Airline Ticket System

PREPARED BY
Dara O'Sullivan - 20080494
Aishling McPartland - 20079978

Assignment 2

Software Engineering Practice

Computer Forensics Year 2

Introduction

This Airline Ticket System is an implementation of a general Airline Ticketing Website such as Ryainr or Delta Airlines, which helps customers to search for available flights to different countries/cities. The project can be seen set into two departments, the administration, and the ticket reservation department. In the ticket reservation department customers are using the website as they seek flights. The customers can filter in order to regulate the size of the results through country, price, and number of stopovers. Once the customer has chosen a flight with corresponding dates and their payment has been processed they then receive a confirmation email and later on if necessary they can make adjustments to booked tickets. In the administration department, the admin can use this system to modify flights, update any cancellations.

During the duration of this project, we first present the Use Cases which describe the possible scenarios that can take place in the system. The Use Cases are then followed by the Class Diagram, which presents the different classes that the system has and their attributes, methods and the relationships among objects. The Sequence Diagram will follow and gives a breakdown of the steps between each use case in sequential order. After this, we have some smaller diagrams such as the State Diagrams and CRC cards which furthermore reinforces the explanation of the system.

Project Assumptions

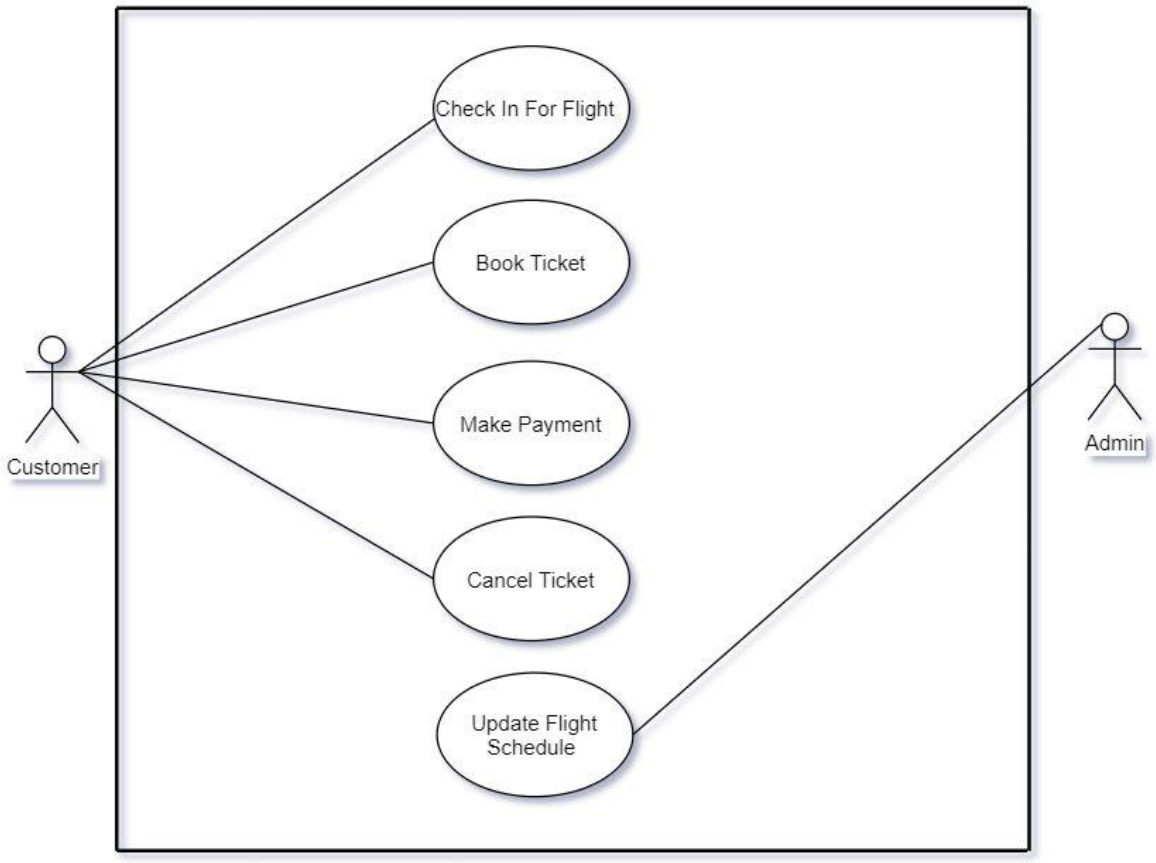
One must assume that the customer has access to their flight information. The customer should not have the same access as an administrator. Any changes that are made to flights or booking information are automatically updated and the people involved are notified of such changes. The user should have full access to the website and should be able to freely browse.

Requirements for the Flight Reservation System

1. The purpose of this is to allow airline tickets to be bought over the internet and can browse the website.
2. Each customer specifies the airport they wish to depart from, a destination airport, and dates for flights leaving and return flights.
3. The customer reserves one departing flight and one returning flight from the options presented by the system.
4. The customer may buy tickets for one or more passengers.
5. Due to the worry of double booking, there should be no more tickets that can be sold for a flight than there are seats on the plane.
6. Each customer is assigned to a specific seat for example 7F.
7. The system calculates the total cost of the tickets by adding the cost of ticket prices (single/return) and if there is any extra luggage.
8. The customer can still change flights if they are unhappy with the price.
9. After a customer has bought a ticket, they will receive a confirmation via email.

10. In this email, it contains a confirmation, as well as an ID so they can access their boarding pass by checking in.
11. The customer can cancel their ticket and receive a refund, as long as it's within 30 days of purchase.
12. The system administrator should have access to the system functions and permissions needed to successfully enact the following use case, Update Flight Schedule.

Use Cases



The relationships between and among the actors and use cases

- Customer Entity: The use cases of the Customer Entity are Check-in For Flight, Book Ticket, Make Payment and Cancel Ticket.
- Admin Entity: The use case of the Admin Entity is Update Flight Schedule.

Stakeholder and people of interest

1. The Customer: This is the person (or people) who wish to be able to browse the website and filter with their specific requirements and needs. They may wish to purchase a ticket and later on they may wish to change flight details, check-in, etc.
2. The Administrator: This is the person (or people) who ensures that all aspects of keeping website content and design fresh, backed up, and fully functional. They may also want to cancel flights or change flight information.
3. The Owners of the airline due to the fact they own the business and want it to succeed.
4. The banks that want to be able to account for every payment made for the booked tickets that are processed and authorized.

