Cyanair AIRLINE

TECHNICAL DESIGN DOCUMENT

CYANAIR AIRLINE BOOKING SYSTEM

JUNE 2020

TABLE OF CONTENTS

TABLE OF FIGURES

Overview

Cyanair, an independent start-up Irish Airline, are currently unable to sell record their passenger data information and details sufficiently due to their current system which operates only paper-based transactions and records customer data through the use of spreadsheets. There is no system that allows flights to be booked electronically for the use of the Flight Representatives to record and retain information.

In this document we will run through the technical system requirements, operating environment, system and subsystem architecture, files and database design, input formats, output layouts and detailed design that relates to the creation of the Cyanair Booking System. The aim of this booking system is to aid Cyanair in terms of their bookings, so that Cyanair can retain customer information more efficiently in order to prevent a loss of data, and to eliminate the use of paper-based transactions.

1 INTRODUCTION

1.1 Purpose and Scope

The purpose and scope of this project involves the creation of a Booking System for Cyanair Airline. We will produce a piece of software that will allow the user to create a booking. There will be two users on this system – Flight Representative and the Administrator.

The Flight Representative will be able to create bookings on behalf of the Passenger using the software created. They will input the details from the perspective passenger that will allow them to book a flight from one departure to one destination, with the additional option of adding a return flight from their destination back to their departure, or an onward leg from their first destination to another destination. The system also asks for the dates of the flights that the passenger would like, and selection of the type of seats, or class that the passenger can choose from – depending and varying on the flight selected. These options will be Economy, Business and First Class. The information regarding the booking of flights will be on a separate form to the passenger details. After the flights have been selected, the dates and class have been chosen, the flight representative can move on to the next page in the system which will allow for the input of the Passenger Details.

This form will require that all fields be completed before moving on to processing the booking and the booking reference number being generated. Here, the Flight Administrator will be able to gather all of the Passenger's details such as their Full Name, Passport Number, Nationality, Phone Number, Email Address and Gender. When this has been completed, the system will then produce a message on the screen asking for the Flight Representative to confirm the booking. After the Flight Representative has confirmed the booking, the program will move to another form that will let the Flight Representative know that the booking has been confirmed and provide a reference number that can be shared with the passenger.

The system will only allow the creation of flight bookings for one passenger at a time. Additional flights may be booked for others, or even the same passengers after the creation of a booking, by the Flight Administrator entering the beginning of the booking system again.

The software will also feature Administrator access to a Maintenance Suite that will allow the Administrator to make changes to the Airport data. The system will allow the administrator to add new airports, view and delete the current airports available. This data will then be updated in the database.

The Maintenance Suite will also feature an option to view all of the Bookings that have been or will be created.

The database for this project will be created using SQLite and the system used to manage this database will be DB Browser.

1.2 Project Executive Summary

Programming Solutions Limited has won a contract in partnership with Cyanair, Ireland's newest start-up airline that specialises in affordable and luxury business travel. The aim of this project is to release a GUI object-oriented program for the company, which can be used internally by the staff and admin working at Cyanair to update and maintain the system. This is to replace their old paper system, as the airline would like to expand, and allow for bookings to be made online.

Cyanair, a start-up Irish Airline, is considering the creation of a Graphical User Interface application to be built using an *Object-Oriented approach.* This application will be developed by a team of experienced developers here at Programming Solutions Limited.

The project requirements involve an **Object-Oriented GUI** program that is going to be used as the airline's booking system. The methods that the airline would like featured in this application will allow a user to select criteria for an intended flight, which include:

1. Departure Airport
2. Destination Airport
3. Type of Seats (Silver, Gold, Platinum)

In our initial meeting with the Cyanair, we have gained further knowledge on their brief and the requirements of the project. They have indicated to us that they would only like the three main criteria for user selection to be included in the scope of the project, as mentioned above.

However, as our group of experts have analysed this project, they have outlined that there are further streams and areas that the airline may want to take a look at, in terms of the scope of the program, to allow additional criteria for the user looking to book a flight using this system.

This potential criterion includes the addition of,

1. Price of flights
2. Dates of flights
3. Duration of flights

However, these are not included within the scope of the project. At the moment, they are merely a talking point between the programmers and Cyanair. These are the recommendations that the programmers have given in order to make this application more usable and deliver a better user experience. They are not added to the scope, but we can speak to Cyanair about the addition of this criteria in future meetings, and the potential of discussing timelines and prices at a further stage.

If the addition of these criteria were to go ahead, a feasibility study would be required at a later stage. We would create this then as a separate project after the first project has been completed. We believe in developing a scalable program for the airline that is adaptable and future-proof, which will allow the application to grow and develop in the future with the addition of more features.

This application will be an automated system – that will be valuable for the customer, in terms of ease of use and valuable for Cyanair as a business. It is beneficial to both the customers and the company.

The current system that Cyanair adopts is not a good system and is outdated as it only allows paper-based transactions and excel spreadsheets to retain sensitive data information. Their recommendations for the new system could also be improved, as outlined before, it needs the addition of price of flights, date of flights and the duration of flights as a minimum. Customers will need this information when they are booking flights, and it has not been outlined by Cyanair as a requirement.

We have recommended to disclose to go with price of flights, dates of flights and duration of flights but they have declined. These would be seen as fundamental parts of booking a flight. These recommendations have been dismissed, but we have made it clear that these could be added to the scope of the project in the future.

1.3 Desired Outcome

The desired outcome of this project goes back to customer’s requirements of this project:

A graphical user interface that allows customers to select criteria an intended flight:

1. Departure Airport
2. Destination Airport
3. Type of Seats (Silver, Gold, Platinum)

1.4 Measurement of Project Success

The success of the project will be measured in terms of creating a successful and useable GUI application that will allow the staff and admin to select criteria for an intended flight on behalf of the passenger and to retain all the booking details of that passenger, in an application that is created using an object-oriented approach.

The application will be password protected and a restricted maintenance suite will allow administrators to add and delete departure and destination airports, and to view all the bookings created.

The application will be free from bugs, and training will be provided to staff and admin at Cyanair to ensure familiarity with the product. Lastly, the project reaching completion within the deadline will also prove as a measurement of the project’s success.

PROJECT INFORMATION

Start Date	1st May 2020
Budget	€125,000
Timeframe	3 Months 13 Weeks
Deadline	31st July 2020
Meeting Regularity	1. May: Twice a week 2. June : Once a week 3. July: Once a week
Payment Process	1. One-third up front 2. One-third after prototype is developed 3. One-third on completion of project (Subject to full contract)
Payment Milestones	1. Start Date – First Payment 2. Functional Tested Prototype – Second Payment 3. Sign-Off, documentation, specifications met, finished product, user manual, training and maintenance plan – Third Payment
Project Expectations	1. Seamless System 2. Intuitive for Users of the system 3. Admins won't need much training 4. Self-intuitive 5. Scalable – possibility of adding requirements in the future 6. Strong communication between both teams, if there are any issues speak to each other 7. Professional to each other

1.4.1 System Overview

The project requirements involve an **Object-Oriented GUI** program that is going to be used as the airline's booking system. The methods that the airline would like featured in this application will allow a user to select criteria for an intended flight, which include:

1. Departure Airport
2. Destination Airport
3. Type of Seats (Silver, Gold, Platinum)

In order to achieve this, we have designed a context diagram for the system. In this diagram, we have identified the requirements that we feel are needed to be integrated within the system. This includes the creation of tables within a database, which will include:

1. Airport
2. Aircraft
3. Bookings
4. Users
5. Flight route

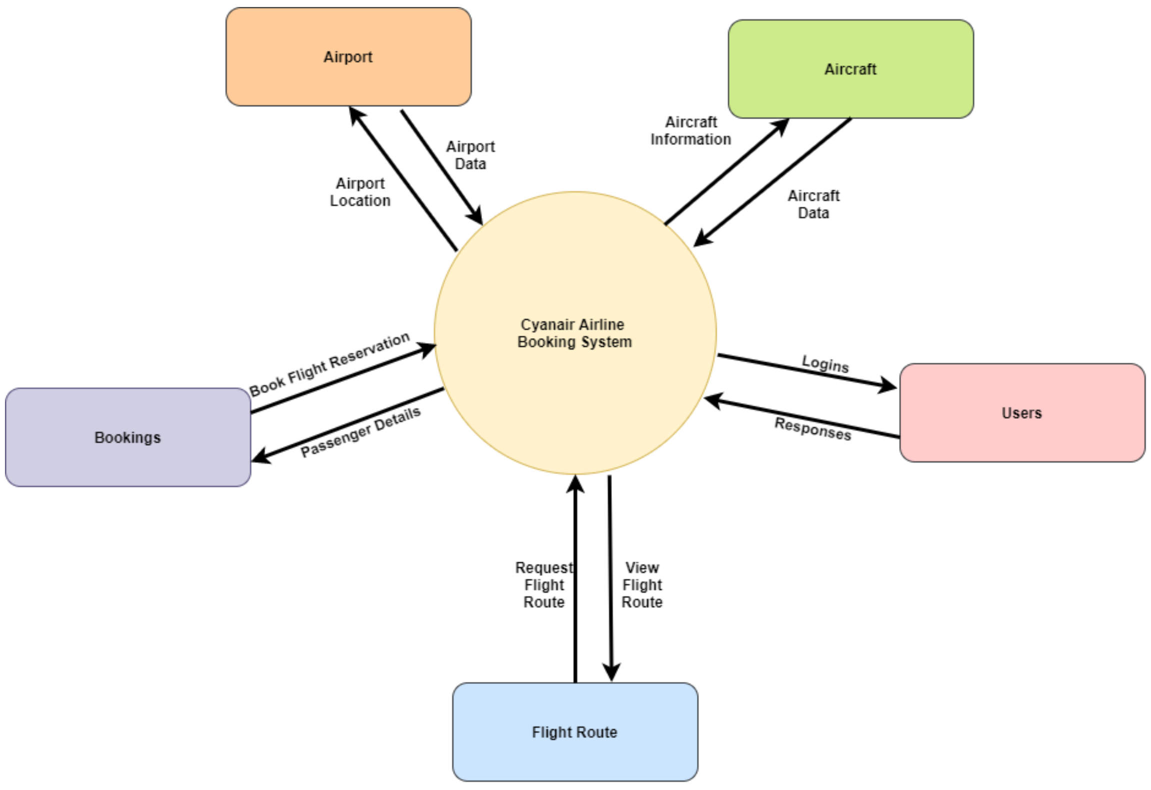


Figure 1: Context Diagram

1.5 Deliverables

This section will provide the assets being produced by the project and the key features.

COMPLETED SOFTWARE USING C#

1. Cyanair.db Database File
2. Help page using HTML integrated within the system

DOCUMENTATION

3. Object Oriented Programming Report Documentation
4. System Design Document Report Documentation
5. Project Scope Statement Documentation
6. Test Log Report Documentation
7. Technical Report Documentation

FUNCTIONALITY

1. The Cyanair Booking System will be protected via a Login (There will be 2 roles – Flight Representative and an Administrator)
2. The system will allow the user to then choose between either the Booking System or the Maintenance Suite. The choice will only be available to the Administrator. The Flight Representative will only have the Booking System as an option.
3. The Booking System will allow the Flight Representative to make bookings on behalf of passengers via the booking process.
4. The system will allow for Departure and Destination flights to be selected, along with the date for the flight.
5. The system will allow One Way, Return, or Onward Leg flights
6. The system will proceed to gather the passenger details
7. After all of the passenger’s information has been gathered, the system will ask for a booking confirmation, with all of the information regarding the flight booking being produced on the screen
8. After the flight has been confirmed, a booking reference number will be generated.
9. The user will then have the option to create a new booking.
10. The Maintenance Suite will allow access to only the Administrator where they will be able to view/add/delete airports and to view bookings.

1.6 Justification

The justification for this project came from the internal issues within Cyanair in regard to how they process their airline bookings. The current system that they are using is manual with the aid of Microsoft Excel. Using this means of storing bookings is prone to many errors, not only with the input of data, but also the loss of data. Flight representatives are currently forgetting to input data into Excel, which is causing a loss of bookings. There are other issues including the creation of duplicate bookings, as well as a lack of organisation, which has led Cyanair to believe that they need to update their booking system. As the company is looking to expand, and gain even more bookings, Cyanair would like software that allows the company to efficiently create bookings and manage these bookings in a proper system.

USE CASE DIAGRAM

The following use case diagram is a representation of the ***Cyanair Booking System***. It graphically displays the relationship between the system, its users and its different functions of the system. This system will allow both the Flight Representative and the Administrator to view the booking system and make bookings on behalf of the passenger. Only the administrator will have access to view bookings, airports, and to add or delete airports.

CYANAIR BOOKING SYSTEM USE CASE DIAGRAM

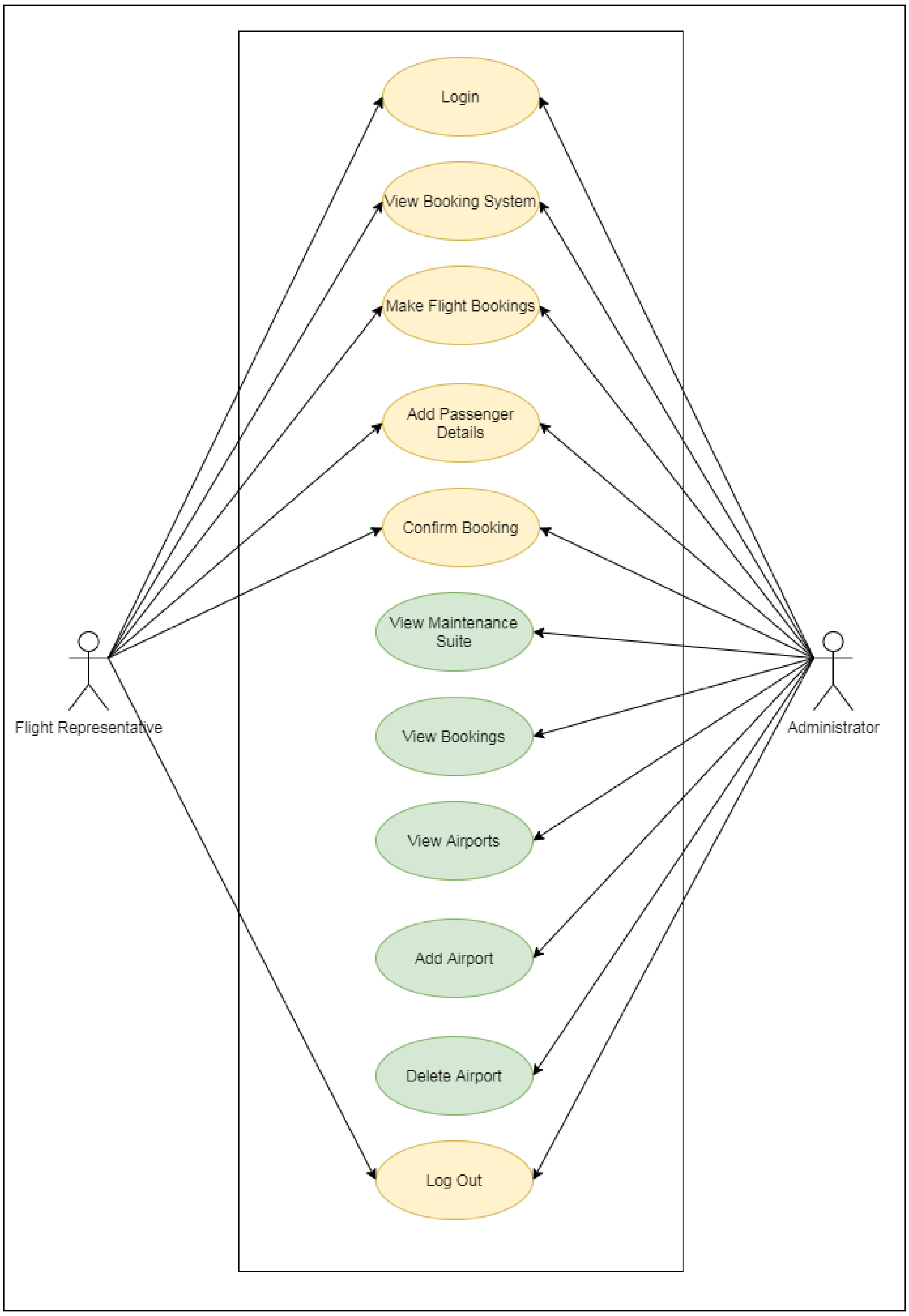


Figure 2: Use Case Diagram

2 SYSTEM REQUIREMENTS

In the Cyanair project, the technical specifications for the users of the application are outlined. This is thinking in terms of the Flight Representative User and the Admin User, and the specifications required for both of these users. It is the specifications of the requirements outlined in the document, the departure airports, destination airports, and the seat types that are to be offered as part of the program. It is where this data is taken and then modelled and structured.

The technical specifications of the project involve designing an Object-Oriented GUI program that is going to be used as the airline's booking system. The methods that the airline would like featured in this application will allow a user to select criteria for an intended flight, which include:

1. Departure Airport
2. Destination Airport
3. Type of Seats (Silver, Gold, Platinum)

Additional requirements for the design aspect of the program includes:

1. 8 Airports – Destination and Departure
2. A graphical user-interface that uses Combo Boxes, Radio Buttons and at least one Check Box
3. The Departure and Destination airports must be displayed in alphabetical order
4. The Departure and Destination airports cannot be the same
5. It should be possible to book an onward leg, from the first destination or back to the original departure airport
6. Passenger name should be recorded, and a Booking Reference assigned
7. When completed, details of the flight(s) booked should be displayed on the screen for the user to confirm
8. On confirmation, details should be written to a database
9. A password-protected Maintenance Suite should be available to allow airports to be added or removed from the lists, and to view booking files

There will be two of users that are planned to use this program, the flight representative that will only have access to the booking system in order to book flights, and an administrator that can control administrative tasks on the system, such as adding and removing airports, and viewing the bookings created. They will also have access to the booking system and can book flights.

FLIGHT REPRESENTATIVE USER REQUIREMENTS

1. A user-friendly graphical user interface that uses combo boxes, radio buttons, and at least one check box.
2. The departure and destination lists must be in alphabetical order.
3. The departure and destination airports cannot be the same.
4. Passenger information must be submitted, such as a name, and a booking reference will be assigned.
5. When this information has been submitted, the details of the flight booked should be available on the screen for a user confirmation.
6. When this has been confirmed, the details will be saved to a database.

ADMINISTRATOR USER REQUIREMENTS

1. An option for administrators to log in that is password-protected.
2. The ability to allow administrators to view the airports available and the bookings created
3. The ability to allow administrators to add and delete departure and destination airports
4. When confirmed, the details will be saved to a database

2.1 Design Constraints

2.1.1.1 System Constraints

There are a few constraints affecting the development of this software, budget being the main obstacle. The budget assigned for this system is €125,000 – there are additional features we feel could be added to the project that are not mentioned within the scope, such as a web-based system, an enterprise database package as opposed to the current database system which is free-software – however, the current budget does not allow for this, which is the cause for these constraints within the project. A higher budget would have given the possibility to rectify these issues and perhaps have a smoother system for Cyanair. Particularly in the case of a web-based system, as the new system will only be used by the Cyanair team, the administrators, and the flight representatives, meaning that the end-user cannot use this system themselves at all, which could have been achieved with a web-based system to allow the user to make their own bookings. As mentioned, the database is used with free software, which can cause some design constraints as it is more restrictive than perhaps enterprise databases, as it is only entry-level. At the moment, this database is not able to handle dates in a way that the developers find satisfactory which has made the use of SQLite a constraint on the project.

2.1.1.2 Assumptions

There are a number of assumptions made in this project, for example, the exact details required from the Passenger when creating a booking was never specified by from the client, therefore, we had to make our own assumptions on what information an airline booking system would be looking to gather.

Another example is the inclusion of the flight representative being able to choose a flight date on behalf of the customer while selecting flights. This was never specified in any of the briefs, requirements or talks with Cyanair – but we have assumed ourselves that this is necessary in order to book a flight, as without it the system would make no sense, and there would be no flights available.

