summary and the payment section. We will look into finishing the rest of the booking procedure by next week.

**Week 10, May 25 – June 02**

**Events of the week**

**Group Meeting 10 –**

Managed to bounce back into schedule. Work completed this week included finishing the booking procedure and additions of more features to each role. We just started the report subsystem this week but it shouldn't be too hard. We plan to do two to three reports that will allow users to gain useful insight into their particular roles. This upcoming week includes finishing up the system as a whole, finishing up the various diagrams for the system and preparing for project presentation.

## Member Contribution Summary

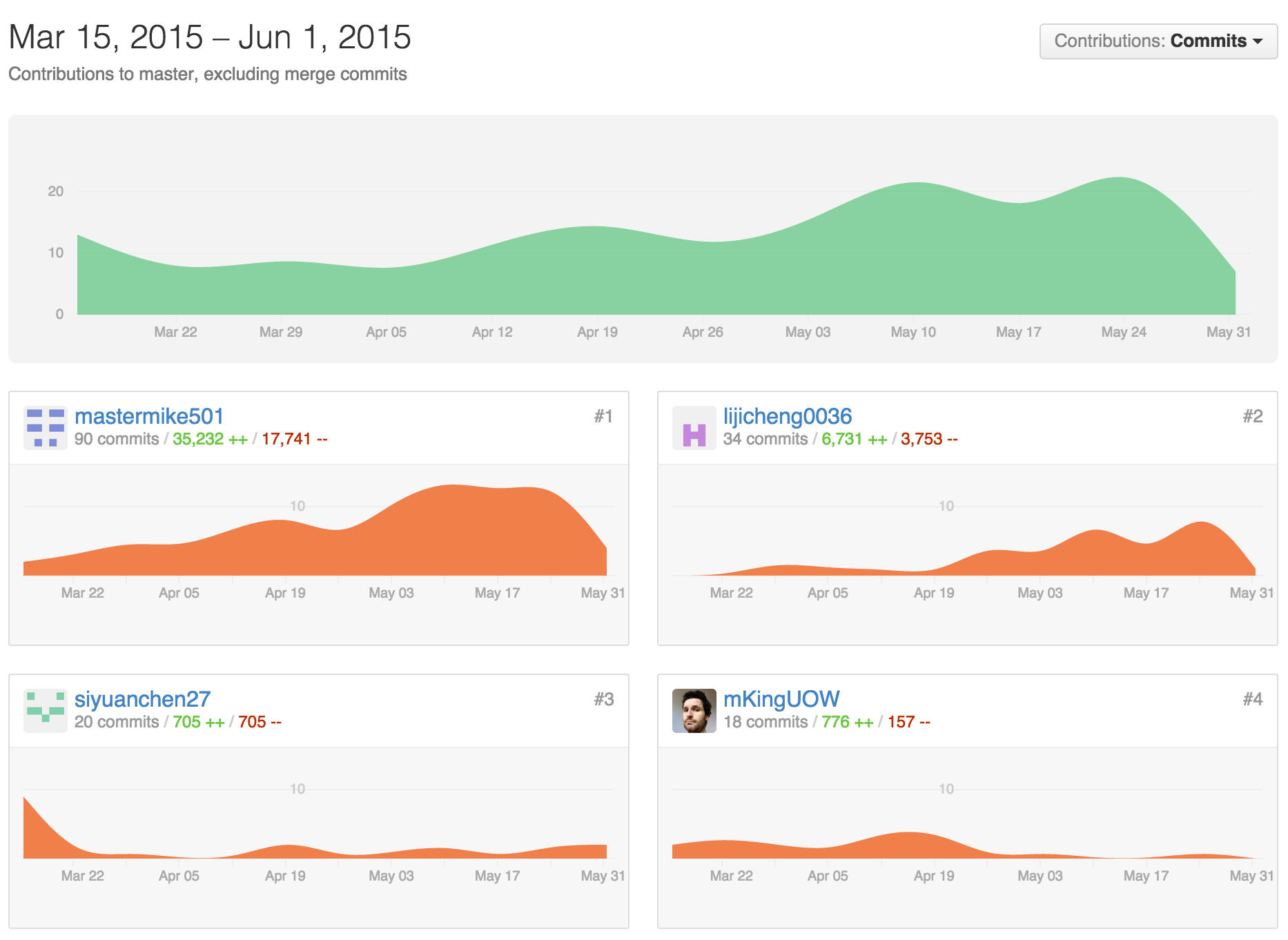
The following table displays member contribution, expressed as a percentage, for the construction phase of the project.

|  |  |  |
| --- | --- | --- |
| Team Member | Contribution | Signature |
| Matthew King | 100% |  |
| Michael Kong | 100% |  |
| Siyuan Chen | 100% |  |
| Jicheng Li | 100% |  |

## Github Commit Logs

This section contains the Github commit log file and graph to demonstrate appropriate collaborative use of a versioning system.

The following screenshot taken from Github shows overall activity as well as individual activity over the course of the project.



The Github commit log file (Git Logfile.txt) can be found on the CD submitted with this report.