



CS-319 Object Oriented Software Engineering
Project Final Report
Airplane Reservation System
Fall 2015

Group 6 - Sec 03

Burak Sefa SERT

Sanem ELBASI

Can Mert YILDIZ

Table of Contents

Table of Figures	4
1. Introduction	6
2. Overview	6
3. Requirement Analysis	7
3.1. Functional Requirements	7
3.1.1 Admin	7
3.1.2 Clerk	8
3.2 Non-functional Requirements.....	9
3.3. Constraints	10
3.4 Scenarios	10
3.5 Use Case Models	12
3.6. User Interface - Navigational Paths and Screen Mock-ups.....	18
4. Analysis	37
4.1Object Model.....	37
4.1.1. Domain Lexicon	37
4.1.2. Class Diagram(s)	38
4.2 Dynamic Models.....	40
4.2.1 State Chart.....	40
4.2.1 Sequence Diagram	42
5. System Design	45
5.1 Design Goals	45
5.2 Sub-System Decomposition	46
5.3 Architectural Patterns	49
5.3.1 Client/Server with Repository.....	49
5.3.2. Model-View-Controller Pattern.....	50
5.4 Hardware/Software Mapping	51
5.5 Addressing Key Concerns	52
5.5.1 Persistent Data Management	52
5.5.2 Access Control and Security.....	52
5.5.3 Global Software Controls	53

5.5.4 Boundary Conditions	53
5.5.5 Software Portability	54
6. OBJECT DESIGN	54
6. 1 Design Patterns	54
6.1.1 Façade Pattern	54
6.1.2 Factory Pattern	56
6.1.3 Observer Pattern	56
6.2 Class Interfaces.....	58
6.2.1 ControllerPackage	58
6.2.2 Database Package	60
6.2.3 Model Package	61
6.2.4 View Package	66
6.3 Specifying Contracts using OCL	72
7. Conclusion and Lessons Learned	77

Table of Figures

Figure 1 - Use Case Diagram.....	12
Figure 2 - Navigational Path.....	18
Figure 3 - Login Page.....	19
Figure 4 - Forgot Password.....	19
Figure 5 - User Main Menu.....	20
Figure 6 - New Reservation Menu.....	21
Figure 7 - Available Flight List	22
Figure 8 - Seat Selection.....	23
Figure 9 - Passengers Information for Reservation.....	24
Figure 10 - Delete Reservation	25
Figure 11 - Reservation List of the Current Passenger	26
Figure 12 - Account Setting for Clerk	27
Figure 13 - Administrator Main Menu	28
Figure 14 - Flight Operations Menu.....	28
Figure 15 - Create Flight Menu	29
Figure 16 - Available Plane List.....	30
Figure 17 - Confirmation of the Flight to be created	31
Figure 18 - List Flights depending on specifications.....	31
Figure 19 - Flight List.....	32
Figure 20 - Change Current Flight Details.....	32
Figure 21 - User Operations Menu	33
Figure 22 - Create User Menu.....	33
Figure 23 - User List to Change/Delete Operations	34
Figure 24 - Change Clerk Details	34
Figure 25 - Airports.....	35
Figure 26 - New Airport Menu	35
Figure 27 - Delete Airport.....	36
Figure 28 - Add/Delete Plane Menu	36
Figure 29 - Class Diagram	38
Figure 30 - State Chart diagram for Reservation.....	40
Figure 31 - State Chart diagram for User	41
Figure 32 - Sequence Diagram for Login	42

Figure 33 - Sequence Diagram for Reservation	43
Figure 34 - Sequence Diagram for Adding New Flight	44
Figure 35 - Package Diagram	47
Figure 36 - ARSView Package.....	47
Figure 37 - ARSController Package	48
Figure 38 - ARSModel Package.....	48
Figure 39 - Client/Server with Repository.....	49
Figure 40 - Client/Server	49
Figure 41 - MVC Architectural Pattern.....	50
Figure 42 - Deployment Diagram	51
Figure 43 - Access Matrix.....	53
Figure 44 -Facade Pattern	55
Figure 45 - Factory Pattern Diagram.....	56
Figure 46 - Observer Pattern Diagram	57
Figure 47 - AppManager	58
Figure 48 - Panel Factory.....	59
Figure 49 - DatabaseConnector	60
Figure 50 – Person.....	61
Figure 51 - User	62
Figure 52 - Seat	62
Figure 53 - Flight.....	63
Figure 54 - Airport	64
Figure 55 - Passenger.....	64
Figure 56 - Clerk	64
Figure 57 - Admin.....	65
Figure 58 - Reservation.....	65
Figure 59 - LoginPanel.....	66
Figure 60 - ClerkMenuPanel	66
Figure 61 - AdminMenuPanel.....	67
Figure 62 - ReservationPanel.....	68
Figure 63 - PersonalSettingsPanel.....	69
Figure 64 - DeleteReservationPanel	70
Figure 65 - ListFlightPanel	70
Figure 66 - FlightPanel.....	71

1. Introduction

As our project, we decided to design and implement an airplane reservation system, in which user -on behalf of a customer- can reserve a place from among available places of a plane with a given date, departure and destination location. For an airline company, it is a problem to manage and store the data related to flights such as information of passengers, reservation details, specifications of planes, place and time of destination and departure. Inevitably, all of these data become very dynamic. Depending on the changing needs of both passengers and airline companies, existing reservations can be cancelled by passengers, companies may organize new flights or stop the flights for a particular route with increasing or decreasing demand and they can add new planes to their air fleets. To this respect, airline companies have to handle and save these dynamically changing variables and our implementation offers a simple solution to solve this problem. The interface of our program which is easy to use enables the clerk to make reservation procedures shorten and also reduce the workload of the clerks with high privileges who must have to take care of adding/removing flights, plane plans and departure-destination places.

This report describes the functional, non-functional and pseudo requirements and also contains use-cases, scenarios and UML diagrams. This reports also give an intuition about how the program will work and what features it has.

2. Overview

First of all, to be clearer, let us define our target end user profile. There will be two different type of end-users who differ on their privileges. One of them, “Clerk”, is simply an employee in an office of airline company and his main task is regulating (i.e. making, cancelling and querying) reservations for customers. On the other hand, “Admin” is the other person whose range of authority is greater and his main task is to decide who will have access to the system. List of Admin’s ability is the following:

- Adding and deleting a seat plan of a plane from the system

- Adding and deleting a clerk from the system

- Changing schedule of a plane

Adding and deleting a flight from the system

In order to store all information (reservations, user information etc.), we will be using an online database so that changes will be visible to all users who use application from different computers and have the necessary privileges. Database system will be MySQL database.

3. Requirement Analysis

3.1. Functional Requirements

3.1.1 Admin

Login:

- Admin will be able to login the system using his/her employee id and password.
- If the id or password is entered incorrectly, the system shall be able to give an error message specifying either id or password is incorrect.

Display Information about Plane and Seats:

- After the login, admin will have an option to see existing planes, their times and also allocated seats in them.

Add Plane or Change Schedule:

- Admin will be able to change the schedule by giving an existing plane to another hour in the schedule if it is empty.
- Admin will be able to add/delete destination and departure cities.
- Admin will be able to add a new plane to the system by indicating plane type and the number and plan of the seats in the plane.

Add/Drop Clerk:

- Admin will be able to add new employees to the system with employee name and email address.
- The system will automatically generate an employee id for the new clerk and will display the new id to the admin.
- Admin will be able to delete a clerk.

Adjust Ticket Prices

- If it is necessary, admin will be able to change ticket prices depending on the airline company.

3.1.2 Clerk

Login:

- Clerk, after registered to the system, can login to the system using employee id and password in order to make reservation.
- If the ID or password is entered incorrectly, the system shall be able to give an error message specifying either id or password is incorrect.

Set Account Details:

- Clerk, after registered to the system, able to change his/her account details including mail address, name and password.

Select Departure/Destination Place:

- Clerk will be able to select passenger's destination and departure place of plane.
- System shall be able to show specific place list consisting of departure and destinations places which clerk will be able to select from.
- System will not list the departure places which don't have available flights.

Select Date:

- User will be able to select the departure date of the flight.
- User will be able to select both departure and return date if the round trip option is checked.
- System shall be able to show flight list on the specified date with specified departure and destination places.
- System will not list the dates which don't have available flights.

Select Time:

- User will be able to select a time interval to narrow the options on the flight list of the given date.

Select Seat:

- After specifying time and place, user will see the available seats in that plane.
- User shall be able to select any empty seat.
- After selection, this seat will not be empty anymore.

Show Reservations / Cancel Reservation:

- User will be able to make a reservation with customers' name, surname, social security number and email address.
- User will be able to see any reservation with social security number of the customer.
- User will be able to cancel any reservation on the reservation list.

3.2 Non-functional Requirements

3.2.1 Privacy and High Security

Since it is a reservation system, we will be holding information of too many passengers and almost all of these information is personal like contact information, identity numbers and even their travel details. For this reason, one of our high priorities is protect the privacy with a high security. Thus, we will encrypt data and request when we send them to database server.

3.2.2 Well Designed Interface

We want to make the GUI of the program both appealing and user friendly so that neither user nor admin have difficulty to use the program.

3.2.3 Extendibility

We aim to design the program in such a way that implementing future changes will not result in the redesigning the whole program.

3.2.4 Cost-Efficiency

It is also one of our concern that making the program cost effective. Our purpose is keeping the cost of the program as low as possible.

3.3. Constraints

- The program will be implemented in java
- Language of the system will be English
- MySQL Database will be used to store data
- The program will be a desktop application

3.4 Scenarios

Scenario #1: Making Reservation

1. Customer, John, goes to the ticket office in order to make a reservation.
2. Office clerk, Mary, asks John date, destination and departure point.
3. John tells date, destination and departure point.
4. Mary, searches for available trips that meet John criterions, informs him about available schedules.
5. John opts for a departure time.
6. According to John's selection, Mary shows John available seats.
7. John selects a seat from amongst available seats.

Scenario #2: Cancelling Reservation

1. Customer, John, goes to ticket office in order to change his reservation.
2. Office clerk, Mary, asks John date, destination and departure point and seat number of the reservation that he wants to change.
3. John tells her date, destination point and departure point.
4. Mary cancels the current reservation of John by using her user privileges.

Scenario #3: Deleting/Adding User to the System

1. The old office clerk, Mary, quits the job.
2. The chief, also the admin, Jasmine deletes Mary from the system by specifying her employee number in order to prevent any unwanted situation.
3. The new office clerk, Anna, replaces Mary's position.
4. The admin adds Anna to the users by entering her employee number and password that Anna decides to the system.
5. Now Anna is capable of making reservation.

Scenario #4: Deleting Plane from the System

1. One of the plane of the airline company becomes unusable due to a malfunction of the radar system of the plane.
2. Company decides to remove this plane from the fleet.
3. The company director informs admin about the change.
4. Admin deletes the plane from the system.

Scenario #5: Adding Plane to the System

1. In order to compensate the plane that is removed from the fleet, the company buys a new plane.
2. After the testing process, plane is ready for the work.
3. Company director informs admin about the change.
4. Admin adds the new plane to the system.

Use Case Diagram:

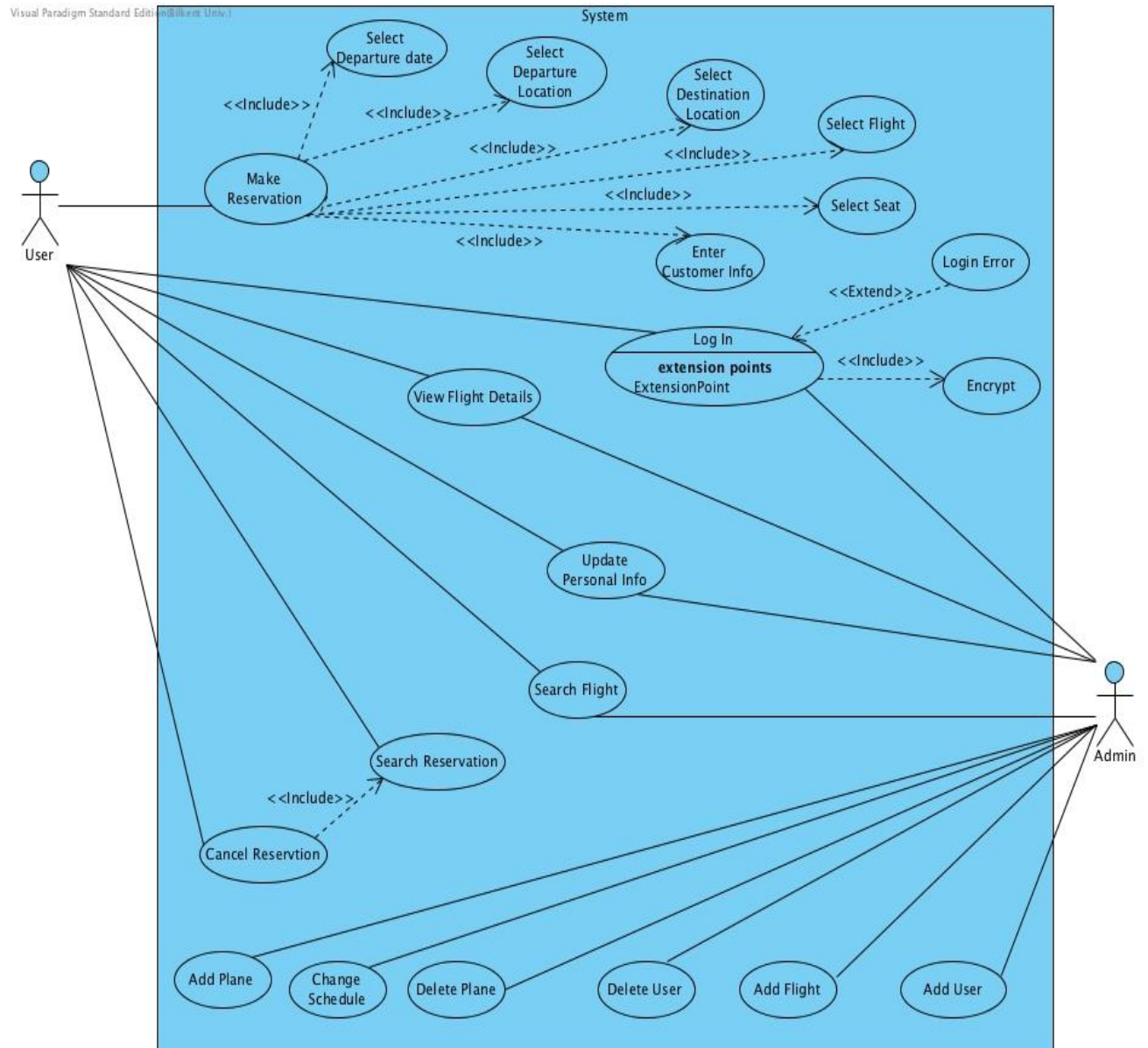


Figure 1 - Use Case Diagram

Use Case #1

Use Case Name: Login

Actor: User (Admin or Clerk both are eligible)

Post-Condition: User logged in to the system.

Main Flow of Events:

- 1) User enters his/her employee id and password on text fields
- 2) User hits the login button
- 3) User logs in successfully if his/her information is correct.

Alternative Flow of Events:

The program throws a login error message.

Use Case #2

Use Case Name: Make Reservation

Actor: Clerk

Pre-Condition: Clerk logged in to the system.

Post-Condition: Clerk has made reservation OR

Clerk has been informed about why he is unable to make reservation

Main Flow of Events:

- 1) Clerk selects time, date, departure and destination points according to customer's wish.
- 2) The program displays flights
- 3) Clerk chooses a flight
- 4) The program displays the seats which are suitable.
- 5) Clerk makes a selection from among those seats.
- 6) Clerk enters the passenger's name, email address, phone number.
- 7) Clerk presses the make reservation button and get the confirmation.

Alternative Flow of Events:

Customer is informed that there is no available seats for given time, date, departure and destination points. As a result either

- Customer stops interaction OR,
- Customer requests reservation for different time.

Use Case #3

Use Case Name: Cancel Reservation

Actor: Clerk

Pre-condition: Clerk has logged in to the system AND

The reservation, that is going to be cancelled, exists.

Post-condition: Clerk cancels reservation OR

The program informs user about why he cannot cancel the reservation.

Main Flow of Events:

- 1) Clerk enters passenger's phone number.
- 2) The program displays the reservations made by the passenger.
- 3) Clerk selects the reservation that passenger wants to cancel.
- 4) Clerk clicks on the cancel button.
- 5) Clerk gets the success notification.

