Airline Software System Report

Rey Garcia & Tori Grasso

Table of Contents

Introduction

Rey Garcia and Tori Grasso, have been given an assignment to create an airline software system. This software will make seating arrangements for a small airline between two cities, X and Y. There are twenty rows, arranged three and three on the flight. The first two rows are business select. Our software will have to deal with the complication of the three different kinds of passengers that come in on a first come first serve basis. Business travelers, tourist, and families. There is an option for passengers to generate a new seat if they do not like the first one they are given. They may choose to accept the new seat or not, but this may only happen a maximum of one time. After seats are assigned, there will be a satisfaction poll generated for customers to fill out. Then, only the manager will be able to access the results. Throughout this report we will outline the important steps it took us to create this software. We will highlight the requirements of the project, our design approach, our coding approach, how we tested our project, and how we plan to maintain our project. Our overall goal was to create an easy-to-use software for customers booking a flight and employees to use.

Requirements

When designing a software system, the first and most important step is the requirement stage. The requirement stage is crucial because it sets the stage for design and implementation. The requirement stage is also necessary to use in creating the test cases and test scenarios. As well as creating user training material or customer support activities. In the grand scheme of things, requirements provide a way to segment a large project. A more formal definition of software requirements is "descriptions of the services that a software system must provide and the constraints under which it must operate" (Software Requirements). Requirements are broken up into two different categories; functional and non-functional.

Functional

A functional requirement can be defined as, "Statements of services that the system should provide, how the system should react to particular inputs and how the system should behave in particular situations" (Software Requirements). For this specific project, our functional requirements include creating the user accounts for both the

passenger and manager, assigning customer seats, reassign seats if needed, producing a ticket for the customer, and producing a satisfactory report for the manager to review. All of these are functional requirements because they are crucial to the system performance.

Non-Functional

A non-functional requirement can be defined as “Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.” (Software Requirements). For this specific project, our non-functional requirements include a friendly GUI that makes it easy to use, checking to see if the login username and password is correct, and showing the already occupied seats on the flight. All of these are non-functional requirements because the system can perform without them. While it may seem as though they are a trivial addition, it must be noted that they add to the overall quality of the system. Without the functional requirements, the system would be mediocre and hard to use.

Analysis

After we have elicited all of the requirements, we needed to analyze and prioritize them. Requirement Analysis is “determining whether the stated requirements are unclear, incomplete, ambiguous, or contradictory, and then resolving these issues” (Requirement Analysis). We had to determine the priority of each requirement and then make a requirement prioritization list. This list consists of the requirement number, description, priority level, and category the requirement falls under. This is important to make as it will be the foundation of the process of building our software.

Req. #	Description	Priority level	Category
1	Create Account	5	UI
2	Login Customer	5	UI
3	Login Manager	5	UI
4	Assign flight seats	5	UI
5	Producing a Ticket for Customer	5	UI
6	Reassign seating	1	UI
7	Satisfaction Report	2	UI
8	User Friendly GUI	2	UI

