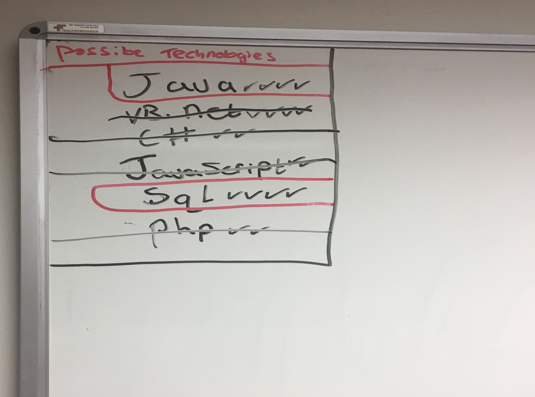
# Introduction

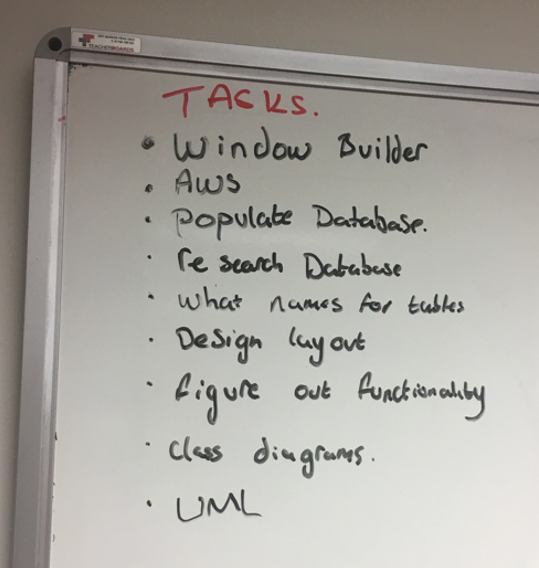
During this group assignment a flight booking system will be produced. The system will be developed for the customer in the form of an on-line system where customers can enter their personal details to search for flights and return a price for the customer depending on destination and frequent flyer details. There will also be an option where the administrator can log in to the system where customer details can be viewed and edited.

This system will be developed by a team and the team will make use of some enterprise applications to aid in the development process. The applications are as follows: GitHub, Jacoco, Jira, Junit, JConsole and Javadocs. The system will be developed using agile methodologies and will comprise of several sprints of 1-2 weeks. The team will carry out scrum meetings weekly to communicate tasks that are completed, upcoming and how everyone is getting on with the development.