Alternative Flow of Events:

If passenger knows the reservation number;

- 1) Clerk enters the reservation number.
- 2) The program displays the reservation.
- 3) Clerk clicks the cancel button.
- 4) Clerk gets the success notification.

If the cancellation request is made 24+ hours before the departure time of plane OR,
The system returns error message that it is too late to make that transaction.

Use Case #4

Use Case Name: Search Flights

Actor: Clerk

Pre-condition: Clerk logged in to the system

Post-condition: Clerk gets the information

Main Flow of Events:

- 1) Clerk selects time, date, departure and destination points according to customer's wish.
- 2) The program displays flights.

Use Case #5

Use Case Name: Search Flights

Actor: Admin

Pre-condition: Admin logged in to the system

Post-condition: Admin gets the information

Main Flow of Events:

- 1) Admin enters the flight Id to the search bar.
- 2) The program displays the flight.

Alternative Flow of Events:

- 1) Admin enters date, departure and destination point.
- 2) The program displays the flight.

If the flight does not exist the program will display a no_flight_found error message.

Use Case #6

Use Case Name: View Flight Details

Actor: Clerk

Pre-condition: Clerk logged in to the system AND
Clerk made a flight search.

Post-condition: Clerk gets the information

Main Flow of Events:

- 1) Clerk selects a flight from the list.
- 2) The program display the flight information with seating plan and available seats.

Use Case #7

Use Case Name: View Flight Details

Actor: Admin

Pre-condition: Admin logged in to the system AND admin made a flight search.

Post-condition: Admin gets the information

Main Flow of Events:

- 1) Admin selects a flight.
- 2) The program displays the flight information with seating plan and available seats.

Use Case #8

Use Case Name: Create User

Actor: Admin

Pre-Condition: Admin logged in to the system.

Post-Condition: Admin adds a new user.

Main Flow of Events:

- 1) Admin clicks to add a new user, under User Menu.
- 2) Admin enters username, surname and email.
- 3) Admin clicks create user button.
- 4) Program creates the user and displays the generated user ID of the user.

Alternative Flow of Events:

The program displays an error.

Use Case #9

Use Case Name: Delete User

Actor: Admin

Pre-Condition: Admin logged in to the system.

Post-Condition: Admin deletes a specific user.

Main Flow of Events:

- 1) Admin clicks to delete a user, under User Menu.
- 2) Admin enters employee id of the new user.
- 3) Program displays the user information.
- 4) Admin clicks to delete user button.
- 5) Program displays a success message.

Alternative Flow of Events:

Program displays a no_user_found error OR displays a delete error message.

Use Case #10

Use Case Name: Change Schedule

Actor: Admin

Pre-Condition: Admin logged in to the system.

Post-Condition: Admin changes schedule of the plane

Main Flow of Events:

- 1) Admin clicks to ChangeFlight button, located under Flight Menu.
- 2) Admin searches flight that he wants to change.
- 3) Program displays the flight information.
- 4) Admin makes the changes.
- 5) Admin clicks save.
- 6) Program sends a notification of the changes to the passengers who has a reservation on that flight.
- 7) Program displays a success message.

Alternative Flow of Events:

Admin can make a flight search with flight's date, departure and destination point.

Use Case #11

Use Case Name: Update Personal Info

Actor: User (Admin or Clerk both are eligible)

Pre-Condition: User logged in to the system.

Post-Condition: User changes his/her personal information such as e-mail password etc.

Main Flow of Events:

- 1) User goes to account settings menu.
- 2) The program displays the user information.
- 3) User changes personal information that he wants to change.
- 4) User confirms and saves changes.
- 5) The program shows success message.

Alternative Flow of Events:

The program will display an error message.

3.6. User Interface - Navigational Paths and Screen Mock-ups

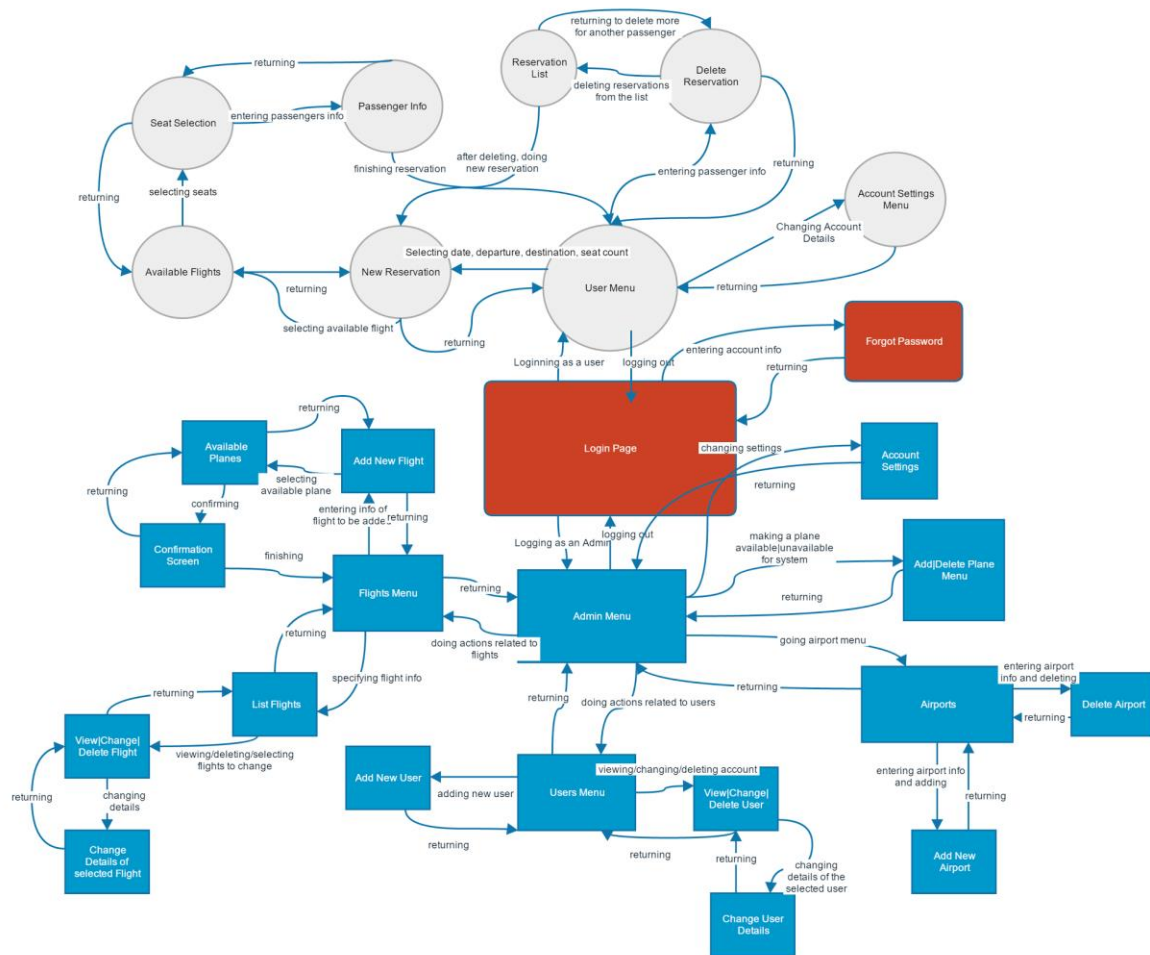
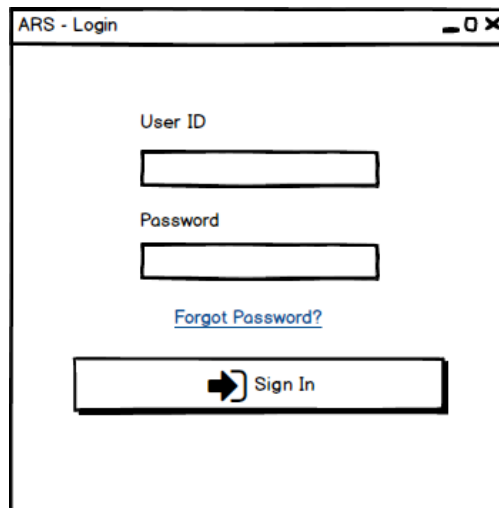


Figure 2 - Navigational Path

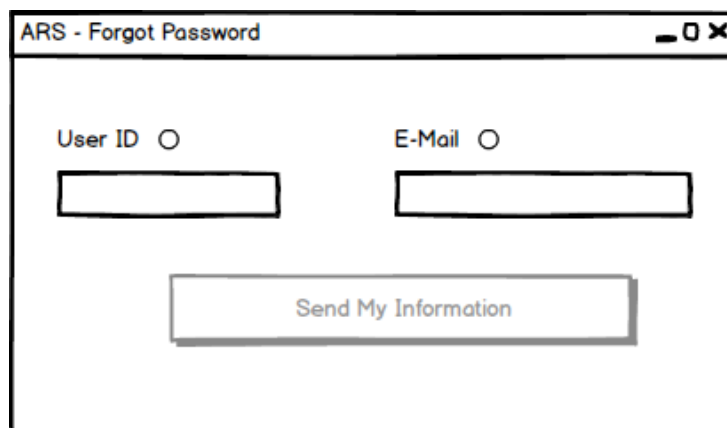
3.6.2 Screen Mock-ups



A screenshot of a web browser window titled "ARS - Login". The window contains a login form with two input fields: "User ID" and "Password". Below the "Password" field is a blue hyperlink labeled "Forgot Password?". At the bottom of the form is a "Sign In" button with a right-pointing arrow icon.

Figure 3 - Login Page

Login page is the first screen that will be shown. User must enter his/her ID and password which obtained from administrator in order to sign in. In case of entering wrong information, a warning window will appear.



A screenshot of a web browser window titled "ARS - Forgot Password". The window contains two input fields: "User ID" and "E-Mail", each with a small circle icon to its right. Below these fields is a "Send My Information" button.

Figure 4 - Forgot Password

Forgot Password: For the users who forgot his/her ID or password, it is possible to get missing information by entering ID or E-mail. The system sends the missing information to mail address of the user.

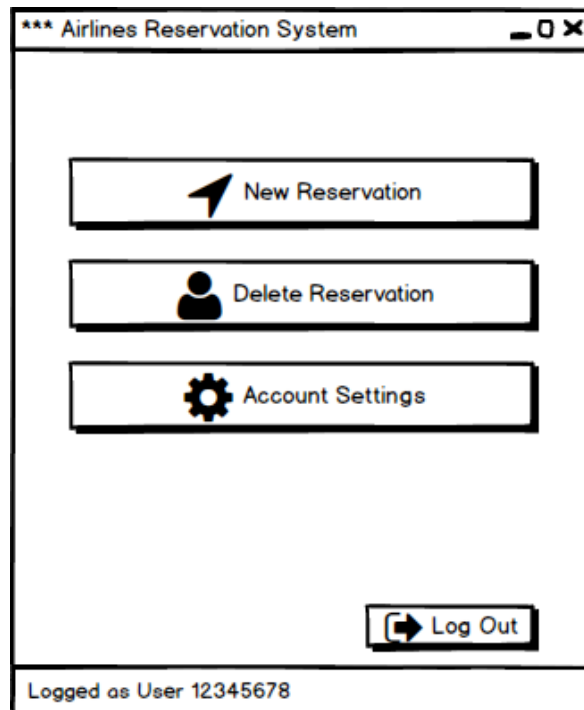


Figure 5 - User Main Menu

User Main Menu: After logging in, both users and admins have a simple main menu. At the left bottom corner of the all windows, it is possible to see the ID and login type. Possible operations of the user is grouped under these 3 buttons and after every 3 button, different operations start systematically branching. The user have to perform all operations related to passenger and these details keep the reservation system simple and fast.

*** Airlines Reservation System - New Reservation

From
Istanbul(Ataturk Airport)
Istanbul(Sabiha Gokcen Airport)
Ankara
Berlin
New York
London

To
Istanbul(Ataturk Airport)
Istanbul(Sabiha Gokcen Airport)
Ankara
Berlin
New York
London

Seat
1
2
3
4
5
6
7

Today

List Available Flights

Return

Logged as User 12345678

Figure 6 - New Reservation Menu

New Reservation Menu: User enters the information related to reservation depending on the expectations of the passenger. New reservation menu lists available airports and seat counts with combo boxes. This menu also has a calendar interface to select the date for flight. After main menu, all of the windows have a return button. It enables user to return previous menu.

*** Airlines Reservation System - Available Flights

Departure	Arrival	Duration	Flight	Total Cost	
11:40	13:30	2h50	TK1725	862TRY	<input type="radio"/>
16:00	17:55	2h55	TK1723	1028TRY	<input checked="" type="radio"/>
19:30	21:30	3h00	TK1727	2018TRY	<input type="radio"/>

> Next

← Return

Logged as User 12345678

Figure 7 - Available Flight List

Available flights: After specifying information of the flight, available flights appear in this windows. We assume that from this point customer (passenger) is able to see the screen of the user to select a proper flight. User is able to select flight with radio buttons. After selection, status of the next button becomes active and status of the radio buttons of the other flight options become inactive.

*** Airlines Reservation System - Passenger Information

Passenger

TC Identity Number

Name*

Surname*

Citizenship*

Phone Number*

Date of Birth*

/

/

E-Mail*

Passenger

TC Identity Number

Name*

Surname*

Citizenship*

Phone Number*

Date of Birth*

/

/

E-Mail*

✓ Finish

← Return


Logged as User 12345678


Figure 9 - Passengers Information for Reservation


Passengers Information: Depending on the passenger count, input text fields with enough number appear. The fields with “*” character must be filled by the user. Otherwise, the finish button does not become active. “TC identity number” is an optional field for Turkish Republic citizens. The name, surname, phone number and date of birth data stored by the application make the passengers unique. In case of any wrong input, system shows an error pop-up.

*** Airlines Reservation System - Delete Reservation

Passenger

TC Identity Number	Name*	Surname*
<input type="text"/>	<input type="text"/>	<input type="text"/>
Phone Number*	Date of Birth*	
<input type="text"/>	<input type="text" value="/ /"/> 	

 Get Reservations

 Return


Logged as User 12345678


Figure 10 - Delete Reservation


Delete Reservation: User is able to delete reservation(s) of the passenger by entering the unique information of the passenger.

*** Airlines Reservation System - Reservation List

Flight	Date	Departure	
TK1725	31.10.2015	11:40	<input type="radio"/>
TK1728	02.01.2016	12:40	<input type="radio"/>
TK1925	31.10.2014	23:30	<input type="radio"/>

 Delete Selected Reservations

 New Reservation

 Return

Logged as User 12345678

Figure 11 - Reservation List of the Current Passenger

Reservation List: Depending on the request of the passenger, user selects the flights which will be deleted by clicking radio buttons. After at least one selection, delete button becomes active. Instead of creating “change reservation” menu, we combined “change” operation with “delete” operation. After delete operation, user has a direct access to new reservation menu.

*** Airlines Reservation System - Change Settings

Employee

Name: John Barker
ID: 12345678
E-mail: 12345678@****.com

Change

Enter Password to Change Settings

New Password

Re-type

Change E-mail

← Return

Save Changes

Logged as User 12345678

Figure 12 - Account Setting for Clerk

Account Settings for Clerk: The user is able to change her/his password or e-mail. Since the IDs are generated from the system, it is not possible to change it. Both password change and mail change, the system asks user for his/her password considering security and privacy issue.