Design

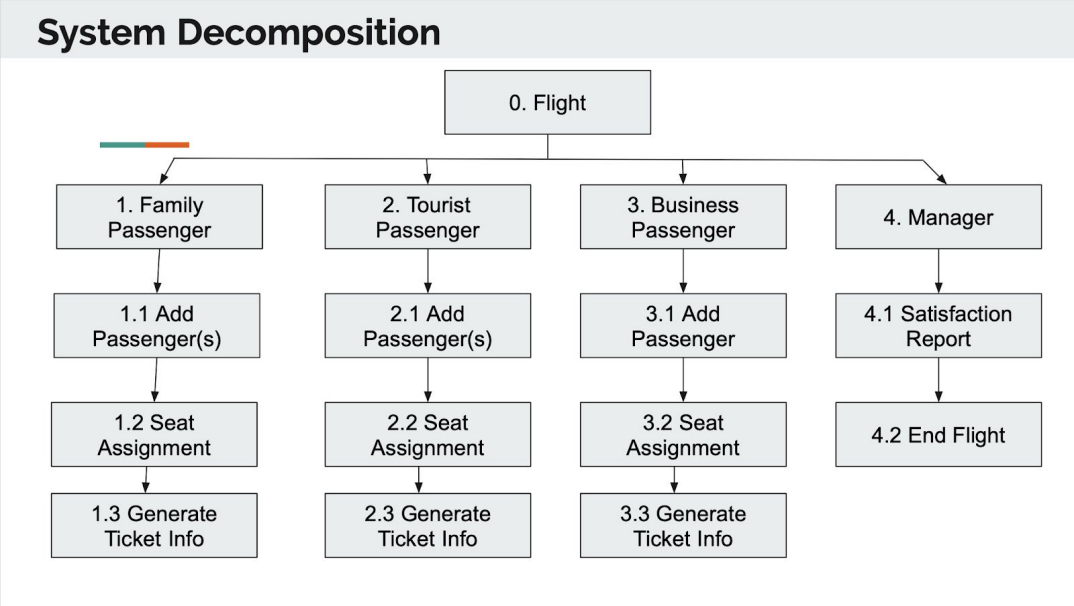
The next step in our project was to choose and implement a design tactic. “Software design deals with how the software is to be structured-that is, what its components are and how these components are related to each other” (Essentials of Software Engineering 130). This can be divided into two different phases; the architectural design phase and the detailed design phase.

Architectural Design

The architectural design phase is a high-level overview of the system and specifies its basic structure. For this airline software, we decided to use an event driven module. An event driven module is a high level design solution based on event dispatcher. It is an interactive program that allows the user to control the action of the program by using input devices such as clicking the mouse or typing on the keyboard. Customers will be given the option to enter information and the software will respond based on that information given. For example, the system will respond differently to each passenger type. A customer that is business select will get a different seat assignment than a family of four.

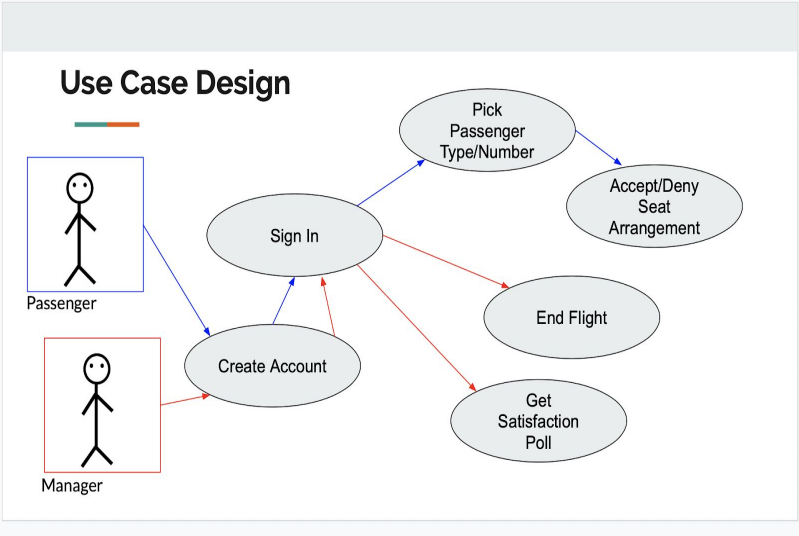
System Decomposition

The next step is to break down what each user will be able to do. First we must break up all of the customers.



The first customer type is a family. A family passenger will be allowed to add as many members of their family (not exceeding five), create a seat assignment, and generate ticket information. The second passenger type is a tourist and they will always travel in pairs, create a seat assignment, and generate their ticket information. The last passenger type is a business passenger who has the option to fly business select or not. This too will create a seat assignment and generate ticket information. The last part of the breakdown accounts for the manager