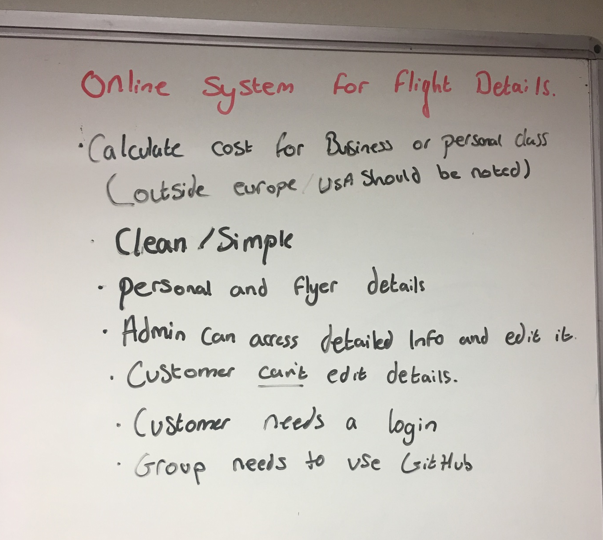
# Images from Group Meetings



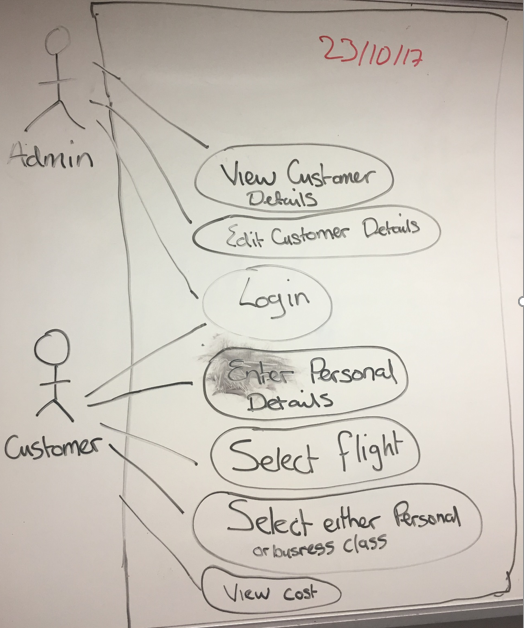
*Fig 2.1 – Possible Technologies*



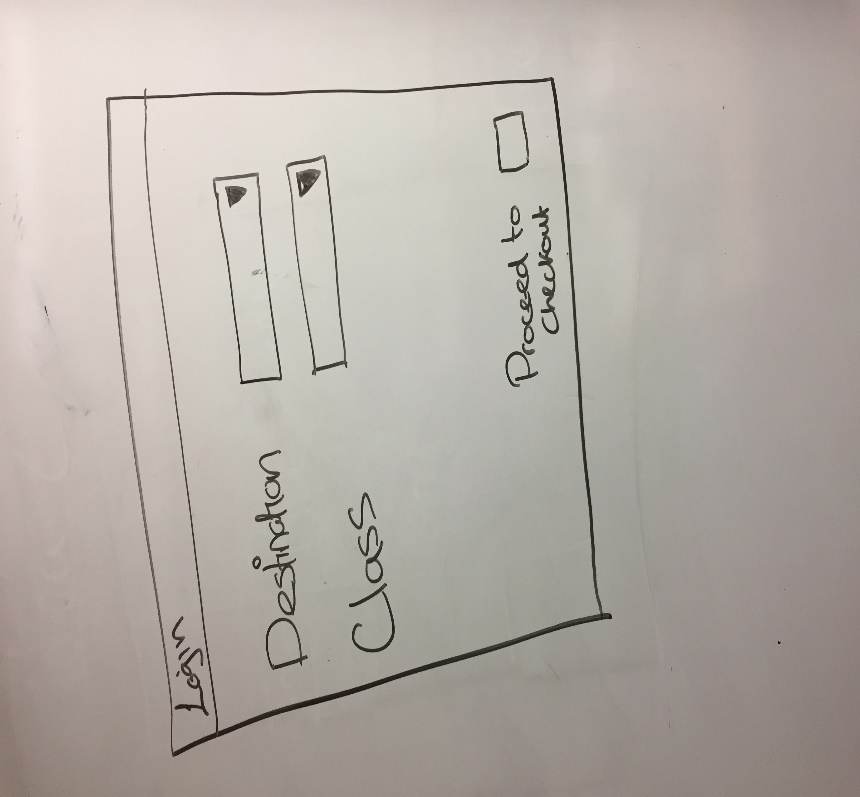
*Fig 2.2 – Tasks to complete the project*



