GROUP 10

FINAL PROJECT PROPOSAL

AIRPORT MANAGEMENT SYSTEM

- Alp Emir BİLEK
- Mohammad Ashraf YAWAR
- İlkan Mert OKUL
- Rahmet Ali ÖLMEZ
- Bilal BAYRAKDAR
- Nevzat SEFEROĞLU
- Ece ERER
- Elif KELEŞ
- Seniha Sena TOPKAYA

CONTENTS

Attention: * -> represents changes

1. PROBLEM DEFINITION

Airport Management System is a system which provide communication between clients and Airline Companies. It organizes whole flight operations.

As days go by, more and more people are using airplanes for transportation. This increase demands for more planes, airports and employees, as the capacity of the current resources start to not suffice the needs. All this growth and increased complexity becomes a huge problem for the passengers, employees, managers and companies.

To keep up with this increase of demand, airline companies and airports need a sophistic ated, easy-to-use management system. Our system is designed to meet these needs.

To its users our system provides:

- Coordinating the airplanes to help them take off and land safely.
- Providing resources for airline companies to manage flight related operations such as ticket sales and check-ins.
- Implementing a display for available tickets, and the details of flights.
- Giving flight information for the passengers who already have tickets, when requested.
- Providing restaurants, banks, stores etc. to meet the needs of the passengers.
- Employing managers and personnel to organize all the resources.

2. USERS OF THE SYSTEM

The users of this system are customers, administrators and personnels of airline companies and the airport and managers of the stores in the airport.
Flight managers and check-in personnel combined for simplicity. Also customer with/ without ti cket option is removed.

2.1. Airport Users

- **Airport Administrator** can add/ remove an Airport personnel, Airline Company, Shop in airport, destination, and set commission rate. Manages the administrative users.
- **Airport Personnel** can dismiss a customer.
- **Customer** can display flights, buy tickets.
- **Shop Manager** can open and close Shops.

2.2. Airline Users

- **Airline Administraton** can recruit/ dismiss a personnel and add/remove destination, flight and aircraft.
- **Airline Personnel** can login and the system.
- ***(Airline Company manager, Airport Traffic Tower, Flight Manager and Guest are removed due to lack of necessity.)

3. REQUIREMENTS

3.1. User Requirements

- All the users should have a menu with choices to pick from.
- The user interface should be clear to understand and easy to use.
- A graphical user interface could be implemented, to improve the convenience of the system.

3.2. System Requirements

- The system will have separate menus for each user type, including administrators, managers, employees and customers. The menus will have a clear, understandable user interface for the system to be user-friendly.
- The system will provide an interface for both airports and airlines, as these closely work together for the passengers to be able to fly.

3.3. Functional Requirements

- The menus for registered users will be accessed only if the correct password is entered. Unless the correct password is entered, a warning will be shown to the user, stating that the password is incorrect.
- The users will only be able to access the operations related to themselves. In other words, no user will be able to make operations that do not belong to them. Also, no private data should be accessed other than the related user.
- Administrators will be able to add managers and employees, employees will add customers, airline companies' manager will add airline companies and store managers will add stores to the system.

- Customers will be able to browse available tickets and see their flight information. They will be able to buy tickets and register to the system to see their flight information from the system.

3.4. Non-Functional Requirements

- Aggregation should be used efficiently across the system where new managers, employees, customers etc. are created for better code reusability.
- All possible exceptions should be handled, and an error message should be shown to the user for each exception.

4. IMPLEMENTATION DETAILS

4.1. *Client

In this module, we have an Interface called UserInterface, which simply has methods to oprate every User's information (like social security number, and a password for system). And a class called Person which implements the PersonInterface.
This is the main class to be used in every administrator, personnel, manager, customer etc.
Basically a person has the information of its name and surname and it is comparable.

- **ArrayList** data structure was used for Airport Personnel. Operations may be easier.
- **TreeMap** data structure was used for Customer. They have keys and values. Methods of this structure are more efficient than others.
- **PriorityQueue** data structure was used for Shop Manager. Shops will be prioritized according to their fees.
- **Queue** data structure was used for Airline Personnel. To keep pilots and personnels working consecutively.

4.2. Airline

This module keeps the data of Aircraft (used by Airline companies), Date and Time, Ticket, Destination and Flight, Airline, AirlineAdmin and AirlinePersonnel, and a storage class to construct data structures.

- Airline can create/ delete tickets, set comission rate.
- AirlineAdmin recruit/ dismiss Personnel, add/ remove destination, aircraft, flight.
- AirlinePersonnel has the information of personnel.
- There are three types of tickets: Economy_Class, Business_Class, First_Class.