2.1.1.3 Exclusions

There are some exclusions within the project that we have identified. Again, some of these exclusions are necessary as there is no room in the budget to accommodate these features, but also, they were never spoken about or agreed with the client as to be requirements. They are simply featuring that we assume would be necessary in an airline booking system, that are currently not being implemented.

At the moment, this system does not show the price of a flight. This is a big exclusion, as it necessary for the passenger to be aware of the cost of a flight. Not only this, but there are no payment options at the moment. This will need to be rectified in some way by the company, to ensure that they are getting payment for bookings. This could even be by allowing the customer to present their Booking Reference number and visiting the airline's HQ to make physical payments if they do not have the ability for software payments. Of course, this course of action is completely up to the client.

Other exclusions are the lack of multi-language support. The system only allows for the English language. As the company is based in Ireland, this shouldn't be an issue, but in the future as the company expands, this could be an area that the client looks to expand, perhaps on future projects.

POSSIBLE FUTURE REQUIREMENTS

In the future of the project, there are a stream of avenues we could go on to add more requirements. Some of these requirements may include the addition of:

2.1.1.4 General Data Protection Regulation (GDPR)

At the moment, ***General Data Protection Regulation (GDPR)*** is a hot topic as laws regarding GDPR compliance are constantly evolving. It is important to stay on top of it, and to be aware of the changes as it can have massive affects to how a system can store its data. GDPR can cause the direction of the development of the system to change if laws are modified or updated. On May 25, 2018, the European Union imposed obligations and regulations regarding the capturing and targeting of data to anyone living within the EU. This had a massive impact on systems and companies not only in the European Union, but all around the world – as visitors from the EU would need their data captured differently, in accordance with the new regulations – as compared to visitors outside of the EU.

This was a very strict law that was created with the aim of tighter online privacy and security for the people living in this union. There are harsh penalties and fines for those who violate the privacy and security standards – so it is something that a company must be mindful of, and then integrate within a system. It is equally as important to keep up to date in regard to the current GDPR laws, and to implement these changes within a system that is live. Essentially, it does have an impact in the development of a system, as the system cannot go live unless it is GDPR compliant, or the company will face fines.

2.1.1.5 Data Encryption

Another potential aspect that could distract the flow of software development is implementing data encryption within the system. As the Cyanair Booking System deals with sensitive and confidential information from Passengers, it is vital to keep this information secure. This can be aided by the use of Data Encryption. The last thing any company would want is for a data leak, so it is essential to implement this encryption correctly. Data Encryption would translate and encrypt data into code that only people with restricted access with the use of a secret key known as a decryption key, or password can use in order to read this information. It is recommended that Cyanair integrates encryption as a data security method in order to secure the data of their passengers.

2.1.1.6 Web-Based System

Another area that has potential in the future for the longevity of this system, would be to perhaps move the application to a web-based system. We feel like this would be a great function for the Cyanair Booking System – so that the company would be able to cut out the middle man – the flight representative – so that passengers are able to book their own flights, or search criteria for a flight, without having to call up the flight representative. We feel that the current system is outdated, as now majority of passengers would book their flights online by themselves.

2.1.1.7 Enterprise Database

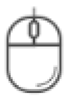
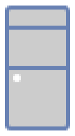
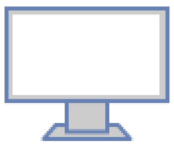
Another potential requirement in the future could be to migrate to an enterprise database. At the moment, the current system is using an entry level database, SQLite, which works fine at the moment when the system is small. What we have conducted from our feasibility studies are that this project in time will become much larger as the company grows. As the company gets larger, more and more bookings will be made, all of which requires a database to store it in – this data is essential to the program and therefore, in the future we would see it as a requirement to update the database system.

3 SYSTEM OVERVIEW

In order for the desired users of the ***Cyanair Booking System*** – the System Administrator(s) and the Flight Representative(s) – to run the program, it is important that they are presented with all of the required hardware and software requirements.

3.1 System Hardware Requirements

The Minimum Hardware Requirements to run the ***Cyanair Booking System*** includes:

	
Keyboard	**Mouse**
	
Computer	**Display Monitor**

1. ***Keyboard*** – A keyboard is required for this system, in order to log in to the booking system, and to record the passenger's inputs (details). It is essential in order to run the program as expected and to capture the booking information.

2. ***Mouse*** – A mouse is essential for the program, as without it the system is completely unusable. The system will go from page to page with the aid of a mouse, in order to input information, it is required, as well as viewing the maintenance suite, to add and view airports, as well as bookings, a mouse is needed. Without a mouse, the system would be useless and unusable. It allows for the control of the graphical user interface of the system.

3. ***Computer*** – A computer is required in order to run this system. Not only is it required to meet the hardware requirements, but also the software requirements. We will speak further about the *minimum system requirements* in the subsequent section to gain further insight.

4. ***Display Monitor*** – In order for the user to use the application, it is necessary to have a display monitor connected to the computer. Without the display monitor, the user cannot use the application as they cannot see or run the system and it's graphical user interface.

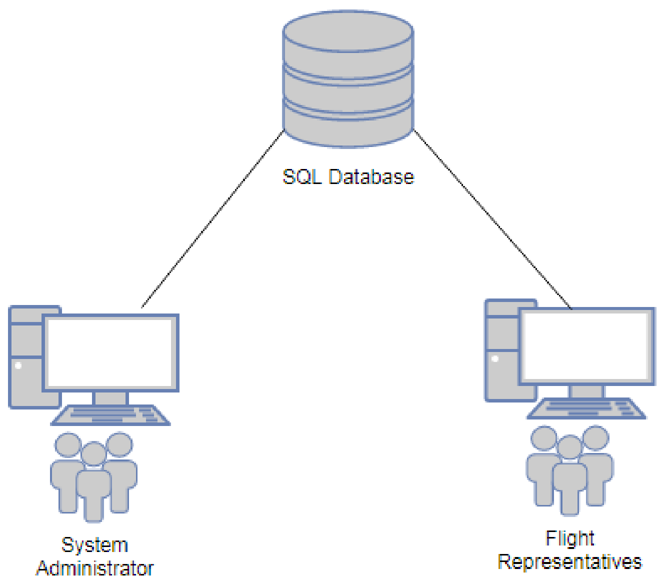


Figure 3: System Hardware Architecture Diagram

3.2 System Software Architecture

This program will be created using the C# Programming Language. The application will be developed using Microsoft Visual Studio Community. The application will be using .NET framework to develop the system. This booking system will feature the use SQLite for its database, and this will be run using the DB Browser application. The ***Cyanair Booking System*** can only be run on PC's using Windows. It is not available on Mac, mobile phones or tablets. As the system will be developed using Windows 7, the minimum requirements in order to run this system can be seen below.

The Minimum Software Requirements to run the ***Cyanair Booking System*** includes:

1. .NET Framework
2. Windows 7 Operating System or Later
3. 1 gigahertz (GHz) or faster 32-bit (x86) or 64-bit (x64) processor*
4. 1 gigabyte (GB) RAM (32-bit) or 2 GB RAM (64-bit)
5. 16 GB available hard disk space (32-bit) or 20 GB (64-bit)
6. DirectX 9 graphics device with WDDM 1.0 or higher driver

This is the ideal system specifications that are required to run this program correctly.

If your system fails to meet the system requirements, there may be performance issues, or the program may be incompatible.

3.2.1 System Creation Tools

Programming Language	C#
Framework	.NET Framework
Software Tools	1. **Visual Studio Community** – used for the creation of the Booking System program and the Graphical User Interface 2. **DB Browser** – used to populate and view database tables. Management of the database used in the system
Database	**SQLite** – Entry Level Free database software
Packages Used	**NuGet**– SQLite Packages and Entity Framework installed on Visual Studio required to connect Database to the System

3.2.2 Packages and Program Installations

In order to create this program, it is essential to follow the technical documentation and specifications to ensure it runs and works as expected.

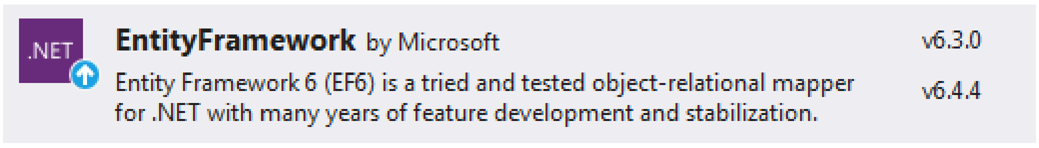
1. Download Visual Studio Community to create C# Form Applications

https://visualstudio.microsoft.com/vs/

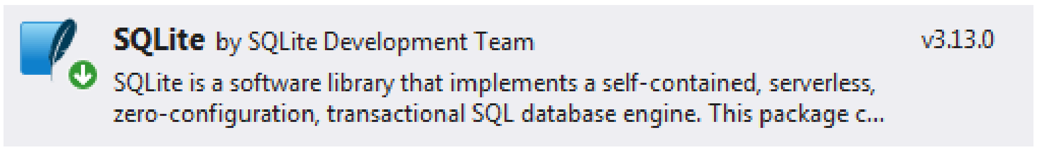
2. Download DB Browser to browse the database that we will be using

https://sqlitebrowser.org/dl/

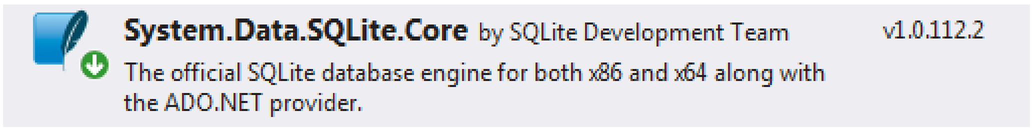
3. Download the following NuGet Packages on Visual Studio:



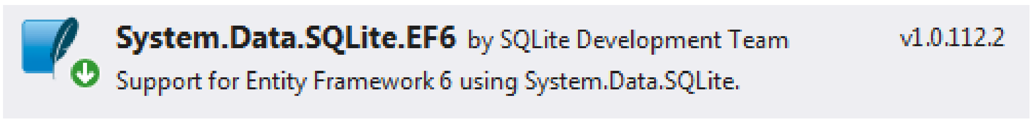
EntityFramework by Microsoft v6.3.0 v6.4.4
Entity Framework 6 (EF6) is a tried and tested object-relational mapper for .NET with many years of feature development and stabilization.



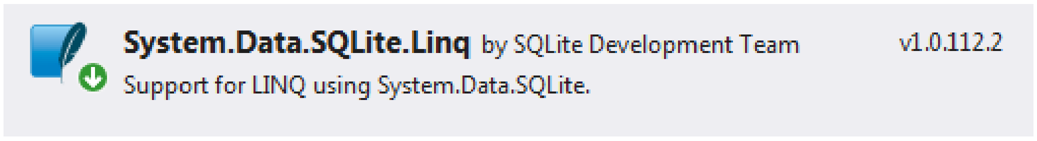
SQLite by SQLite Development Team v3.13.0
SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. This package c...



System.Data.SQLite.Core by SQLite Development Team v1.0.112.2
The official SQLite database engine for both x86 and x64 along with the ADO.NET provider.



System.Data.SQLite.EF6 by SQLite Development Team v1.0.112.2
Support for Entity Framework 6 using System.Data.SQLite.

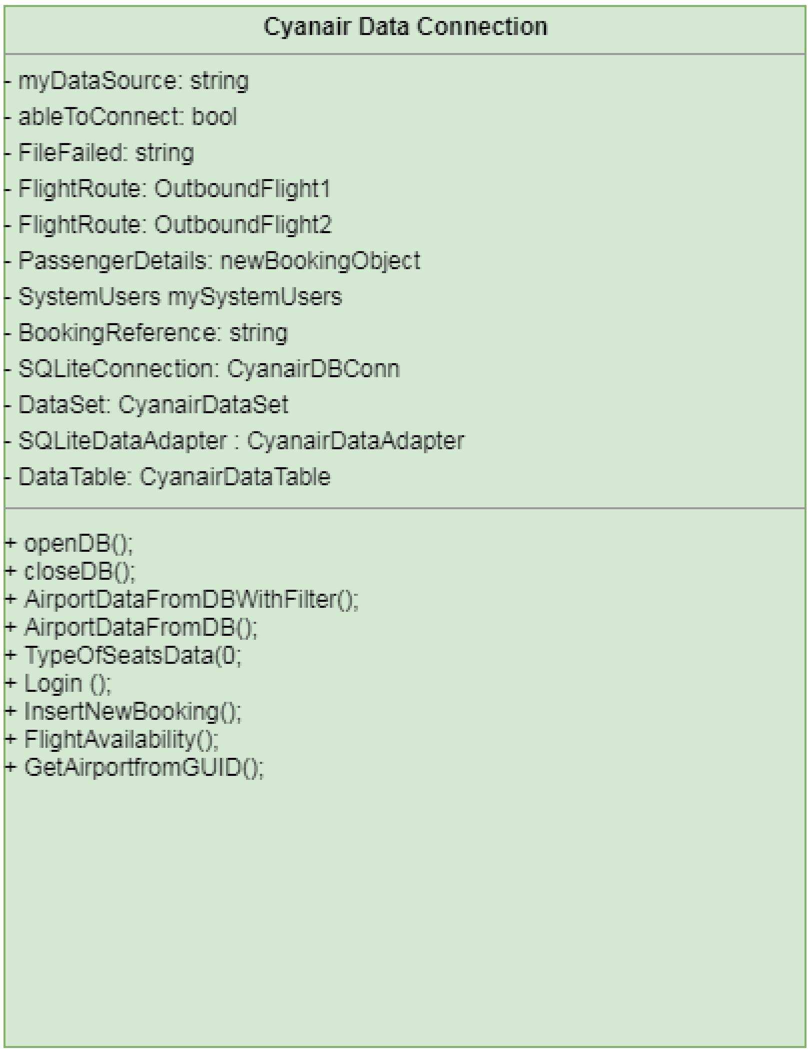


System.Data.SQLite.Linq by SQLite Development Team v1.0.112.2
Support for LINQ using System.Data.SQLite.

4. Download the Cyanair.db database file and create a folder in `C:\\Data\\Cyanair.db.`

5. Create a Database Connection using SQL in C#

In order to access data from an external database, Cyanair.db, we will need to establish a connection. We will create this as it's own class, titled, ***CyanairDataConnection***. We will store all of the functions relating to the connection between the database and the system here.



```
Cyanair Data Connection

- myDataSource: string
- ableToConnect: bool
- FileFailed: string
- FlightRoute: OutboundFlight1
- FlightRoute: OutboundFlight2
- PassengerDetails: newBookingObject
- SystemUsers mySystemUsers
- BookingReference: string
- SQLiteConnection: CyanairDBConn
- DataSet: CyanairDataSet
- SQLiteDataAdapter : CyanairDataAdapter
- DataTable: CyanairDataTable

+ openDB();
+ closeDB();
+ AirportDataFromDBWithFilter();
+ AirportDataFromDB();
+ TypeOfSeatsData(0;
+ Login ();
+ InsertNewBooking();
+ FlightAvailability();
+ GetAirportfromGUID();
```

Figure 4: Data Connection Class

The connection will manage a connection to the database.

The command will execute a query command on the database.

The data adapter will exchange data between the data set and the database.

CYANAIR DATA CONNECTION CLASS

In order to use the connection class, it is essential to declare all of the components required to connect to the database, as well as providing a location to the data source, i.e. the database location.

```
static string myDataSource = @"Data Source = C:\\data\\Cyanair.db";

public SQLiteConnection CyanairDBconn = new SQLiteConnection(myDataSource);

public DataSet CyanairDataSet = new DataSet();
public SQLiteDataAdapter CyanairDataAdapter = new SQLiteDataAdapter();
private DataTable CyanairDataTable = new DataTable();
```

Figure 5: Data Connection Class Declarations

Once all of these variables have been assigned, we will create a function that allows the developer to open and close the database connection when necessary, instead of having to constantly call it all each time and to save the amount of code in the program.

```
public void openDB()
        {
            if (CyanairDBconn.State != System.Data.ConnectionState.Open)
            {
                CyanairDBconn.Open();
            }


        }
```

Figure 6: Open Data Connection

```
public void closeDB()
        {
            if (CyanairDBconn.State == System.Data.ConnectionState.Open)
            {
                CyanairDBconn.Close();
            }


        }
```

Figure 7: Close Data Connection

LOGIN WITHIN CYANAIRDATACONNECTION CLASS

The login SQL statement will be created as a function within the CyanairDataConnection class. This function will allow the data connection to connect to the database, and open the database. It will then read the information provided in the users table which will try to match and validate to what is entered by the user within the program. If the information provided matches the information within the database, then the user will be able enter, however, if it does not, access will be denied and they will be prompted that their information was valid, and they can try to login again.

```
public bool Login(string UserName, string Passwordy,SystemUsers mySystemUsers)
        {

            bool CanAccess = false;

            this.openDB(); //opens the database if not already opened ! makes
sure its available

            string mySQLcommandstring = "SELECT Guid,Role FROM Users WHERE
Username='"+ UserName + "' AND Password='"+ Passwordy + "';";
            DataTable loginDt = new DataTable();

            CyanairDataAdapter = new SQLiteDataAdapter(mySQLcommandstring,
this.CyanairDBconn);

            CyanairDataAdapter.Fill(loginDt);

             if (loginDt.Rows.Count  > 0) //> 1
             {
                // MessageBox.Show("You have logged in ! ");

                mySystemUsers.BookedByUser_Guid =
loginDt.Rows[0].ItemArray[0].ToString();
                mySystemUsers.UserRole =
loginDt.Rows[0].ItemArray[1].ToString();

                CanAccess = true;

            }


             this.closeDB();

            return CanAccess;



        }
```

Figure 8: Login Function

ADD AIRPORT

```
if (txtAirportName.Text != "" && txtAirportCode.Text != "")
                {
                SQLiteConnection MaintenanceSuiteConnection = new
SQLiteConnection(@"Data Source = C:\\data\\Cyanair.db");
                //CREATES A NEW GUID FOR THE INSERT INTO THE DB
                Guid AirportGuid = Guid.NewGuid();


                //add guid later?                           //WHERE Deleted=0
                string mySQLCommandString = "INSERT INTO Airport
(Guid,Airport,AirportCode) VALUES('"
                                                + AirportGuid.ToString() + "','"
                                                 + txtAirportName.Text + "','"
                                                 + txtAirportCode.Text + "')";
                MaintenanceSuiteConnection.Open();

                SQLiteCommand sqlInsertAirport = new
SQLiteCommand(mySQLCommandString, MaintenanceSuiteConnection);
                sqlInsertAirport.ExecuteNonQuery();
               // SqlDataAdapter MaintenanceSuiteDataAdapter = new
SqlDataAdapter(mySQLCommandString, MaintenanceSuiteConnection);
               // MaintenanceSuiteDataAdapter.SelectCommand.ExecuteNonQuery();
                MaintenanceSuiteConnection.Close();
                lblAirportAdded.Visible = true;
                MessageBox.Show("INSERTED SUCCESSFULLY ! NEW RECORD ADDED !");
                ReloadDataINDGV();
                txtAirportName.Text = "";

                txtAirportCode.Text = "";
```