UC1 - Check-in For Flight

- **Primary actor**: Customer.
- **Description:** This use case describes how a user can check-in online.
- **Stakeholders and interests:** Customer, Owner, and Admin.
- **Level**: User goal
- **Trigger:** Users' flight is in less than 24 hours.
- **Precondition:** User has not checked in yet.
- **Postcondition:** User has then checked in.
- **Success Guarantee**: User is checked in, receives a seat/flight number and a boarding pass.
- **Minimal Guarantee:** User has the ability to access the website.
- **Main success scenario:**
    1. The system validates her ID.
    2. The system validates users' tickets.
    3. The system checks flight updates such as delays.
    4. The system allows the user to check-in.
- **Fail Scenario:**

    <u>FS1 - Invalid ID entered</u>

    1. At step 1, the system returns an invalid ID message.
    2. System returns user to step 1.

    <u>FS2 - Invalid Tickets</u>

    1. At step 2, the system returns that the ticket is invalid.
    2. System returns user to step 1.

UC2 - Book Ticket

- **Primary actor**: Customer.
- **Description:** This use case describes how a user can book a ticket.
- **Stakeholders and interests:** Customer, Owner, and Admin.
- **Level:** User goal
- **Trigger:** User needs a flight.
- **Precondition:** User has a valid passport.
- **Postcondition:** User has then booked their ticket.
- **Success Guarantee:** Users have a seat (or several seats) reserved on a particular flight and the number of available seats on the flight is reduced by the number of seats reserved.
- **Minimal Guarantee:** The user has the ability to access the website and can browse all available tickets.
- **Main success scenario:**
    1. User fills in the details of their departing and arriving flight locations and suitable times.
    2. System processes the data and displays available flights.
    3. User chooses which flights they would like to book and seat or seats.
    4. System checks that the chosen flight still has available seats and books the seat or seats selected.
    5. The user enters their private information including name, passport number, etc.
    6. System sends a confirmation email.
    7. System marks the seats as booked and reduces available seats by how many seats booked.
- **Fail Scenarios:**

    <u>FS1 - No flights match the criteria.</u>

    1. At step 1, the system checks to see for the availability of seats on the departing and arriving flights. There are no more seats available.
    2. The system then sends the user back to the homepage where they can search for alternative flights returning to step 1

    <u>FS2 - No seats left on the flight chosen.</u>

    1. At step 4, the system cannot find any seats left on the flight chosen.
    2. System returns to step one and allows the customer to change the criteria they have entered to try to find available seats on alternative flights.

    <u>FS3 - User does not receive a confirmation email</u>

    1. System books the seat but does not send a confirmation email.
    2. User contacts the airline support team.

FS4 - Double booking issue

1. System did not reduce available seats by how many seats booked.
2. There is now a double-booking issue.
3. User is refunded the price of the ticket.

- **Alternative Scenario:**

AS1 - Information given was incomplete.

1. At step 5, system requests missing information from user.
2. User then supplies missing information.
3. User can continue to step 6.

UC3 - Make Payment

- **Primary actor**: Customer.
- **Description:** This use case describes how a user makes the payment.
- **Stakeholders and interests:** Customer, Owner, Admin, and Banks.
- **Level:** User goal
- **Trigger:** User needs to pay for a flight.
- **Precondition:** User has booked a flight.
- **Postcondition:** User then pays for the booked flight.
- **Success Guarantee:** User has successfully paid for their ticket, and now has a confirmed seat on the flight chosen.
- **Minimal Guarantee:** The user has the ability to access the website, browse with ease and is able to book a ticket.
- **Main success scenario:**
    1. User enters all credit or debit card information in.
    2. System then ensures that the payment is authorized by the bank.
    3. Once the payment is made, the booking is then set.
- **Fail Scenario:**

FS1 - The payment is not made

1. At step 2, system reports failure to user due to insufficient funds.
2. System then makes the previously booked seat available.

FS2 - Authorization failure

1. At step 2, system reports the authorization failure to the user.
2. The system returns to step 1 and allows the user to re-enter their payment information and try again.

- **Alternative Scenario:**

  AS1 - User enters the incorrect card information

  1. At step 1, system presents a message to the user that the card information that has been entered is incorrect.
  2. User re-enters their information.

UC4 - Cancel Ticket

- **Primary actor:** Customer or the Admin
- **Description:** This use case describes how a user can cancel a ticket.
- **Stakeholders and interests:** Customer, Admin, Owner, and Banks.
- **Level:** User goal
- **Trigger:** User needs to cancel a ticket for a flight.
- **Precondition:** User has bought a ticket for a flight.
- **Postcondition:** User cancels the flight they previously booked.
- **Success guarantee:** Users no longer have a seat reserved on a particular flight and the number of available seats on the flight is increased by the number of cancelled booked seats.
- **Minimal Guarantee:** The user has booked and paid for a ticket or several tickets.
- **Main success scenario:**

  1. User selects a link to cancel their ticket.
  2. The system places the seat(s) as currently available and increases available seats by the number of seats that were previously booked.
  3. The system obtains and verifies bank details of customers and refunds the price of the ticket.

- **Fail Scenario:**

  FS1 - The refund cannot be made

  1. At step 3, the user cancels their ticket and the seat(s) is now available for a new customer.
  2. Refunds cannot be processed because they have passed the grace period.