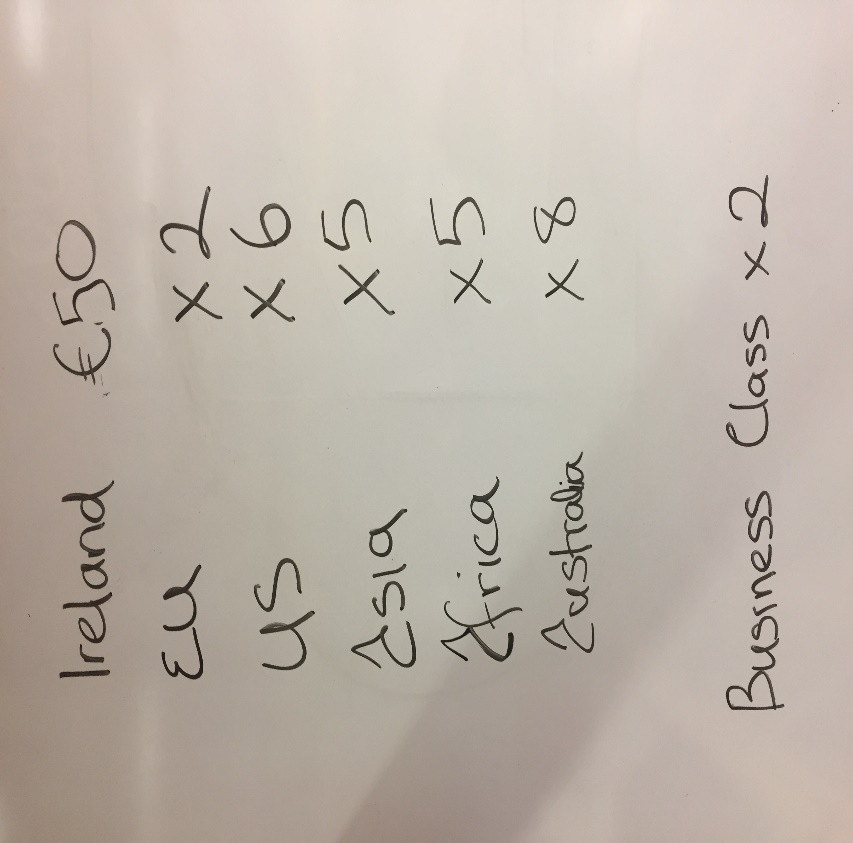
*Fig 2.3 – Group requirements*



*Fig 2.4 – UML Use Case Diagram*



*Fig 2.5 – Customer Login Window*



*Fig 2.6 – Costings for Travel Destinations*

# UML Diagrams

## Class Diagram

# 

# Use Case Descriptions

## 4.1 Register Client

|  |  |
| --- | --- |
| Use Case | Register Client |
| Objective | To register a client on the system |
| Precondition | The client wishes to register their details on the system. |
| Main Flow | 1. Client enters details. (Name, Address, DOB, Contact Details) 2. Client registers on system. |
| Alternative Flow |  |
| Post Condition | 1. Client has been registered. |

## 4.2 Select Flight

|  |  |
| --- | --- |
| Use Case | Select flight. |
| Objective | To select a flight. |
| Precondition | A client wishes to select a flight and view the cost. |
| Main Flow | 1. Client logs in. 2. Client views available flights. 3. Client selects flight. 4. Client selects business or personal class. 5. Client views cost. |
| Alternative Flow | 1. No log in details, see use case register client. |
| Post Condition | A client has selected a flight and viewed the cost of the flight. |

## 4.3 Edit Flight Details

|  |  |
| --- | --- |
| Use Case | Edit Flight Details |
| Objective | To edit flight details |
| Precondition | The administrator wishes to edit flight details on the system. |
| Main Flow | 1. Administrator logs in 2. Administrator edits flight details. |
| Alternative Flow |  |
| Post Condition | 1. Administrator has edited flight details. |

## 4.4 Edit Client Details

|  |  |
| --- | --- |
| Use Case | Edit Client Details |
| Objective | To edit client details on the system |
| Precondition | The administrator wishes to edit client details |
| Main Flow | 1. Administrator logs in. 2. Administrator edits client details |
| Alternative Flow |  |
| Post Condition | 1. Administrator has edited client details. |

# Business Requirements

The business requirements for this flight booking system are as follows:

## 5.1 Functional Requirements

### Administrator

* Log In to System
* View Client Details
* Edit Client Details
* View Flight Details
* Edit Flight Details

### User

* Register
* Log In
* View Available Flights
* Select Flight
* Select Business or Personal Class
* View Cost of Flight

## 5.2 Non-Functional Requirements

### Availability

* The system should be 99.999% available.

### Security

* The system should only be available to registered users
* Passwords must contain numbers and letters and at least one capital letter.
* Users will have three attempts to login after which they will be locked out of the system.
* Once the client enters details it should not be able to be changed by the client.

### Confidentiality

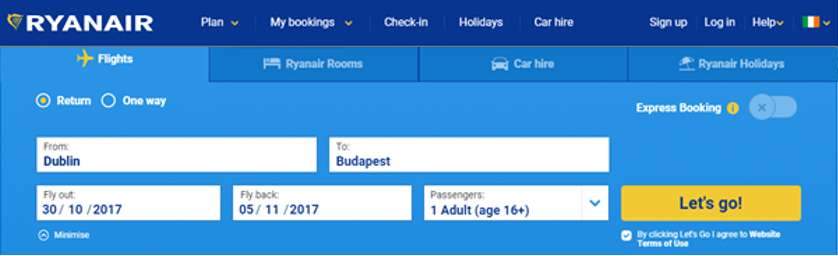
* Only the administrator may view the other client’s details.

### Usability

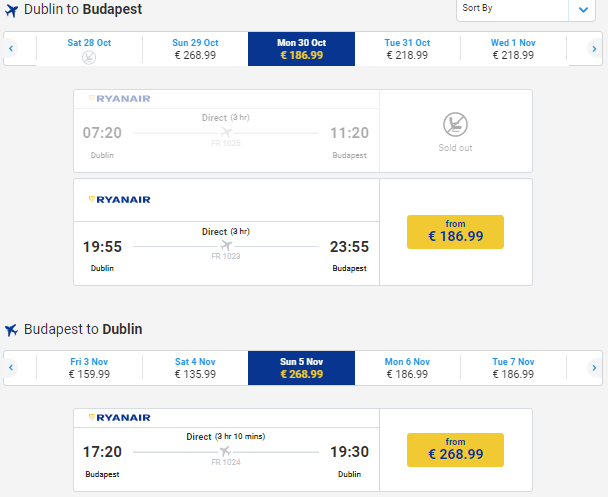
* The system shall be easy to use by people without training.