Figure 9: Add Airport

DELETE AIRPORT

```
private void btnDelete_Click(object sender, EventArgs e)
        {
            DeleteRow();
            ReloadDataINDGV();


                                                                                    }

private void ReloadDataINDGV()
        {
            //SQL
            SQLiteConnection MaintenanceSuiteConnection = new
SQLiteConnection(@"Data Source = C:\\data\\Cyanair.db");
            //CREATES A NEW GUID FOR THE INSERT INTO THE DB
            Guid AirportGuid = Guid.NewGuid();

            MaintenanceSuiteConnection.Open();
            //add guid later?                           //WHERE Deleted=0
            string mySQLCommandString = "SELECT * FROM Airport;";
            SQLiteDataAdapter MaintenanceSuiteDataAdapter = new
SQLiteDataAdapter(mySQLCommandString, MaintenanceSuiteConnection);
            DataTable dtDGV = new DataTable();
            MaintenanceSuiteDataAdapter.Fill(dtDGV);
            dgvAirports.DataSource = dtDGV;
            dgvAirports.Visible = true;
            MaintenanceSuiteConnection.Close();
        }
```

```
//FUNCTION THAT ALLOWS A ROW SELECTED IN THE DATAGRID VIEW TO BE DELETED !
        private void DeleteRow()
        {
            SQLiteConnection MaintenanceSuiteConnection = new
SQLiteConnection(@"Data Source = C:\\data\\Cyanair.db");
            SQLiteCommand DeleteCommand = new SQLiteCommand();
            //MaintenanceSuiteConnection.ConnectionString=

            if (dgvAirports.Rows.Count > 1 && dgvAirports.SelectedRows[0].Index !=
dgvAirports.Rows.Count -1)
                {
                DeleteCommand.CommandText = "DELETE FROM Airport WHERE Uid=" +
dgvAirports.SelectedRows[0].Cells[0].Value.ToString()
                    + "";
                MaintenanceSuiteConnection.Open();
                DeleteCommand.Connection = MaintenanceSuiteConnection;
                DeleteCommand.ExecuteNonQuery();
                MaintenanceSuiteConnection.Close();
                dgvAirports.Rows.RemoveAt(dgvAirports.SelectedRows[0].Index);
                MessageBox.Show("deleted !!!!");
            }
```

Figure 10: Delete Airport

VIEW AIRPORTS

```
private void LoadDataFromAirportsINDGV()
        {
            //DATA GRID VIEWER
            SQLiteConnection MaintenanceSuiteConnection = new
SQLiteConnection(@"Data Source = C:\\data\\Cyanair.db");
            MaintenanceSuiteConnection.Open();
            //add guid later?                          //WHERE Deleted=0
            string mySQLCommandString = "SELECT * FROM Airport";
            SQLiteDataAdapter MaintenanceSuiteDataAdapter = new
SQLiteDataAdapter(mySQLCommandString, MaintenanceSuiteConnection);
            DataTable dtDGV = new DataTable();
            MaintenanceSuiteDataAdapter.Fill(dtDGV);
            dgvAirports.DataSource = dtDGV;
            dgvAirports.Visible = true;
            MaintenanceSuiteConnection.Close();
        }
```

Figure 11: View Airports

VIEW BOOKINGS

```
private void LoadDataFromBookingsINDGV()
        {
            //SQL
            SQLiteConnection MaintenanceSuiteConnection = new
SQLiteConnection(@"Data Source = C:\\data\\Cyanair.db");
```

```
            MaintenanceSuiteConnection.Open();
            //add guid later?                            //WHERE Deleted=0
            string mySQLCommandString = "Select * FROM PassengerDetails LEFT JOIN 
Booking ON PassengerDetails.Uid = Booking.Uid; ";
            SQLiteDataAdapter MaintenanceSuiteDataAdapter = new 
SQLiteDataAdapter(mySQLCommandString, MaintenanceSuiteConnection);
            DataTable dtDGV = new DataTable();
            MaintenanceSuiteDataAdapter.Fill(dtDGV);
            dgvBookings.DataSource = dtDGV;
            dgvBookings.Visible = true;
            MaintenanceSuiteConnection.Close();

        }
```

Figure 12: View Bookings

BOOKING SYSTEM

```
//FUNCTION TO FILL COMBOBOX WITH THE DESTINATION/DEPARTURE AIRPORT LEG 1 AND LEG 2
        public void AirportDataFromDB()
        {
            this.openDB(); //opens the database if not already opened ! makes sure 
its available
            string mySQLcommandstring = "SELECT * from Airport ORDER BY Airport ;";
            CyanairDataAdapter = new SQLiteDataAdapter(mySQLcommandstring, 
this.CyanairDBconn);

            CyanairDataAdapter.Fill(CyanairDataSet, "DepartureLeg1Airport");
            CyanairDataAdapter.Fill(CyanairDataSet, "DestinationLeg1Airport");
            CyanairDataAdapter.Fill(CyanairDataSet, "DepartureLeg2Airport");
            CyanairDataAdapter.Fill(CyanairDataSet, "DestinationLeg2Airport");
        }
        //FUNCTION TO FILL COMBOBOX WITH THE TYPE OF SEATS AVAILABLE

public void TypeOfSeatsData()
        {
            this.openDB(); //opens the database if not already opened ! makes sure 
its available
            string mySQLcommandstring = "SELECT * from SeatClass ;";
            CyanairDataAdapter = new SQLiteDataAdapter(mySQLcommandstring, 
this.CyanairDBconn);

            CyanairDataAdapter.Fill(CyanairDataSet, "ClassTypeLeg1");
            CyanairDataAdapter.Fill(CyanairDataSet, "ClassTypeLeg2");
        }



 public string GetAirportfromGUID(string AirportGuid)
        {

            DataTable dt = new DataTable();
            this.openDB(); //opens the database if not already opened ! makes sure 
its available
            string mySQLcommandstring = "SELECT * from Airport WHERE Guid='" + 
AirportGuid  + "' ;";
```

```
            CyanairDataAdapter = new SQLiteDataAdapter(mySQLcommandstring, 
this.CyanairDBconn);

            CyanairDataAdapter.Fill(dt);
            //FILLS IN THE FORM WITH THE AIRPORT NAME AND CODE !
            return dt.Rows[0].ItemArray[2].ToString() + " (" + 
dt.Rows[0].ItemArray[3].ToString() + ")";

        }
  public bool FlightAvailability(FlightRoute myFlightRoutetoCheck)
        {

            DataTable dt = new DataTable();
            bool FlightisOK = false;
            string myDateOnly = myFlightRoutetoCheck.DateOfFlight.Substring(0, 10); 
// 12/06/2020


            this.openDB(); //opens the database if not already opened ! makes sure 
its available
            string mySQLcommandstring = @"SELECT Uid,FlightNo,TimeOfFlight,Duration 
FROM FlightRoute " +
                                    " WHERE " +
                                    "DepartureAirport_Guid='" + 
myFlightRoutetoCheck.DepartureAirport_guid + "'  AND " +
                                    "DestinationAirport_Guid='" + 
myFlightRoutetoCheck.DestinationAirport_guid + "' AND " +
                                    "DateOfFlight='" + myDateOnly + "'" +
                                    "; ";
            CyanairDataAdapter = new SQLiteDataAdapter(mySQLcommandstring, 
this.CyanairDBconn);

            CyanairDataAdapter.Fill(dt);

            if (dt.Rows.Count > 0)
            {
                FlightisOK = true;
                myFlightRoutetoCheck.FlightNo = dt.Rows[0].ItemArray[1].ToString();
                myFlightRoutetoCheck.TimeOfFlight = 
dt.Rows[0].ItemArray[2].ToString();
                myFlightRoutetoCheck.Duration = dt.Rows[0].ItemArray[3].ToString();

            }

            return FlightisOK;


        }



/FUNCTION TO FILL COMBOBOX WITH THE DESTINATION/DEPARTURE AIRPORT LEG 1 AND LEG 2
        public void AirportDataFromDBWithFilter(string ExcludeAirportFromList, 
string SelectedTable)
        {

            if (CyanairDataSet.Tables.Contains(SelectedTable))
            {
                CyanairDataSet.Tables.Remove(SelectedTable);
            }
            this.openDB(); //opens the database if not already opened ! makes sure
its available
            string mySQLcommandstring = "SELECT * from Airport WHERE Guid != '"+
ExcludeAirportFromList + "' ORDER BY Airport ;";
            CyanairDataAdapter = new SQLiteDataAdapter(mySQLcommandstring,
this.CyanairDBconn);

            CyanairDataAdapter.Fill(CyanairDataSet, SelectedTable);

        }
```

Figure 13: Flight Selection – Booking System

4 SYSTEM DESIGN

4.1 Use Case Diagram

The following use case diagram is a representation of the ***Cyanair Booking System***. It graphically displays the relationship between the system, its users and its different functions of the system. This system will allow both the Flight Representative and the Administrator to view the booking system and make bookings on behalf of the passenger. Only the administrator will have access to view bookings, airports, and to add or delete airports.

CYANAIR BOOKING SYSTEM USE CASE DIAGRAM

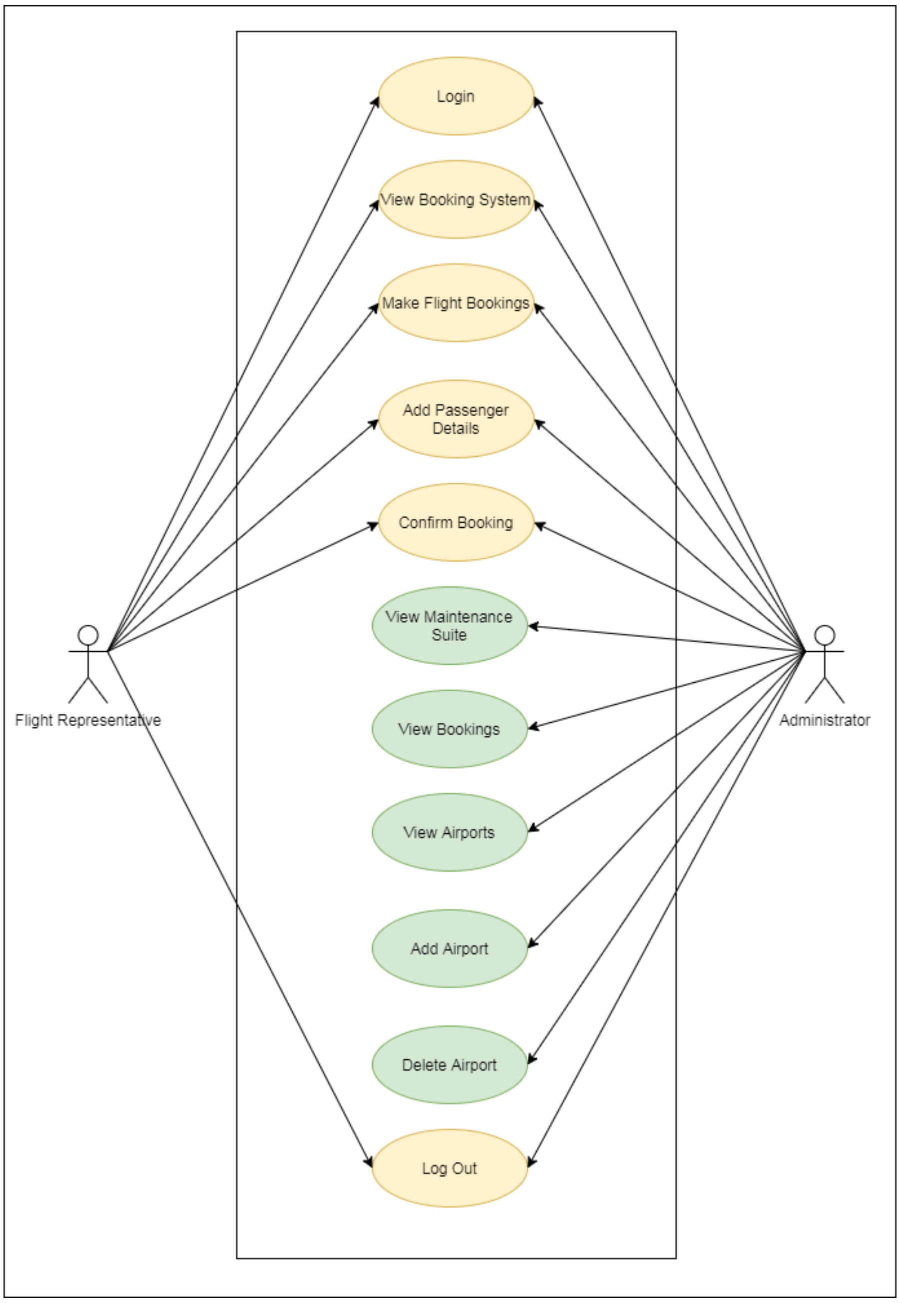


Figure 14: Use Case Diagram

4.2 Class Diagram

This diagram is a visual representative of the classes that are to be used in the ***Cyanair Booking System***. As we are able to see from the above Class Diagram, all of the classes connect to the main class, "Bookings". There is an additional class that is used as a data connection between the system and the database. The "Person" class is a super-class to the "PassengerDetails" to which "PassengerDetails" inherits objects from the "Person Class". Both "Flight Route" and "System Users" declare their own objects which are used within the system and the database.

CYANAIR BOOKING SYSTEM CLASS DIAGRAM

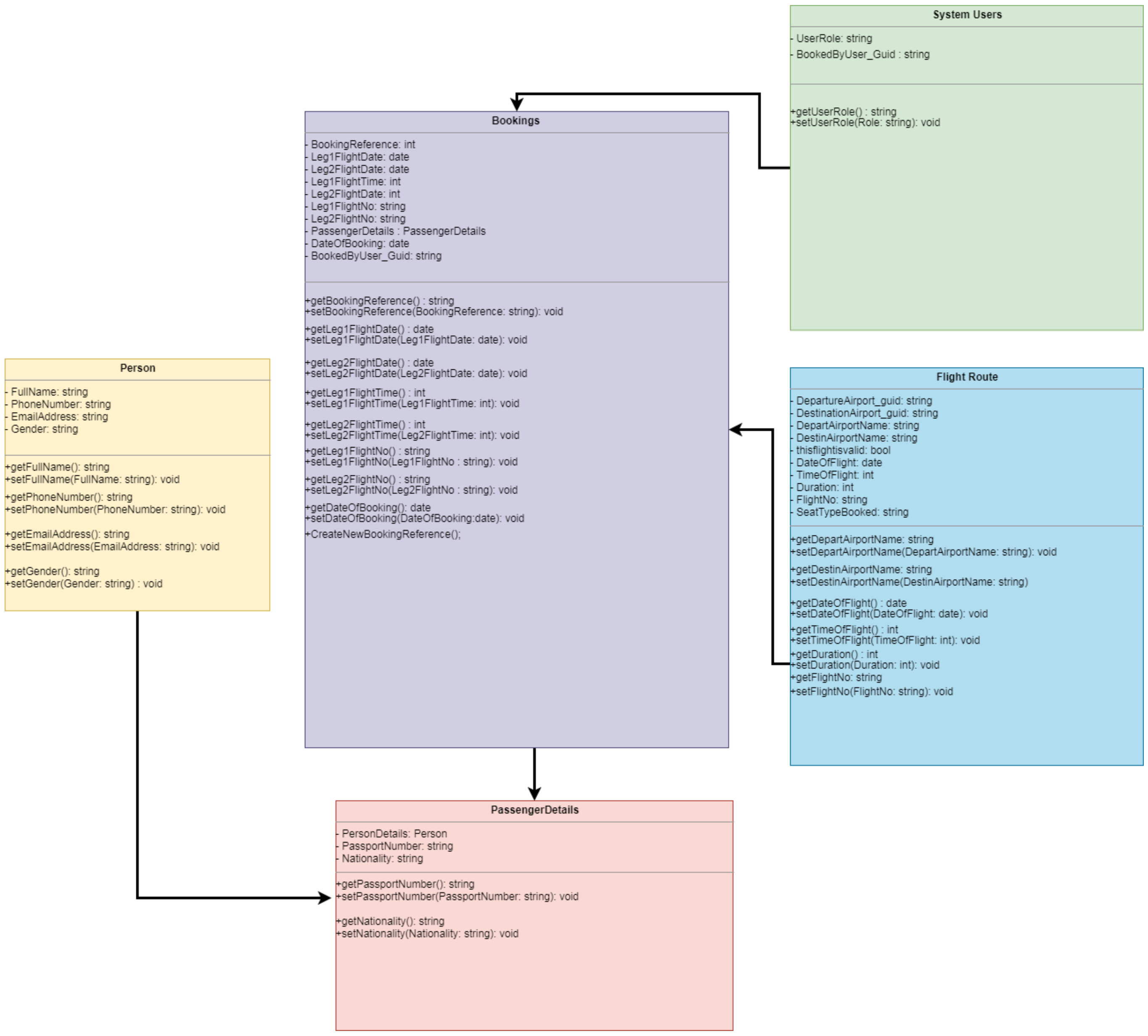

System Users
- UserRole: string
- BookedByUser_Guid : string
+getUserRole() : string
+setUserRole(Role: string): void
Bookings
- BookingReference: int
- Leg1FlightDate: date
- Leg2FlightDate: date
- Leg1FlightTime: int
- Leg2FlightDate: int
- Leg1FlightNo: string
- Leg2FlightNo: string
- PassengerDetails : PassengerDetails
- DateOfBooking: date
- BookedByUser_Guid: string
+getBookingReference() : string
+setBookingReference(BookingReference: string): void
+getLeg1FlightDate() : date
+setLeg1FlightDate(Leg1FlightDate: date): void
+getLeg2FlightDate() : date
+setLeg2FlightDate(Leg2FlightDate: date): void
+getLeg1FlightTime() : int
+setLeg1FlightTime(Leg1FlightTime: int): void
+getLeg2FlightTime() : int
+setLeg2FlightTime(Leg2FlightTime: int): void
+getLeg1FlightNo() : string
+setLeg1FlightNo(Leg1FlightNo : string): void
+getLeg2FlightNo() : string
+setLeg2FlightNo(Leg2FlightNo : string): void
+getDateOfBooking(): date
+setDateOfBooking(DateOfBooking:date): void
+CreateNewBookingReference();
Person
- FullName: string
- PhoneNumber: string
- EmailAddress: string
- Gender: string
+getFullName(): string
+setFullName(FullName: string): void
+getPhoneNumber(): string
+setPhoneNumber(PhoneNumber: string): void
+getEmailAddress(): string
+setEmailAddress(EmailAddress: string): void
+getGender(): string
+setGender(Gender: string) : void
Flight Route
- DepartureAirport_guid: string
- DestinationAirport_guid: string
- DepartAirportName: string
- DestinAirportName: string
- thisflightisvalid: bool
- DateOfFlight: date
- TimeOfFlight: int
- Duration: int
- FlightNo: string
- SeatTypeBooked: string
+getDepartAirportName: string
+setDepartAirportName(DepartAirportName: string): void
+getDestinAirportName: string
+setDestinAirportName(DestinAirportName: string)
+getDateOfFlight() : date
+setDateOfFlight(DateOfFlight: date): void
+getTimeOfFlight() : int
+setTimeOfFlight(TimeOfFlight: int): void
+getDuration() : int
+setDuration(Duration: int): void
+getFlightNo: string
+setFlightNo(FlightNo: string): void
PassengerDetails
- PersonDetails: Person
- PassportNumber: string
- Nationality: string
+getPassportNumber(): string
+setPassportNumber(PassportNumber: string): void
+getNationality(): string
+setNationality(Nationality: string): void