Containers of this module:

- **Map** data structure was used for Ticket. Map is ideal to use when we have an identifying value and a key value linked to it.
- **ArrayList** data structure was used for Aircraft. Since our frequent operation is get(), ArrayList is more efficient to keep our data.
- **MapGraph** data structure was used for Destination. We represent flight destinations as graph since it is practical, when calculating paths/ distance.

4.3. Airport

The basest type of the System. Anything will be managed from Airport class. Airport module includes information about flights, tickets, customers, personnels, airline companies, shops. Stores many lists for Customers, Airline Companies and Shops. These types have aparted Managers(ShopManager,CompanyManager). Managers can add, remove objects from their own branch lists. Airport Admin is the superior of that managers. It can manage the whole system. Shop Manager manages shop situation(open-close) and fee. Airport Admin manages shops, airline companies, destination of flights and Airport personnels. Airport personnel can dismiss customer. Customers can display flights and buy ticket or search his/her ticket by using PNR.

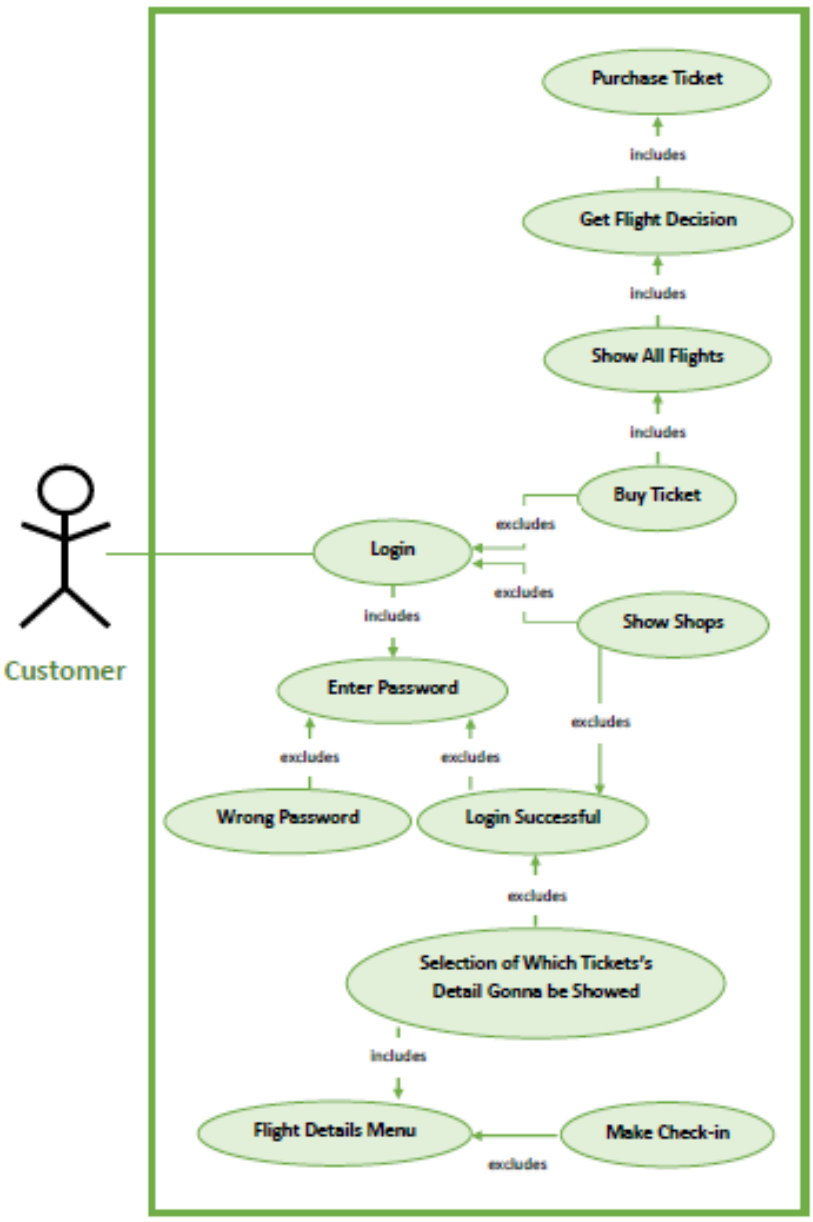
- **TreeMap** data structure was used for Airline, Flights and Customer. They have keys and values. Methods of this structure are more efficient than others.
- **PriorityQueue** data structure was used for ShopManager. Shops will be prioritized according to their fees.
- **ArrayList** data structure was used for Place and AirportPersonnel. Operations may be easier.