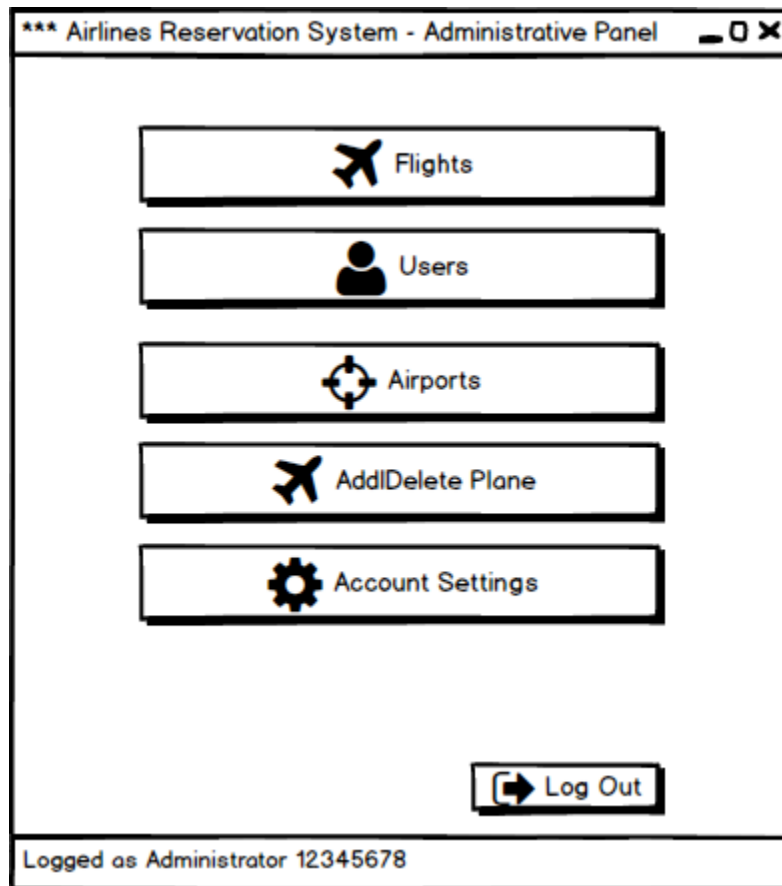


Figure 13 - Administrator Main Menu

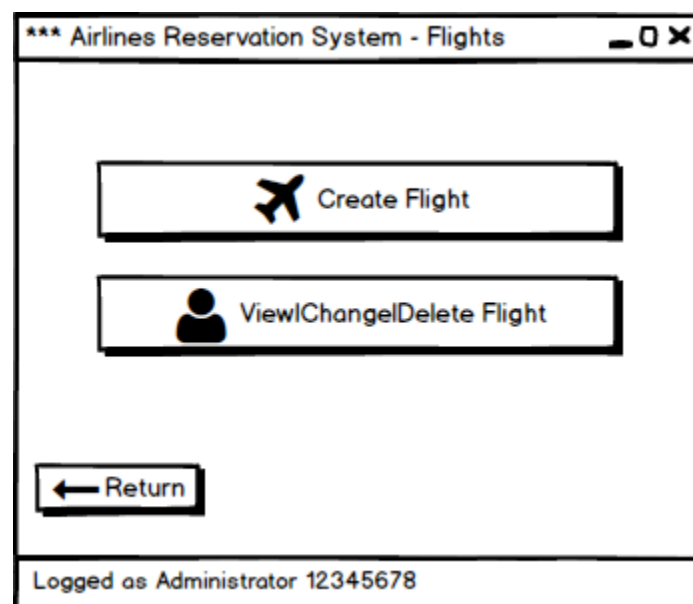


Figure 14 - Flight Operations Menu

*** Airlines Reservation System - Create Flight

From

- Istanbul(Ataturk Airport)
- Istanbul(Sabiha Gokcen Airport)
- Ankara
- Berlin
- New York
- London

To

- Istanbul(Ataturk Airport)
- Istanbul(Sabiha Gokcen Airport)
- Ankara
- Berlin
- New York
- London

Date of Departure

Today

Time of Departure

00:00

List Available Planes

Return

Logged as Admin 1234567

Figure 15 - Create Flight Menu

Create Flight Menu: Administrator is able to create new flights depending on the requests of the airline company. S/he enters the necessary information related to flight.

*** Airlines Reservation System - Available Planes

PlaneID	Plane Model	
1	AIRBUS - A310	<input type="radio"/>
2	AIRBUS - A457	<input type="radio"/>
3	AIRBUS - A546	<input type="radio"/>
4	AIRBUS - A985	<input type="radio"/>
5	AIRBUS - X356	<input type="radio"/>

Create Flight

Return

Logged as Administrator 123

Figure 16 - Available Plane List

Available plane list: After specifying flight information, available plane list appears. Administrator is able to select one of them by clicking radio button. After this, other radio buttons become inactive and create flight button becomes active.

*** Airlines Reservation System - Confirm

Flight Information

Flight	From	To	PlaneID	PlaneModel
TK5467	Istanbul(Ataturk Airport)	Ankara	4	AIRBUS-A985

Confirm

Return

Logged as Admin 12345678

Figure 17 - Confirmation of the Flight to be created

Confirmation: Since creating flight is a serious operation, system shows again the details of the flight together which will be created and waits for confirmation from administrator. After confirmation the application returns main administrator menu.

*** Airlines Reservation System - List Flights

Consider

☐ From: Today To: / /

☐ Starting From: 00:00 To: 00:00

☐ DepartureAirport: Destination:

List Flights

Return

Logged as Admin 12345678

Figure 18 - List Flights depending on specifications

List Flights: Before the operations related to existing flights, the administrator specifies the flights which will be listed. S/he is able to add search options by clicking the radio buttons.

*** Airlines Reservation System - Flights

Flight	AvailableSeats	From	To	Departure	Arrival	PlaneID	Plane Model	ChangeDetails
TK1000	10	Istanbul(Ataturk Airport)	Ankara	11:30	12:30	3	AIRBUS-A380	<input type="radio"/>
TK1010	100	Ankara	Istanbul(Sabiha Gokcen Airport)	11:30	12:30	4	AIRBUS-X760	<input type="radio"/>
TK1050	56	Istanbul(Ataturk Airport)	Berlin(Tegel Airport)	19:30	22:30	5	AIRBUS-X543	<input type="radio"/>
TK2000	0	Istanbul(Ataturk Airport)	Barcelona	11:00	14:00	6	AIRBUS-A380	<input type="radio"/>

Delete Selected Flights

← Return

Logged as Admin 12345678

Figure 19 - Flight List

Flight List: The administrator is able to delete selected flights. Radio buttons let the administrator select more than one flight. After at least selection, delete button becomes active. It is also possible changing details of a flight by clicking changing details button.

*** Airlines Reservation System - Change Current Flight Details

Flight	AvailableSeats	From	To	Departure	Arrival	PlaneID	Plane Model
TK1000	10	<input type="text"/>	<input type="text"/>	<input type="text" value="00:00"/>	<input type="text" value="00:00"/>	3	AIRBUS-A380

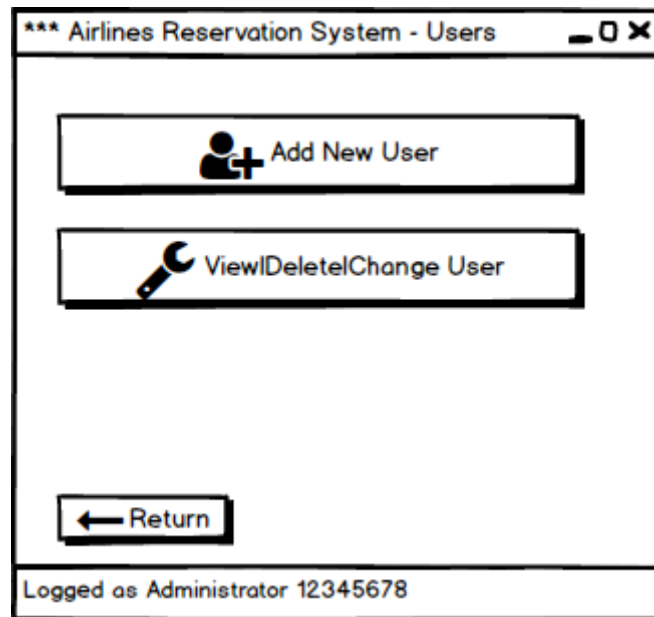
Confirm Changes

← Return

Logged as Admin 12345678

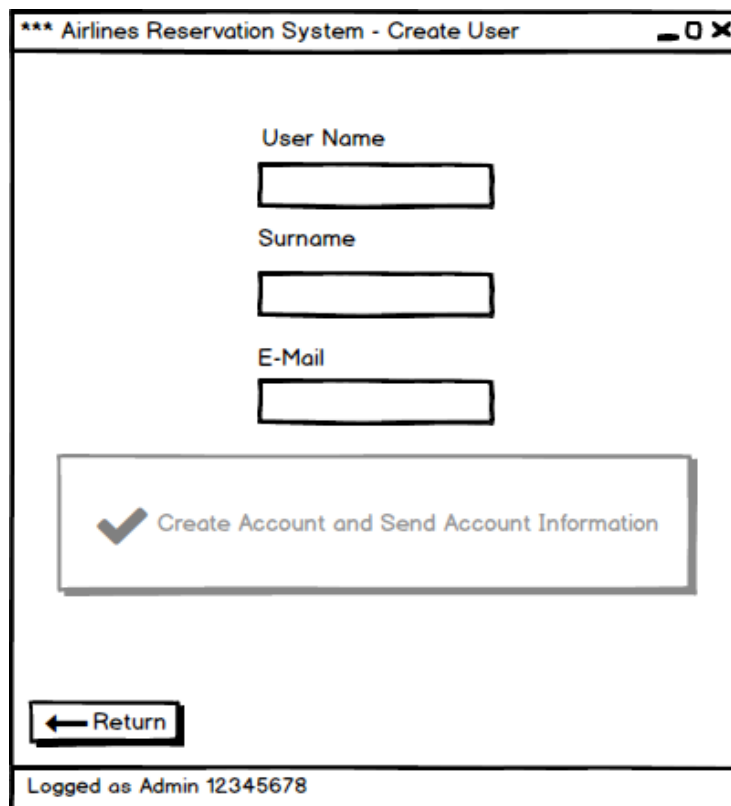
Figure 20 - Change Current Flight Details

Change Current Flight Details: The administrator is able to change destination and departure airports and deadlines. This operations results with cancelling all reservations of the current flight.



A screenshot of a software window titled "Airlines Reservation System - Users". The window contains three buttons: "Add New User" with a person icon, "View/Delete/Change User" with a wrench icon, and "Return" with a left arrow icon. At the bottom, a status bar reads "Logged as Administrator 12345678".

Figure 21 - User Operations Menu







A screenshot of a software window titled "Airlines Reservation System - Create User". The window contains three text input fields labeled "User Name", "Surname", and "E-Mail". Below these fields is a button labeled "Create Account and Send Account Information" with a checkmark icon. At the bottom left is a "Return" button with a left arrow icon. The status bar at the bottom reads "Logged as Admin 12345678".


Figure 22 - Create User Menu

Create User Menu: Only the administrator is able to add a new user. After filling the text fields and pressing create button, the user receive his/her account information.

*** Airlines Reservation System - User List

ID	Name	Surname	E-Mail	ChangeDetails
1	John	Carpenter	jcarpenter@gmail.com	 Change Details <input type="radio"/>
2	Asli	Taner	aTaner@gmail.com	 Change Details <input type="radio"/>
3	Ahmet	Yilmaz	aYilmaz@gmail.com	 Change Details <input type="radio"/>

 Delete Selected Users

 Return


Logged as Admin 12345678


Figure 23 - User List to Change/Delete Operations

User List: Only administrator can delete the users. He or she is able to select more than one user by using radio buttons.

*** Airlines Reservation System - Change User Details

ID	Name	Surname	E-Mail
2	<input type="text"/>	<input type="text"/>	<input type="text"/>

 Confirm Changes

 Return

Logged as Admin 12345678

Figure 24 - Change Clerk Details

Change Clerk Details: Administrator is able to change information of the clerks by filling related text fields.

*** Airlines Reservation System - Airports

Add New Airport

Delete an Airport

Return

Logged as Administrator 12345678

Figure 25 - Airports

*** Airlines Reservation System - New A

AirportName

City

Country

Add

Return

Logged as Admin 12345678

Figure 26 - New Airport Menu

New Airport: The administrator is able to add new airports to the system depending on the requests of the airline company.

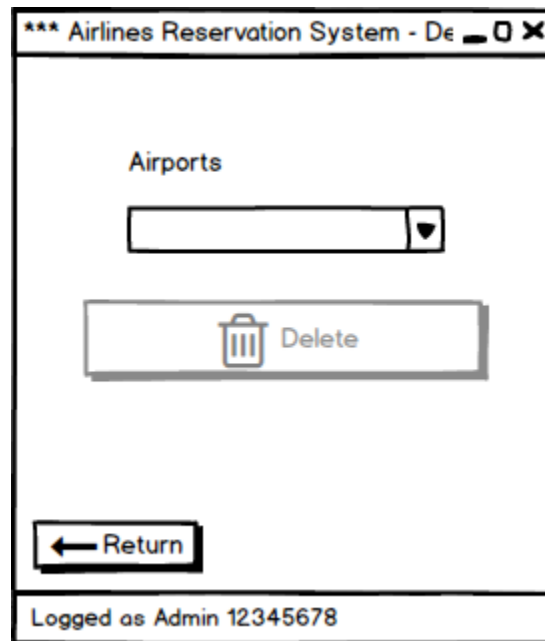


Figure 27 - Delete Airport

Delete Airport: The administrator is able to delete an airport depending on the requests of the airline company. S/he selects an airport from the list in the combo box and presses delete button.

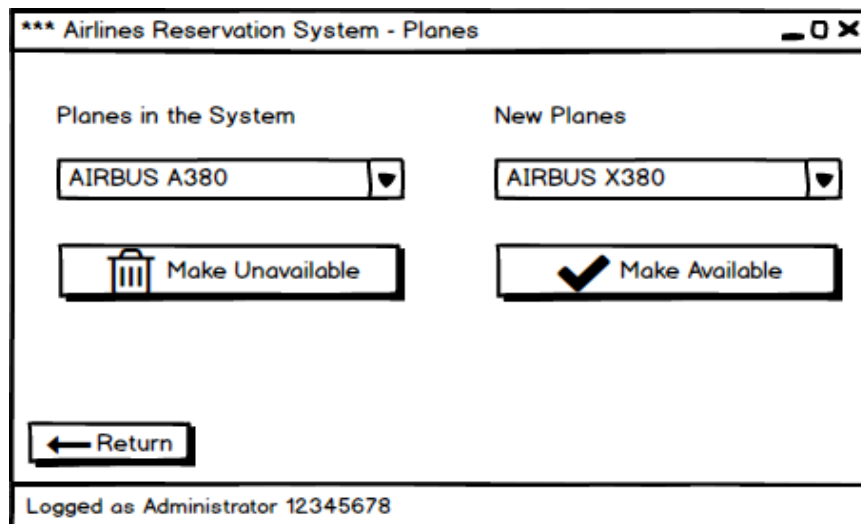


Figure 28 - Add/Delete Plane Menu

Add/Delete Plane: The administrator is able to make planes available or unavailable for flights. The system has a default list of planes and after this operations, planes and seat plans become available when administrator creates a flight.

4. Analysis

4.1 Object Model

4.1.1. Domain Lexicon

Admin: An employee of the airline company who has privileged rights. Admin will use the application to organize the flight schedule.

User: An employee of the airline company who does makes the reservations of the passengers.

Passenger: A customer of the airline company who is either about to make a reservation or has a reservation.

ID: An ID is the unique identifier of the airline company' employees.

Password: Combination of numbers and characters that is necessary to be able to login the system. Each ID has a password.

Destination: It is the name of the airport that the flight will arrive.

Departure: It is the name of the airport that the flight will depart

Reservation: Withholding seats.

4.1.2. Class Diagram(s)

Visual Paradigm Standard Edition@ikent Univ.)