Figure 15: Class Diagram

CYANAIR DATA CONNECTION CLASS

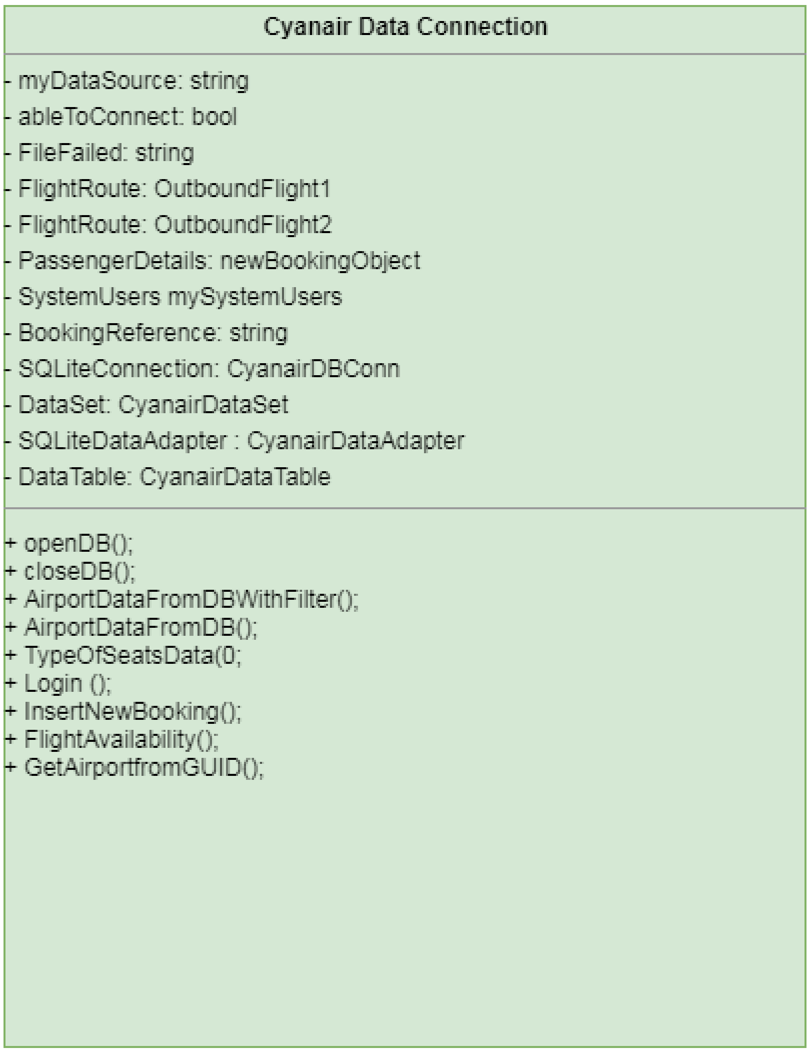

Cyanair Data Connection
- myDataSource: string
- ableToConnect: bool
- FileFailed: string
- FlightRoute: OutboundFlight1
- FlightRoute: OutboundFlight2
- PassengerDetails: newBookingObject
- SystemUsers mySystemUsers
- BookingReference: string
- SQLiteConnection: CyanairDBConn
- DataSet: CyanairDataSet
- SQLiteDataAdapter : CyanairDataAdapter
- DataTable: CyanairDataTable
+ openDB();
+ closeDB();
+ AirportDataFromDBWithFilter();
+ AirportDataFromDB();
+ TypeOfSeatsData(0;
+ Login ();
+ InsertNewBooking();
+ FlightAvailability();
+ GetAirportfromGUID();