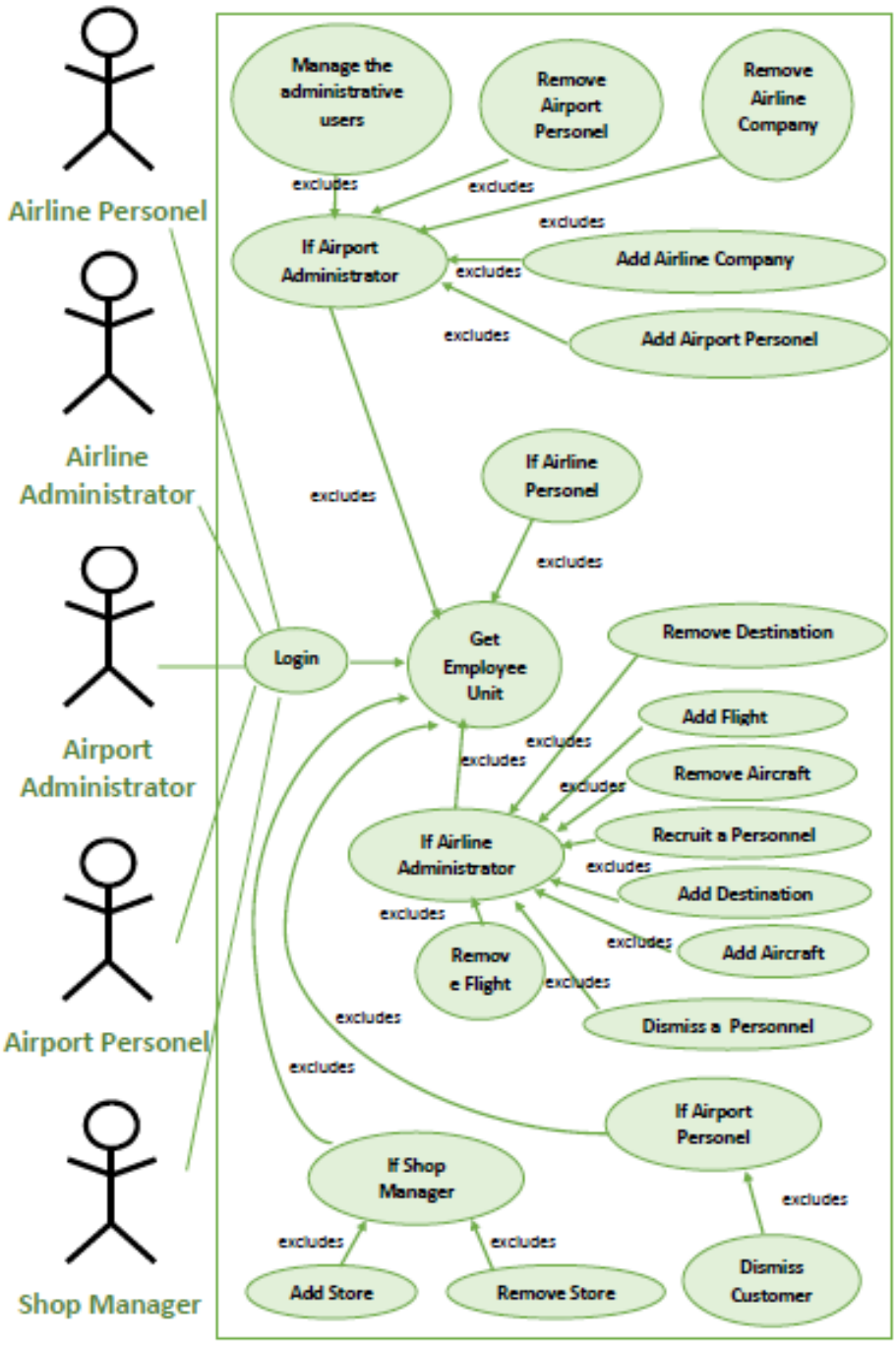
These structures are implemented and used in this project:

- Abstract Graph
- Breadth First Search
- Depth First Search
- Edge
- Graph
- List Graph
- Map Graph
- Skip List