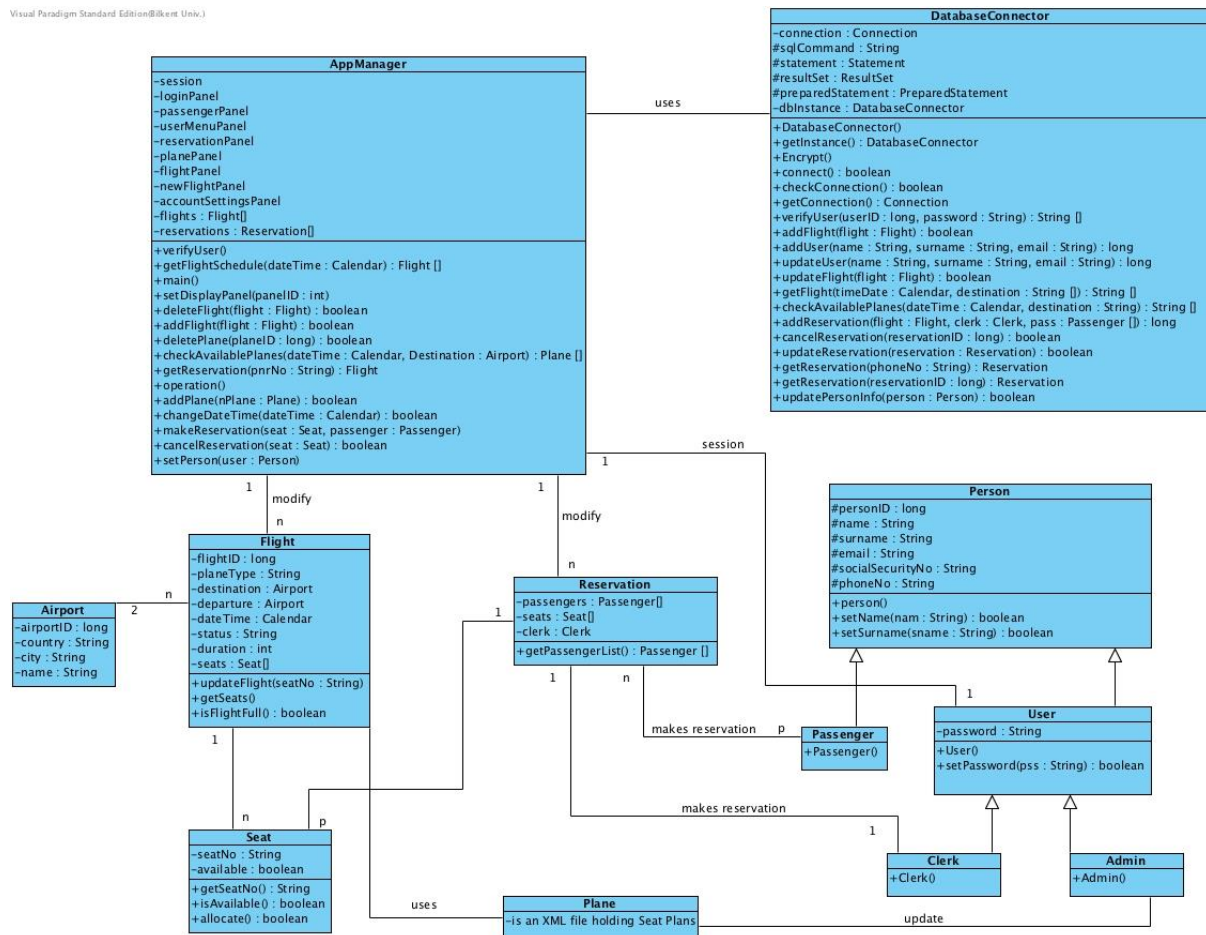


Figure 29 - Class Diagram

To make the class diagram clear, GUI classes and panels are not included. Brief description of classes and their most important methods in a nutshell is given below:

AppManager: This class has access to DatabaseConnector and gets necessary information from its operations. This class also has entity classes and initializes their objects. It holds the session of the user. Most importantly, this class has the main method and has the app logic.

Flight: This is a model class that holds the information of a flight such as departure and a destination airports, date, time and seats. It also has a unique identifier flightID.

Airport: This is a model class that holds the information of an airport such as country, city and name. It also has a unique identifier airportID.

Plane: This is external xml file that holds the seat plans of different types of planes.

Seat: This is a model class that holds the information of a seat such as the availability of the seat and the passenger on the seat if the seat is reserved and the flight that seat belongs.

Reservation: This is a model class that holds the information of a reservation such as clerk that made the reservation, the passenger information, seat information and a unique reservation ID.

DatabaseConnector: This class holds the connection to the database and is responsible from every query to the database. It's most important methods are getFlights which will return all of the flights on a specific date and to a specific destination from specific departure airport and getAvailablePlanes which will return the available planes that can be assigned to a new flight at a specific date from specific departure airport.

Person: This is a super class that holds the information about a person such as name, surname, email and social security number.

User: This is a model class who is child of the Person class. User class holds the information about a user such as the ID and the password of the user.

Clerk: This is a model class who is child of the User class. This class is defined to separate the access rights of the Clerk's and Admin's.

Admin: This is a model class who is child of the User class. This class is defined to separate the access rights of the Clerk's and Admin's.

Passenger: This is a model class who is child of the Person class.

4.2 Dynamic Models

4.2.1 State Chart

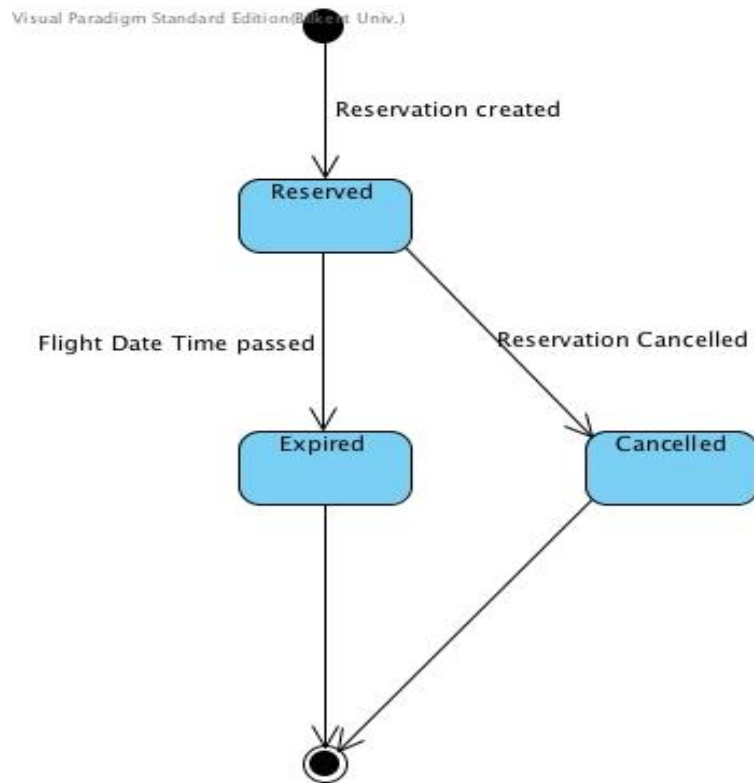


Figure 30 - State Chart diagram for Reservation

When a reservation is created initially it has the reserved state. If the flight's date and time is passed then the reservation has the expired state. Before reservation gets expired it can be cancelled then it briefly has the cancelled state.

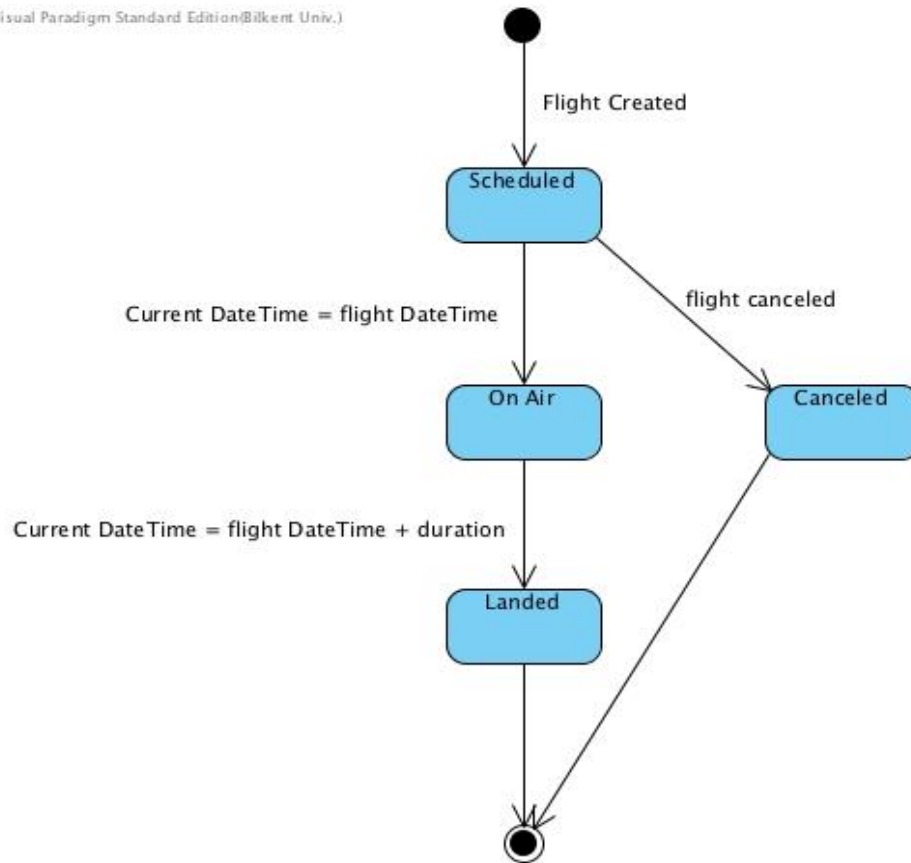


Figure 31 - State Chart diagram for User

When a flight is created it initially has the scheduled state. When the flight's time and date comes the flight has the state on air, until the specified duration of the flight is ended. Then, it has the landed state. A flight can be canceled before its departure time and date, in this case flight has cancelled state.

4.2.1 Sequence Diagram

Login to the System:

UML Paradigm Standard Edition(Bilkent Univ.)

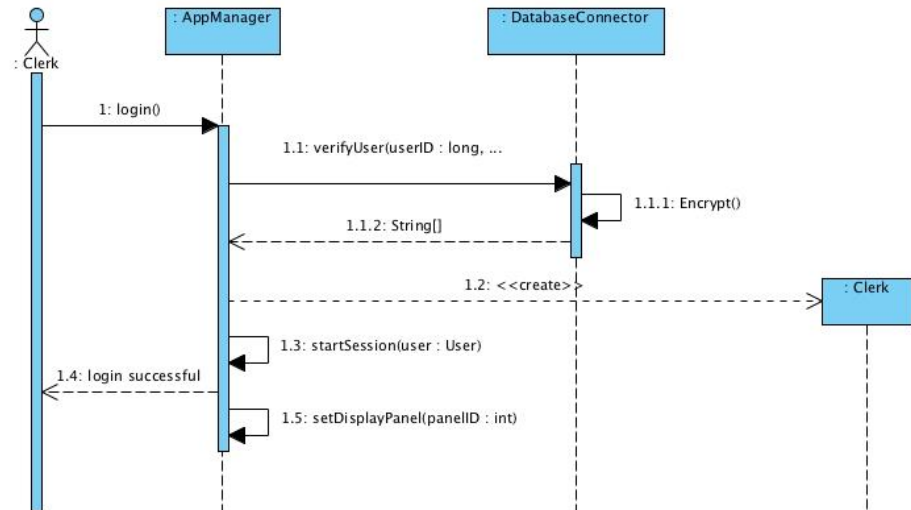


Figure 32 - Sequence Diagram for Login

This sequence diagram explains when a clerk or admin logs in to the program. User enters his/her userID and password, when user clicks the login button, AppManager decides whether the user exists by using DatabaseConnector to check, the verify method returns the information about the user object with its type. Then AppManager creates the Clerk object and starts a session. Then notifies the user that the login was successful and displays the main Clerk screen.

Reserving Seat:

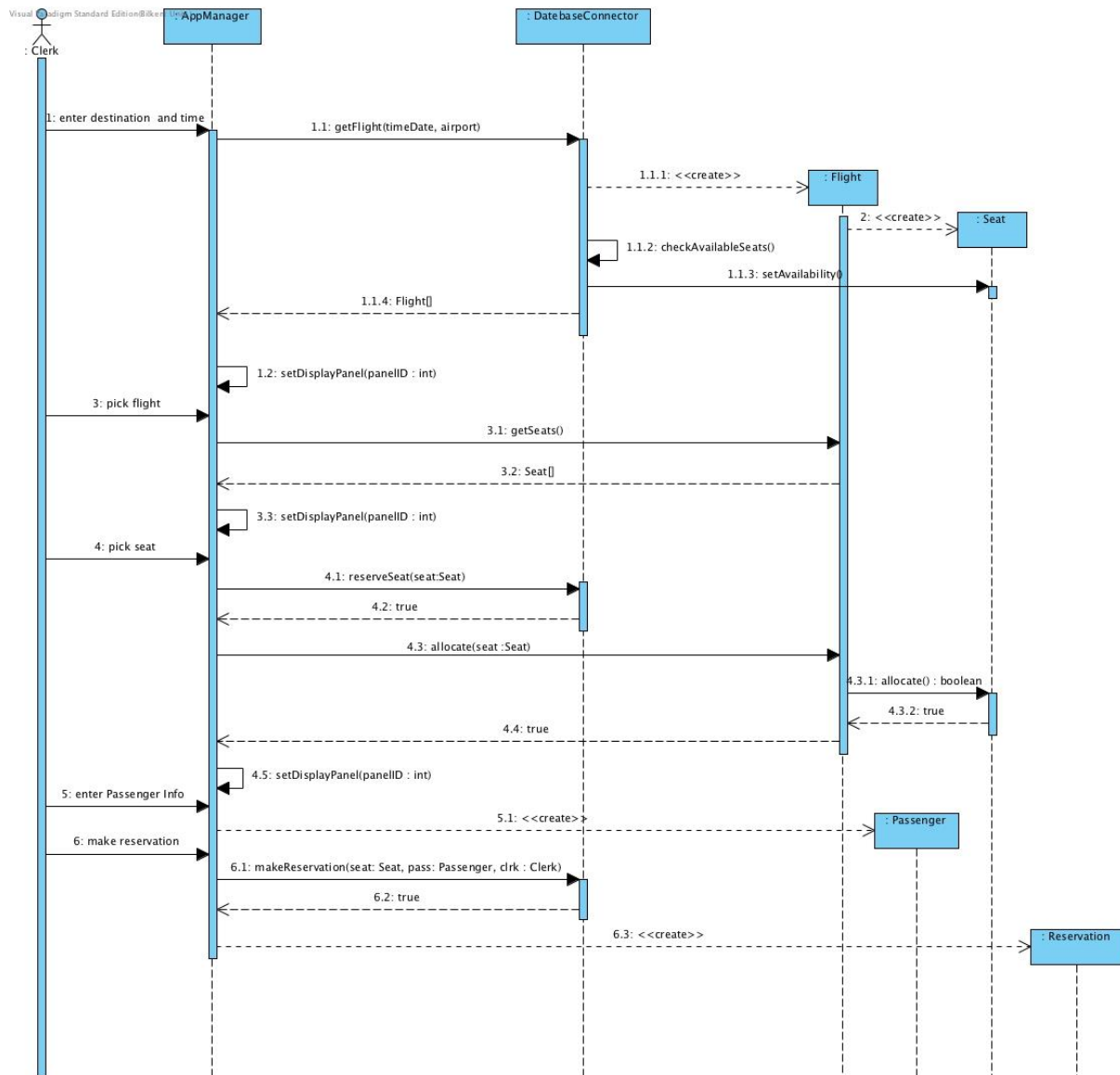


Figure 33 - Sequence Diagram for Reservation

Clerk enters the date, departure airport and destination airport, then clicks next. AppManager uses DatabaseConnector to get available Flights. DatabaseConnector Return availability Flights with seats availability checked. AppManager displays the available flights.

Clerk chooses one of the Flights. AppManagers gets the seats of the flight and displays the seats to the Clerk.

Clerk chooses a seat, AppManager reserves the seat first at the database so that another clerk don't see it available. Then AppManager changes the state of the seat object to allocate. AppManager directs Clerk to passenger information panel.

Clerk enters the name, email, and phone number of the passenger. AppManager creates the Passenger object.

Clerk clicks to make a reservation AppManager updates the Database and creates the reservation object and notifies Clerk that reservation has created successfully and displays the reservationID to the Clerk.