Figure 16: Cyanair Data Connection Class

CONTEXT DIAGRAM

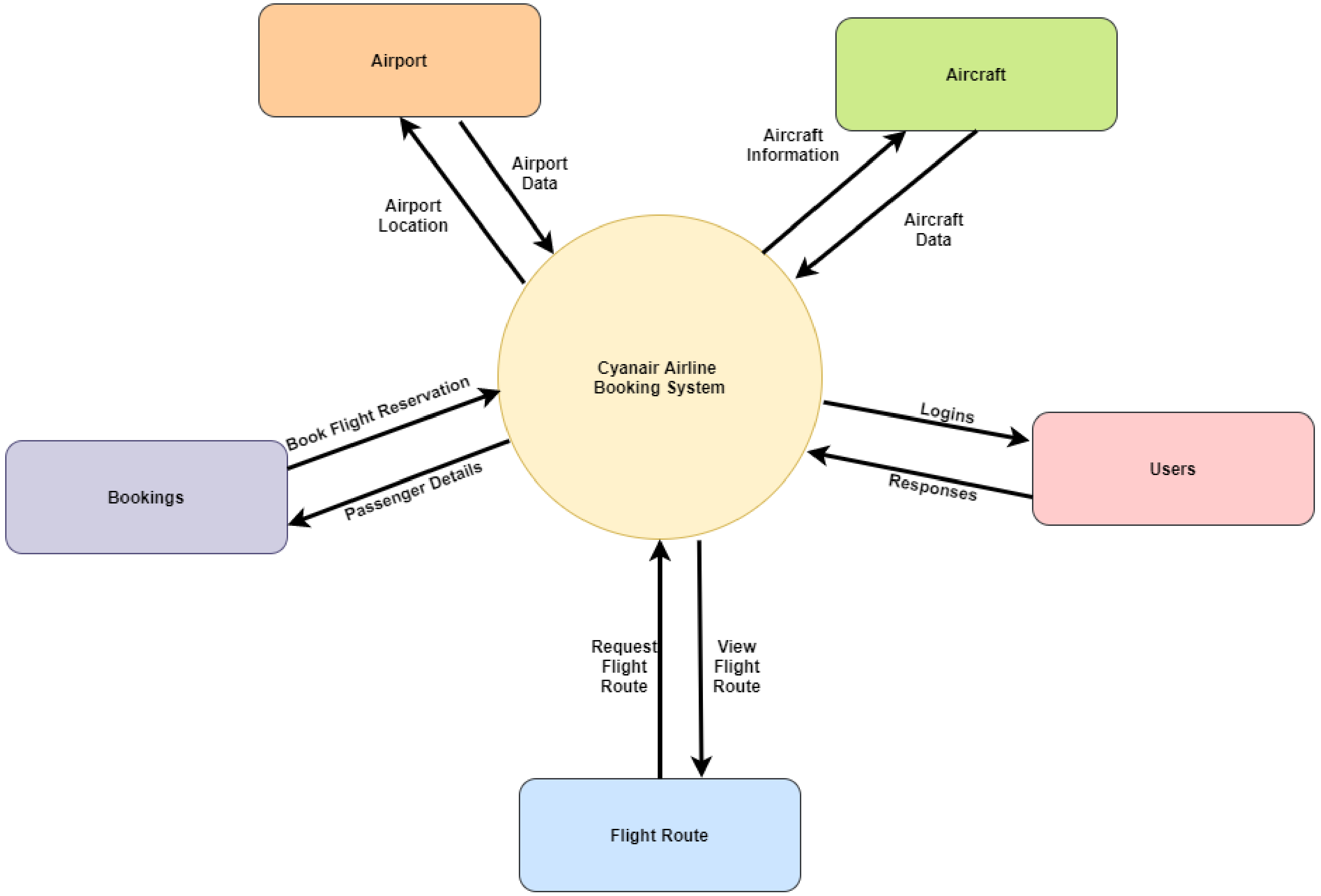


Figure 17: Context Level Diagram

LEVEL 1 DFD BOOKINGS

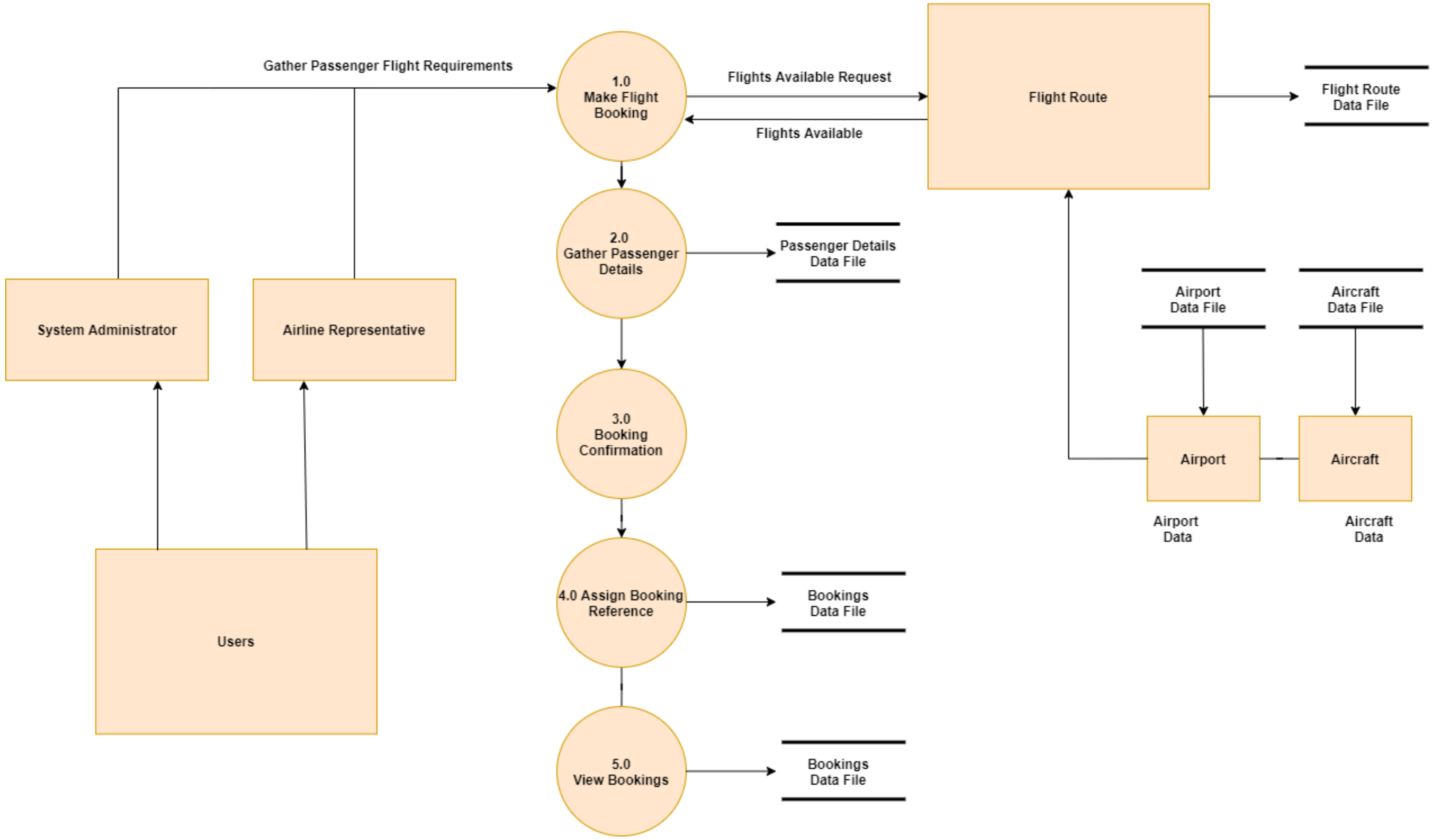


Figure 18: Level 1 DFD Booking System

LEVEL 1 DFD ADMINISTRATOR

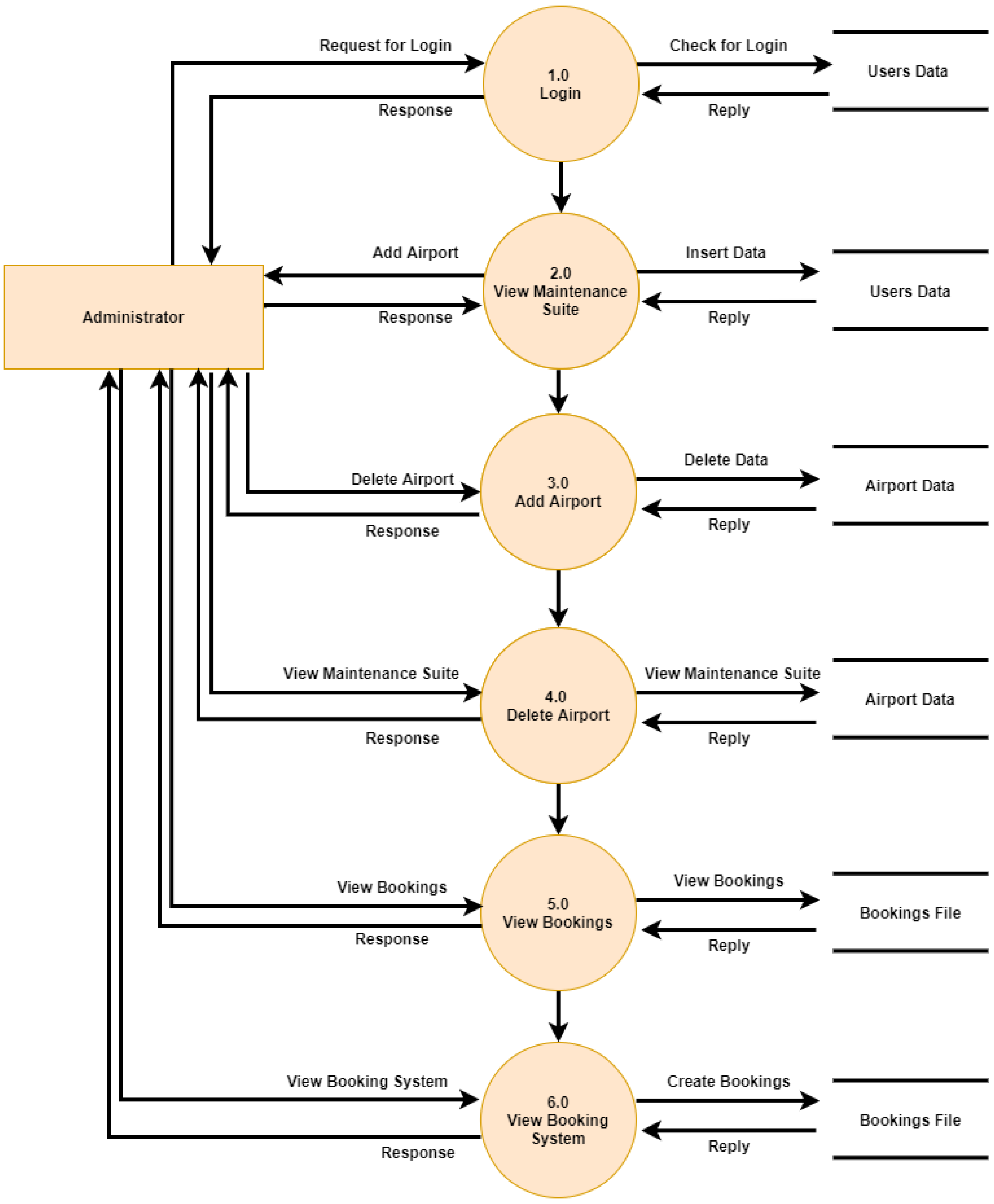


Figure 19: Level 1 DFD Administrator

LEVEL 1 DFD FLIGHT REPRESENTATIVE

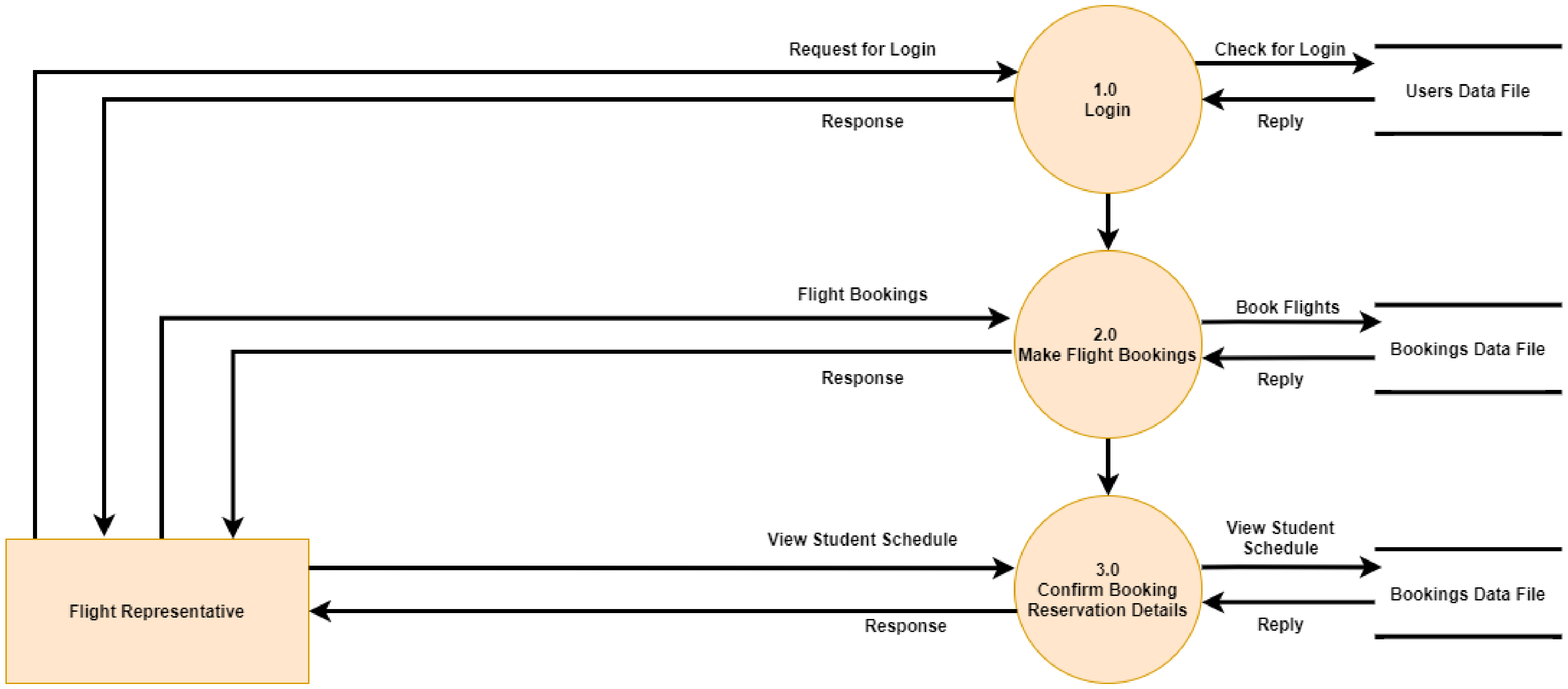


Figure 20: Level 1 DFD Flight Representative

LEVEL 1 DFD AIRPORT

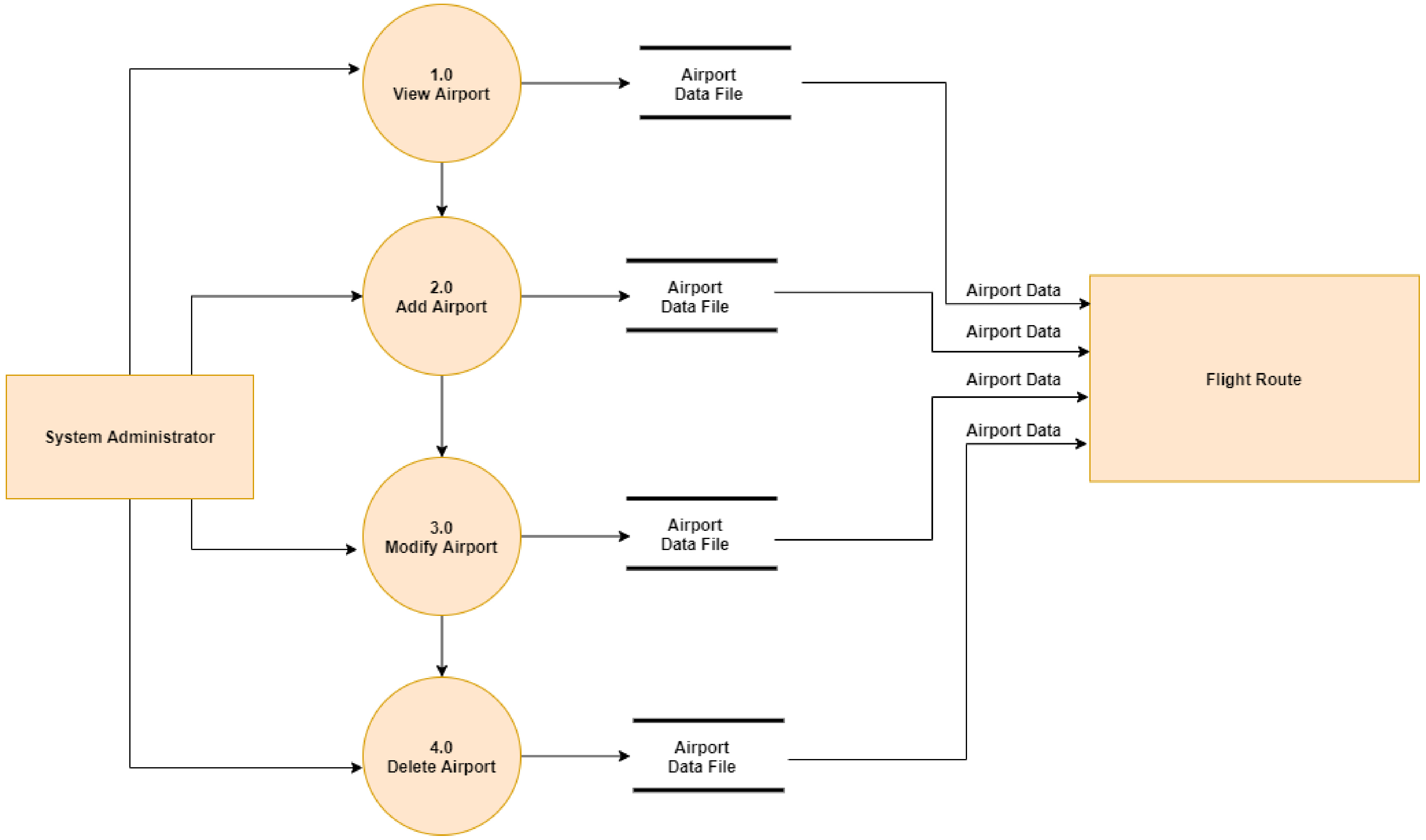


Figure 21: Level 1 DFD Airport

LEVEL 1 DFD AIRCRAFTS

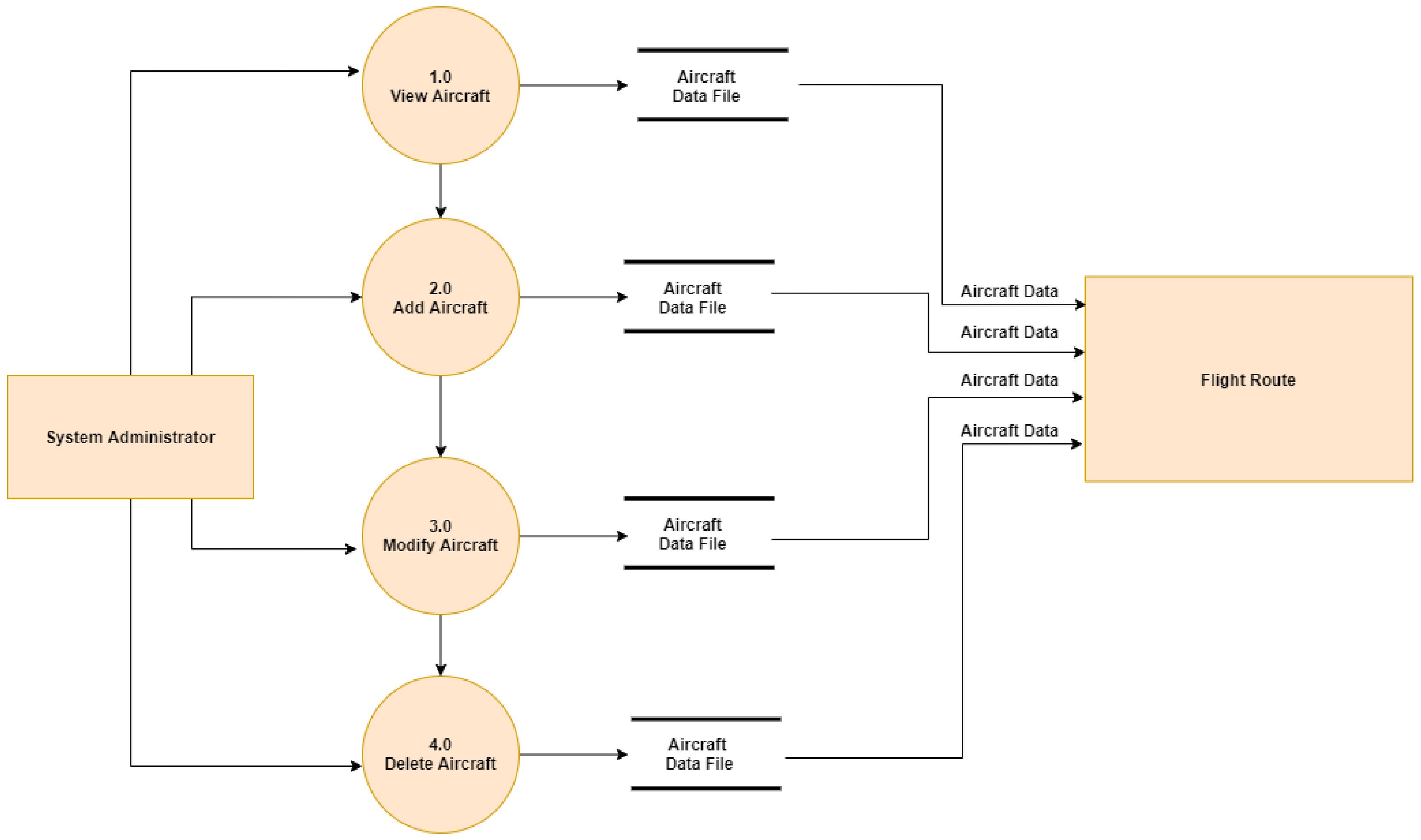


Figure 22: Level 1 DFD Aircraft

LEVEL 1 DFD USERS

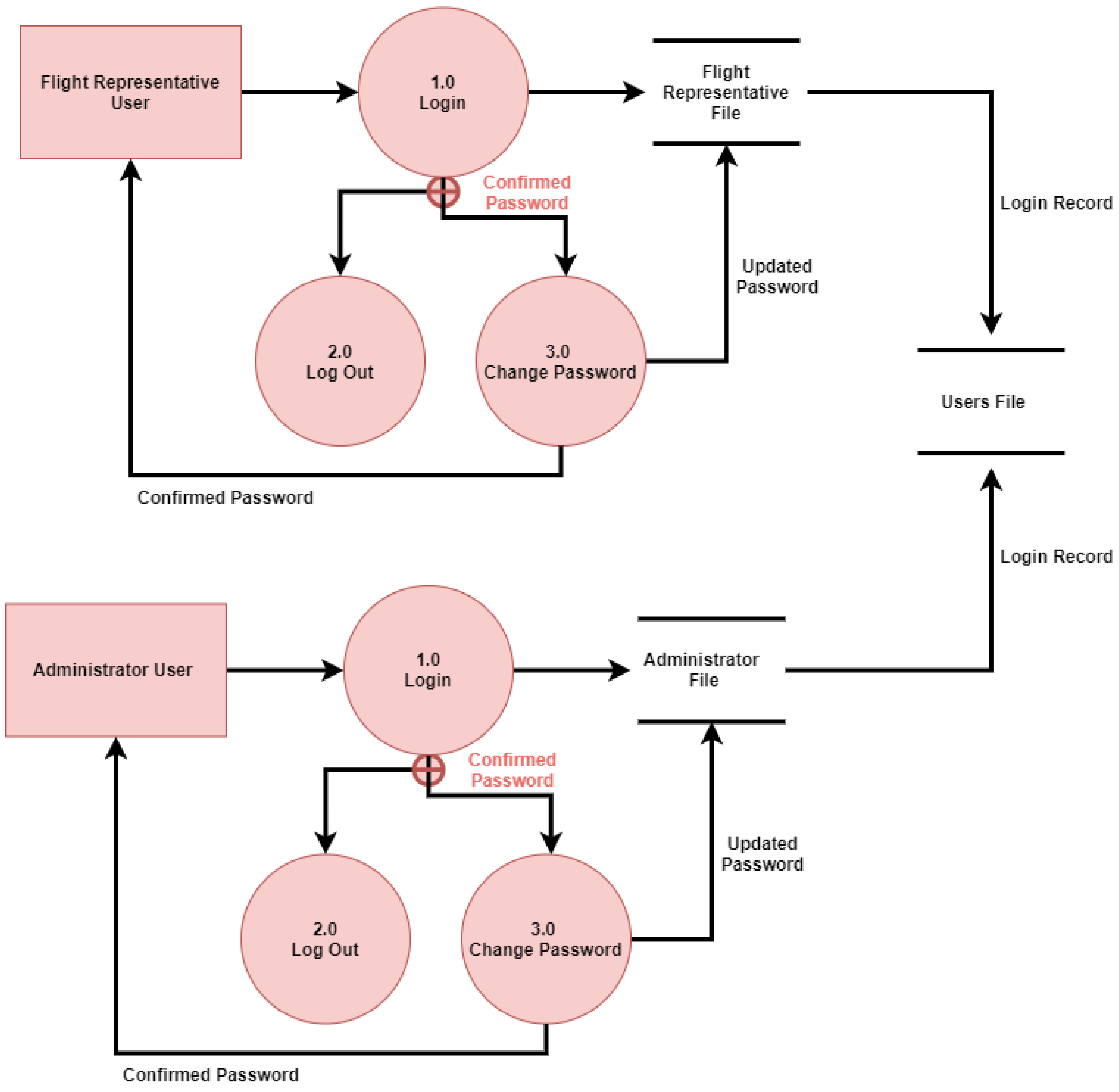


Figure 23: Level 1 DFD Users

5 FILE AND DATABASE DESIGN

As previously mentioned in this document, the management system used for the "Cyanair.db" database is DB Browser. We have found through research that this is the most ideal program to browse and create our data on, as it is easy to use and user-friendly. The main positive of this management system is that the software is free, compared to other programs that exceed our budget. In the future, we would like to upgrade this management system if possible, to an enterprise system that can handle larger data.

DB Browser for SQLite can be downloaded at: https://sqlitebrowser.org/

We are using SQLite for the connection between the database and the booking system, with the installation of additional NuGet packages on Visual Studio.

5.1 Database Management System Files

As we have created a database to store our data, we need to define a data dictionary for each of the tables within the database. The use of the data dictionary is to provide a more detailed account about each of the contents within the database. This includes the names of the fields, the type of data it represents, the size of this data, the format, an example of this data and a description of the data.

There will be six tables used within this database at present, which include:

1. ***Airport***– this table stores all of the information relating to the airport which is required for the flight information and bookings.

2. ***Booking*** – this table stores all of the information regarding a booking made using the system. This table features links from other tables.

3. ***Passenger Details*** – this table will store the passenger's details which will then be merged with the booking table in order to represent all of the information regarding a booking and the details of the passenger who has booked the flight(s).

4. ***Flight Route*** – this table stores all of the information regarding a flight route.

5. ***Aircraft*** – this table stores information regarding the aircraft. It is linked to the Flight Route table.

6. ***Users*** – this table stores information regarding the users of the program. It stores their username, password and role which is required for the login of the system.

5.1.1 Data Dictionary

TABLE: AIRPORT – DATA DICTIONARY

Field Name	Field type	Field Size	Format	Example	Description
Uid	Bigint	20		1 …	This is the unique ID for an airport
Guid	Varchar	255		1234abc-5678-de9-0fgh-123ijk45678lm9	This is the general ID for an airport
Airport	Varchar	255		Dublin Airport	This is the name of the Departure Airport
AirportCode	Varchar	3	ABC	DUB	This is the Airport Code for the Airport
Deleted	Tinyint	1		Default 0	This is to delete an Airport

Figure 24: Airport Data Dictionary

TABLE: BOOKING – DATA DICTIONARY

Field Name	Field type	Field Size	Format	Example	Description
Uid	Bigint	20		1 …	This is the unique ID for the Bookings
Guid	Varchar	255		1234abc-5678-de9-0fgh-123ijk45678lm9	This is the general ID for the booking
BookingReference	Varchar	20	CY01012020-00000	CY03062020-12345	This is the booking reference number for the reservation
DepartureLeg1Airport_Guid	Varchar	255		Dublin Airport	This is the name of the Departure Airport
DestinationLeg1Airport_Guid	Varchar	255		London Stansted	This is the name of the Destination Airport
DepartureLeg2Airport_Guid	Varchar	255		Dublin Airport	This is the name of the Departure Airport
DestinationLeg2Airport_Guid	Varchar	255		London Stansted	This is the name of the Destination Airport
Leg1FlightDate	Date	10	9999/99/99	2020/06/03	This is the date of the flight
Leg2FlightDate	Date	10	9999/99/99	2020/06/03	This is the date of the flight

Leg1FlightTime	**Numeric**	**2,2**	**00.00**	**10.15**	**This is the time of the flight**
Leg2FlightTime	**Numeric**	**2,2**	**00.00**	**10.15**	**This is the time of the flight**
Leg1FlightNo	**Varchar**	**6**	**CY0000**	**CY1234**	**This is the Flight Number for a flight**
Leg2FlightNo	**Varchar**	**6**	**CY0000**	**CY1234**	**This is the Flight Number for a flight**
PassengerDetails_Guid	**Varchar**	**255**		**1234abc-5678-de9-0fgh-123ijk45678lm9**	**This is the Guid of the Passenger Details Table**
DateOfBooking	**Date**	**10**	**9999/99/99**	**2020/06/03**	**This is the dates of the booking**
BookedByUser_Guid	**Varchar**	**25**		**KatieKeogh12**	**This is the name of the user who booked the flight – depends on the name of the representation/user who logged in to the system**
Deleted	**Tinyint**	**1**		**Default 0**	**This is to delete a Booking**

Figure 25: Booking Data Dictionary

TABLE: PASSENGER DETAILS – DATA DICTIONARY

Uid	**Bigint**	**20**		**1 …**	**This is the unique ID for the Passenger Details**
Guid	**Varchar**	**255**		**1234abc-5678-de9-0fgh-123ijk45678lm9**	**This is the general ID for the passenger details**
PassengerFullName	**Varchar**	**255**		**Thomas Swift**	**This is the name of the Passenger**
PassportNumber	**Varchar**	**255**	**PA0000000**	**PA1234567**	**This is the Passport Number of the Passenger**
Nationality	**Varchar**	**255**		**Irish**	**This is the Nationality of the Passenger**
PhoneNumber	**Varchar**	**255**		**+353 85 / +22 /**	**This is the phone number of the Passenger**
EmailAddress	**Varchar**	**255**	**xxx@xxxx.ie**	**cyanair@gmail.com**	**This is the email address of the Passenger**
Gender	**Char**	**1**	**(M/F)**	**M**	**This is the gender of the Passenger**
Deleted	**Tinyint**	**1**		**Default 0**	**This is if a passenger has been deleted or not**

Figure 26: Passenger Details Data Dictionary

TABLE: FLIGHT ROUTE – DATA DICTIONARY

Field Name	Field type	Field Size	Format	Example	Description
Uid	Bigint	20		1 …	This is the unique ID for the Flight Route
Guid	Varchar	255		1234abc-5678-de9-0fgh-123ijk45678lm9	This is the general ID for the Flight Route
DepartureAirport_guid	Varchar	255		Dublin Airport	This is the name of the Departure Airport
DestinationAirport_guid	Varchar	255		London Stansted	This is the name of the Destination Airport
Aircraft_guid	Varchar	255		1234abc-5678-de9-0fgh-123ijk45678lm9	This is the Aircraft available for the flight route
DateOfFlight	Date	12	9999/99/99	2020/06/03	This is the dates of the flight
TimeOfFlight	Numeric	2,2	00.00	10.15	This is the time of the flight
Duration	Numeric	2.1		5.5	This is the duration of the flight
FlightNo	Varchar	6	CY0000	CY1234	This is the Flight Number for a flight
Deleted	Tinyint	1		Default 0	This is to delete a Flight Route

Figure 27: Flight Route Data Dictionary

TABLE: AIRCRAFT – DATA DICTIONARY

Field Name	Field type	Field Size	Format	Example	Description
Uid	Bigint	20		1 …	This is the unique ID for the Aircraft
Guid	Varchar	255		1234abc-5678-de9-0fgh-123ijk45678lm9	This is the general ID for the Aircraft
RegistrationNumber	Varchar	10	EI0000	EI1234	This is the Registration Number of the Aircraft
Model	Varchar	255		Boeing 747 Airbus 380	This is the Aircraft available for the flight route
ClassType	Varchar	255		Economy, Business,First Class	This is the type of seats available on the plane
NoOfEconomy	Int	3		48	The number of economy seats available on the plane
NoOfBusiness	Int	3		24	The number of business seats available on the plane
NoOfFirstClass	Int	3		12	The number of economy seats available on the plane
Deleted	Tinyint	1		Default 0	This is to delete an Aircraft

Figure 28: Aircraft Data Dictionary

TABLE: USERS – DATA DICTIONARY

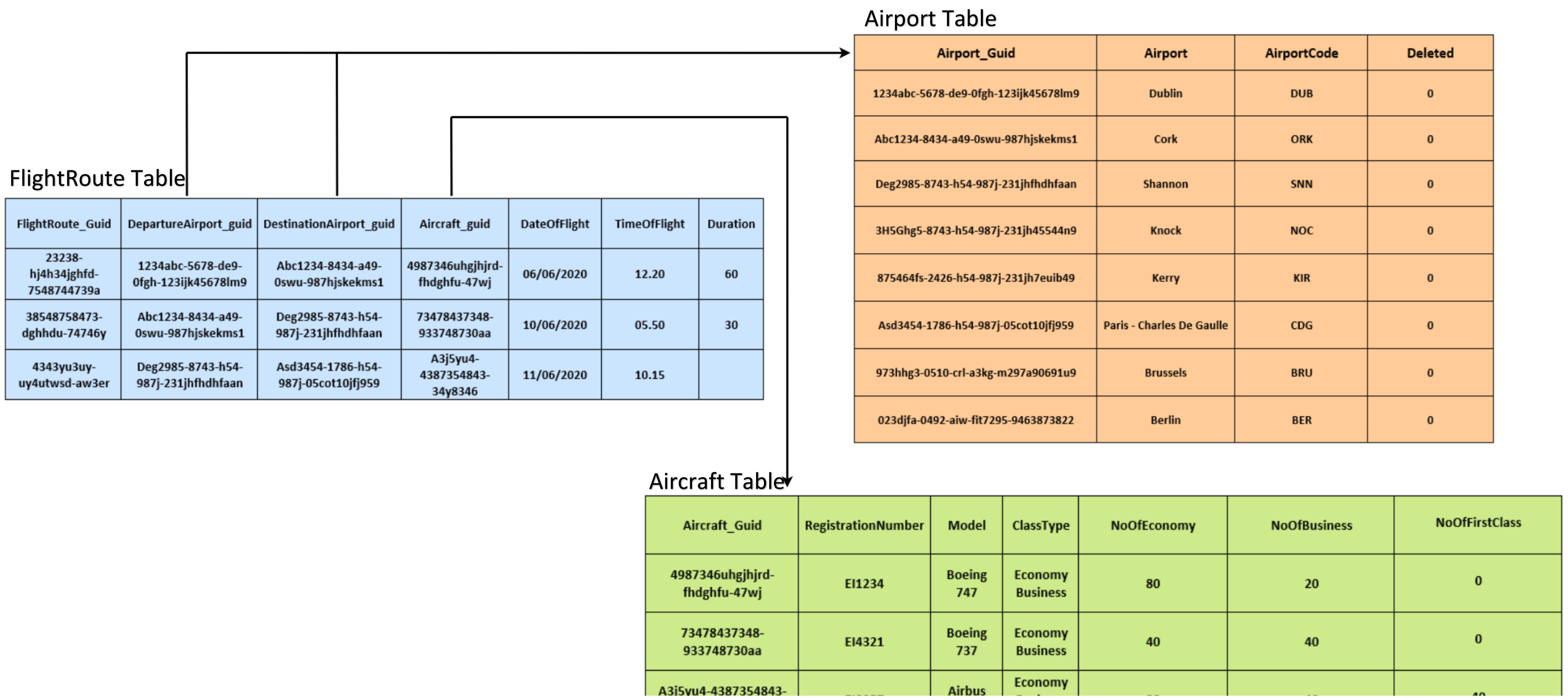
Field Name	Field type	Field Size	Format	Example	Description
Uid	Bigint	20		1 …	This is the unique ID for the attendance of users
Guid	Varchar	255		1234abc-5678-de9-0fgh-123ijk45678lm9	This is the general ID for the User
Username	Varchar	25	Username123	KatieKeogh12	This is the username of the user
Password	Varchar	25		Password12345	This is the password of the user
Role	Varchar	255		Administrator Airline Representative	This is the role of the user – Administrator and the Airline Representative that can book flights
Deleted	Tinyint	1		Default 0	This is to delete a User

Figure 29: Users Data Dictionary

5.1.2 Entity Relationship Diagram

The Entity Relationship diagram gives a graphical representation of the data in the database tables along with displaying the links between each of these tables.

