

Model Driven Software Engineering

(COEN 6312)

Project Deliverable 3

Submitted to

Dr. Abdelwahab Hamou-Lhadj

Submitted by

Nareshkumar M. Sisodiya 27650817

Arjun Lokhande 27411111

Binu Basil John 27421753

Anant Mathur 27323670

Khushboo Handa 27323794

February 29, 2016

Contents

[List of Figures 3](#_Toc444472269)

[1. Introduction 4](#_Toc444472270)

[2. Class Diagram Description 5](#_Toc444472271)

[2.1 Person 5](#_Toc444472272)

[2.1.1 User 6](#_Toc444472273)

[2.1.2 Administrator 6](#_Toc444472274)

[2.2 Online Air Reservation System 6](#_Toc444472275)

[2.3 Ticket 7](#_Toc444472276)

[2.4 Catalog 7](#_Toc444472277)

[2.5 Flight 8](#_Toc444472278)

[2.6 Payment 8](#_Toc444472279)

[3. OCL Constraints 9](#_Toc444472280)

# List of Figures

**No table of figures entries found.**

# 1. Introduction

For any given software system to function efficiently, it is essential to visualize, design, construct and document it through a standardized procedure. UML or Unified Modeling Language achieves these goals by providing a graphical representation of the software system.

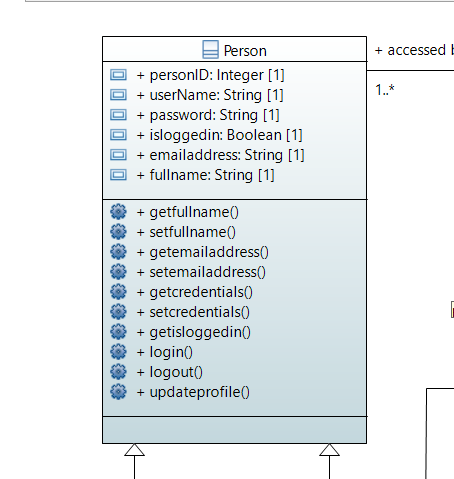
Flyair online reservation system has taken the initial steps of its formation through its standardized procedure in the previous deliverable, by defining the functional requirements of our system using Use Case Diagrams that provide interactions between the use cases and the actors of the system. It prioritises the simplicity of usage for the clients throughout the process of search and reservation of tickets. This is highly reflected in the entire system as well as its construction from its core. The main objective of this deliverable is to explore the detailed class diagram of the system with the classes, attributes and functions associated and list out the constraints that is applied to the same, using OCL.

The Class diagram is created after a close analysis of the system from the point of view of a customer and an administrator by outlining the possibilities and requirements during the entire flow of the system operation. The OCL constraints helped, to go further into the finer details of the system operation listing out the restrictions and conditions that should be implemented to get the smooth and accurate flow of the system. This required the vigorous effort and immense imagination of the entire team. Thus, the result is keeping all the possible situations in the flow of operation within the boundaries of the system design.

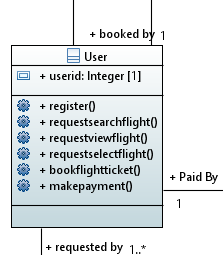
The OCL constraints help to go further into the finer details of the system operation listing out the restrictions and conditions that should be implemented to get the smooth and accurate flow of the system. Thus, the result is keeping all the possible situations in the flow of operation within the boundaries of the system design