Adding New Flight:

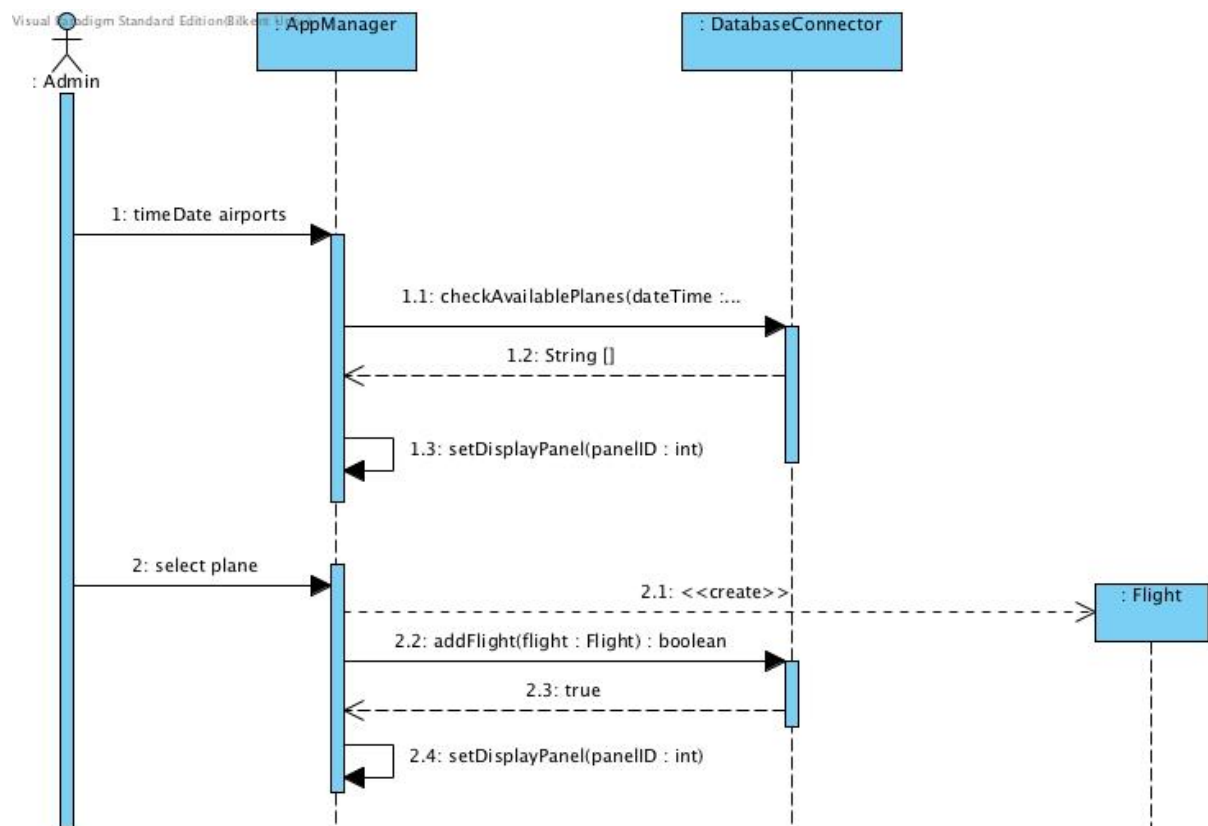


Figure 34 - Sequence Diagram for Adding New Flight

Admin enters date, departure airport and destination airport. AppManager checks the database by using DatabaseConnector to get the available planes. Then, AppManager displays the plane list to the Admin.

Admin chooses one of the planes from the list. AppManager creates the Flight. Then adds the flight to the database so that Clerks can see the flights. The AppManager notifies the Admin that flight has been created successfully.

5. System Design

5.1 Design Goals

In the design part of the project, one of the most important parts is determining design goals, since design goals help us determine the most important features which are related to non-functional requirements of the project. As developers of Airline Reservation Systems, we decided to give priority to the following design goals:

User friendliness: Our target audience are airline companies and they must satisfy their passengers. A customer who will make a reservation for a flight surely wants to finish all of the procedures as quickly as possible therefore our one of the aims is to make the interface simple and easy to understand. We will also reduce the number of steps for a reservation by reducing interaction between clerk and software with less buttons, inputs and menus.

Reliability: An airline company cannot have a chance or excuse for loss of details related to the reservations. Since they must provide service for their customers 24 hours nonstop without data loss, one of the main goals is to prevent unexpected crashes and also backup information.

Modifiability: We will use MVC pattern in our project so that it will be easy to add new features or change existing implementation.

Robustness: We will also consider the unexpected or unwanted inputs and scenarios. The program will prevent these situations by giving proper warning messages for wrong outputs. Our interface also reduces the risk of wrong inputs and unexpected scenarios. For example, for an unavailable seat, it is not possible to click the button for reservation or our interface will hide the unavailable seats, flights or flight dates to not let unexpected scenarios happen. Also, the strong encapsulation will not let the users interfere the core processes of our implementation.

Portability: Since our program will be used by different companies and different branches of them which have varying work places, conditions and budget, one of our main goals is portability therefore we will implement it in Java which can run on almost every platform.

Maintainability: Our system includes a very dynamic data flow. Adding, removing and changing information on the system are the most frequently used functions therefore maintainability is another aim of our design.

5.2 Sub-System Decomposition

Sub-system decomposition is one of the important details because when we divide the problem into smaller parts, it becomes easier to understand, produce solutions and make progress. For Airline Reservation System (ARS), we have three main subsystems which are called ARSView, ARSController and ARSModel based on MVC pattern.

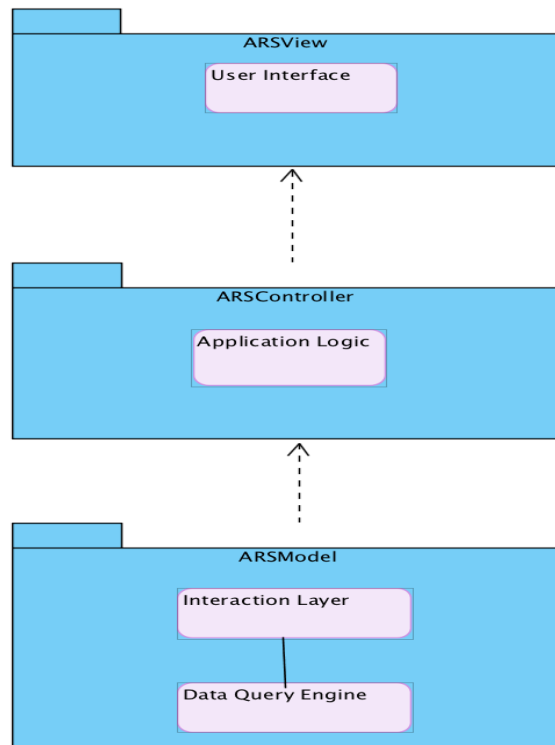


Figure 35 - Package Diagram

ARSView: This sub-system is responsible for all graphical user interfaces and their transitions. It consists of panels. ARSView collects the necessary information from ARSModel and changes the current panel when it is needed. For every page, we have different panel classes which are also in this package. PageManager class gets the necessary information from ARSModel and draws the current page.

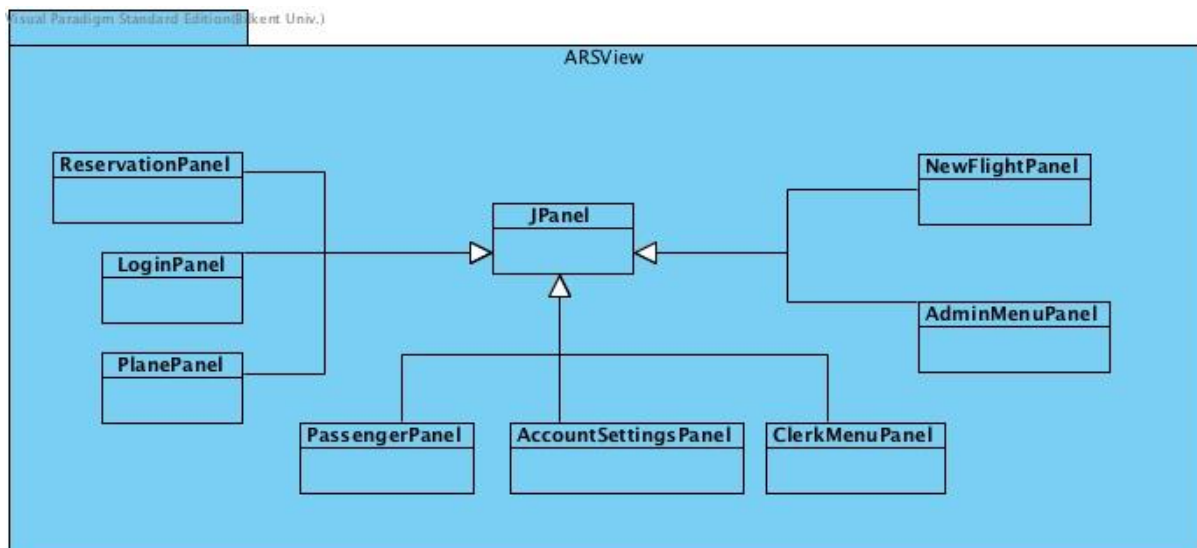


Figure 36 - ARSView Package

ARSController: This sub-system is responsible for initializing and controlling models and views, provides the modification and creation of the data. As user interacts only with the controller, ARSController should interact with ARSModel and as ARSModel interact with ARSView the needed pages will be drawn. As ARSController receives inputs from user, it calls the necessary functions of ARSView or ARSModel. AppManager class is in this package.

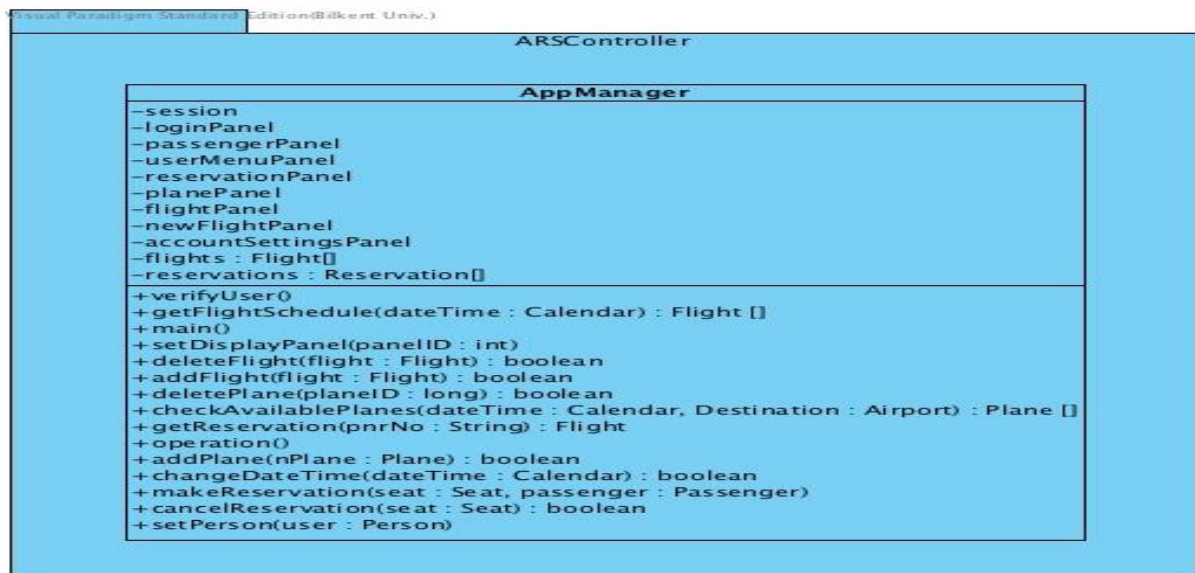


Figure 37 - ARSController Package

ARSModel: This sub-system is responsible for encapsulating all saved data such as reservations, passengers, flight and plane (seat plans) details, destination-departure places, user account information and so on. With this sub-system the system checks and fetches the data on the database when needed. In ARSModel, we also have domain objects' classes.

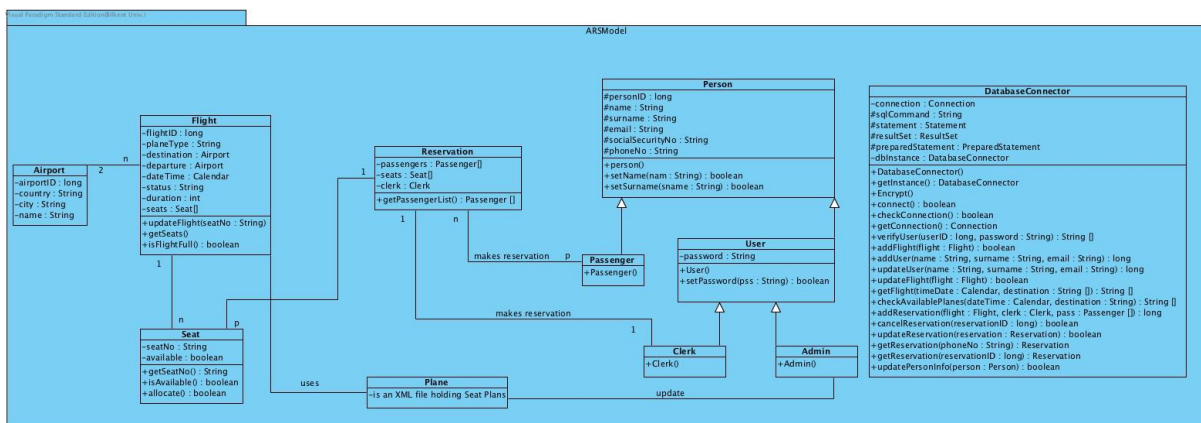


Figure 38 - ARSModel Package

5.3 Architectural Patterns

In ARS, Client Server and MVC architectural patterns are applied.

5.3.1 Client/Server with Repository

Client Server pattern will be used to manage stored data. Repository pattern and client server are linked and work together in our system. Therefore, we consider these two patterns together. Repository pattern includes a database structure where the all data of the system are stored.

The structure of Airplane Reservation System requires an architecture where the users interact with the Database of the system as a client. Since Database System holds the all data from users with MySQL database, client server pattern is applied to our system. There are many clients connected to the one common server and data can be sent to database from user or can be reached from the database by users at the same time.

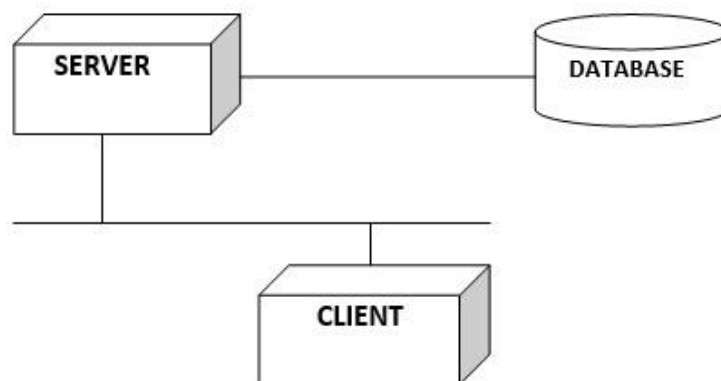


Figure 39 - Client/Server with Repository

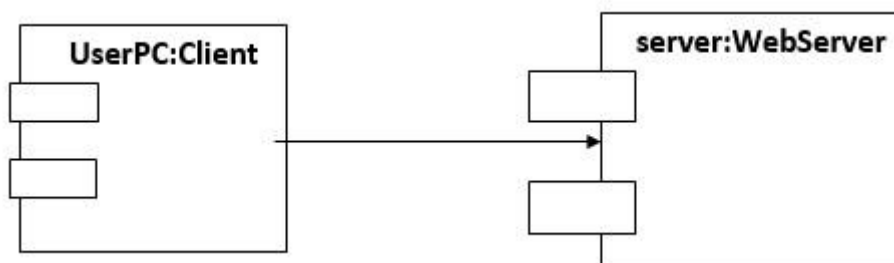


Figure 40 - Client/Server

5.3.2. Model-View-Controller Pattern

MVC composed of 3 components; model, view and controller. MVC is based on separation of concerns. For instance, MVC is used for separating data management, transactions of user entries and user interfaces from each other to provide many different users with many different views simultaneously. Airplane Reservation System should provide many views for users. Since the transitions between user interfaces should be rapid and produce new user interfaces, applying MVC pattern is very useful for our system. Also, MVC pattern increases the portability and modifiability in our system. MVC makes model classes reusable without modification. The system is divided to three subsystems, ARSModel, ARSView and ARSController. The diagram for our MVC is shown below.

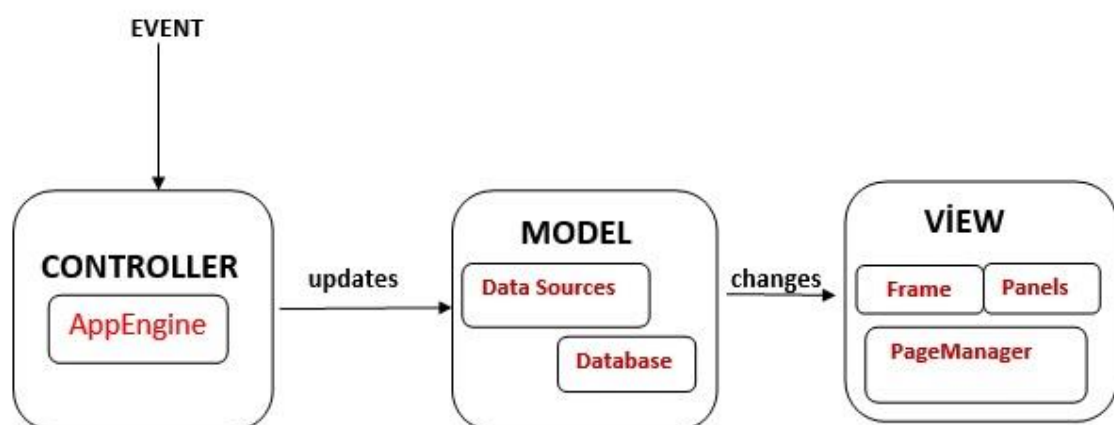


Figure 41 - MVC Architectural Pattern

ARSModel: The model part is related to data management. Mainly, how to store data by using setters and getters is needed. Any property that can change should have listeners when any changes required. The system requires rapid updates on the view when needed and listeners allow us to have multiple views of the same data. In model, we have DatabaseConnector class, which has access on MySQL Database and boundary objects' classes.