FLIGHTROUTE RELATIONSHIP WITH AIRPORT AND AIRCRAFT TABLES



Airport Table

Airport_Guid	Airport	AirportCode	Deleted
1234abc-5678-de9-0fgh-123ijk45678lm9	Dublin	DUB	0
Abc1234-8434-a49-0swu-987hjskekms1	Cork	ORK	0
Deg2985-8743-h54-987j-231jhfhdhfaan	Shannon	SNN	0
3H5Ghg5-8743-h54-987j-231jh45544n9	Knock	NOC	0
875464fs-2426-h54-987j-231jh7euib49	Kerry	KIR	0
Asd3454-1786-h54-987j-05cot10jfj959	Paris - Charles De Gaulle	CDG	0
973hhg3-0510-crl-a3kg-m297a90691u9	Brussels	BRU	0
023djfa-0492-aiw-fit7295-9463873822	Berlin	BER	0

FlightRoute Table

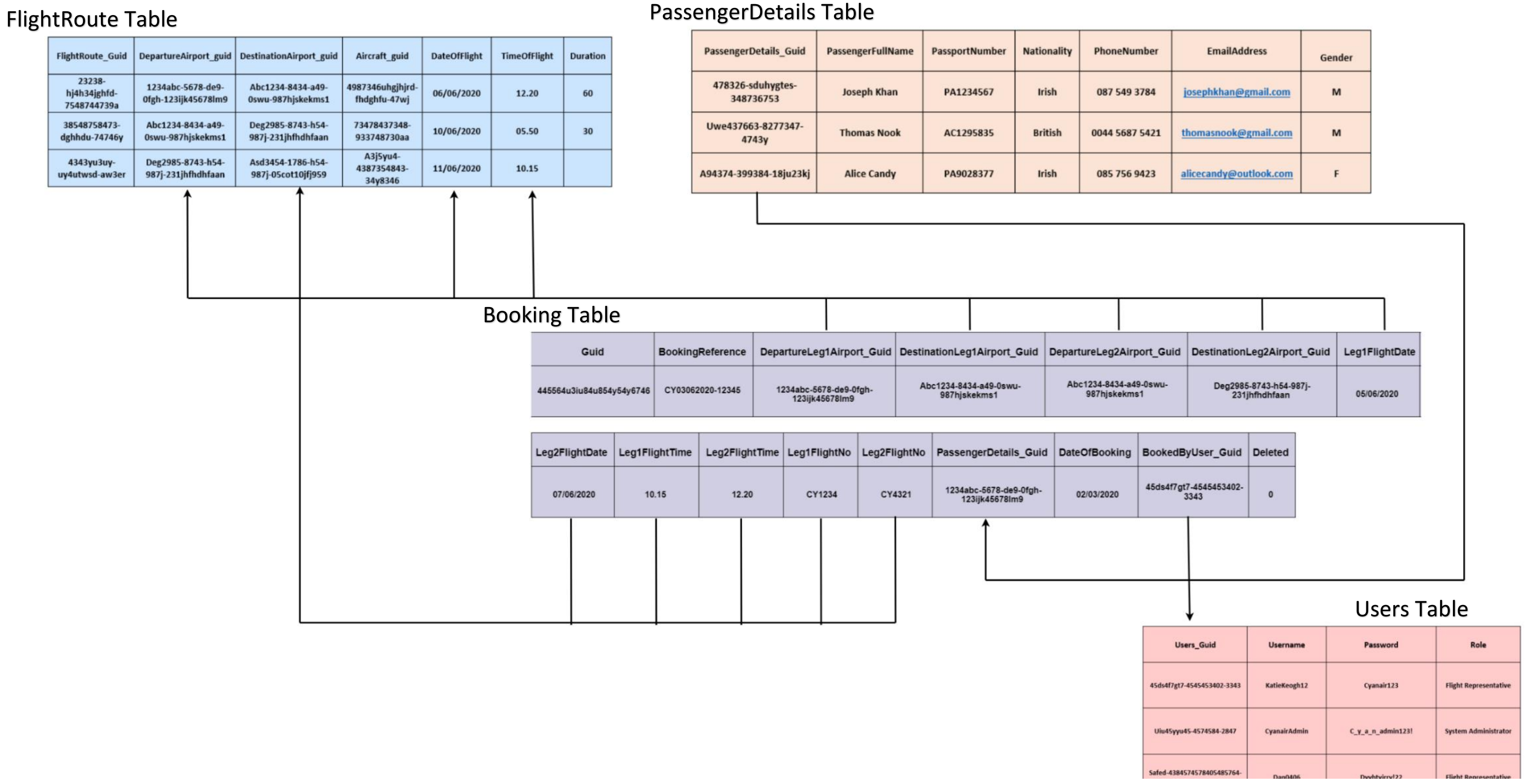
FlightRoute_Guid	DepartureAirport_guid	DestinationAirport_guid	Aircraft_guid	DateOfFlight	TimeOfFlight	Duration
23238-hj4h34jghfd-7548744739a	1234abc-5678-de9-0fgh-123ijk45678lm9	Abc1234-8434-a49-0swu-987hjskekms1	4987346uhgjhjrd-fhdghfu-47wj	06/06/2020	12.20	60
38548758473-dghhdu-74746y	Abc1234-8434-a49-0swu-987hjskekms1	Deg2985-8743-h54-987j-231jhfhdhfaan	73478437348-933748730aa	10/06/2020	05.50	30
4343yu3uy-uy4utwsd-aw3er	Deg2985-8743-h54-987j-231jhfhdhfaan	Asd3454-1786-h54-987j-05cot10jfj959	A3j5yu4-4387354843-34y8346	11/06/2020	10.15	

Aircraft Table

Aircraft_Guid	RegistrationNumber	Model	ClassType	NoOfEconomy	NoOfBusiness	NoOfFirstClass
4987346uhgjhjrd-fhdghfu-47wj	EI1234	Boeing 747	Economy Business	80	20	0
73478437348-933748730aa	EI4321	Boeing 737	Economy Business	40	40	0
A3j5yu4-4387354843-	[illegible]	Airbus	Economy	[illegible]	[illegible]	[illegible]

Figure 30: Flight Route, Airport, Aircraft Entity Relationship

BOOKING TABLE RELATIONSHIP WITH FLIGHTROUTE, PASSENGERDETAILS, USERS TABLES


FlightRoute Table

FlightRoute_Guid	DepartureAirport_guid	DestinationAirport_guid	Aircraft_guid	DateOfFlight	TimeOfFlight	Duration
23238-hj4h34jghfd-7548744739a	1234abc-5678-de9-0fgh-123ijk45678lm9	Abc1234-8434-a49-0swu-987hjskekms1	4987346uhgjhjrd-fhdghfu-47wj	06/06/2020	12.20	60
38548758473-dghhdu-74746y	Abc1234-8434-a49-0swu-987hjskekms1	Deg2985-8743-h54-987j-231jhfhdhfaan	73478437348-933748730aa	10/06/2020	05.50	30
4343yu3uy-uy4utwsd-aw3er	Deg2985-8743-h54-987j-231jhfhdhfaan	Asd3454-1786-h54-987j-05cot10jfj959	A3j5yu4-4387354843-34y8346	11/06/2020	10.15	

PassengerDetails Table

PassengerDetails_Guid	PassengerFullName	PassportNumber	Nationality	PhoneNumber	EmailAddress	Gender
478326-sduhygtes-348736753	Joseph Khan	PA1234567	Irish	087 549 3784	josephkhan@gmail.com	M
Uwe437663-8277347-4743y	Thomas Nook	AC1295835	British	0044 5687 5421	thomasnook@gmail.com	M
A94374-399384-18ju23kj	Alice Candy	PA9028377	Irish	085 756 9423	alicecandy@outlook.com	F

Booking Table

Guid	BookingReference	DepartureLeg1Airport_Guid	DestinationLeg1Airport_Guid	DepartureLeg2Airport_Guid	DestinationLeg2Airport_Guid	Leg1FlightDate
445564u3iu84u854y54y6746	CY03062020-12345	1234abc-5678-de9-0fgh-123ijk45678lm9	Abc1234-8434-a49-0swu-987hjskekms1	Abc1234-8434-a49-0swu-987hjskekms1	Deg2985-8743-h54-987j-231jhfhdhfaan	05/06/2020

Leg2FlightDate	Leg1FlightTime	Leg2FlightTime	Leg1FlightNo	Leg2FlightNo	PassengerDetails_Guid	DateOfBooking	BookedByUser_Guid	Deleted
07/06/2020	10.15	12.20	CY1234	CY4321	1234abc-5678-de9-0fgh-123ijk45678lm9	02/03/2020	45ds4f7gt7-4545453402-3343	0

Users Table

Users_Guid	Username	Password	Role
45ds4f7gt7-4545453402-3343	KatieKeogh12	Cyanair123	Flight Representative
Uiu45yyu45-4574584-2847	CyanairAdmin	C_y_a_n_admin123!	System Administrator
Safed-4384574578405485764-	Dan0406	Dvyhtyirry!22	Flight Representative

Figure 31: Booking, Flight Route, Passenger Details, Users Entity Relationship

5.2 Non-Database Management System Files

PROJECT FILES/DOCUMENTS:

1. Cyanair Booking System Software – 21/06/2020

The completed software has been created with C#. This file is an output of the system that has been created, however, it is to be used as the Cyanair Booking System – an application that will capture and store data. This program will require inputs, such as passenger's information and details, and the outputs of the system is that it will store bookings and generate a booking reference number.

2. Object Oriented Programming Report Documentation – 05/06/2020

A report outlining basic knowledge of Object-Oriented Programming will be produced to ensure that the project team has an understanding of the programming paradigm being implemented within the system, and to re-iterate the importance of OOP such as the creation of objects, with their attributes and methods. This file is an output of the system in regard to documentation.

3. Scope Sign Off Document – 15/06/2020

The scope document was in order to ensure that both the client and the project team were on the same page about the requirements and the work to be done for the required system. This document details the scope of the project, the expected deliverables, functionality, justification, constraints, assumptions, exclusions, and a sign off from the client agreeing to the scope of the work.

4. System Design Document – 19/06/2020

This document will be created aimed at the client. It will give a walkthrough of the entire project – a summary, its requirements, the scope, deliverables, point of contact, visual representations of the work to be carried out, the class diagrams to be used within the software, how the system will look and feel, etc. This document will outline all the work necessary to make this project happen.

5. Technical Design Document – 22/06/2020

This document will be much like the System Design Document – however, it is aimed at other members in the team, and people with a software background – therefore, it will be much more technical. It is not aimed at the client.

6. User Help Page – 16/06/2020

An html file will be created that will feature information regarding the booking system and information how to use it. This document will be integrated within the Cyanair Booking System and will be present on each page within the system, represented by a ‘ ❓ ’ button. It is to be created for the users of the system, to give a greater insight into how the system itself functions.

7. Test Log Report Documentation – 22/06/2020

This document will feature a test log of all the features that are in the booking system. This document will give a further insight into the system, with its functionalities and to make sure that requirements that were previously specified are being met. This test log will outline any errors that are present within the document, along with which features, and aspects of the system are working correctly and as expected.

8. System Analysis & Design Document – 14/04/2020

This document was created when a feasibility study was conducted to determine whether or not this project would be a good fit for the company. When this feasibility study was being conducted, a lot of research in regard to the system and the project was undertaken. This information will be helpful and possibly referenced within the documents of this system.

6 SYSTEM DESIGN

In regard to human-machine interface, we have produced prototype models of what we envision the ***Cyanair Booking System*** will look like. The goal we have for the graphical user interface is a simple white form, with the Cyanair logo, and the information required to make a booking. Our aim was to keep the program simple and functional, while remaining user-friendly.

Note: The following images are only prototypes of the program and may *differ to the final end product.*

LOGIN SCREEN

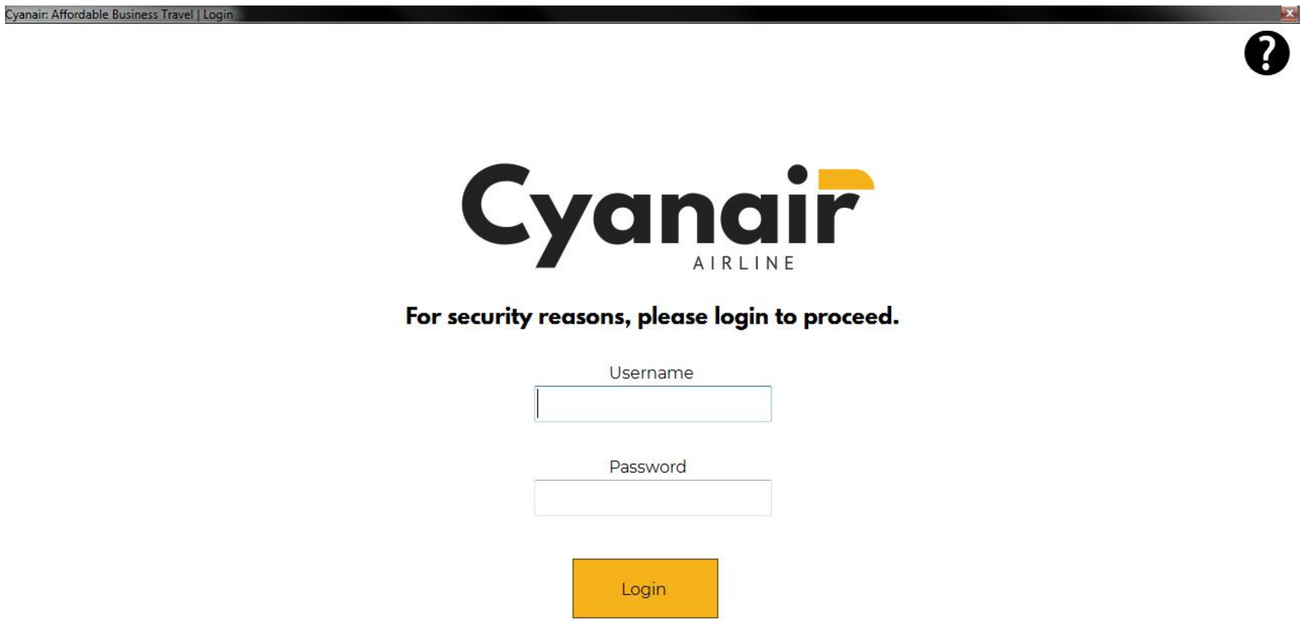

Cyanair: Affordable Business Travel | Login
Cyanair
AIRLINE
For security reasons, please login to proceed.
Username
Password
Login

Figure 32: Login Screen

Process	Details	Parameter & Additional Information
Purpose	The login screen requires the user to login in order to access the system. Based on the login provided, each login is assigned a role by the system administrator, which will either allow them to enter the system as an administrator or a flight representative. The maintenance suite can only be accessed by the administrator, so the flight representative is restricted from accessing this form.	The username and password must be valid in order for the user to enter the system.

Form Called From	System Load	This is the opening screen of the Cyanair Booking System
Data calls on Form	Login();	`= "SELECT Guid,Role FROM Users WHERE Username='"+ UserName + "' AND Password='"+ Passwordy + "';";`
Form Controls	**Labels x 3** **Textbox x 2** **Buttons x 2** **Picturebox x 1** **Panel x 1**	*lblWelcome*: informs user to login for security purposes *lblUsername*: labels the username text box *lblPassword*: labels the password text box *txtUsername*: textbox used for the data entry of the username *txtPassword*: textbox used for the data entry of the password. Password char is set to * *btnHelp*: this button is on each page of the system. Allows the user to search for help options in a new window *btnLogin*: this button when clicked will check and validate the login based on the inputs of txtUsername + txtPassword *pbxLogo*: this is a picturebox which contains an image of the Cyanair logo *pnlWelcomeScreen*: panel used to store all of the design components together
Form Behaviour	Form checks the contents inputted from txtUsername + txtPassword and matches to the	If the login is valid, the user will be able to enter the program.

	contents in the Users class within the database. It validates a login if the credentials are correct	If the login is incorrect, the user will be presented with a message on the screen to let them know their details are incorrect. If the fields are left blank and the user tries to login, a message will appear on the screen informing them that they cannot login with empty fields
Form Objects & Class	SystemUsers Class CyanairDatabaseConnection Class	Refer to Class Diagram CyanairDatabaseConnection.Login()

CHOICE SCREEN

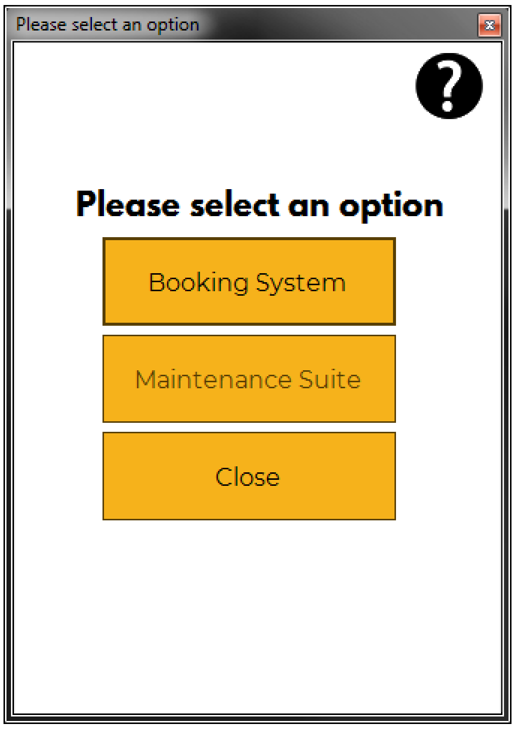


Figure 33: Choice Screen (Flight Representative View)

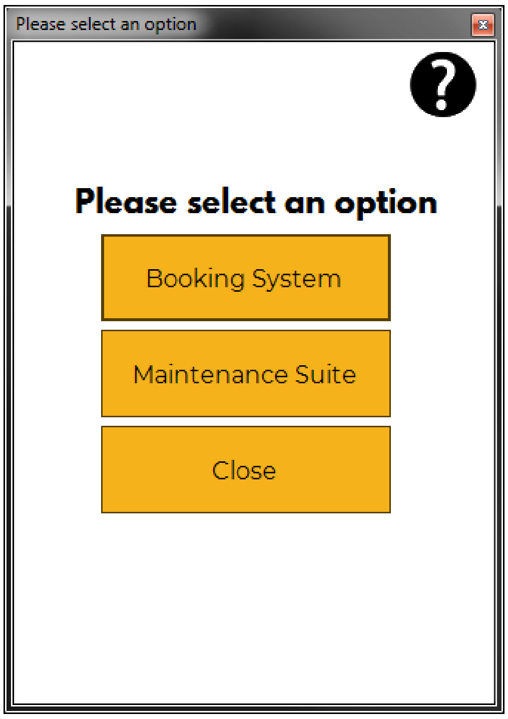


Figure 34: Choice Screen (Administrator View)