# FlyAir (2).png2. Class Diagram Description

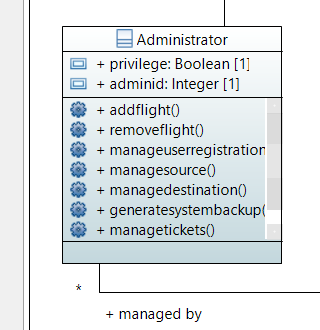
## 2.1 Person



### 2.1.1 User

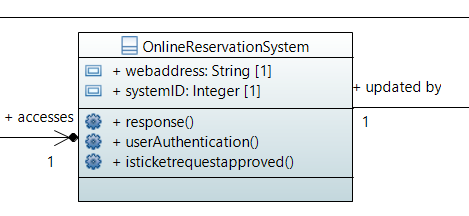


### 2.1.2 Administrator



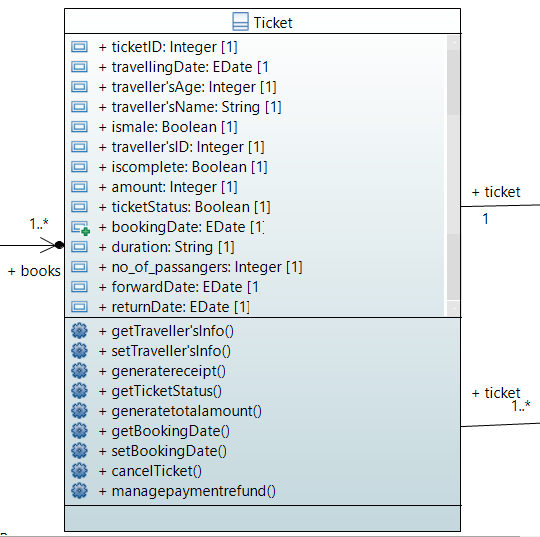
Admin class will define the role of an administrator in the system. This class will inherit the Person class. The primary task of Admin class is perform administrative functions in flight reservation system. Administrator will manage flight details, user registrations and routinely take system back-up. Admin class can manage overall system.

## 2.2 Online Air Reservation System

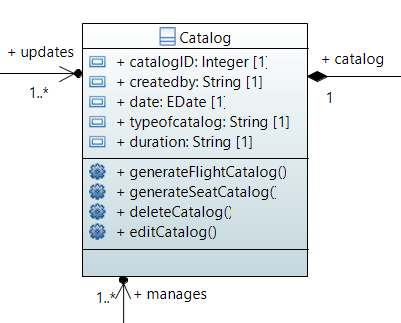


OnlineReservationSystem class plays a vital role in responding to the request initiated by the person. Also User authentication as well as approval of ticket request will be handled by this class.

## 2.3 Ticket

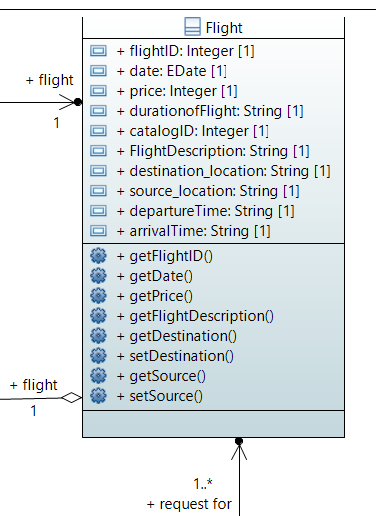


## 2.4 Catalog

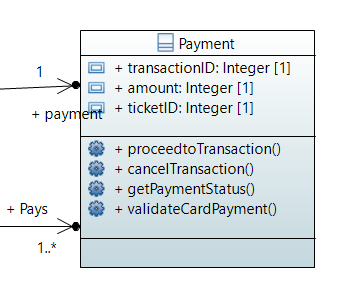


Catalog class holds all required data for flights and tickets. Upon receiving different combinations of requests, it can serve list of flight or list of available seats.

## 2.5 Flight



## 2.6 Payment



# 3. OCL Constraints

* Each person who logs in to the system to book a ticket must have an unique ID :

Context : Person

inv : self.allinstances()->forall(P1,P2:Person | P1 <> P2 implies P1.userID <> P2.userID);

* Each Ticket booked by the user for scheduled travel should have an unique ID :

Context : Ticket

inv : self.allinstances()->forall(T1,T2:Tickets | T1<>T2 implies T1.ticketID <> T2.ticketID);

* A logged in user planning a travel can book at most 10 tickets of the flight :

Context : User

inv : self.book->no\_of\_passengers <= 10;

* Source and Destination location of the flight cannot be the same :

Context : Flight

inv : self.destination\_location <> self.source\_location;

* User must provide Traveller's information while booking ticket :

Context : Ticket

int : self.travellersName->notEmpty() AND self.travellersAge->notEmpty() AND self.travellersID->notEmpty();

* The return journey date must be after the forward journey date :

Context : Ticket

inv : self.forwardDate < self.returnDate;