ARSView: In the view, we consider the operations when the state changes and ways to display these changes to the user. Panels are included in the view part. The model data is displayed and actions of the user are sent to the controller by button click in the view part.

ARSController: The controller part has the most significant role in the system because the controller puts classes in an order to produce operations. When the user makes an action in the view part, controller decides which operation will be done next. Each class in the view part interacts with only controller class. As a result, we get high coupling and low cohesion which is one of the important key principles of software engineering.

5.4 Hardware/Software Mapping

The programming language used to develop Airplane Reservation System is Java. We chose Java since it has good features to apply our requirements and design goals. For example, it provides good environment to catch errors in any step we want and it contributes to robustness of our system. In addition Java is a human readable programming language and since this project is a team project and team members should be able to understand others' code we chose Java. Also, we are planning to use Eclipse IDE to develop this system since it has good tools and plug-ins to make our job easier while implementing the GUI.

User can use this program in any computer that is connected to Internet. However, to use this system, customer computer should have JRE (Java Runtime Environment).



Figure 42 - Deployment Diagram

As it shown in Deployment Diagram, we decided to use Webserver which will fetch the necessary data from MySQL Database. As this approach, many users can use the system at the same time.

5.5 Addressing Key Concerns

5.5.1 Persistent Data Management

Data management is a one of the challenging task for applications which demands online data storage. Our program demands a database in order to store reservations, user privilege etc. Since our database is going to store different kind of information, it will consist of many tables, therefore we need to reflect on the design of the database. Our database will be a MySQL database.

The other thing about the data management of our program is that handling race condition. Since, multiple users may manipulate specific data in the database, race hazard may occur. We will follow a widely-known precaution (i.e. sequencing problems) against possible race hazards.

5.5.2 Access Control and Security

Our program has access control mechanism between admin and user. Admin cannot reach every method in the system as well as user. Both of them have different objects to reach in the system. User can login to the system, make reservation, delete or show reservations and update his/her personal information, such as e-mail. However, admin can login to the system; add or drop planes from flee, change time schedule of the flights and regulating user privileges. In that case user can only reach changed of flight information. With this main actors methods will differ.

In addition to these details, we are planning to encrypt vulnerable data to protect them. We will use the implementations which Java provided like Java. Security package and Cipher class.

	User	Schedule	Reservation	Plane
User	UpdatePersonallInfo	ViewSchedule	ViewReservation MakeReservation	
Admin	AddUser DeleteUser	ChangeSchedule	ViewReservation	AddPlane DeletePlane

Figure 43 - Access Matrix

One of the main concern about security is sending server a login request. In order to prevent any kind of unwanted situation, data will be encrypted.

5.5.3 Global Software Controls

In our program, system runs with the workflow of the main actors. We have an explicit control system that has a centralized control object. We control our system with event driven explicit control. AppEngine class is the centralized class of our system that controls other classes.

5.5.4 Boundary Conditions

Initialization:

- The program will start whenever the user opens executable file.
- Login screen will appear firstly.
- Login screen has interface for user where establishment of database connection starts.
- In that screen user can only login to the system with valid user name and password.
- If the information of user is wrong the error message will appear and user cannot reach the next screen.
- If the information of user is correct, user can reach main menu screen.

Termination:

- User can exit the program whenever he clicks close button. Then the execution of the program will be stop as well as the database connection and user will log out to the system.

- If user terminates the program before completing his action, system will not ask user to save the changes.
- All data that are not saved will be deleted when the user terminate the program.

Failure:

- If database connections lost an error message will be shown on the screen. To connect again to the database program must be restarted.
- While saving action if there is a fail, program will ask user to repeat the last action.

5.5.5 Software Portability

Since Java is one of those languages which allows developer to create environment-independent software, we choose Java as implementation language for our software. Our program will run on every platform that Java Runtime Environment is installed.

6. OBJECT DESIGN

6. 1 Design Patterns

In this chapter, three design patterns used in Airline Reservation System project will be explained. These are Façade Pattern, Factory Pattern and Observer Pattern.

6.1.1 Façade Pattern

Façade design pattern is a structural pattern and it is one of the most important patterns in our project since it provides several methods for a set of objects with a unified interface. Therefore, Façade makes the implementation easier and more readable for all persons involved in project. It also lowers the coupling and provides high cohesion by reducing complexity, which are important key principles for software engineering.

We used AppManager class as our façade class and define most of the useful methods here. This class also accesses the database and gets necessary information.

By using this class, we eliminated unnecessary work and instead of getting information from database in all classes, we only use this class to access entity objects' and database. The following code segment is an example of How we use AppManager class which is our façade:

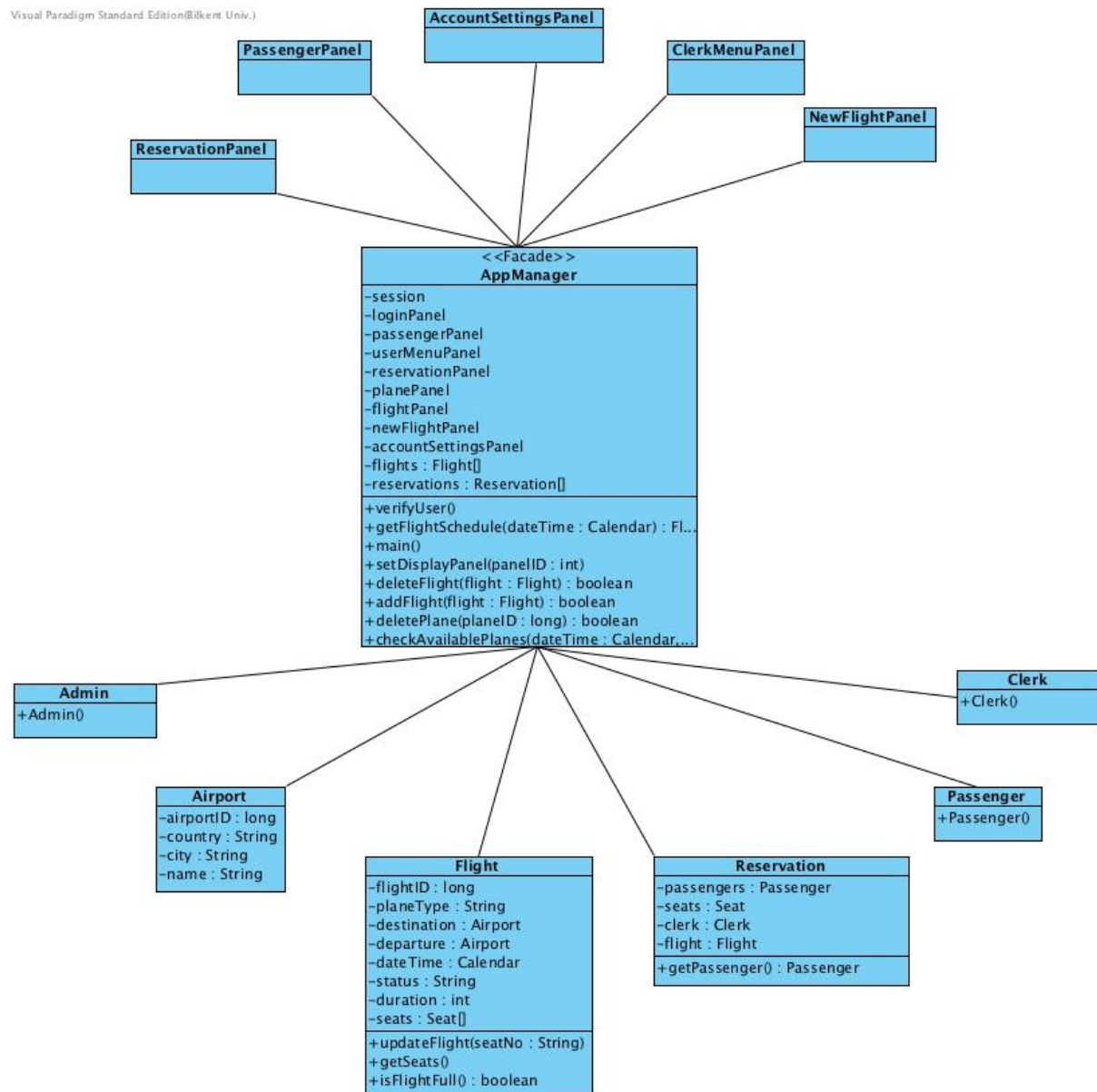


Figure 44 -Facade Pattern

For simplicity and clarity, we did not include all panels and all entity classes reached by AppManager. However, as shown in Figure 44, panels have access to entity classes and Database through AppManager.

6.1.2 Factory Pattern

Factory design pattern is behavioral pattern which helps us to separate creation of panels from controller object. As you may observe, we have two different types of users, clerk and admin, which they differ in job and privilege. Therefore, we have needed different types of panels in our program for each types of user. Our PanelFactory class is designed for this job. Once the types of user, who signs in, is determined, our PanelFactory class generates the panel needed. To be more specific, if the type of user is a clerk then our PanelFactory class generates MakeReservationPanel, DeleteReservationPanel, AccountSettingPanel, PassengerInformationPanel etc. On the other hand, if the type of user is an admin then our PanelFactory class generates AdminMainPanel (which enables admin to control Clerks' privileges and his own account settings)

Visual Paradigm Standard Edition(Bilkent Univ.)

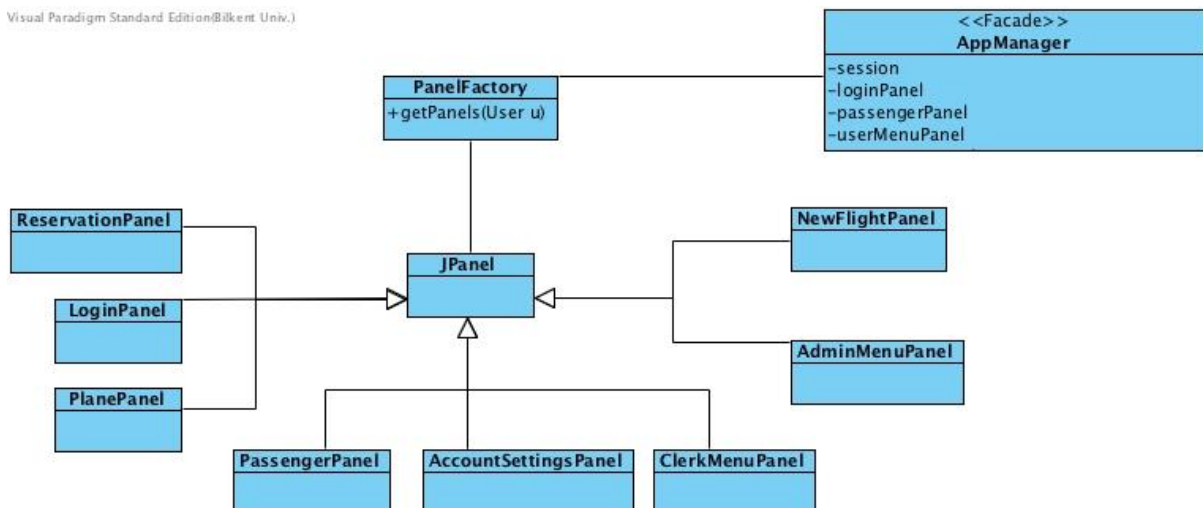


Figure 45 - Factory Pattern Diagram

6.1.3 Observer Pattern

As we used Model-View-Controller pattern as an architectural pattern, using observer pattern was inevitable for us. The reason behind is the change in model

results in change in views. Thus, model should notify the views namely panels and panels should be updated.

In our design, AppManager class which belongs to Model package is subject and panels which are belong to View package are observer. According to changes AppManager receives, it notifies views and current panel will be updated.

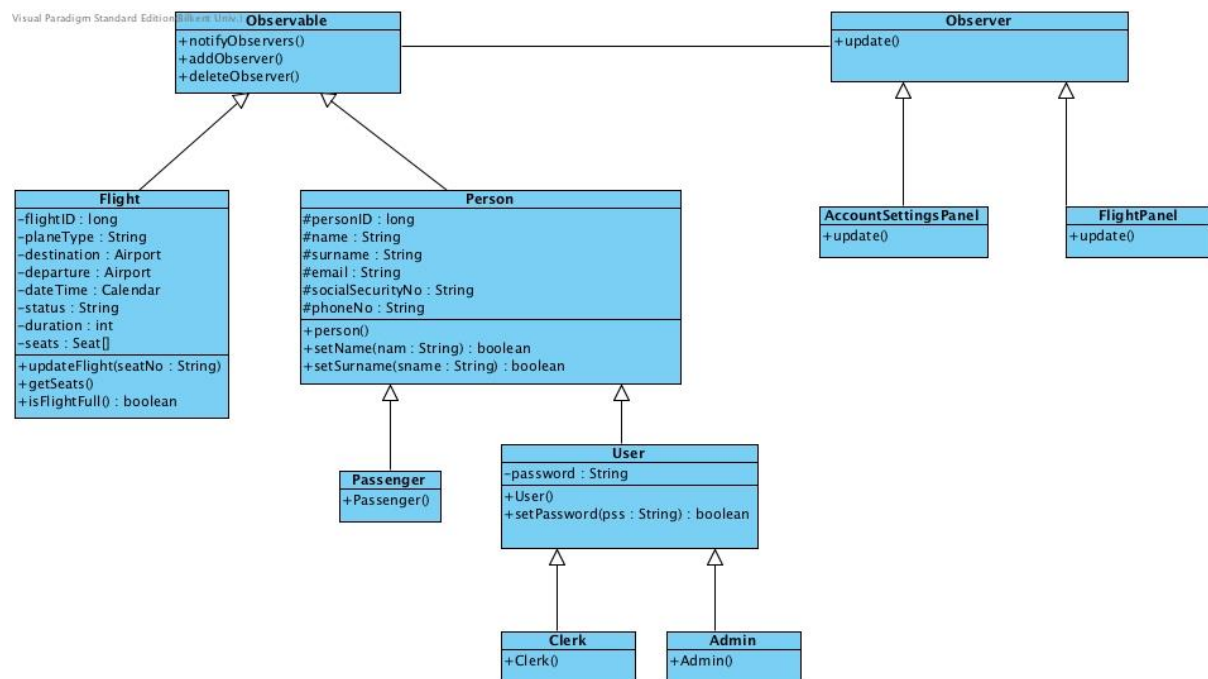


Figure 46 - Observer Pattern Diagram

6.2 Class Interfaces

6.2.1 ControllerPackage

6.2.1.1 AppManager

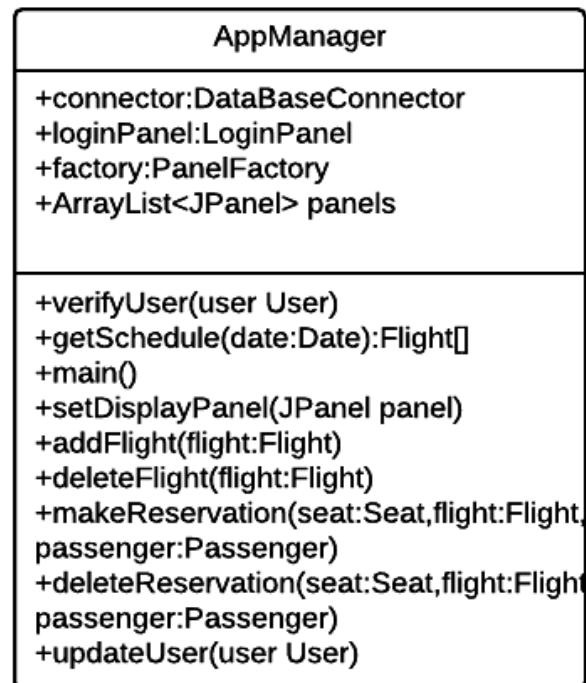


Figure 47 - AppManager

Connector: connector is the instance of databaseConnector which establishes connection with the database, all database operations are done upon the variable.

loginPanel: loginPanel is the initial panel of the app.

factory: PanelFactory is the class that generates panels according to user type.

panels: panels that PanelFactory returns are held inside panels arraylist.