UC5 - Update Flight Schedule

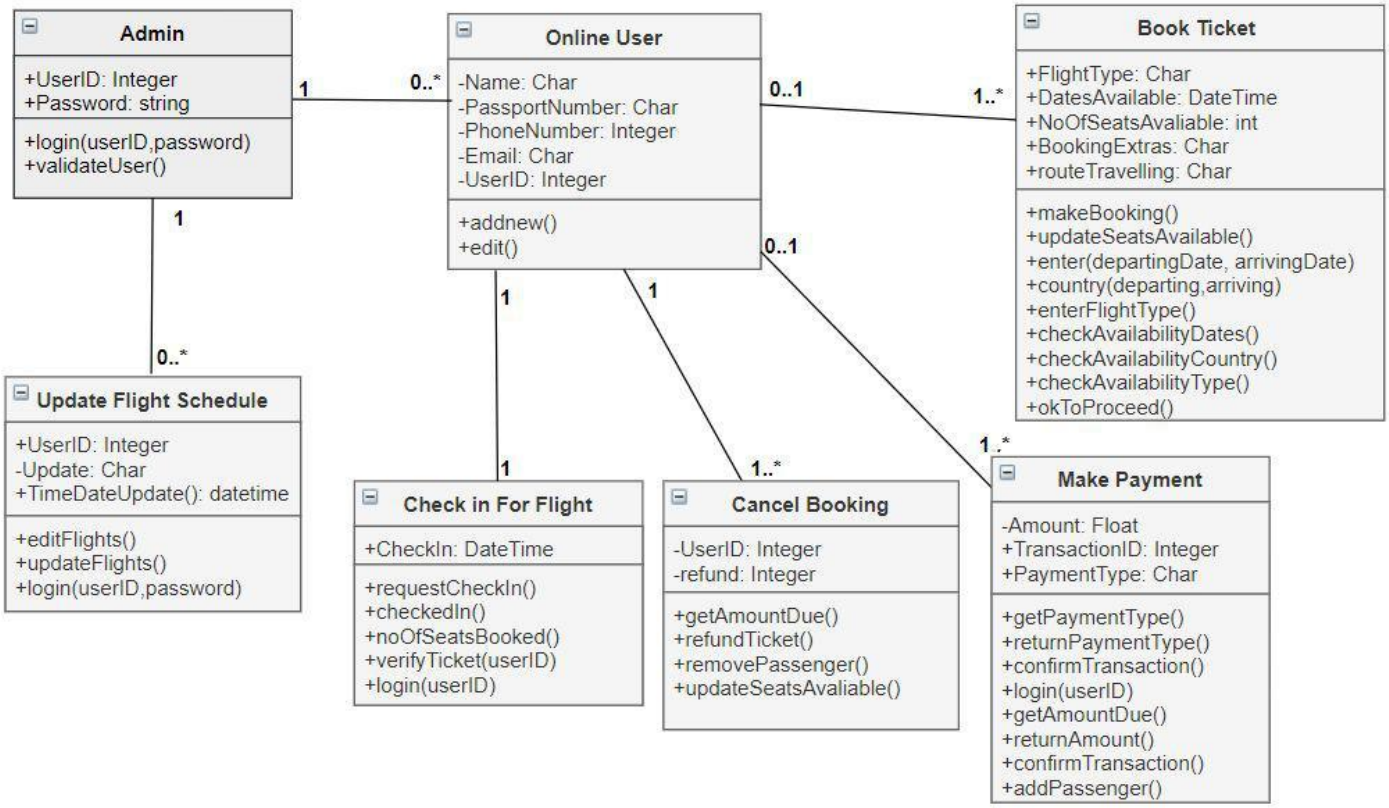
- **Primary actor:** Admin.
- **Description:** This use case describes how the Admin wishes to update the flight schedule such as delays in schedule or cancellations.
- **Stakeholders and interests:** Customer, Admin, Banks, and Airline.

- **Level:** Admin goal.
- **Trigger:** Admin wishes to update the flight schedule.
- **Precondition:** That the user is an administrator and has an account.
- **Postcondition:** Admin can successfully change flight details.
- **Success Guarantee:** The admin is able to enter in their ID, and are able to successfully update the flight schedule.
- **Minimal Guarantee:** The user has access to their account.
- **Main success scenario:**
    1. Admin is verified via ID and password and is able to edit the flight schedule.
    2. The admin makes changes and saves them.
    3. System is then updated with the new changes made.
- **Fail Scenario:**

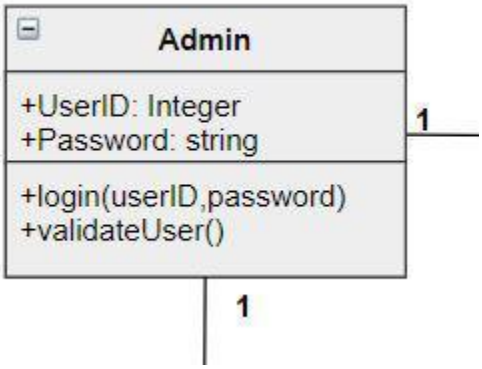
    <u>FS1 - The changes are not updated</u>

    1. At step 3, system does not update when the changes made.
    2. This leads to issues with the flight schedules.

Class Diagram

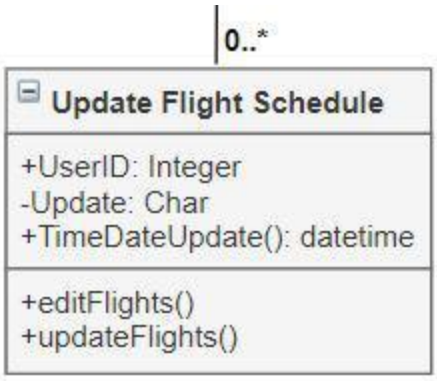


The class diagram above illustrates which classes are required for this project to run successfully, how each class works together. Below I have a detailed explanation of each class.

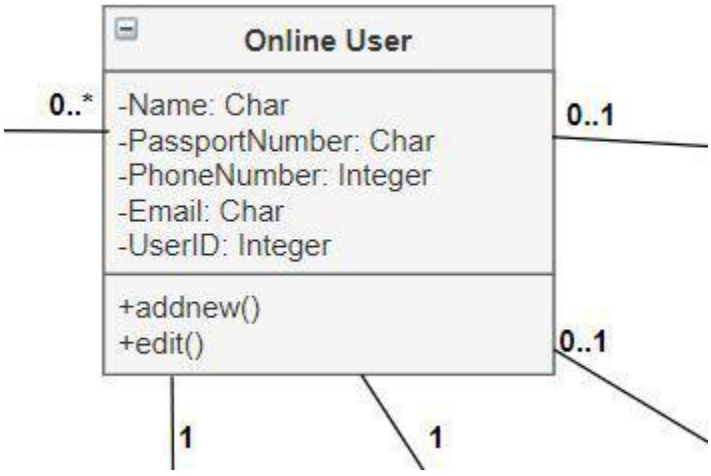


This is the Admin class; it has the multiplicity of 1 and is connected to the Customer class and the Update Flight Schedule class. The method it contains is login(userID,password) which takes in the

userID and password in order to login the user and then the validateUser(), which allows the Admin to be validated.



The next class that we're detailing is the Update Flight Schedule. It has a multiplicity of 0..* meaning there are 0 or many updates to the flight schedule and is connected to the Admin class as they manage it. The first method that the class contains is editFlights(), which allows the Admin to update the flight schedule. The next method it contains is the updateFlights(), which allows the admin to update the flight schedule such as delays, cancellations, and other issues concerning the flight schedule and these changes will be put into place and the system will be updated.