# . Review of Existing Systems

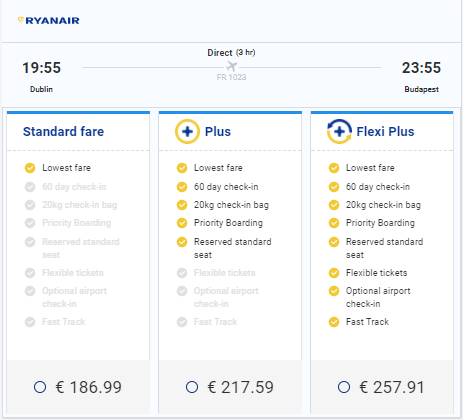
This section samples an existing system which serves a similar purpose to the system we have been assigned for our team project. It has been examined to so that we can create our own GUI that works in a similarly efficient way.



*Fig 6.1 - Ryanair’s Initial Menu*



*Fig 6.2 - Price for flight(s) displayed*



*Fig 6.3 -* Additional options displayed

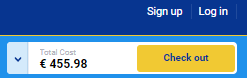


Fig 6.4 - Cost displayed with no user login required

Our system will be developed in Java using eclipse. The GUI above can be similarly represented on a Java GUI platform as the Java components that exist are quite similar to this web based system’s components (e.g. JButton, JTextfield, JCheckBox, JDialog, JRadioButton etc.).

# DevOps Development Tools

## Jacoco

## Junit/TestNG

## 7.3 Jira

## JConsole/jRAT

## 7.5 Javadoc

# 8. Code Created

## 8.1 Consideration for security of code

## Consideration for performance of code

# 9. Team Collaboration

## 9.1 Agile Development

## GIT

# 10. Database

## 10.1 Amazon RDS Cloud Database Connect On SQLWorkBench

1. Add connection
2. Hostname: devops.clql55s9fxrz.eu-west-1.rds.amazonaws.com
3. Username: DevOps
4. Password: groupthree

Test connection and should work, my connection name is DevOps but don’t think it matters what you call it, I added some sample tables.

## 10.2 Some Sample Java Code to Connect to Database

/\*\*

\* Database handler

\*

\* @author Mark Glenn

\*/

import javax.swing.\*;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

@SuppressWarnings("serial")

public class DatabaseHandler extends javax.swing.JFrame {

// credentials for database including AWS RDS database endpoint and JDBC

// driver

final String JDBC\_DRIVER = "com.mysql.jdbc.Driver";

final String DB\_URL = " devops.clql55s9fxrz.eu-west-1.rds.amazonaws.com";

final String USER\_NAME = "DevOps";

final String PASSWORD = "groupthree";

Connection conn = null;

Statement stmt = null;

ResultSet rs = null;

// connect to database

public void connectToDatabase() {

try {

// STEP 1 - Load the JDBC driver

java.lang.Class.forName(JDBC\_DRIVER);

System.out.println("STEP 1 COMPLETE - Driver Registered...");

// STEP 2 - Open a connection

conn = DriverManager.getConnection(DB\_URL, USER\_NAME, PASSWORD);

System.out.println("STEP 2 COMPLETE - Connection obtained...");

// STEP 3 - Create Statement object

stmt = conn.createStatement();

System.out.println("STEP 3 COMPLETE - Statement object created...");

} catch (ClassNotFoundException e) {

System.out.print("Connection Error");

//JOptionPane.showMessageDialog(null,"Could not load driver.\n" + e.getMessage());

} catch (SQLException e) {

System.out.print("Connection Error");

//JOptionPane.showMessageDialog(null,"Problem with SQL.\n" + e.getMessage());

}

}

// connect and do query

public void doQuery(String query) {

try {

java.lang.Class.forName(JDBC\_DRIVER);

System.out.println("STEP 1 COMPLETE - Driver Registered...");

// STEP 1 - Open a connection

conn = DriverManager.getConnection(DB\_URL, USER\_NAME, PASSWORD);

System.out.println("STEP 2 COMPLETE - Connection obtained...");

// STEP 2 - Create Statement object

stmt = conn.createStatement();

System.out.println("STEP 3 COMPLETE - Statement object created...");

System.out.println("STEP 4(a) COMPLETE - Query executed and database found...");

System.out.println("STEP 4(b) COMPLETE - Query executed.");

} catch (Exception ex) {

JOptionPane.showMessageDialog(null, ex.getMessage());

}

}

}// end of class

## 10.3 Database persistence technology

# 11. Conclusions