functions. The manager will generate a satisfaction report as well as end

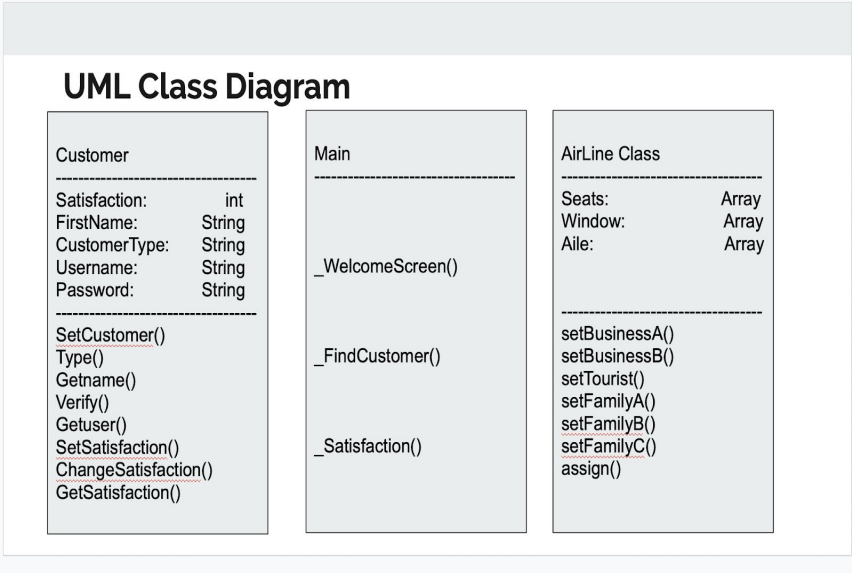


the flight.

You can see in this use case design how the system will be laid out. The passenger user will have its own specific path and within the passenger each passenger type will have its own similar but unique path. The manager's UI will look quite different than the passenger's view.

Class Diagram

The class diagram is an important step to plan out before any coding is done so that you know exactly what you need to do. We created three main classes; the customer class, main class, and the airline class.



The customer class deals with anything customer related. This would entail items such as the type of customer, name, username, password, and satisfaction. Some of the methods within this class perform functions like verifying the user, getting their satisfaction, and returning their necessary information to the airline class. The airline class has each case of a customer handled and assigns the seats. The main class incorporates everything and handles the GUI of what users see when they use the program. You can see in the class diagram the different methods and variables/variable types we intended to use.

Hi-Fi

We then needed to design the GUI, which is a pivotal part of the system because it is the only thing that the customer can see, as they do not need to see the back-end portion."The user interface (UI) is the part of the software most visible to the user and is one of the most important to get right" (Essentials of Software Engineering 152). This step is crucial to the design phase, as we needed to create a prototype for each window of our system. We wanted to make it very understandable for users to use and easy to look at. We did a basic pen and paper design (lo-fi) and then transferred that information into a google document using shapes and text boxes to get a feel for what the system would look like (see appendix). There is a variety of different windows that we needed to prototype and those include the welcome window, creating account window, customer sign in window, manager sign in window, seat assignment window, ticket generator window, and the satisfaction result window. We wanted to be able to move from window to window and keep it as simple as possible. We did not want to bombard the customer with too much information in one window to make them feel overwhelmed. At the same time we did not want to waste time and

resources on making too many windows. We had to decide on a happy medium that would have just the right amount of information on each window.

Coding

The coding stage of software engineering is a crucial step and is directly related to the design phase. Several tasks involve both the backend and frontend aspects of the application. Due to the event driven structure used, there was a lot of swapping between GUI coding and backend coding. A few helper methods were used to lighten the load on the main airline function. 3 classes where used. (1) the seating class was in charge of assigning a seat to the current customer that was passed in. It takes into account what type they are and assigns accordingly. Within it is a reading function for clearing out past seats that were assigned and gives new one. It also hold the entire seating chart for the flight with open slots and taken slots. (2) The customer class and was used to create each customer. It holds their information to be able to use the flight. (3) The main function takes both classes and brings them into one task. It hold each customer with their seating assignments as well as having the

authority to give each customer a seat. It holds all the code for GUI from the welcome screen to the exit flight screen. Since there would be multiple paths to take, it is necessary to include if-statements within if-statements that would exist inside the single loop until the program is finished.