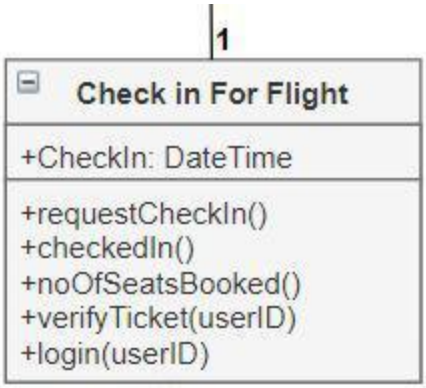


The Online User has multiplicities to the following classes:

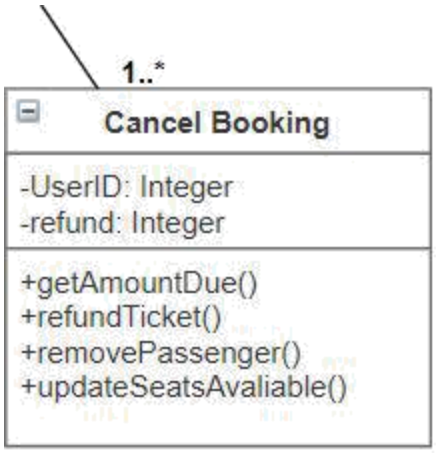
- Admin: "0..*" - This means that there can be 0 or many online users.
- Check-in For Flight: "1" - This means there is one user per check-in.
- Cancel Booking: "1" - This means that there is one user per cancellation.
- Make Payment: "0..1" - This means that there are 0 or 1 users per payment, as a user may decide to leave the website and not make the payment.
- Make Booking: "0..1" - This means that there are 0 or 1 users per booking as a user if unhappy can leave the website and carry forward with the booking.

14

The Online User has two methods, addnew() which allows their information to be saved and edit() which allows them to change their information entered such as phone number. The Online User also stores various personal details required for a booking. It takes the customer's name and passport number for the booking, the phone number, and email in order to send them a confirmation email which contains the user ID. The userID which allows the user to log in when needed.

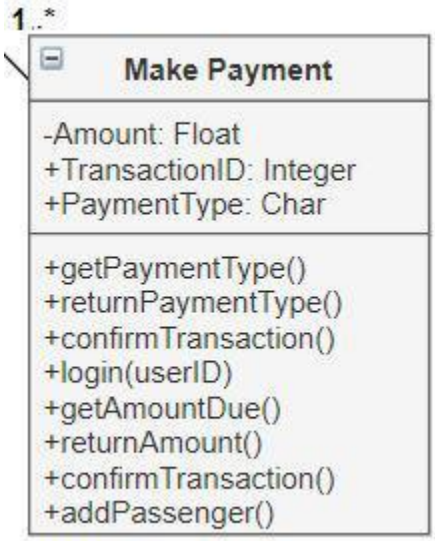


The next class we have is the Check-in For Flight class, which has a multiplicity of 1 meaning there can be one check-in per user at a time, there cannot be 2 users checking into one ticket for one flight. The first method contained is requestCheckIn() which sends a request to check in the user. It has the checkedIn() method, allowing the user to be checked in. The next method is noOfSeatsBooked(), this method keeps track of who has checked in and if a person does not check-in their seat can be bought, when they are checked in, they now have a seat on that flight. The verifyTicket(userID) allows the system to verify the ticket and the login(userID) allows for the user to login and check-in.



The Cancel Booking class has a multiplicity of 1..* meaning that there can be 1 or many cancellations because there must be at least one user because it's a cancellation, there must be at least one user buying the ticket. It holds the user ID and information for refunds. This class has the following methods:

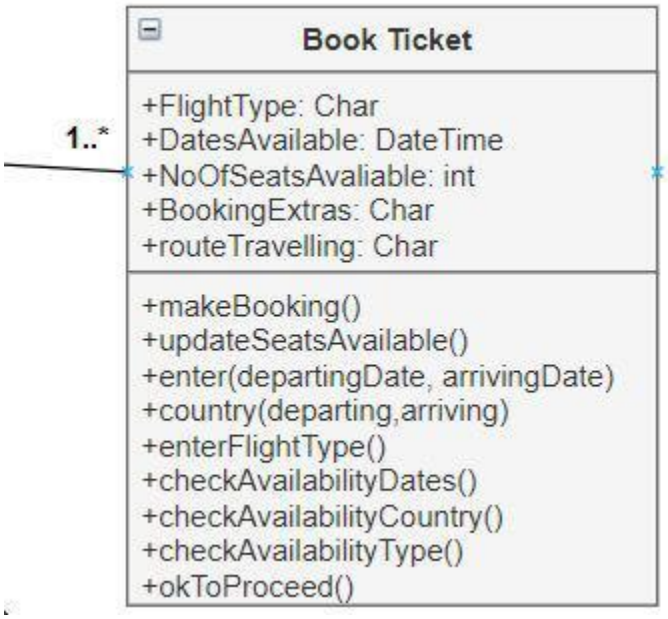
- getAmountDue(): Here the method extracts the data to recover how much is due for the refund.
- refundTicket(): Here the method is used to provide the user with a refund.
- removePassenger(): This method removes the user from the system.
- updateSeatsAvaliable(): This method allows the seat that the user has canceled can now become available for purchase.



The Make Payment class has the multiplicity of 1..*, meaning that there is at least 1 user making a payment and can be more is they buy multiple tickets with alternating cards. It holds information such as the amount due, the transaction ID and the payment type.

The following is the types of methods that are contained in the class:

- getPaymentType(): This method takes in the payment type which could be credit, debit or American express.
- returnPaymentType(): This returns what kind of payment type the user will be making.
- confirmTransaction(): This method ensures that the payment has gone through and the user receives a confirmation that the purchase has gone through. It also allows the banks to verify the transaction.
- login(userID): This allows the system to collect the flight data as they log in.
- getAmountDue(): This method will present the user with the amount due.
- returnAmount(): The system now returns the amount due to the user.
- addPassenger(): This adds the passenger to the flight as they have paid so now their booking is confirmed.