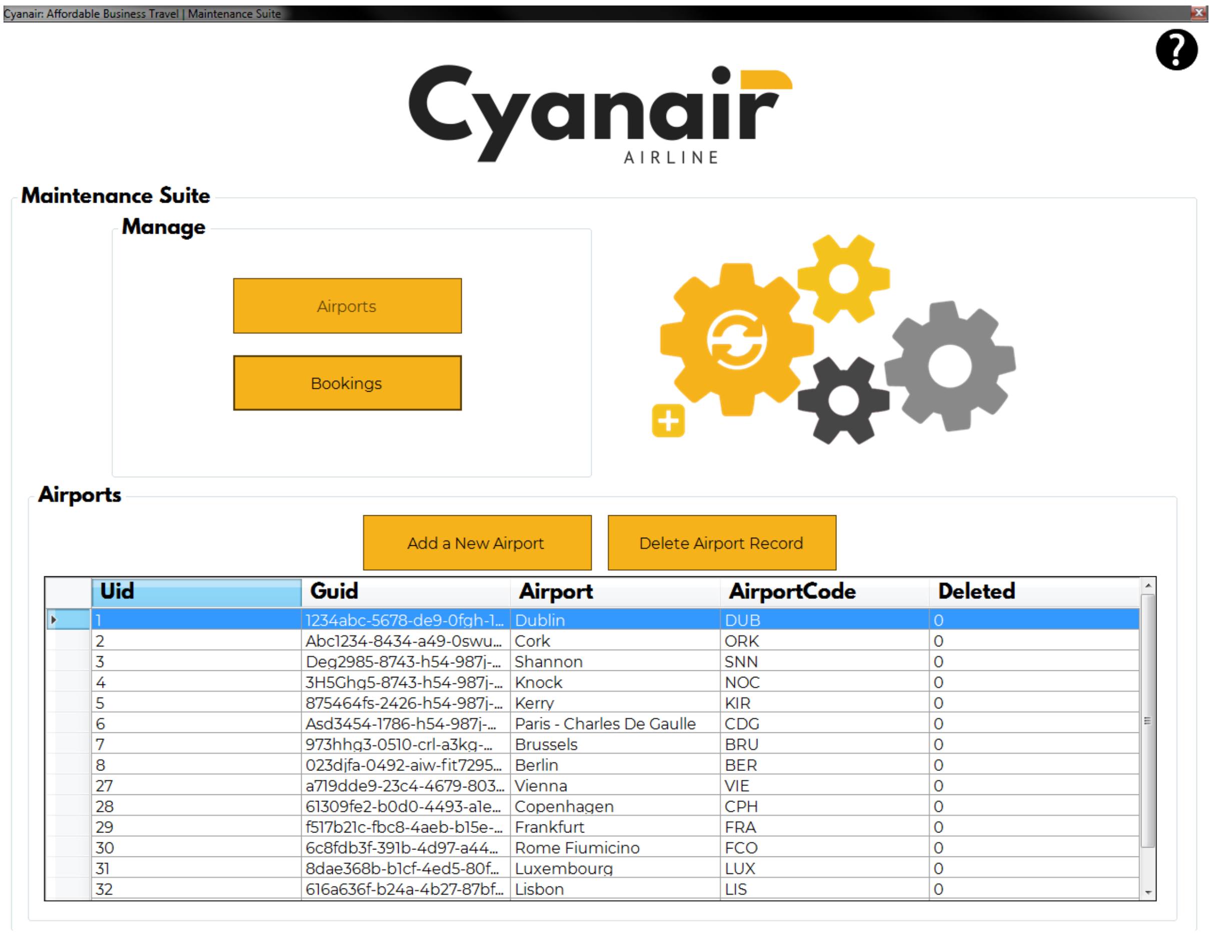
Process	Details	Parameter & Additional Information
Purpose	As mentioned previously, each user is assigned a role by the administrator – either a flight representative or an administrator. Administrator has access to the entire system, but the flight representative is restricted to the booking system only. After a successful login, users will be presented with another screen, the choice screen. This choice screen allows the user to select an option from the list of options, which should include: Booking System, Maintenance Suite and Close. However, if the user is a flight representative, they will not have the choice to select the maintenance suite. They will only be able to enter the booking system or end the program. This screen is seen as an input, as it requires the user of the system to select an option in order to proceed, which depends on the input of the user.	Options are based on role assigned to login details.

Form Called From	frmLogin	This form is called when the user has successfully logged in

Data calls on Form	frmChoice_Load	`if (mySystemUsers.UserRole == "Flight Representative")`
		`{`
		`btnMaintenanceSuite.Enabled = false;`
		`}`

Form Controls	**Buttons x4** **Label x 1** **Picturebox x 1**	btnBookingSystem: when clicked, opens the Booking System btnMaintenanceSuite: Restriced access. By default the button is set to disabled. If the user logged in is registered as an administrator, then the button will be enabled. When clicked, opens the Maintenance Suite btnClose: When clicked closes the system btnHelp: this button is on each page of the system. Allows the user to search for help options in a new window lblChoice: Informs the user to make a choice based on the selection on screen pbxLogo: this is a picturebox which contains an image of the Cyanair logo
Form Behaviour	Form will display 3 buttons. However, if the user is a Flight Representative, they will be unable to click the Maintenance Suite, as this is restricted to only the Administrators of the program.	2 buttons will open another form, or there is the option to close the system. A help button is available on screen for help if required.

Form Objects & Class	SystemUsers Class CyanairDatabaseConnection Class	Refer to Class Diagram CyanairDatabaseConnection.Login()



Uid	Guid	Airport	AirportCode	Deleted
1	1234abc-5678-de9-0fgh-1...	Dublin	DUB	0
2	Abc1234-8434-a49-0swu...	Cork	ORK	0
3	Deg2985-8743-h54-987j-...	Shannon	SNN	0
4	3H5Ghg5-8743-h54-987j-...	Knock	NOC	0
5	875464fs-2426-h54-987j-...	Kerry	KIR	0
6	Asd3454-1786-h54-987j-...	Paris - Charles De Gaulle	CDG	0
7	973hhg3-0510-crl-a3kg-...	Brussels	BRU	0
8	023djfa-0492-aiw-fit7295...	Berlin	BER	0
27	a719dde9-23c4-4679-803...	Vienna	VIE	0
28	61309fe2-b0d0-4493-a1e...	Copenhagen	CPH	0
29	f517b21c-fbc8-4aeb-b15e-...	Frankfurt	FRA	0
30	6c8fdb3f-391b-4d97-a44...	Rome Fiumicino	FCO	0
31	8dae368b-b1cf-4ed5-80f...	Luxembourg	LUX	0
32	616a636f-b24a-4b27-87bf...	Lisbon	LIS	0

Figure 35: Maintenance Suite

Process	Details	Parameter & Additional Information
Purpose	The maintenance suite screen provides inputs to the system. As per the requirements and specifications, the maintenance screen is to be able to add an airport – which is an input to the system as it requires the user (administrator) to provide the details in order to create a new airport. This screen is also allows the user to view both the airport information, as well as the booking information. It displays the information to the user. As this current graphical user interface is only a prototype, we can already see that the design of this screen in particular needs to be updated, as	The user can only access this form if they have logged in successfully as an administrator. They will have the option to view the maintenance suite in the choice form enabled, whereas other users such as the flight representatives will not be able to click this option.

	there is no need for the next, last, first and previous buttons, as all of the information is to be presented in the representation of a form in the data grid viewer. This screen also needs somewhere for the user to be able to input a new airport, so it is important to keep in mind that this is still only a prototype, but the graphical user interface can be improved.	
Form Called From	frmChoice	This form is called from the Choice Form based on the selection of the user.
Data calls on Form	1. AddAirport(); 2. ReloadDataINDGV(); 3. LoadDataFromAirportsInDVG(); 4. LoadDataFromBookingsINDGV(); 5. DeleteRow(); 6. ViewAirports();	1. `"INSERT INTO Airport (Guid,Airport,AirportCode) VALUES('" + AirportGuid.ToString() + "','" + txtAirportName.Text + "','" + txtAirportCode.Text + "')";` 2. `"SELECT * FROM Airport;";` 3. `"SELECT * FROM Airport";` 4. `"Select * FROM PassengerDetails LEFT JOIN Booking ON PassengerDetails.Uid = Booking.Uid; ";` 5. `"DELETE FROM Airport WHERE Uid=" + dgvAirports.SelectedRows[0].Cells[0].Value.ToString()` 6. `"SELECT * FROM Airport";`

Form Controls	**Buttons x 7** **Groupbox x 4** **Label x 5** **Textbox x 2** **Panel x 3** **Picturebox x2**	*btnAirports*: When clicked, this will allow the *grpAirportCode* to open which displays the option add/delete airports and to view the airports in a data grid viewer. *btnBookings*: When clicked, this will allow the *grpBookings* to open which displays the bookings made in a data grid viewer. *btnSave*: This button will only appear after the user has clicked *btnAirports* and *btnAddAirport*. It will allow the user to input an airport name and airport code to *txtAirportName + txtAirportCode.* *btnDelete*: This button will only appear after the user has clicked *btnAirports* and *btnDeleteRecord*. It will provide instructions and then allow the user to delete a row they have selected from the database. *btnAddAirport*: this button will only be visible after btnAirports has been selected. When clicked, it will show the *grpAirportManagement* groupbox which will show the *pnlAddAirport* to allow the user to input information. *btnHelp*: this button is on each page of the system. Allows the user to search for help options in a new window *grpMaintenanceSuite*: this groupbox contains all of the design components of the form

		grpManage: this groupbox contains the buttons relating to either the airports or the bookings. The buttons in this groupbox will either pull up the information and datagridviewers based on the selection of the user. *grpAirportManagement*: this groupbox will only appear once the btnAirport button has already been clicked, and the user selects to add or delete an airport. *grpBookings*: this groupbox will appear when the user has selected the btnBookings *grpAirportCode*: this groupbox will appear when the user has selected the btnAirport *lblAirportName*: this label is stored in the *pnlAddAirport* which relates to the textboxes to add an airport name/code *lblAirportCode*: this label is stored in the *pnlAddAirport* which relates to the textboxes to add an airport name/code *lblDelete*: this label provides information relating to how to delete a record *lblDelete1*: this label provides information relating to how to delete a record *lblDelete2*: this label provides information relating to how to delete a record

		txtAirportName: this textbox allows the user to input an airport name used for when adding an airport *txtAirportCode*: this textbox allows the user to input an airport code used for when adding an airport *pnlMaintenanceSuite*: this panel contains all of the design components within the form *pnlDeleteAirport*: this panel contains all of the design components relating to deleting an airport record *pnlAddAirport*: this panel contains all of the design components relating to adding an airport record *pbxLogo*: this is a picturebox which contains an image of the Cyanair logo *pbxMaintenance:* this is a picturebox that displays a maintenance image
Form Behaviour	When loaded, the form displays the Cyanair Logo, the grpManage box which contains the buttons to choose between the Airports and Bookings. There will also be a Maintenance image.	When the Bookings option has been clicked, the form will display the grpBookings groupbox which also contains it's data grid viewer to display the bookings records. When the Airports option has been clicked, the grpAirportCode will be displayed, and two buttons which have further choices to add or delete airports and their respective functions.

Form Objects & Class	CyanairDatabaseConnection Class	CyanairDatabaseConnection.AddAirport();
		CyanairDatabaseConnection.ReloadDataINDGV();
		CyanairDatabaseConnection.LoadDataFromAirportsInDVG();
		CyanairDatabaseConnection.LoadDataFromBookingsINDGV();
		CyanairDatabaseConnection.DeleteRow();
		CyanairDatabaseConnection.ViewAirports();

BOOKING SCREEN

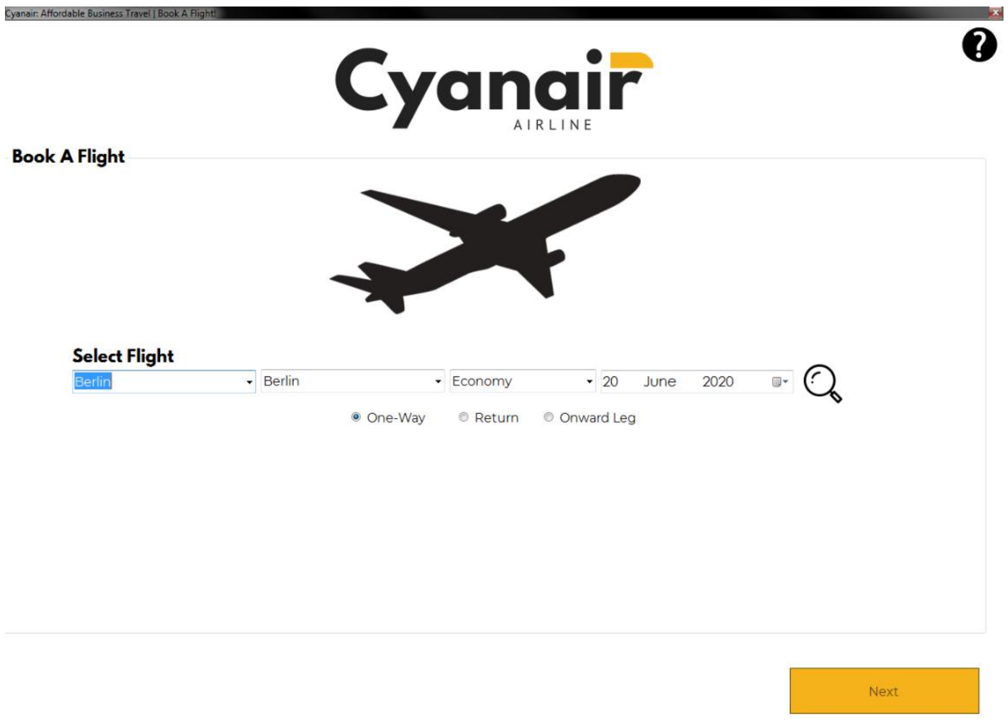
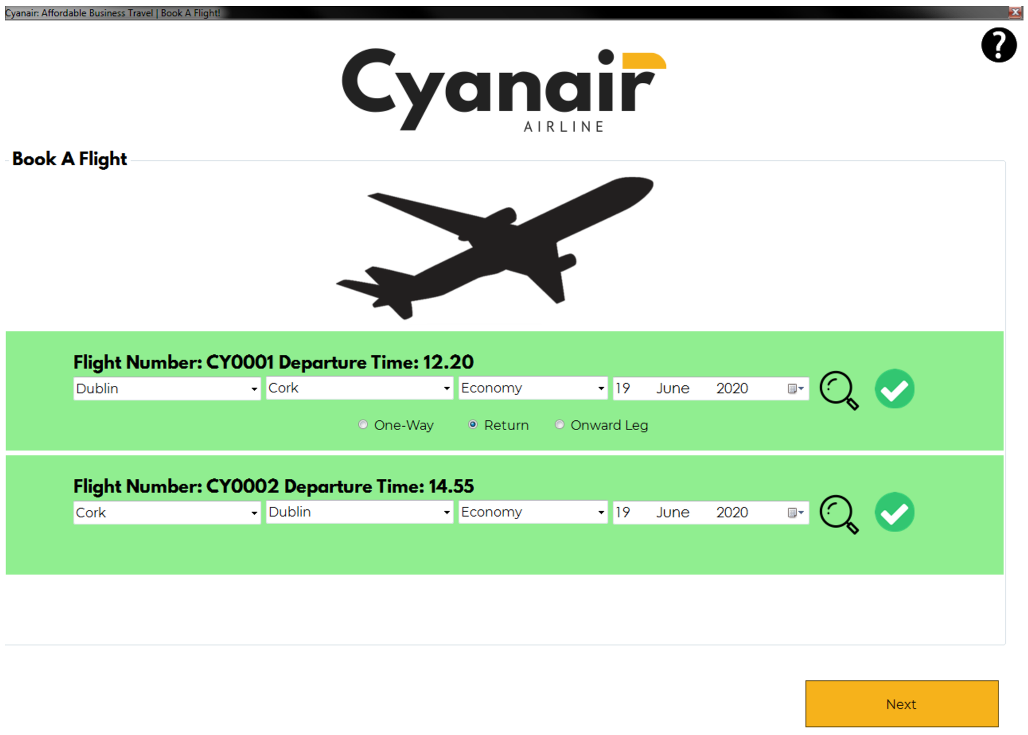


Figure 36: Booking Screen



Flight Number: CY0001 Departure Time: 12.20
Dublin | Cork | Economy | 19 June 2020
One-Way | [✓ Return] | Onward Leg

Flight Number: CY0002 Departure Time: 14.55
Cork | Dublin | Economy | 19 June 2020

Figure 37: Booking Screen

Process	Details	Parameter & Additional Information
Purpose	If the user selects the booking system from the choice screen, they will be presented with the flight booking screen. This screen intends to allow the flight representative to select flight(s) on behalf of the passenger. The passenger should have the option of a flight that is one-way, return flight, or an onward leg. Each flight option requires a departure flight, a destination flight, a type of seat from the options – Economy, Business or First Class – and a date. Once these have all been selected, the flight representative should click the 🔍 button to search for the flight availability. If the flight is available, the panel of the respective flight (e.g. leg 1 – one way) should change to green, and the 🔍 button should change to ✓. However, if the flight is not available, ✖ should appear instead of either of the images. The next button will not be able to be clicked until the flight has been confirmed. The same applies to if the flight is a return or an onward leg. The user will not be able to progress with the program until both of the flights are confirmed as available. When a return flight is selected, the departure for the return flight should automatically populate to the destination that was selected from the first flight. The destination for the return flight should also automatically populate to the first leg's departure airport. When the onward leg has been selected, the departure airport should be automatically populated with the destination from the first flight. For all of the flights, if it is one-way, return or onward leg, they should all automatically remove an airport when the airport has been selected as a destination/departure, i.e. meaning that a passenger cannot fly from Dublin to Dublin. Once the flights have been selected and confirmed, the user can progress to the next page. This screen in the system is an input as it gathers and inputs the flights required on behalf of the passenger, and stores it within an object, that can be carried on from screen to screen, until it is required to be stored into the database.	User can book flights – one-way, return or onward leg on behalf of passenger. Flight must be confirmed as valid in order to progress with booking.