Implementation

For the implementation process, we used the pair programming method as it is what made the most sense for a group of two. Pair programming ensures "...that all production code is written with two programmers working at the same machine or facility. This is the revision step taken to the extreme" (Essentials of Software Engineering 88). By executing the pair programming method, it allowed us to more carefully write our code so that when we needed to revise our code there were few mistakes. Another thing we did was follow SOFA. SOFA stands for short, one thing, few arguments, and abstraction. By using SOFA, our goal was to have little code smells as possible. Another important characteristic that we followed to make our code the best quality is camel casing variables, like customerType or WelcomeScreen. This made the code easy to read and we were able to find what we were looking for faster. In addition to

that, if anyone else were to read our code they would not have a hard time (see appendix).

Testing

Testing is a vital part of the software engineering process because this is where you can see if your code actually works like it is supposed to. For software, we focus more on the validation of the program. In other words, did we build the right thing? Is it what the customer wanted? Are the specifications correct? We used some of the more informal methods of testing in the process of writing the code. For example, we performed black-box testing, which is “testing based solely on specifications, without looking at the code”(Essentials of Software Engineering 204). Another method we used was mutation testing, so we checked if there was an error in the code and then is there a break in the test. Our main testing approach was to use the sandwich integration on all of the methods. This proved to be very useful, especially since there were several methods to be tested. We used separate text files that had separate combinations of seats, customers with already set satisfaction results, and a full or almost

full flight scenarios that we used to test the different possible outcomes of a flight.

Maintenance

We have completed the project but, as many people know, there is always room for improvement. From the first time we presented this project and performed a demo, we had already made a couple of improvements. The first being that you are able to see what seats are not available when you are being given your own options for seating. We also incorporated a satisfaction survey that asks the customers for their opinions, rather than assuming their satisfaction score. We have also been working on encrypting the passwords for the users for security reasons, but unfortunately ran into some problems. If we had more time to implement this change we would make sure the password is hidden when it is being typed and use a database instead of a file to store user information. We have no experience with databases so we just used a file to store this information. In the future we would create a database and hash the passwords in their as well. Another thing we would have liked to improve is the overall GUI and design of the system

so it could run as smooth as possible. Lastly, we would obviously maintain the system when any problems are to arrive.

Conclusion

Overall, we were able to produce a functioning airline software system that produces a seat and ticket for the three different types of customers; business, traveler, and family. The first challenge we had to overcome was deciding what the requirements were. Once we listed all of our requirements and determined which were functional and which were non-functional, we could prioritize them. Building off of the prioritization list we then needed to come up with a design plan. Within our design we needed to think of the kind of architectural design pattern we wanted to use. We decided that an event driven module was the best way to approach the problem. Then, we needed to create a system decomposition, class diagram, and a lo-fi/hi-fi sketch of what our GUI would look like. With all of these tools we were able to start the implementation process in which we decided to pair program. Coding in a pair may have taken longer than normal but it decreased the revision process. We then needed to test our code in which we used sandwich

integration to test different methods. In conclusion, we created the best airline software we could based on what the customer wanted.

References

"Requirement Analysis." *Software Development Process – Activities and Steps*, pp. 83–93., doi:10.1007/978-1-4612-3980-2_11.

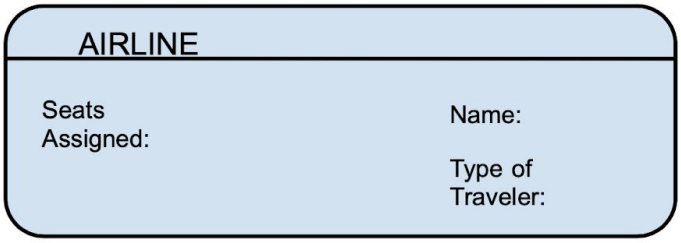
"Software Requirements." *Software Requirements*, 2004, doi:10.1109/9781118156674.ch3.

Tsui, Frank F., et al. *Essentials of Software Engineering / Frank Tsui, Orlando Karam and Barbara Bernal*. Jones & Bartlett Learning, 2014.