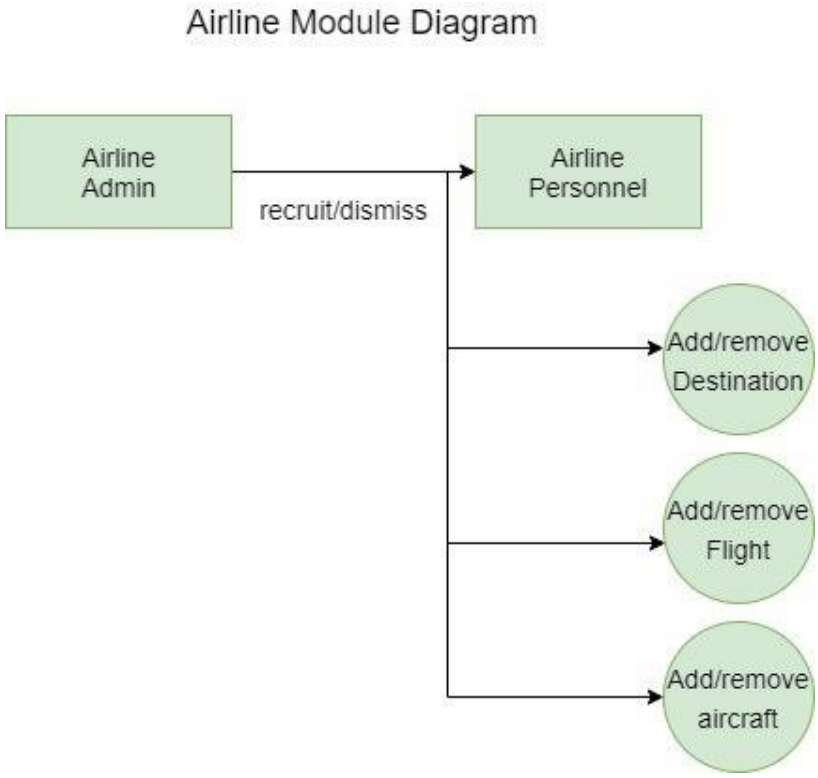
5. DIAGRAMS

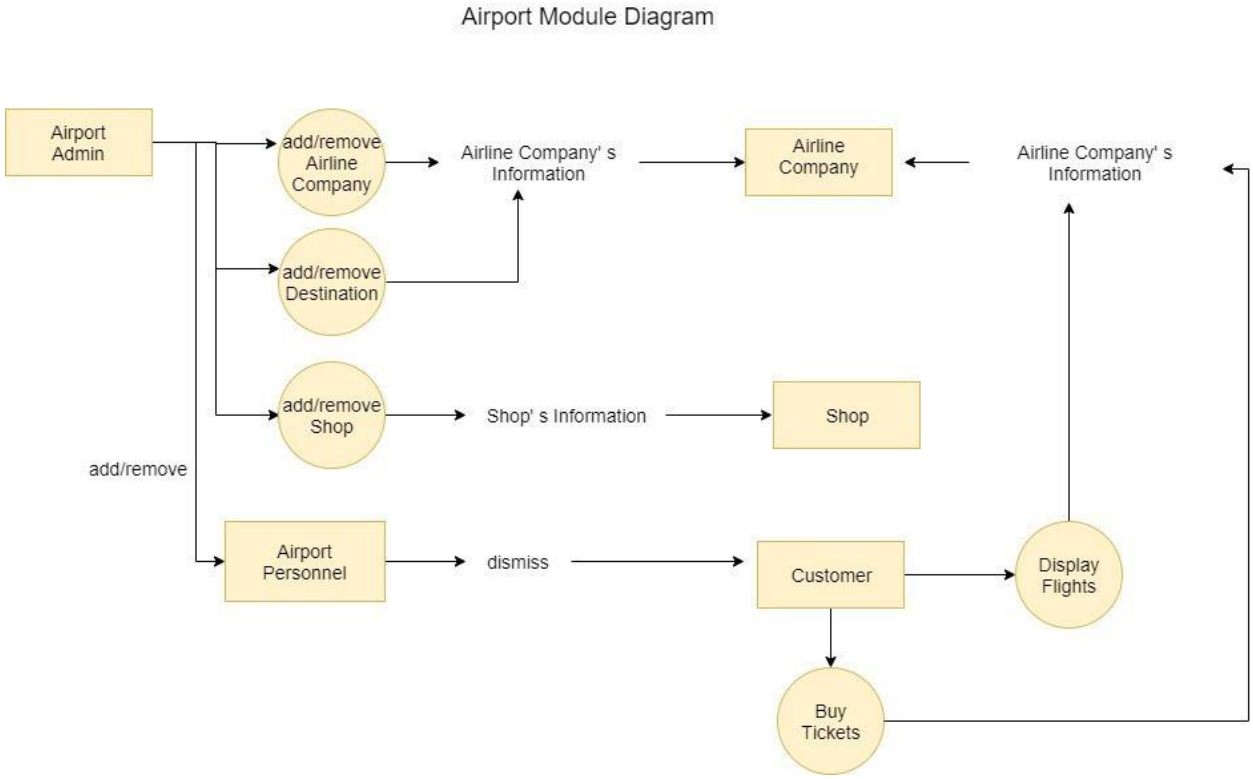
5.1. Use-Case Diagram




Airline Personel
Airline Administrator
Airport Administrator
Airport Personel
Shop Manager
Manage the administrative users
Remove Airport Personel
Remove Airline Company
If Airport Administrator
Add Airline Company
Add Airport Personel
If Airline Personel
Login
Get Employee Unit
Remove Destination
Add Flight
Remove Aircraft
Recruit a Personnel
If Airline Administrator
Add Destination
Remov e Flight
Add Aircraft
Dismiss a Personnel
If Airport Personel
If Shop Manager
Add Store
Remove Store
Dismiss Customer
excludes