Form Called From	frmChoice	This form is called from the Choice Form based on the selection of the user.
Data calls on Form	1. myNewCyanDatabaseConnection.AirportDataFromDB(); 2. myNewCyanDatabaseConnection.AirportDataFromDBWithFiler(); 3. TypeOfSeatsData();	1. `"SELECT * from Airport ORDER BY Airport ;";` 2. `"SELECT * from Airport WHERE Guid != '"+ ExcludeAirportFromList + "' ORDER BY Airport ;";` 3. `"SELECT * from SeatClass ;";`
Form Controls	**Buttons x8** **Groupbox x 1** **Panel x3** **Combobox x 6** **DateTimePicker x 2** **Radio Button x3 3** **Picturebox x 2**	*pbxLogo*: this is a picturebox which contains an image of the Cyanair logo *pbxPlane*: this is a picturebox which contains an image of an airplane *grpBookAFlight*: this groupbox contains all of the design components relating to the booking of a flight *pnlBookingSystem*: this panel contains all of the design components in the form *pnlLeg1*: this panel contains all of the design components to book one flight *pnlLeg2*: this panel contains all of the design components to book a return/onward leg *cmbDepartureLeg1*: Departure Airport options are stored here

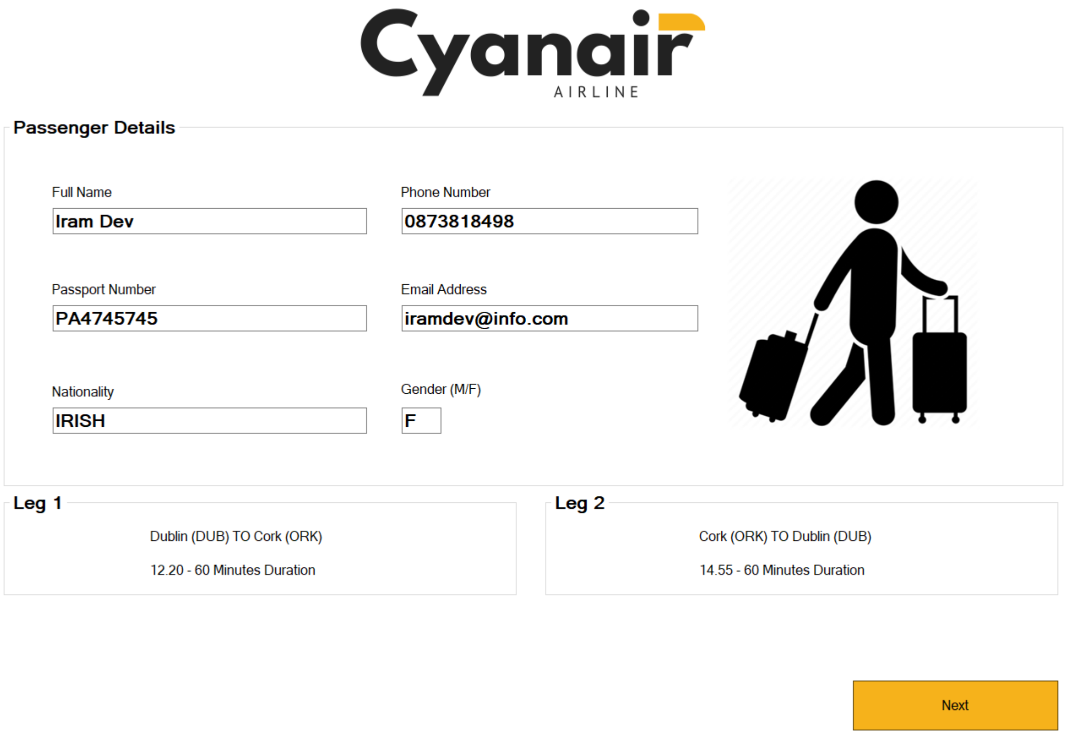
		cmbDestinationLeg1: Destination Airport options are stored here *cmbDepartureLeg2*: Departure Airport options are stored here *cmbDestinationLeg2*: Destination Airport options are stored here *cmbTypeOfSeatLeg1*: Seat class options are stored here *cmbTypeOfSeatLeg2*: Seat class options are stored here *dtpFlightDateLeg1*: Allows date for a flight to be selected *dtpFlightDateLeg2*: Allows date for a flight to be selected *btnSearchAvailabilityFlight1*: includes image which will search in the database if the flight exists with the details selected by the user. If it does exist, a confirmation btn will appear next to it. If it does not exist, an error message and btn will appear. *btnSearchAvailabilityFlight2*: includes image which will search in the database if the flight exists with the details selected by the user. If it does exist, a confirmation btn will appear

		next to it. If it does not exist, an error message and btn will appear. *btnFlightUnavailable1*: If flight details are not valid and do not exist, this image will appear *btnFlightUnavailable2*: If flight details are not valid and do not exist, this image will appear *btnFlightAvailable1*: If flight details are valid and exist, this image will appear. This is required for the flight(s) in order for the user to progress to the next page. *btnFlightAvailable2*: If flight details are valid and exist, this image will appear. This is required for the flight(s) in order for the user to progress to the next page. *btnHelp*: this button is on each page of the system. Allows the user to search for help options in a new window *btnNext*: By default this button is set to disabled. It will only become enabled when the flight(s) have been confirmed as available. It will take the user to the passenger details form to input their details in regards to the booking. *rbtOneWay*: Radio button choice. Will allow the user to only book a single flight *rbtReturn*: Radio button choice. Will allow the user to book a return flight. Departure airport on the

		return flight will automatically select the destination from leg 1, and the destination on leg 2 will automatically select the departure on leg 1. *rbtOnwardLeg*: Radio button choice. Will allow the user to book an onward leg. The departure flight on this leg will automatically select the destination flight from leg 1.
Form Behaviour	The form will allow the user to book flights – one-way, return or onward leg on behalf of passenger. Flight must be confirmed as valid in order to progress with booking.	The user must select the search button to search for flight availability. If a flight is not available, an error will appear on screen. If the flight is available, a confirmation image will appear on the screen. The user will not be able to progress to the next page until the flight(s) have been confirmed as available.

Form Objects & Class	FlightRoute Class
	
	CyanairDatabaseConnection Class
		Refer to Class Diagram
		
		CyanairDatabaseConnection.AirportDataFromDB();
		
		CyanairDatabaseConnection. AirportDataFromDBWithFiler ();
		
		CyanairDatabaseConnection. TypeOfSeatsData ();

PASSENGER DETAILS



Iram Dev

0873818498

PA4745745

iramdev@info.com

IRISH

F

Dublin (DUB) TO Cork (ORK)

12.20 - 60 Minutes Duration

Cork (ORK) TO Dublin (DUB)

14.55 - 60 Minutes Duration

Figure 38: Passenger Details

Process	Details	Parameter & Additional Information
Purpose	After the user has successfully selected and confirmed the flights required, the next screen on the system requires further input of the passenger's details. This screen will require the user to gather the passenger's details, such as their full name, passport number, nationality, phone number, email address and gender information. The system will not allow the user to progress until all of the details have been selected. If any of the fields have been left blank, a message should appear to let the user to know what fields have not been completed. Though not in the prototype, the system may also carry the information regarding the booking which will be stored in a Booking object – and can be displayed at the bottom of the screen to give a little information in regards to the flight(s) that was/were selected, before the full information is provided on the following screen.	User must fill in passenger's details relating to whom the booking is for. The user cannot leave any fields empty or they will not be able to progress within the program.
Form Called From	frmBookingSystem	This form is called from the Booking System.

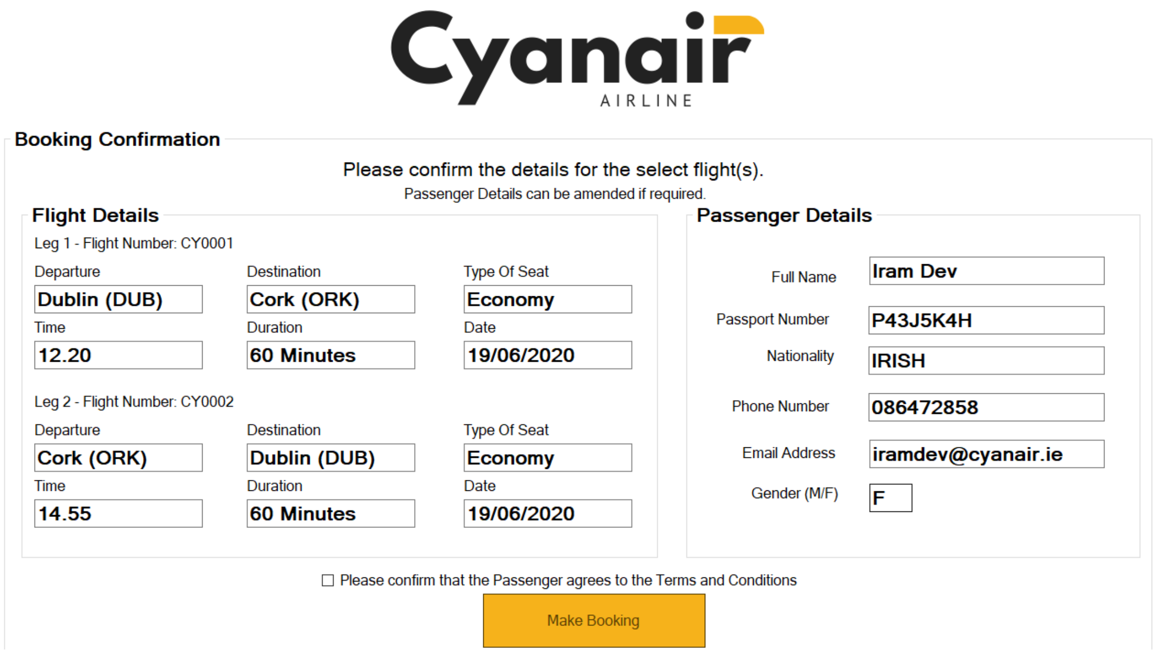
Data calls on Form	Public CyanairDataConnection myNewCyanDatabaseConnection;
	
	public FlightRoute OutboundFlight01;
	
	public FlightRoute OutboundFlight02;	

Form Controls	**Buttons x 2** **Label x 10** **Picturebox x 2** **Groupbox x 3** **Panel x 1** **Textbox x 6**	*pbxLogo*: this is a picturebox which contains an image of the Cyanair logo *pbxPassenger*: this is a picturebox which contains an image of a passenger *btnNext*: when click confirms that all fields have been field, and opens the next page, booking confirmation *btnHelp*: this button is on each page of the system. Allows the user to search for help options in a new window *grpPassengerDetails*: Contains all of the design components relating to the passengers details *grpLeg1*: displays all of the design components relating to information about the leg 1 flight *grpLeg2*: displays all of the design components relating to information about the leg 1 flight *lblOutwardFlightLeg1DepDest, lblOutwardFlightLeg1TimeDur*: labels used to display the leg 1 departure, destination, time and duration

		lblOutwardFlightLeg2DepDest, lblOutwardFlightLeg2TimeDur: labels used to display the leg 2 departure, destination, time and duration *lblFullName*: labels the textbox for the full name of the passenger *lblPassportNumber*: labels the textbox for the passenger numberof the passenger *lblNationality*: labels the textbox for the nationality of the passenger *lblPhoneNumber*: labels the textbox for the phone number of the passenger *lblEmail*: labels the textbox for the email of the passenger *lblGender*: labels the textbox for the gender of the passenger *txtFullName*: this textbox allows for the input from the user *txtPassportNumber*: this textbox allows for the input from the user *txtNationality*: this textbox allows for the input from the user

		txtPhoneNumber: this textbox allows for the input from the user *txtEmailAddress*: this textbox allows for the input from the user *txtGender*: this textbox allows for the input from the user
Form Behaviour	The form will input the details of the passenger. It requires that all fields are filled in order to progress. Also displays information about the flight(s) selected.	All fields must be inputted. If the fields are not all completed, an error message will appear on screen. If they are, the program will continue to the next page.
Form Objects & Class	PassengerDetails Class Person Class FlightRoute Class CyanairDatabaseConnection Class	Refer to Class Diagram

BOOKING CONFIRMATION


Cyanair AIRLINE

Booking Confirmation

Please confirm the details for the select flight(s).
Passenger Details can be amended if required.

Flight Details

Leg 1 - Flight Number: CY0001

Departure: Dublin (DUB)	Destination: Cork (ORK)	Type Of Seat: Economy
Time: 12.20	Duration: 60 Minutes	Date: 19/06/2020

Leg 2 - Flight Number: CY0002

Departure: Cork (ORK)	Destination: Dublin (DUB)	Type Of Seat: Economy
Time: 14.55	Duration: 60 Minutes	Date: 19/06/2020

Passenger Details

Full Name	Iram Dev
Passport Number	P43J5K4H
Nationality	IRISH
Phone Number	086472858
Email Address	iramdev@cyanair.ie
Gender (M/F)	F

☐ Please confirm that the Passenger agrees to the Terms and Conditions

Figure 39: Booking Confirmation

Process	Details	Parameter & Additional Information
Purpose	The final page within the booking system is the booking confirmation screen, which appears after the user has already inputted both the flight details and the passenger details. This screen will “confirm” this information, by letting the flight representative ensure and check that all of the information is correct. This screen will allow the passenger details to be amended if necessary, however, the flight details will only be allowed to be viewed, so the field will be presented as ‘read-only’. This is because the flight details require an availability check. Once the user has confirmed that the information in regards to the passenger is correct, they are presented with a checkbox that asks the user to confirm that the passenger agrees to the terms and conditions. The full terms and conditions can be read within the help system integrated within the program should the passenger enquire. The user is not able to make the booking if the passenger does not agree to the terms and conditions.	User is able to view all of the information regarding the booking here. This includes the information for the flight(s) selected and all of the passenger details that were inputted. The user is able to amend the passenger details if required, however, the flight details cannot be amended. The user must check the checkbox in order to create the booking. Once the make booking button has been selected, a booking reference number will be generated and the user will be informed that their booking has been inputted into the database. The user can then close the program.

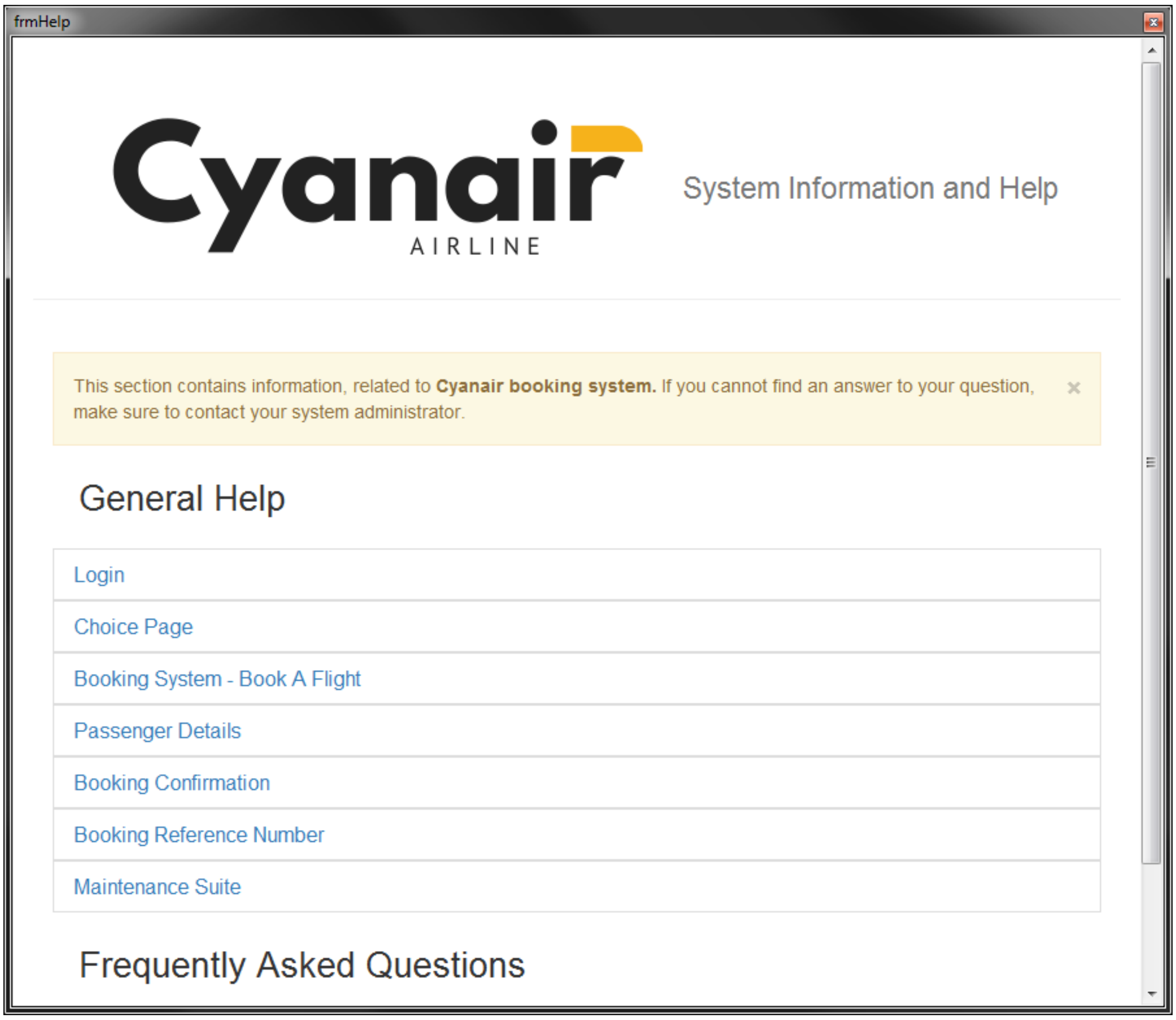
	Once the 'make button' is clicked, another message should appear on screen asking the user if they are sure they would like to create a new booking, requiring either a yes or no answer. If no is selected, it will close the message box and nothing else will change, the user can always just click the button again once they have made the amendments that they may require. If yes is selected, then the program will generate and assign a booking reference to the booking that has just been created, along with a message appearing on screen that displays the booking reference, and that lets the user know that the booking has been successfully created and inputted into the booking system database. As this screen supplies a booking reference number – this is an output of the system. Once the booking has been completed, and the booking reference number supplied, the program will allow the user to close the booking system. If the user would like to make another booking they can run the program again. We have identified that this may not be the best practice, but if the project was to be extended or expanded within the future, and there was more time and resources available, we would look to bring the user directly back to the option page so that they would have the option to enter the booking screen again to create a new booking, or to end the program.	
Form Called From	frmPassengerDetails	This form is called from the Passenger Details form.

Data calls on Form	`InsertNewBooking();`	`@"INSERT INTO PassengerDetails (Guid,PassengerFullName,PassportNumber,Nationality,`
		
		`PhoneNumber,EmailAddress,Gender)`
		`@"INSERT INTO Booking (Guid,BookingReference,DepartureLeg1Airport_Guid,DestinationLeg1Airport_Guid,`
		
		`DepartureLeg2Airport_Guid,DestinationLeg2Airport_Guid,`
		
		`Leg1FlightDate,Leg2FlightDate,Leg1FlightTime,Leg2FlightTime,PassengerDetails_Guid,`

		`DateOfBooking,BookedByUser_Guid,Leg1FlightNo,Leg2FlightNo)`
Form Controls	**Buttons x 3** **Label x 24** **Picturebox x 1** **Groupbox x 3** **Panel x 6** **Textbox x 19** **Checkbox x1**	*pbxLogo*: this is a picturebox which contains an image of the Cyanair logo *btnHelp*: this button is on each page of the system. Allows the user to search for help options in a new window *btnMakeBooking*: This button depends on the checkbox being checked in order for it to become enabled. By default it is set to disabled. Once it is clicked, a message will ask the user if they are sure they would like to create the booking. If the answer is yes, then it will create a new booking and generate a booking reference number. *btnClose*: When clicked this will close the program *pnlBookingConfirmation*: this panel contains all of the design components of the form *grpBookingConf,* this groupbox displays all of the design components relating to the booking confirmation. Includes sub-groupboxes *grpFlightDetails*,: this groupbox features the *pnlFlight1* and *pnlFlight2* and its labels and textboxes. *pnlFlight1* : (*lblDepartureLeg1, lblDestinationLeg1, lblTypeOfSeatLeg1, lblTimeLeg1, lblDurationLeg1, lblDateLeg1, txtDepartureLeg1, txtDestinationLeg1, txtTypeOfSeatLeg1, txtTimeLeg1, txtDurationLeg1, txtDateLeg1)*

		pnlFlight2: (*lblDepartureLeg2, lblDestinationLeg2, lblTypeOfSeatLeg2, lblTimeLeg2, lblDurationLeg2, lblDateLeg2, txtDepartureLeg2, txtDestinationLeg2, txtTypeOfSeatLeg2, txtTimeLeg2, txtDurationLeg2, txtDateLeg2)* *grpPassengerDetails*: this groupbox contains all of the labels and textboxes relating to the passenger details. *(lblFullName, lblPassportNumber, lblNationality, lblPhoneNumber, lblEmail, lblGender,, txtFullName, txtPassportNumber, txtNationality, txtPhoneNumber, txtEmailAddress, txtGender)* *pnlBookingReference*: this panel contains the information relating to the booking reference number. This panel will appear after the user has confirmed they would like to make a new booking. The booking reference number will be generated and inserted into the textbox
Form Behaviour	The form will input the details of the passenger. It requires that all fields are filled in order to progress. Also displays information about the flight(s) selected.	All fields must be inputted. If the fields are not all completed, an error message will appear on screen. If they are, the program will continue to the next page.
Form Objects & Class	PassengerDetails Class Person Class FlightRoute Class CyanairDatabaseConnection Class	Refer to Class Diagram CyanairDatabaseConnection.`InsertNewBooking();`

HELP SCREEN



This section contains information, related to **Cyanair booking system.** If you cannot find an answer to your question, make sure to contact your system administrator.

General Help

- Login
- Choice Page
- Booking System - Book A Flight
- Passenger Details
- Booking Confirmation
- Booking Reference Number
- Maintenance Suite

Frequently Asked Questions

Figure 40: Help Screen

This Help screen will be available to view from each screen in the system. It will pop out and can be closed by clicking the x button, without exiting the program itself. It contains information regarding each page in the system. It has been created with HTML and the form simply contains a web browser element from the design components with the data source.