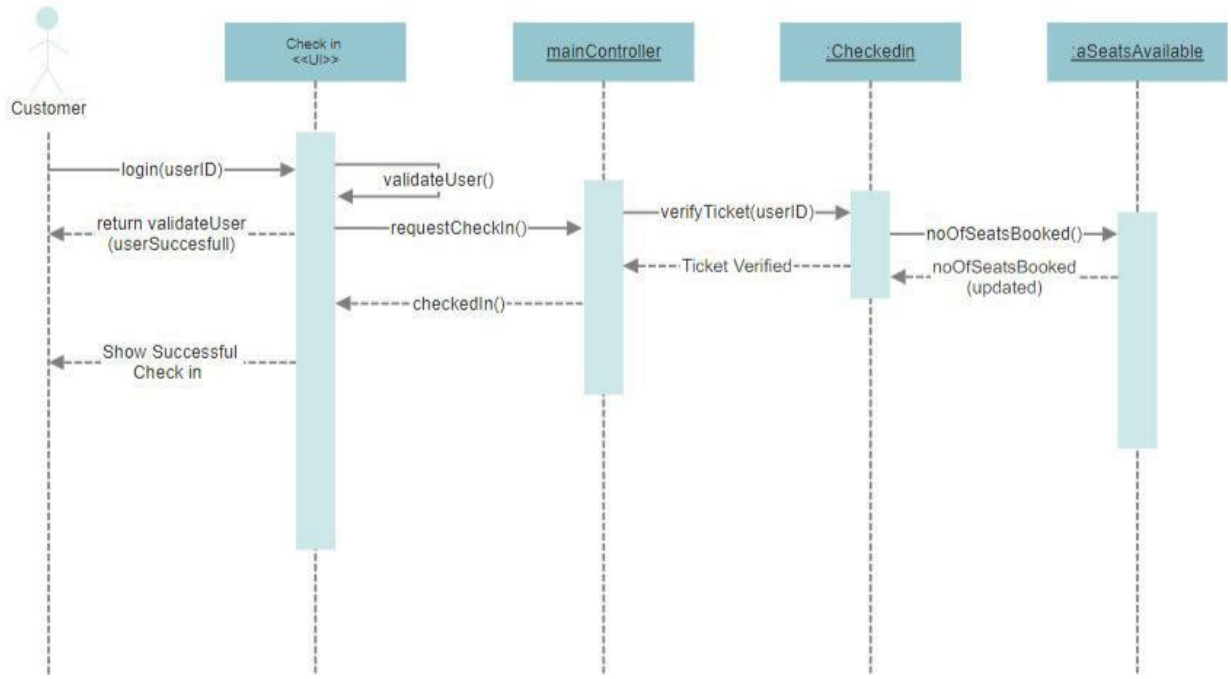
The Book Ticket class has a multiplicity of 1..* meaning that there has to be at least one person or more. The booking class is in charge of presenting a menu to the user and allows them to filter by dates available. The class holds information such as the flight type, the dates available and the number of seats available. As well as any booking extras such as extra luggage. The methods it contains is the following:

- makeBooking(): This is a method that actually creates the booking.

- updateSeatsAvailable(): This method updates the seats available as people book seats, they should be taken off the menu.
- enter(departingDate, arrivingDate): This method allows the user to enter in their departing date and arriving date.
- country(departing, arriving): This allows the user to enter in the country they are departing and the country they wish to arrive in.
- Enter FlightType(): This method returns the type of flight that the user has selected (Economy, Business or First Class).
- checkAvailableDates(): This ensures that the dates chosen are available.
- checkAvailableCountry(): This ensures that the country for departing and arriving flights chosen are available.
- checkAvailableType(): This ensures that the type of flight chosen is available.
- okToProceed(): This lets the user know they are okay to proceed.

Sequence Diagrams

SD 1 - Checking in For Flight

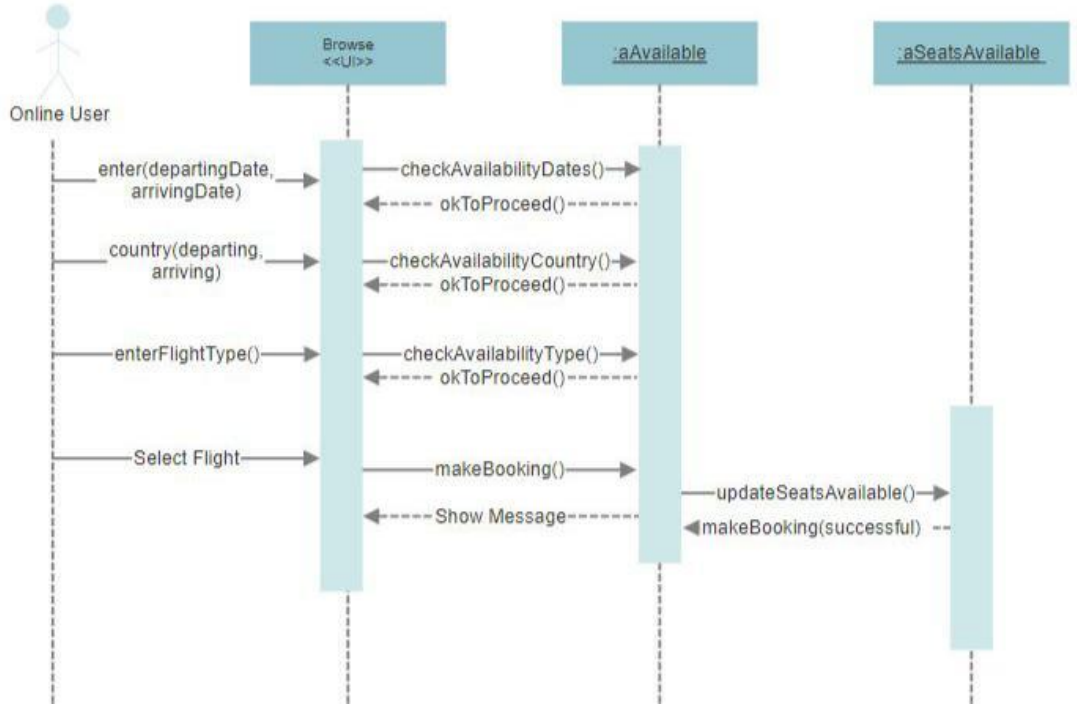


Here the user checks into their flight online. Users first enter their user ID using the method login(userID). The system then checks the user details and returns them if the user is validated.

When the ID is validated with the method validateUser(), then the user is then allowed to check-in for their flight online with requestCheckIn() and is ensures the ticket is verified verifyTicket(userID) and the ticket is then verified they can be checked in with checkedIn(). The number of seats can be updated because they have now been checked with noOfSeatsBooked() and are then updated noOfSeatsBooked(updated).

Here we can see there is the Check-in user interface that allows the user to see the online platform, the controller acts as a mediator between the UI and the system. We can also see that the Checkedin is an object of a class, the check-in for flight class that we have in the class diagram. It verifies the user's ticket. The last object of a class isaSeatsAvailablewhich updates the seats available because if a user does not check in someone else can purchase the last minute.