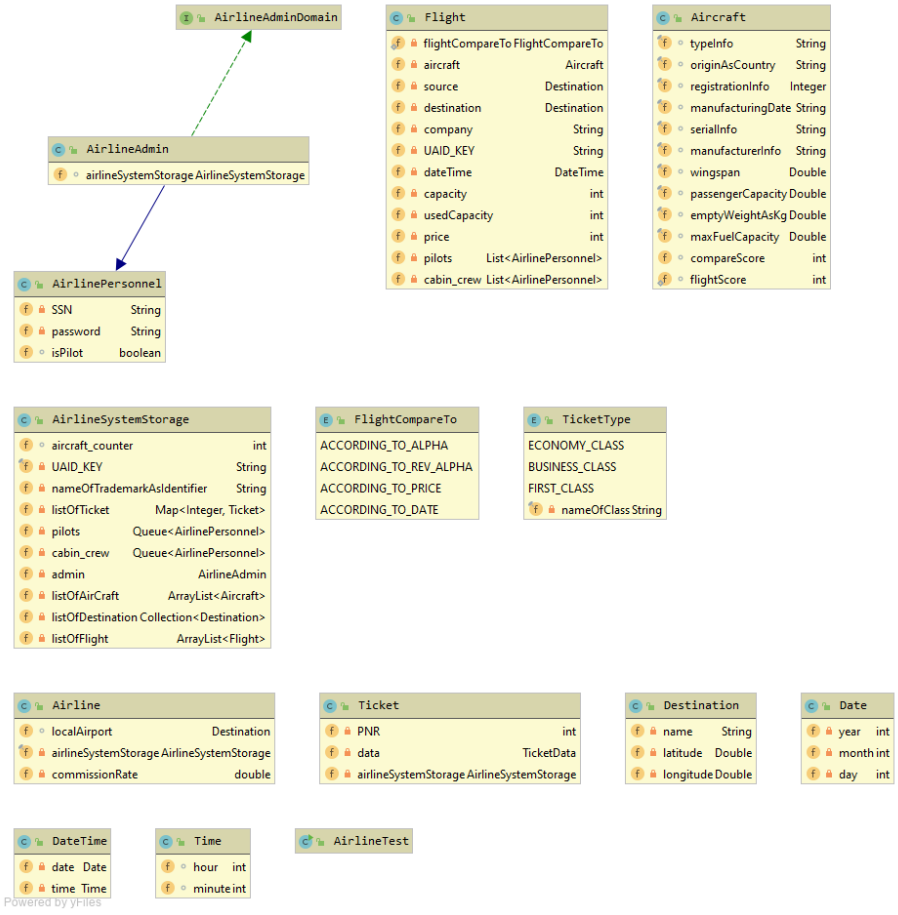
5.2. Module diagram



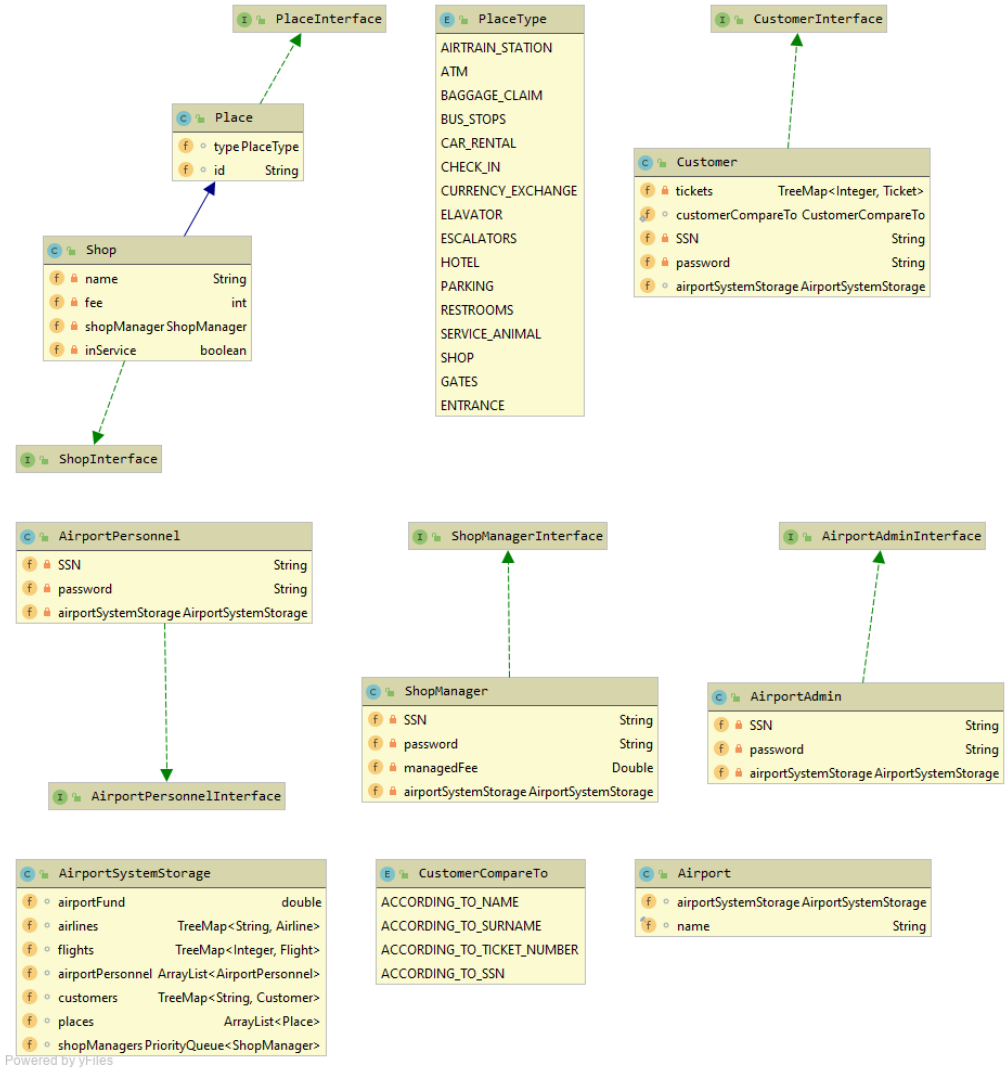


5.3. Class Diagrams

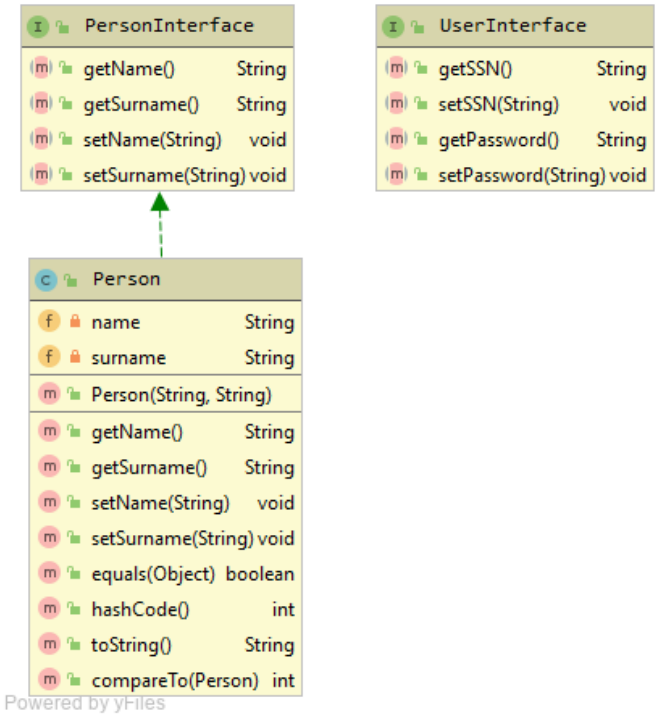
5.3.1. Airline Class Diagram



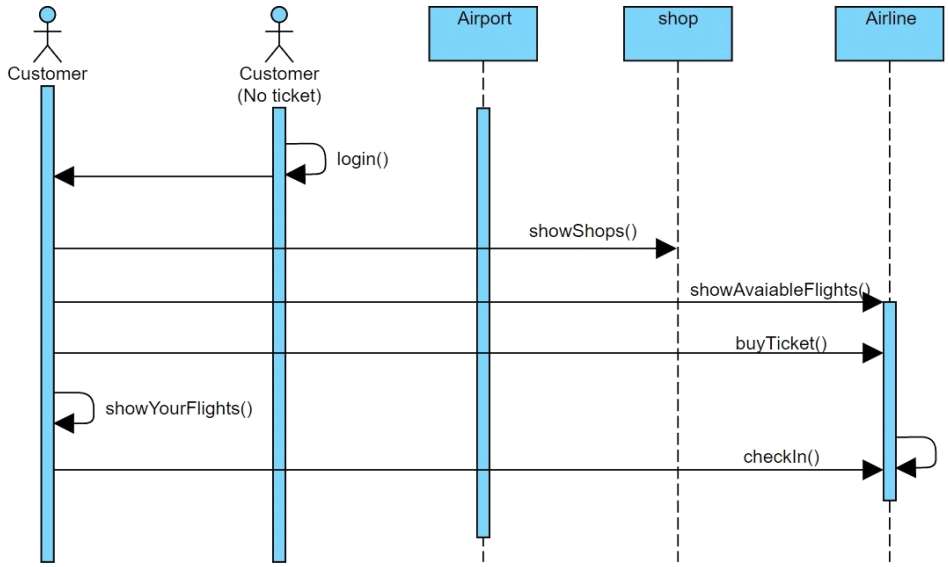