Methods:

verifyUser: verifies that if the user exists using connector.

getSchedule: return the flights that are arranged in given date.

setDisplayPanel: set the panel that is being showed.

addFlight: adds flight database using connector.

deleteFlight: deletes the given flight from database.

makeReservation: makes reservation by changing database using connector

deleteReservation: deletes a reservation

updateUser: updates user information in the database

6.2.1.2 PanelFactory.java

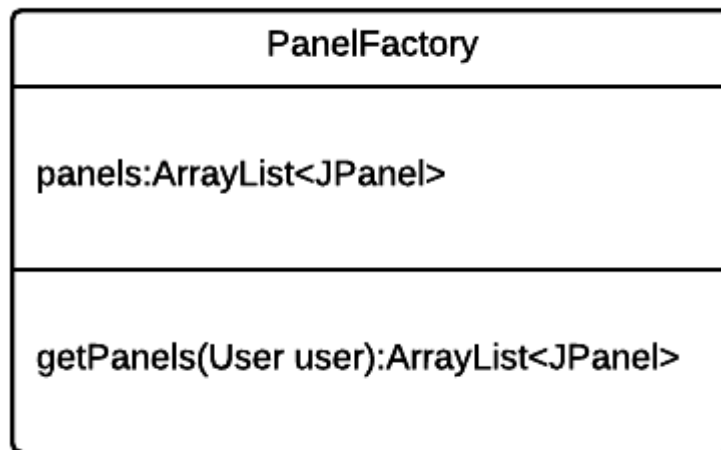


Figure 48 - Panel Factory

PanelFactory is a class which generates only necessary panels according to type of user that logs in.

getPanels(User user):ArrayList<JPanel>: gets the user that logs in and considering which object the user instanceof generates necessary panels.

6.2.2 Database Package

6.2.2.1 DatabaseConnector

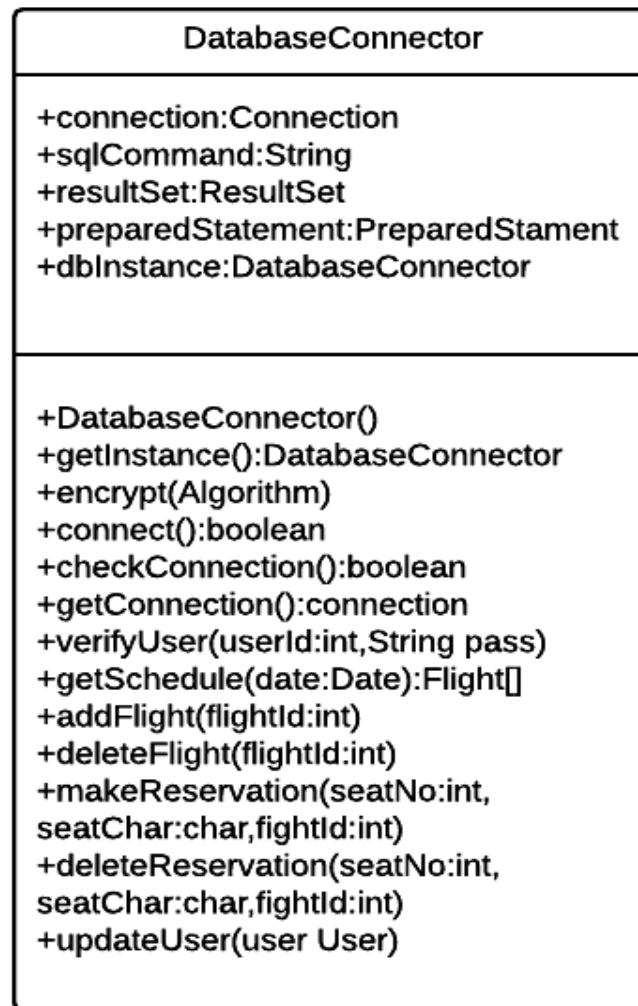


Figure 49 - DatabaseConnector

DatabaseConnector is a class that is responsible for establishing connection with database. Since we already mentioned most of the operations above, we will just explain, we have not mentioned yet.

encrypt(): the method that encrypts users password using specific algorithm

connect(): establish connection with a database, in case of success returns true otherwise return false

checkConnection(): checks if connection lost, returns false if so

6.2.3 Model Package

6.2.3.1 Person.java

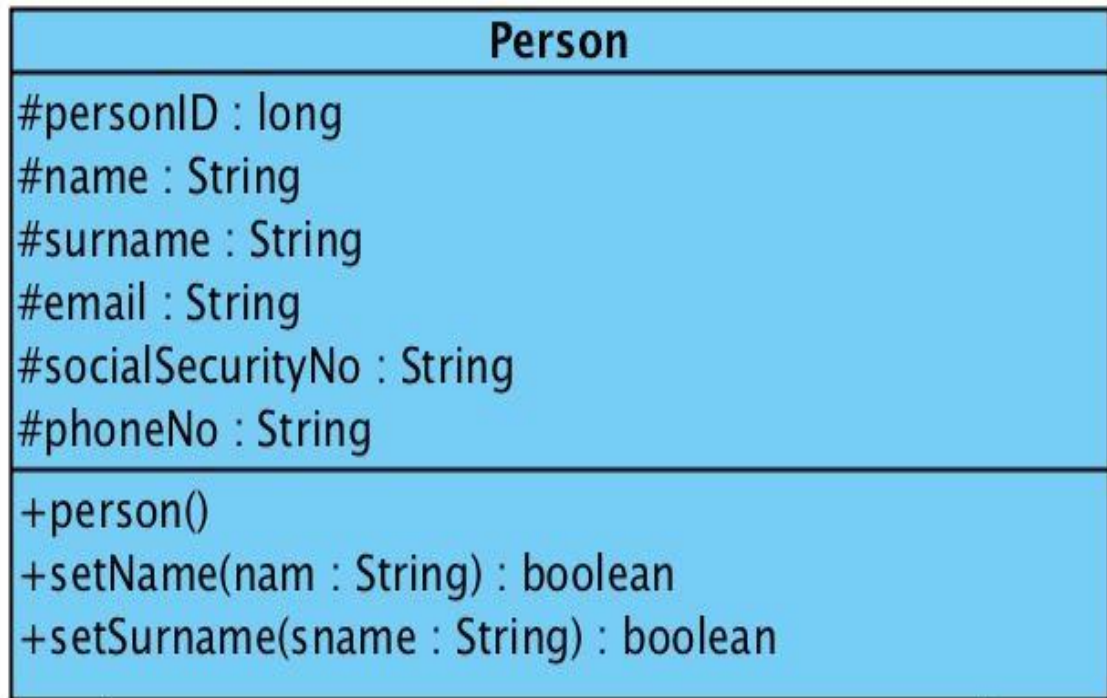


Figure 50 – Person

Person class is an abstract class which is a parent of both user class.

Extends: Observable

Methods:

getter and setters

6.2.3.2 User.java

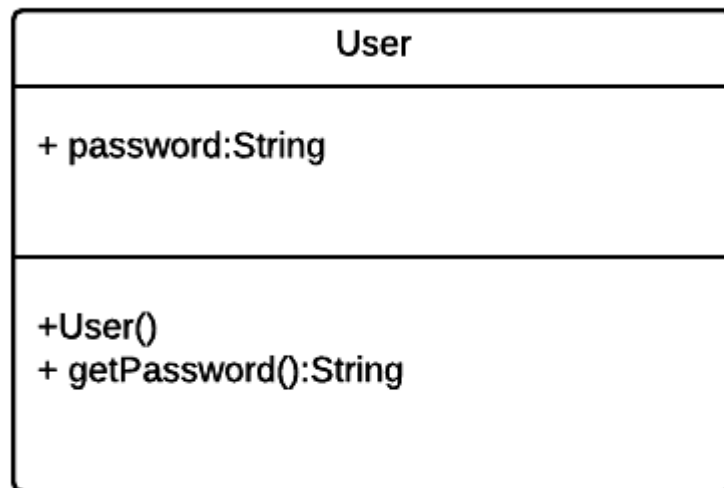


Figure 51 - User

Extends: Person

Child class of the person class which holds the password of a user.

Methods:

getPassword(): returns the password of a user.

getters and Setters that overwritten.

6.2.3.3 Seat.java

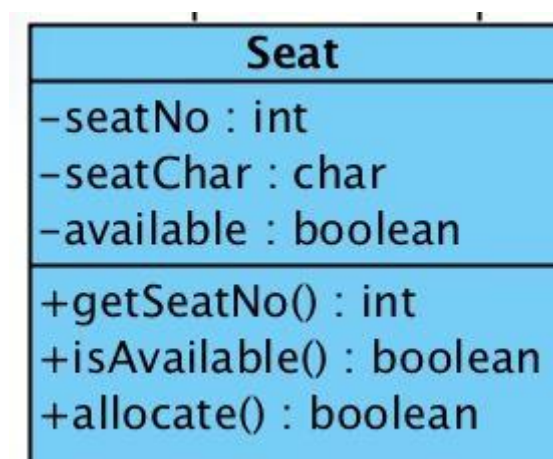


Figure 52 - Seat

Seat class holds the information about a seat of a plane. Each seat is represented by one number and one char.

Methods

+getSeatNo() : Returns the number of the seat

+allocate(): Mark the seat as unselectable

+isAvailable(): Returns the seat's availability

6.2.3.4 Flight.java

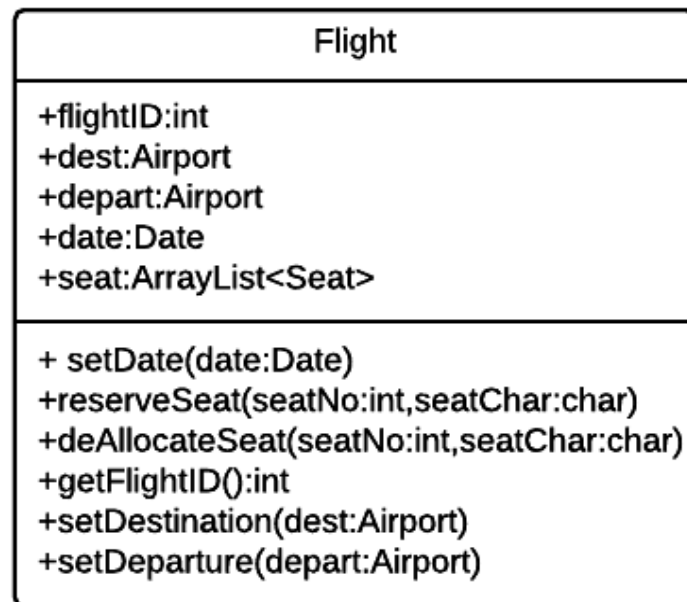


Figure 53 - Flight

Flight class holds information about flights which consist of destination point departure point flight id date and array of seats.

Implements: IObservable

Methods:

setDate(date:Date): sets the date

reservaSeat(seatNo:int,seatChar:char): reserves the specified seat

deAllocateSeat(seat:No:int,seatChar:char): deallocates the specified seat.

getFlight():returns the flight id

setDestination(des:Airport): sets destination point as specified airport

setDeparture(depart:Airport) sets departure point as specified airport

6.2.3.5 Airport.java



Figure 54 - Airport

Airport object which only consist of an id, name, country and city.

Methods: getters and setter

6.2.3.6 Passenger.java

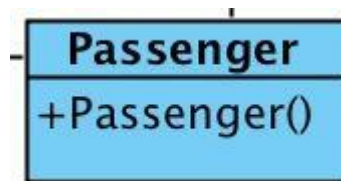


Figure 55 - Passenger

Extends: Person

6.2.3.7 Clerk.java

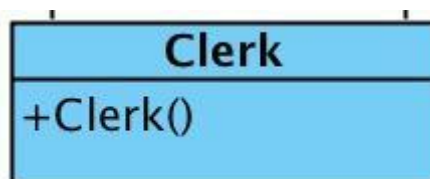


Figure 56 - Clerk

This class is used to determine the access authority to features.

Extends: User

6.2.3.8 Admin.java



Figure 57 - Admin

This class is used to determine the access authority to features.

Extends: User

6.2.3.9 Reservation.java

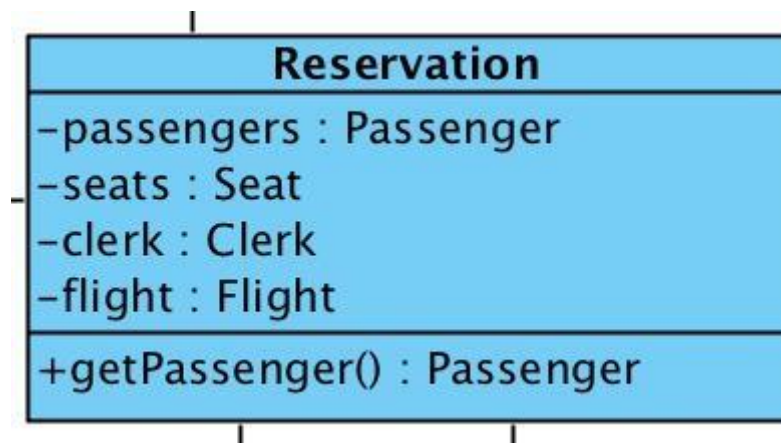


Figure 58 - Reservation

Reservation class holds the information about passenger that reserves, reserved seat, clerk that have done the reservation and the flight.

6.2.4 View Package

6.2.4.1 LoginPanel

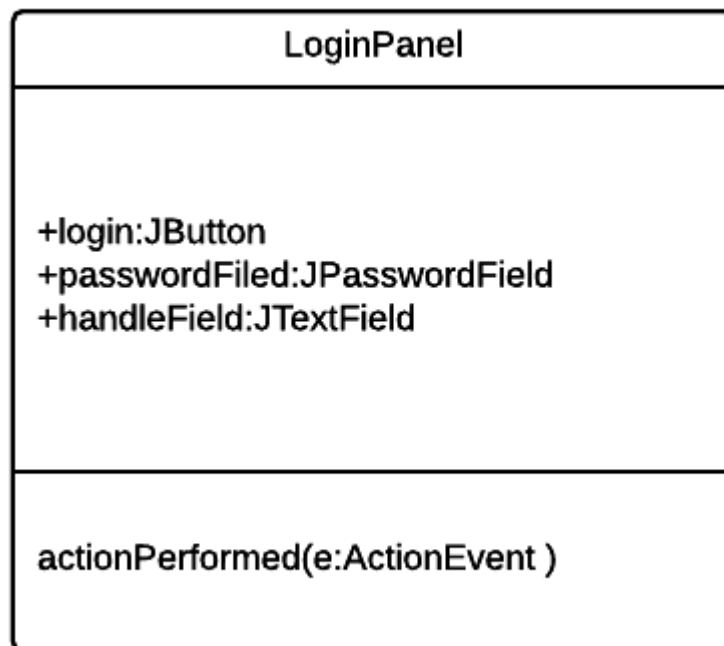


Figure 59 - LoginPanel

Two text field and one button. handleField is for username and passwordField is for password.

6.2.4.2 ClerkMainMenu

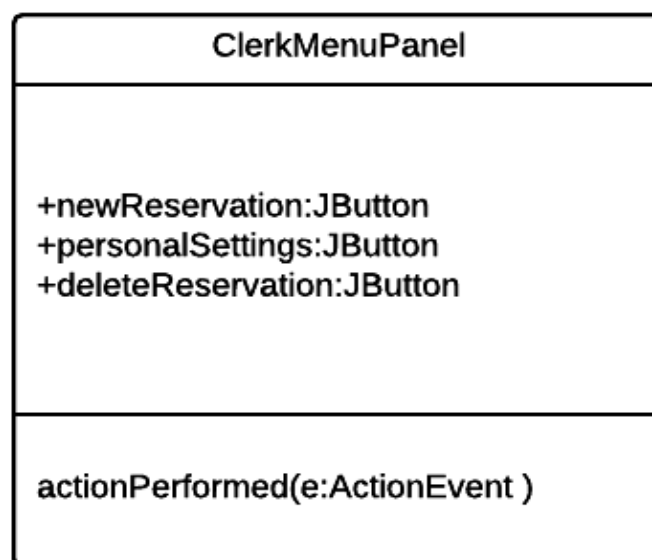


Figure 60 - ClerkMenuPanel

The main panel for clerks which can lead to new reservation panel delete reservation panel and personal settings menu.