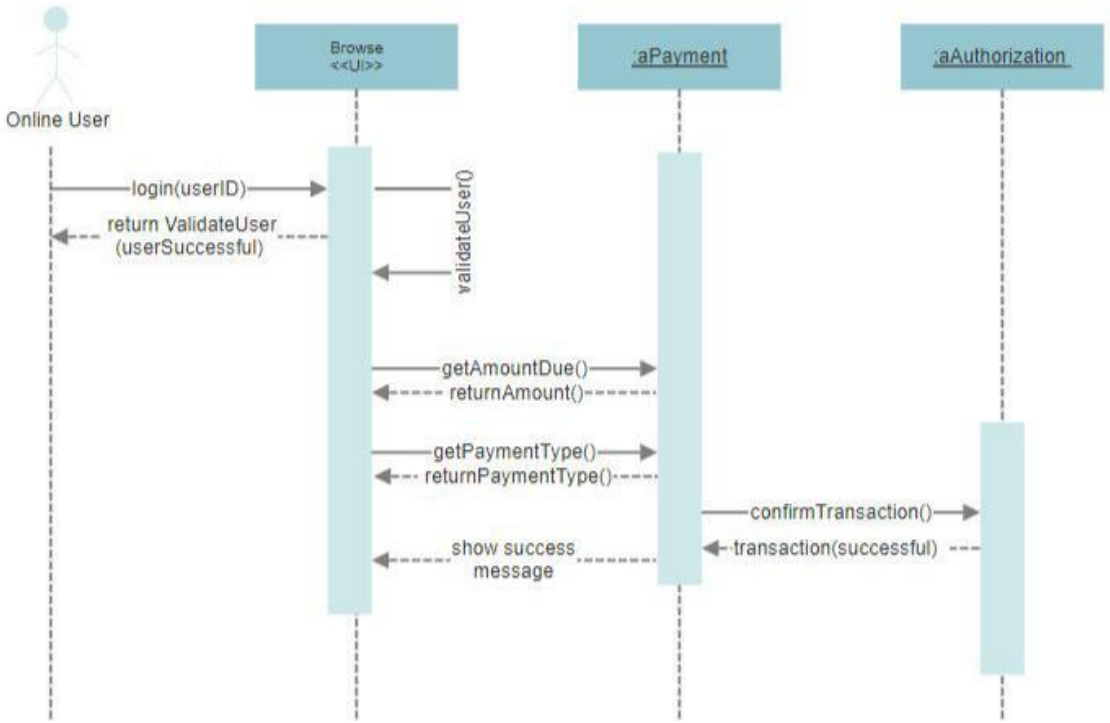
SD 2 - Book Ticket



In the second sequence diagram, the steps for booking a ticket are explained. The user enters in their preferences such as country of destination and country of departure with country(departing,arriving, the dates/times with enter(departingDate, arrivingDate)and flight type (such as Economy, Business or First Class) with enterFlightType().Each method is checked with either checkAvailabilityDates(), checkAvailabilityCountry() orcheckAvailabilityType()which ensures the dates, country or type of

flight chosen are available, if available it returns the method okToProceed()allowing the user to proceed. The user then chooses to makeBooking()which allows the user to make the booking. The seats available are then updated with updateSeatsAvailable()and are returned with makeBooking(successful) which then displays the successful booking to the user.

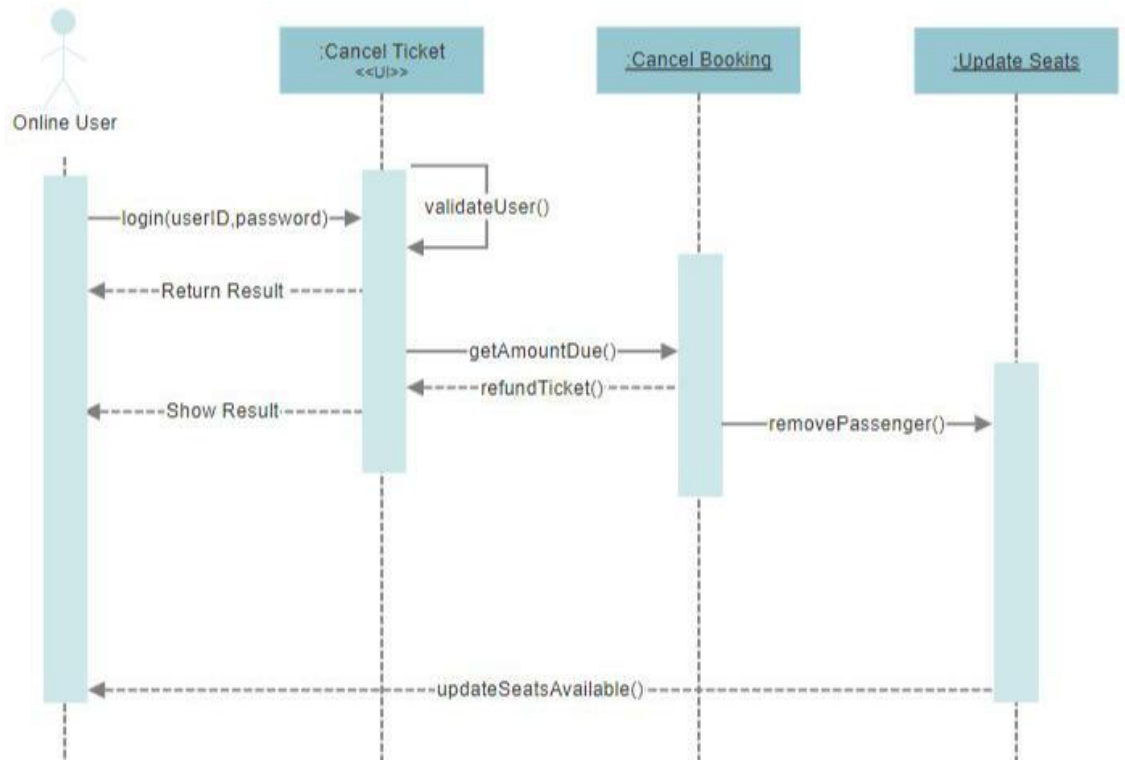
SD 3 - Make Payment



In this sequence diagram, the user attempts to make payment for their booking. Firstly, they login with login(userID). It then validates the user with validateUser()that ensures the users ID is correct.