Appendix

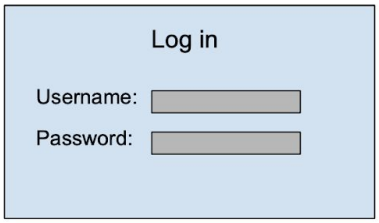
Hi-Fi

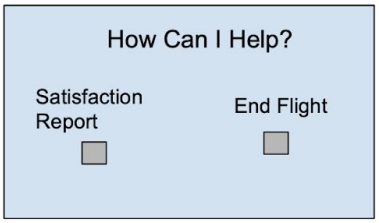
Ticket Generator



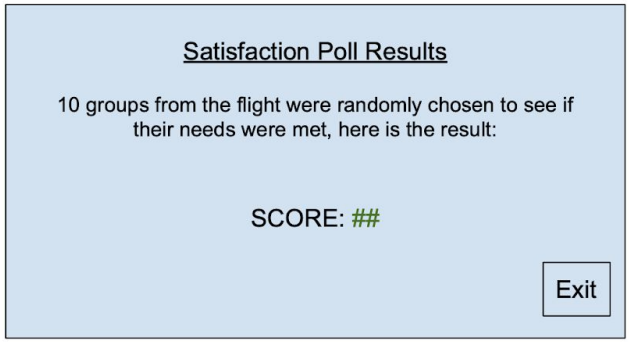
Thank you and have a great flight!