5.3.2. Airport Class Diagram

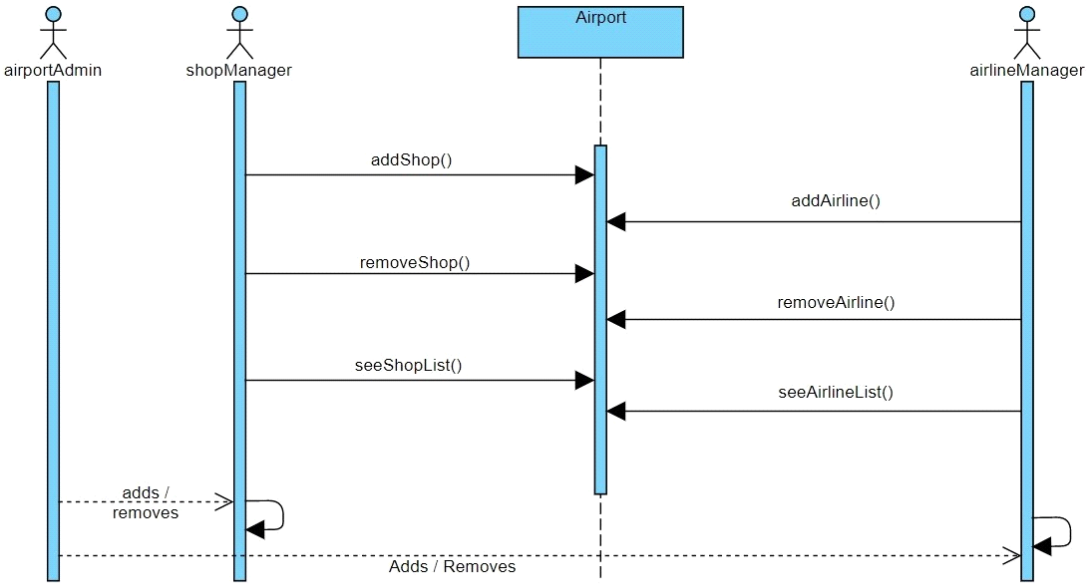


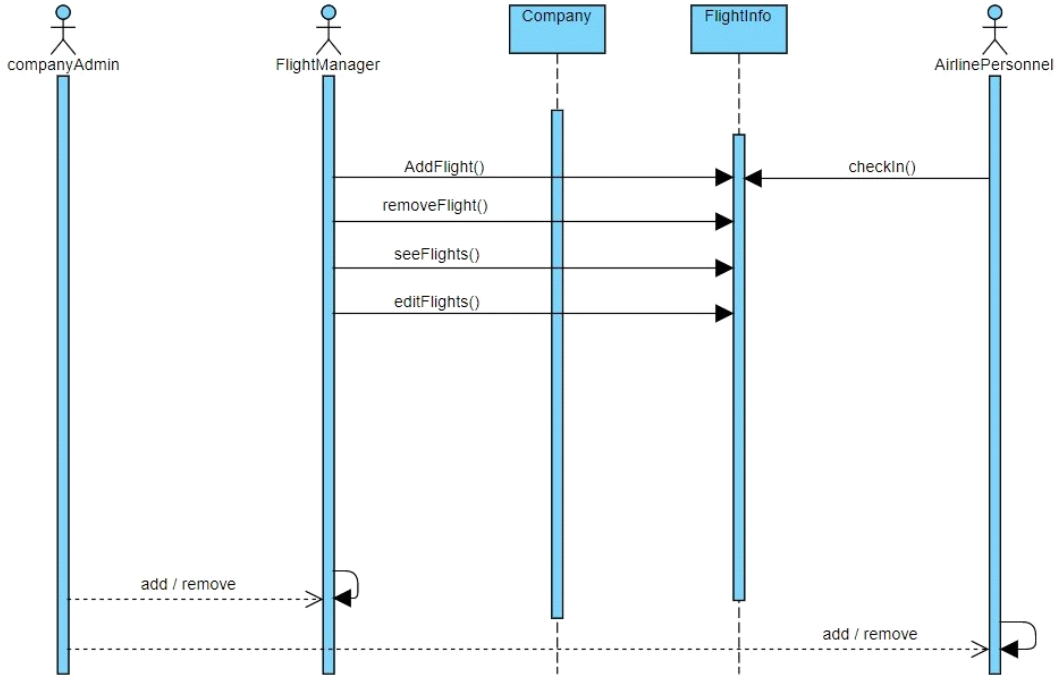
5.3.3. Client Class Diagram



5.4. Sequence Diagrams




airportAdmin
shopManager
Airport
airlineManager
addShop()
addAirline()
removeShop()
removeAirline()
seeShopList()
seeAirlineList()
adds / removes
Adds / Removes


companyAdmin
FlightManager
Company
FlightInfo
AirlinePersonnel
AddFlight()
checkIn()
removeFlight()
seeFlights()
editFlights()
add / remove
add / remove