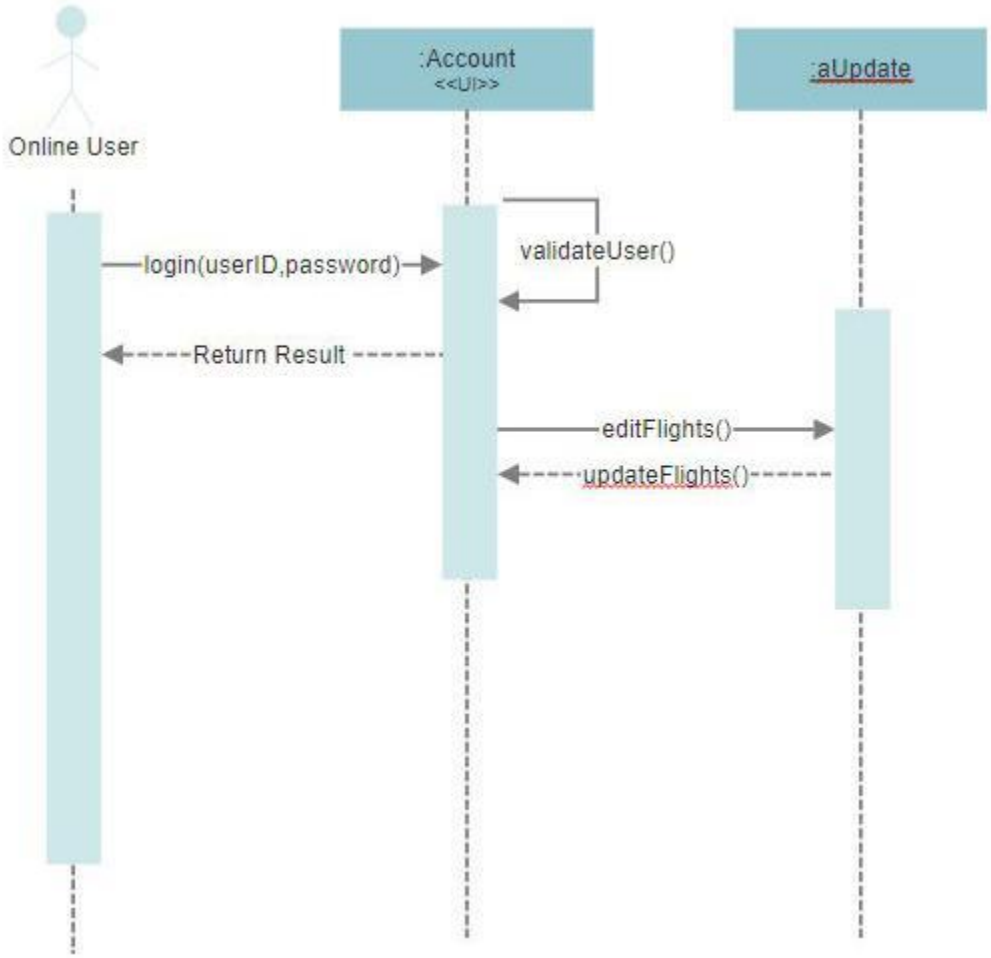
The flight information is then retrieved and returned. The amount that is due is due with getAmountDue()and returns the amount due to display to the user, returnAmount(). When the user attempts to pay they enter in their preferred method of payment type (a credit or debit card). They fill in their card information and are then paid with getPaymentType()and returned with returnPaymentType().The payment must then be authorized by the bank to ensure there are no fraudulent charges and is confirmed withcorfirmTransaction() and is returned with the message transaction(successful).Once this confirmation has come through, the payment can be made and the booking is confirmed and a message is displayed to the user.

SD 4 - Cancel Ticket Booked



Here is a sequence diagram for a customer trying to cancel their booked ticket. The customer enters their user ID to login, login(userID). The system has to validate the user with validateUser(). They then select the cancel ticket link on the website. The system collects the amount due with getAmountDue()and refunds the tickets with refundTicket().The system must then remove the passenger from the flight and update the seats available so that someone wishes to book a seat it's now available, this is done with the removePassenger().The seats are updated with updateSeatsAvailable().

SD 5 - Update Flight Schedule

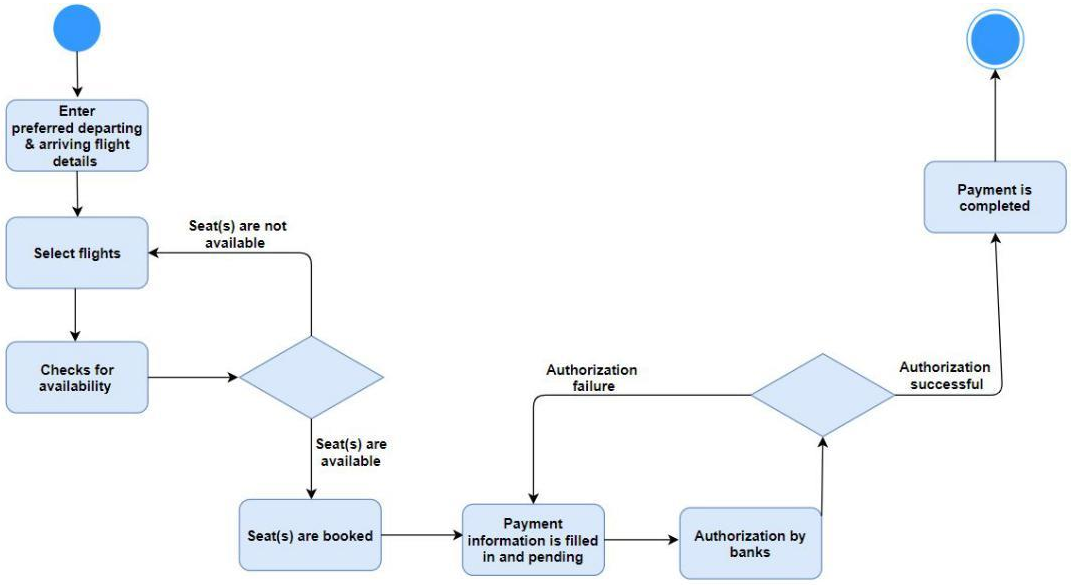


In the last sequence diagram, we can see the Admin logs in and is verified, this is done with login(userID,password).They then make adjustments to the flight schedule which is then sent to the database with editFlights()and the changes are saved and updated with updateFlights().The screen now displays the changes the admin has made.

The diagram shows the UI and then the aUpdatewhich is an object of the Update Flight Schedule class. I have two elements of a class, editFlights() and updateFlights() which allows for the changes to be made.

State Diagrams

Booking and Paying for a ticket



Here in a state diagram, we have a full circle that represents the initial state of the User. Here the user is on the flights' website. The first state is where the user enters in their preferred flight information such as departing/arriving airports as well as preferred times and they are selected.
In the third state is where the system checks for availability where two scenarios are possible.