Manager Sign In

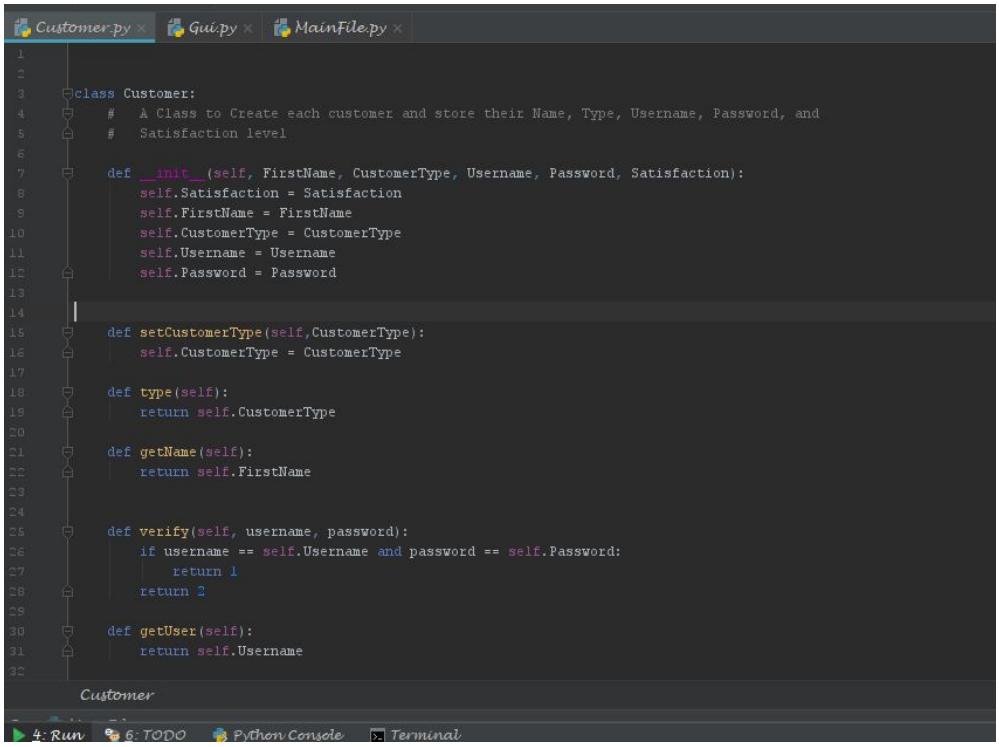




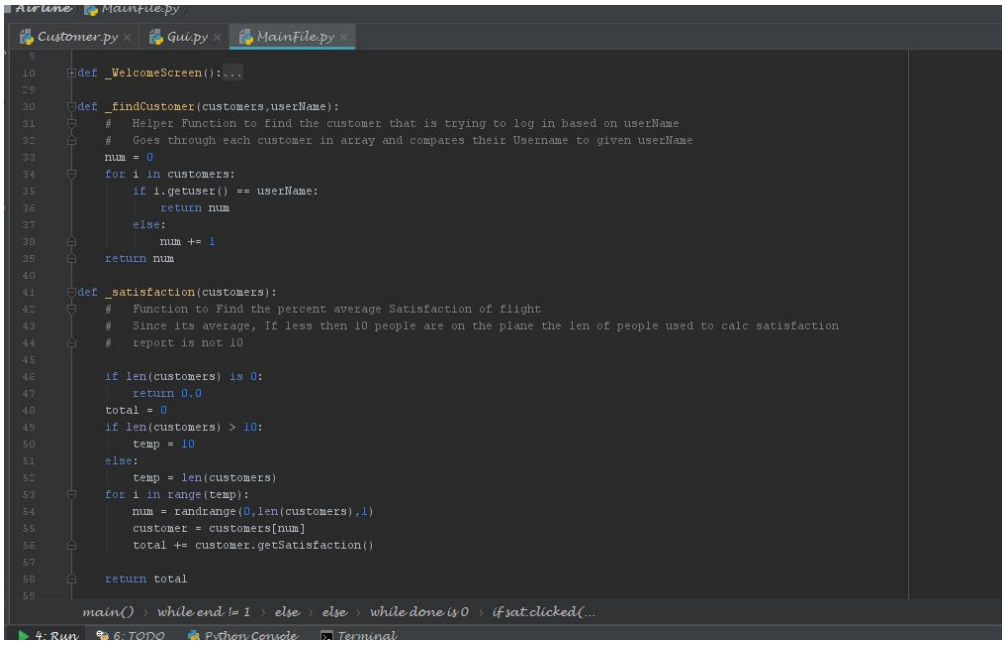
Satisfaction Results



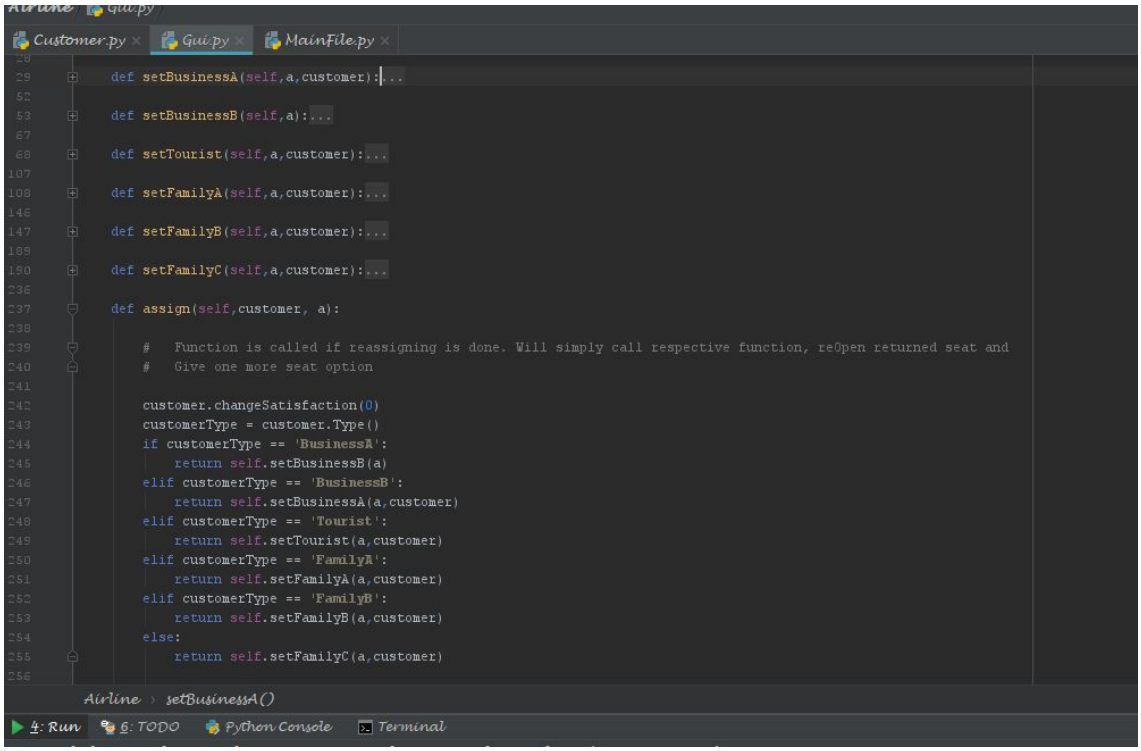
Source Code



```
class Customer:
    #   A Class to Create each customer and store their Name, Type, Username, Password, and
    #   Satisfaction level

    def __init__(self, FirstName, CustomerType, Username, Password, Satisfaction):
        self.Satisfaction = Satisfaction
        self.FirstName = FirstName
        self.CustomerType = CustomerType
        self.Username = Username
        self.Password = Password


    def setCustomerType(self,CustomerType):
        self.CustomerType = CustomerType

    def type(self):
        return self.CustomerType

    def getName(self):
        return self.FirstName


    def verify(self, username, password):
        if username == self.Username and password == self.Password:
            return 1
        return 2

    def getUser(self):
        return self.Username
```



```
def _WelcomeScreen():...

def _findCustomer(customers,userName):
    #   Helper Function to find the customer that is trying to log in based on userName
    #   Goes through each customer in array and compares their Username to given userName
    num = 0
    for i in customers:
        if i.getuser() == userName:
            return num
        else:
            num += 1
    return num

def _satisfaction(customers):
    #   Function to Find the percent average Satisfaction of flight
    #   Since its average, If less then 10 people are on the plane the len of people used to calc satisfaction
    #   report is not 10

    if len(customers) is 0:
        return 0.0
    total = 0
    if len(customers) > 10:
        temp = 10
    else:
        temp = len(customers)
    for i in range(temp):
        num = randrange(0,len(customers),1)
        customer = customers[num]
        total += customer.getSatisfaction()

    return total
```



```
    def setBusinessA(self,a,customer):...

    def setBusinessB(self,a):...