6.2.4.3 AdminMainMenu

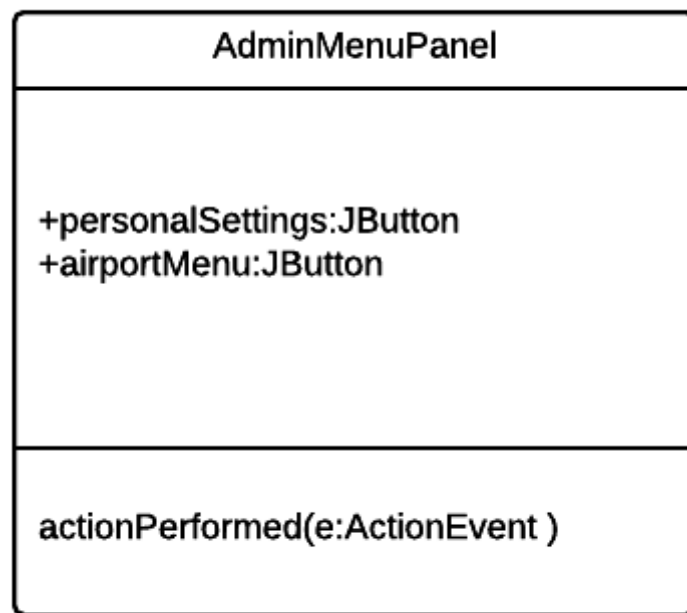


Figure 61 - AdminMenuPanel

personalSetting button leads to personal settings and airportMenu button leads to airport menu.

6.2.4.4 ReservationPanel

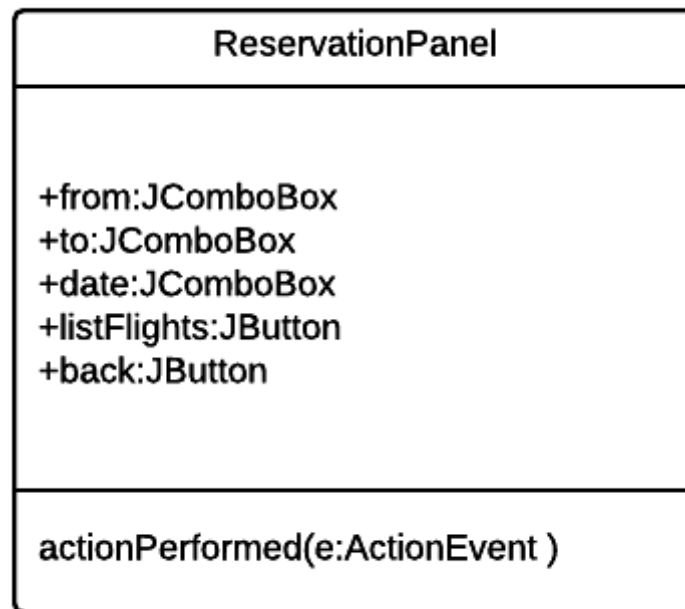


Figure 62 - ReservationPanel

User select departure destination and date in this panel. After that, **listFlights** button leads to panel that shows suitable flights. Back button returns to **ClerkMainPanel**.

6.2.4.5 PersonalSettingPanel

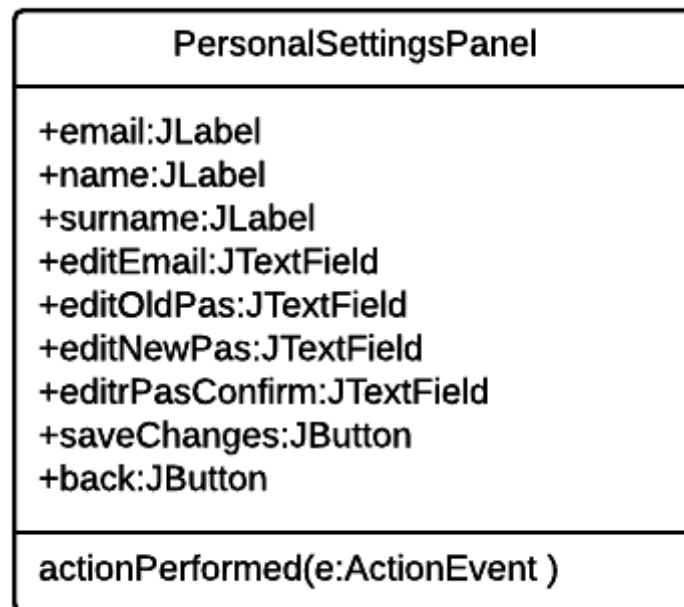


Figure 63 - PersonalSettingsPanel

Implements: Observer

The panel, that is common for both clerk and admin, which they can alter their information. In this panel JLabels are to show current information and JTextFields are to change the information.

6.2.4.6 DeleteReservationPanel

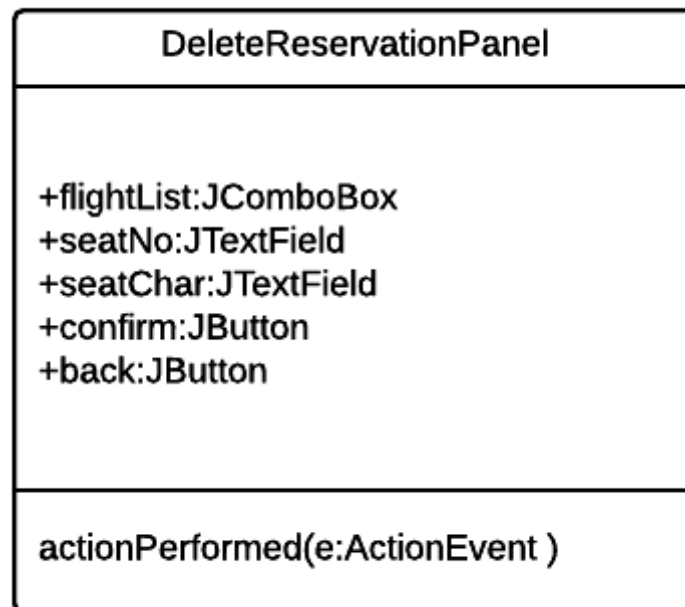


Figure 64 - DeleteReservationPanel

In this panel, user selects the flight and the seat that is going to be deleted. Confirm button saves the changes back button goes back to ClerkMainPanel.

6.2.4.7 ListFlightPanel

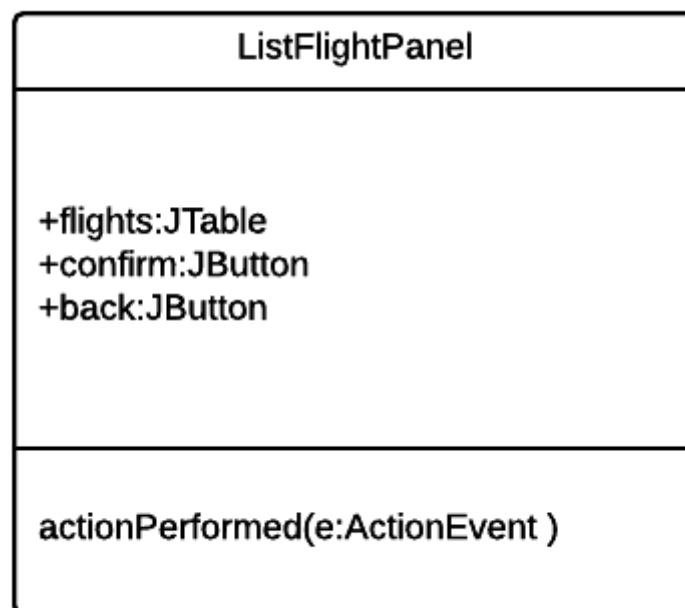


Figure 65 - ListFlightPanel

The panel that shows up while making a new reservation. flights is the table that shows the flights that suitable for user criterion. Once user selects the flight and hits the confirm button, FlightPanel shows up. If the user hits back button, it returns the NewReservationPanel.

6.2.4.8 FlightPanel

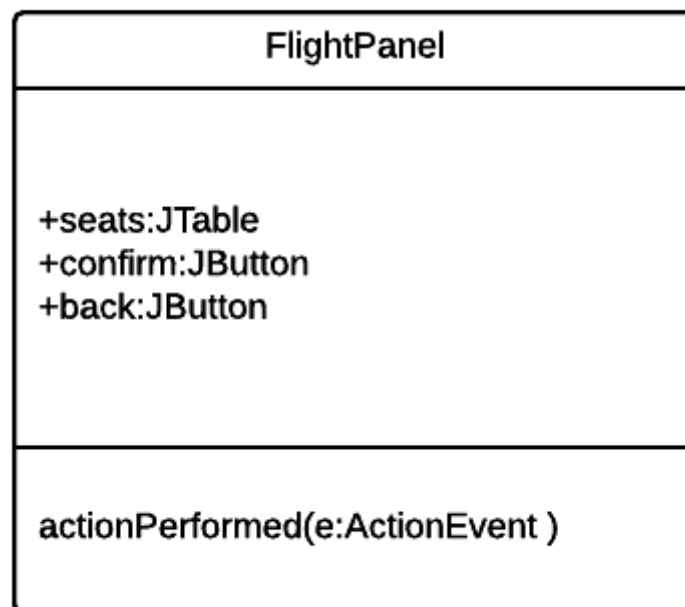


Figure 66 - FlightPanel

Implements: Observer

That is the last panel that user interacts while making reservation. FlightPanel shows the seats in that flight(available seats are represented as enable checkbox and unavailable ones are represented as disabled checkbox). After user hit confirm button it saves the changes and returns to the ClerkMainPanel. “back” Button leads to the ListFlightPanel in which Clerk can select a flight.

6.3 Specifying Contracts using OCL

AppManager:

1) //to add flight, current user must be admin which we control with isAdmin method, it returns true //if the current user is admin

context AppManager::addFlight(f : Flight)

pre: self.user -> isAdmin(user : User)

2) //to delete flight current user must be admin and flight should be empty which we control with //isEmpty method

context AppManager::deleteFlight(f : Flight)

pre: self.user -> isAdmin(user : User) and f -> isEmpty()

3) //to register a new user to the system, user should not be registered before
// emails are unique in the system we check if the email already exists

context AppManager:: addUser(user : User)

pre: db -> isExists(user->getEmail())

4) //to delete user current user must be admin and given id should be grater than zero and should //not be equal to itselfs id. It deletes given user id from databasse

context AppManager::deleteUser(id : int)

pre: self.user -> isAdmin(user : User) and id > 0 and self.user->getID()<>id

post: db -> deleteUser(id);

5) //to add reservation, current user must be clerk.

context AppManager:: addReservation()

pre: self.user -> !isAdmin(user : User)

6) //to delete reservation, current user must be clerk.

context AppManager:: addReservation()

pre: self.user -> !isAdmin(user : User)

Person:

7) //id attribute of Person class should be greater than 0

context Person inv:

self.id>0

8) //getID method of Person class returns id attribute as result

context Person::getID()

post: result = self.id

User:

9) //getPassword method of User class returns password attribute as result

context Person::getPassword()

post: result = self.password

Flight:

10)//setTime operation takes a long parameter and sets the date attribute to this parameter

context DestinationDeparture::setTime(time : long)

post: self.date = date->setTime(long)

11) //to reserve a seat in a flight, seat should be available as pre-condition and allocated seat size must //be incremented at the end

context Flight::reserveSeat(seatNo : int, seatChar: char)

pre: self.isAvailable(seatNo : int, seatChar: char)

post: self.reservedSeatSize-> self.reservedSeatSize @pre+1

12)//getFlightID operation of Flight class return flightID as a result

context Flighth::getFlightID()

post: result: flightID

13) //to deallocate a seat in a flight, seat should be reserved before as pre-condition and allocated //seat size must be decremented at the end

context Flight: **deallocateSeat**(seatNo : int, seatChar: char)

pre: !self.isAvailable(seatNo : int, seatChar: char)

post: self.reservedSeatSize-> self.reservedSeatSize @pre-1

14) // flightID attribute of Flight class should be greater than 0

context Flight inv:

self.flightID > 0

15) //destination and departure places must be different.

context Flight inv:

self.destination <> self.departure

16)//**getTime** operation of Flight class return the time as a result

context Flight::getTime()

post: result = self.date.getTime()

17) //**getDate** operation of Flight class return the date as a result

context Flight::getDate()

post: result = self.date.getDate()

18) //setDate operation takes a Date parameter and sets the date attribute to this parameter

context Flight::setDate(date : Date)

post: self.date = date

19)//setDestination operation of Flight class sets the destination airport to the given Airport //parameter

context Flight::setDestination(dest : Airport)

post: self.destination = dest

20)//seDeparture operation of Flight class sets the Departure Airportto the given Airport //parameter

context Flight:: setDeparture (depart : Airport)

post: self. departure = depart

Seat:

21)//seatNo attribute in Seat class should be more than zero

context Seat inv:

self.seatNo > 0

22) //setAllocated operation of Seat class makes boolean allocated attribute of this class true.

context Seat::setAllocated()

post: self.allocated = true

23) //setDeallocated operation of Seat class makes boolean allocated attribute of this class false.

context Seat::setDeallocated()

post: self.allocated = false

24)//getSeatNo operation returns seatNo as a result

context Seat::getSeatNo()

post: result = self.seatNo

Airport:

25) //airportID must be greater than zero.

context Airport inv:

self.airportID > 0

AdminMainPage:

26) //if personal setting button is pressed in AdminMainPage, it sets the panel 2

//which is personal settings panel

context AdminMainPage:: actionPerformed(a: ActionEvent)

pre: a -> getSource() = self.settings

post: AppManager -> frame -> setPanel(2);

27) //if airports button is pressed in AdminMainPage, it sets the panel 3

// which is airportMenu panel

context AdminMainPage:: actionPerformed(a: ActionEvent)

pre: a -> getSource() = self.airport

post: AppManager -> frame -> setPanel(3);

ClerkMainPage:

28) //if new reservation button is pressed in ClerkMainPage, it sets the panel 5

//which is new reservation panel

context ClerkMainPage:: actionPerformed(a: ActionEvent)

pre: a -> getSource() = self.settings

post: AppManager -> frame -> setPanel(5);

29) //if delete reservation button is pressed in ClerkMainPage, it sets the panel 8

// which is delete reservation panel

context ClerkMainPage:: actionPerformed(a: ActionEvent)

```
pre: a -> getSource() = self.airport  
post: AppManager -> frame ->setPanel(8);
```

30) //if personal setting button is pressed in ClerkMainPage, it sets the panel 2

//which is personal settings panel

context ClerkMainPage:: actionPerformed(a: ActionEvent)

```
pre: a -> getSource() = self.settings
```

```
post: AppManager -> frame ->setPanel(2);
```

7. Conclusion and Lessons Learned

To sum up, our airplane reservation system will be a desktop application in which reservations can be regulated. This will make reservation easy to follow and change. Also, user-friendly environment makes both user and admin work easy and faster.

Throughout analysis process, we have learned many useful things that, potentially, has significant importance in our future business life. Planning and analyzing are the phases that determines the path that we should follow throughout the development process. State charts and sequence diagrams are the tools that visualize plan that we have made therefore, they have great importance. This process has greatly helped us to comprehend how to use those tools.

Last but not least, analysis part teach us how to start a project from scratch. If we started our project before doing any analysis, we would have completely lost and we would not be able to proceed.

Throughout this report, we learned many useful things that has significant importance in our future business life. Use-case and class diagrams were two of them. We also learned how to draw state charts and sequence diagrams which show the dynamic behavior of objects and understood their importance in design step. Our project has obstacle-free process so far and it is extendible in the future. If there will

be any other need about airline reservation system, our project can easily be extended to meet with these new needs.

In the system design part, we learned about sub-system decomposition and its' importance on software development. In addition, we learned about architectural patterns, hardware/software mapping and how to draw deployment diagrams. Finally, we addressed the key concerns related with our project. In object design, we used design patterns to ease our job and make the implementation more readable and reusable. We also learned why water-fall approach is not very usable since we found problems with analysis in further steps including system and object design and fixed them.

If we consider our adopted process, we think that diagrams and patterns in this report will make our job very easier in the implementation part. As the most important activities in software engineering development activities are requirements elicitation, analysis, design, implementation and testing, we are done with the most important ones and left with implementation which will be easier with the help of this report.