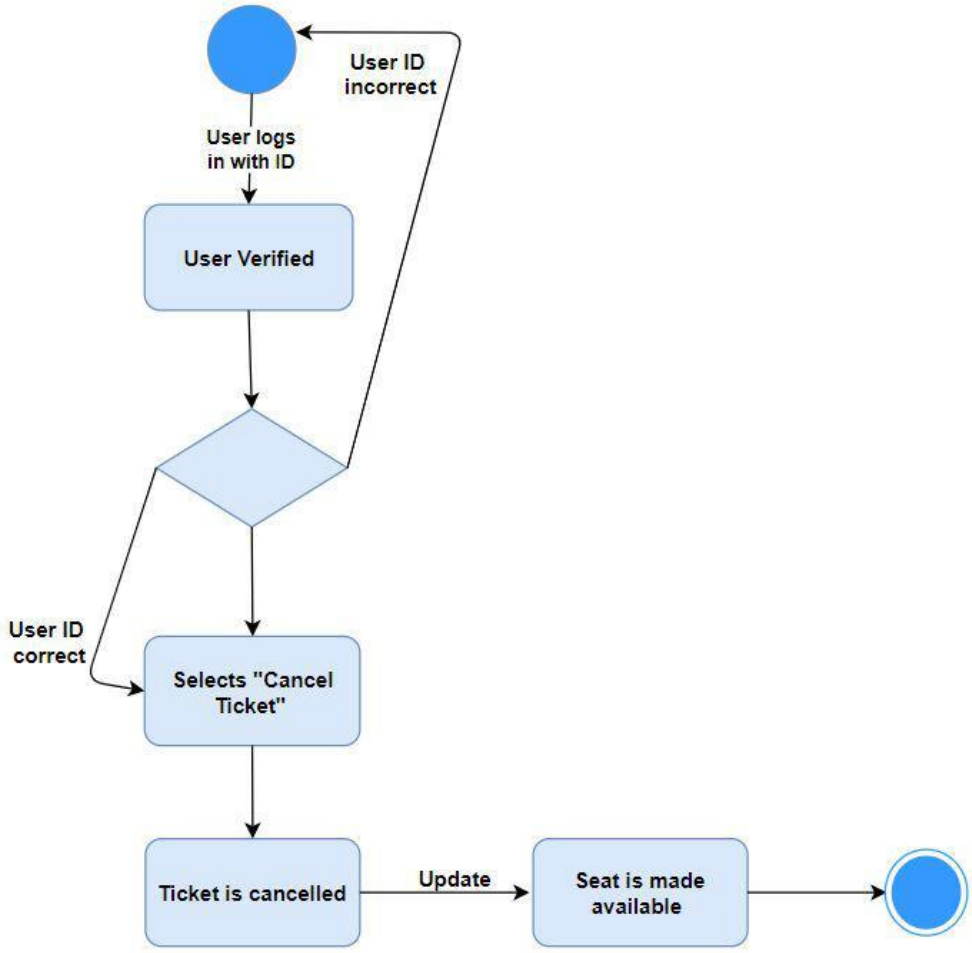
1. The seat(s) are not available: Here the system will revert the user back to the second state where they must select flights. Here they will be presented with a menu displaying other flight options.
2. The seat(s) are available: Here the system moves to the next state and the seats are booked.

Once the seat is booked, we move onto the next state where the payment information such as credit or debit card information is then pending. The next state is then the authorization by the banks for the pending payment. Here two scenarios are possible:

1. There is an authorization failure: Here the payment has not gone through and the system sends the user back the previous state with recommendations. Here the user can contact their bank and attempt the state again or, exit.
2. The payment is successful: Once the payment was successful, the user now has a seat(s) booked on the flights chosen.

The booking has now been finalized and this is the end of the state diagram, represented by a smaller circle surrounded by a larger circle.

Cancelling a purchased ticket



Here we can see that the initial state is the user logging in with their ID. In the first state, the user must be verified. Here one of two things that could happen:

1. The user ID is incorrect: If this is the case, the user cannot be allowed access and the system sends the user back to the login page.
2. The user ID is correct: If the user ID is correct then the user is permitted to click the “Cancel Ticket” and then this is processed. The system grabs the data necessary to cancel the correct number of seats on the specified flight.

After that stage, the ticket has then been cancelled. The number of seats available for purchase is now updated. Once that stage has been complete, the state diagram is complete.

CRC Cards

The top of the card is the name of the class. The left side is a list of responsibilities of the class. The right side shows other classes with which the class must collaborate to fulfil its responsibilities.

Check in For Flight	
• Keeps track of which users have checked in. • Validates User ID • Any updates should be brought up to the user such as delays.	• The Customer

View Available Flights	
• Display all available flights to the customer • Filters through customers' needs.	• The Customer

Book Ticket	
• Displays booking menu to the user. • Ensures that the ticket or tickets are booked and there are spaces reserved on the flight. • It also ensures that the number of seats booked is taken away from the total number of available seats. • It should also ensure that the customer receives a confirmation email.	• The Customer • Make Payment • Admin

Make Payment	
• Securely store the customer's credit/debit card information • That the payment has been authorized by the bank • That the payment is made	• The Customer • Book Ticket

Cancel Ticket	
• Ensures the users' ticket is cancelled • The seat or seats that were previously booked are now available for purchase • If a refund is entitled then they should be refunded	• The Customer • Book Ticket • Make Payment

Update Flight Schedule	
• Admin is verified • Admin can successfully change details of flight schedule • The system is then updated	• The Customer • System Administrator • Make booking • Make Payment • Check in

Test Code

```
import static org.junit.Assert.*;

import org.junit.Before;

import org.junit.Test;


public class AirlineTicketSystemTest {

	private Bookings bookings;

	private Fights flights;


	@Before

	public void setUp() {

		flights= new Flights(4637);

		flights= new Flights(23);

		flights.cancelFlight(23);

		bookings = new Bookings(135,&amp;quot;Dara O Sullivan&amp;quot;, 4637, 6A, false);

		bookings = new Bookings(973,&amp;quot;Aishling McPartland&amp;quot;, 4637, 26D, true);

		bookings.changeSeat(973, 7A);

		booking.cancelBooking(4637);

	}

	@Test

	public void testCancelBooking() {

		assertEquals(1, 4637.seatsBooked());

	}

	@Test
```

```
public void testBookedFlights() {
        assertEquals(1 , 973.bookedFlights());
}
@Test
public void testSeatBooked() {
        assertTrue(4637.seatBooked(26D));
}
@Test
public void testGetSeatNo() {
        assertEquals(26D, 973.returnSeatNumber());
}
@Test
public void testChangeSeat() {
        assertEquals(7A, 135.returnSeatNumber());
}
@Test
public void testAvailableFlights() {
        assertEquals(1, flights.getTotal());
}
@Test
public void testCancelFlight() {
        assertFalse(flights.exist(23));
}
```

```
    @Test
    public void testGetBookingName() {
        assertEquals("Aishling McPartland", bookings.getName(973));
    }
}
```