    def setTourist(self,a,customer):...

    def setFamilyA(self,a,customer):...

    def setFamilyB(self,a,customer):...

    def setFamilyC(self,a,customer):...

    def assign(self,customer, a):

        #   Function is called if reassigning is done. Will simply call respective function, reOpen returned seat and
        #   Give one more seat option

        customer.changeSatisfaction(0)
        customerType = customer.Type()
        if customerType == 'BusinessA':
            return self.setBusinessB(a)
        elif customerType == 'BusinessB':
            return self.setBusinessA(a,customer)
        elif customerType == 'Tourist':
            return self.setTourist(a,customer)
        elif customerType == 'FamilyA':
            return self.setFamilyA(a,customer)
        elif customerType == 'FamilyB':
            return self.setFamilyB(a,customer)
        else:
            return self.setFamilyC(a,customer)
```

Testing

```
def __init__(self):

    #   All the seats and the spots of Window Seats as well

    self.Windows = [0,5,6,11,12,17,18,23,24,29,30,35,36,41,
    self.aile = [2,3,8,9,14,15,20,21,26,27,32,33,38,39,44,45
    self.Seats = [0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0,0,0,0,0,0,0,0,
                  0,0,0,0,0,0]
```

```
#   Isn't 0, Seat Assignments will stop
end = 0

airline = Airline()
#'''Fake customer Account'''

#z = Customer('r','','r','r',0)
#x = Customer('e','','e','e',0)
#y = Customer('y','','y','y',0)
#w = Customer('n','','n','n',0)
#v = Customer('a','','a','a',0)
#u = Customer('l','','l','l',0)
#customers.append(z),customers.append(y),customers.append(x),customers.append(w),customers.append(v),customers.append(u)

while end != 1:
    WelcomeWin, TopButton, BottomButton = _WelcomeScreen()
    #   If bottom button is clicked you'll need to create an account and the use that new account to log into
    #   the program
    if BottomButton.clicked(WelcomeWin.getMouse()) is True:...
    #   Need to clarify if you're the manager so you have manager commands or if your
main()
```