6. TEST CASES

6.1. Airport Test Case

Test ID	Scenario	Expected Result
1	Hiring a new Airport Personnel	A new AirportPersonnel added to List
2	Login to System	Checks the userId and Password
3	Making a Selection from MENU	Jumps the selected Label
4	Adding couple new Airline Companies	New Airlines added to List
5	Displaying the Airline Companies	Listing the Airline Companies on the screen
6	Removing a new Airline Company	Firing the given Company from the Airport
7	Adding a new airport administrator	Airport administrator is added to the list
8	Removing an airport administrator	Airport administrator is removed from the list
9	Adding a new shop to the system	A shop is added to the list
10	Removing a shop from the system	The shop is removed from the list
11	Adding a shop manager	An instance of a shop manager is added to the list
12	Removing shop manager	Shop manager is removed from the list
13	Attempting to remove an object (i.e. Airport admin.) when the list is empty	An error message is shown to user

6.2. **Airline Test Cases**

Test ID	Scenario	Expected Result
01	Recuiting a new employee	A new employee added to company
02	Dissmissal of an employee	An existent employee removed from company
03	Accepting customer's flight request	Proper request of customer is accepted and ticket is created for this flight
04	Rejecting customer's flight request	Unproper request of customer is rejected
05	Destination adding for flight routes	New destination is added to destination container
06	Destination removing from flight routes	An existent destination is removed from destination container
07	New flight creation request from airport	New fligth is added to fligth container
08	Create UAID key	New uaid key is created for new ticket
09	Aricraft listing/sorting/removing/adding	Given processes are handled
10	Employee listing/sorting/removing/adding	Given processes are handled
11	Destination listing/sorting/removing/adding	Given processes are handled
12	Ticket listing/sorting/removing/adding	Given processes are handled
13	Flight listing/sorting/removing/adding	Given processes are handled

6.3. Client Test Case

Test ID	Scenario	Expected Result
01	Buying ticket	The customer receives the ticket.
02	Log in	The customer logs in to the system with her id and password.
03	Requesting ticket	The customer looks at the flight information and requests a ticket for the date s\he wishes. If there is a ticket for that date, s\he can buy it later.
04	Check-in	Registration of customers is done before the flight.
05	Cancel ticket	Customer's previously bought ticket is cancel.

Test Case Airline

Exceptions:

Date - Checks whether if day, month, year is valid. (positive number)

Error message: ” wrong day value entered!”

TicketData - Checks whether if customer or flight is null.

Error message: “Flight or customer or systemStorage cannot be null."

Flight – Checks whether flight in the ticket is avaliable

Error message: "Choosen filght's tickets are currently unavailable! "

RecruitPersonnel - Checks whether airlinePersonnel is null

Error message: “AirlinePersonnel cannot be null."

DismissPersonnel - Checks whether airlinePersonnel is null

Error message: “AirlinePersonnel cannot be null."

Test Case Airport

Exceptions:

DismissCustomer – Checks whether SSN is null or empty

Error message: "SSN cannot be null."

Error message: “SSN cannot be empty."

SetPassword – Checks whether password is null

Error message: "Password cannot be null."

AddAirportPersonnel – Checks whether airportPersonnel is null

Error message: "AirportPersonnel cannot be null."

RemoveAirportPersonnel – Checks whether airportPersonnel is null

Error message: "AirportPersonnel cannot be null."

AddShop – Checks whether shop is null

Error message: "Shop cannot be null."

RemoveShop – Checks whether shop is null

Error message: "Shop cannot be null."

AddAirline – Checks whether airline is null.

Error message: "Airline cannot be null."

RemoveAirline – Checks whether airline is null.

Error message: "Airline cannot be null."

SetCommissionRate – Checks whether Commission-Rate is positive.

Error message: "Commission-Rate cannot be negative number."

AddDestination – Checks whether destination is null

Error message: “Destination cannot be null."

RemoveDestination – Checks whether destination is null

Error message: “Destination cannot be null."

AuthenticateManager – Checks whether shop is null

Error message: “Shop cannot be null."

OpenShop - Checks whether authenticateManager(shop) is ok

Error message: “Authentication failed for shop manager."

CloseShop - Checks whether authenticateManager(shop) is ok

Error message: “Authentication failed for shop manager."

SearchWithPNR – Checks whether airportSystemStorage.getFlights().containsKey(PNR)) is false.

Error message: “Given PNR has no ticket, please check your information carefully!"

Customer – Checks whether SSN is used before.

Error message: "Given SSN is currently used!"