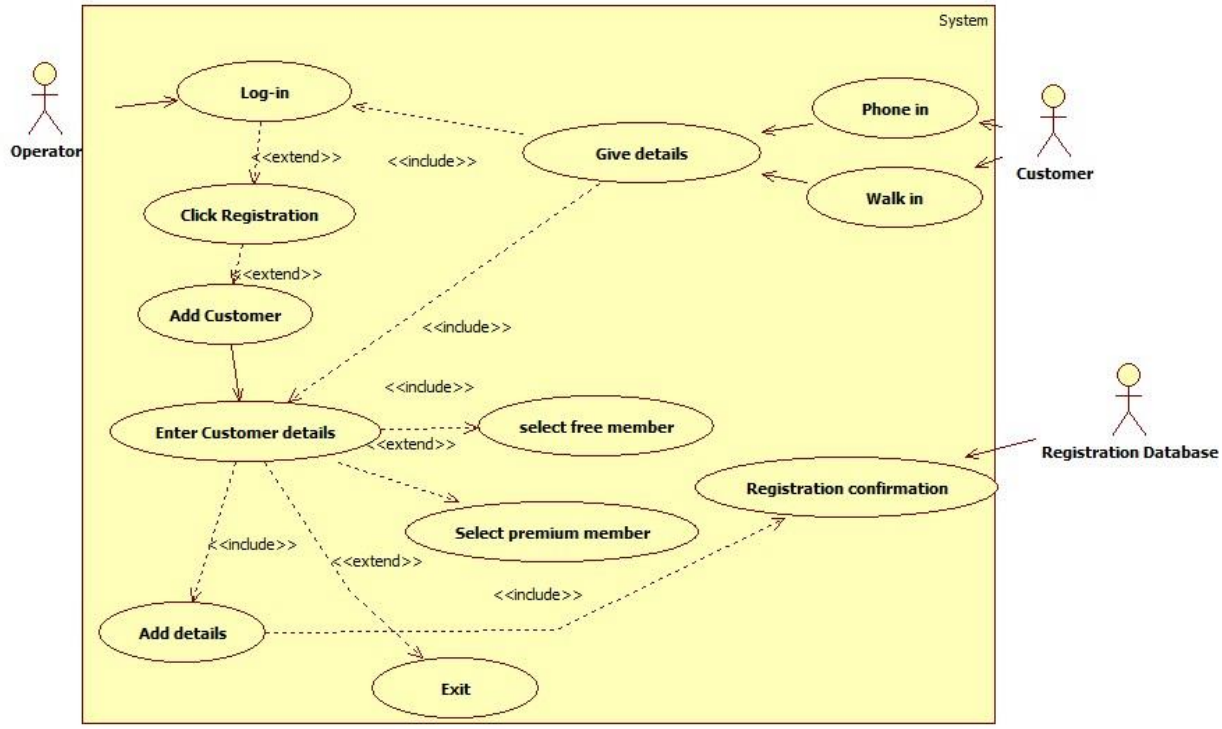
Individual Report

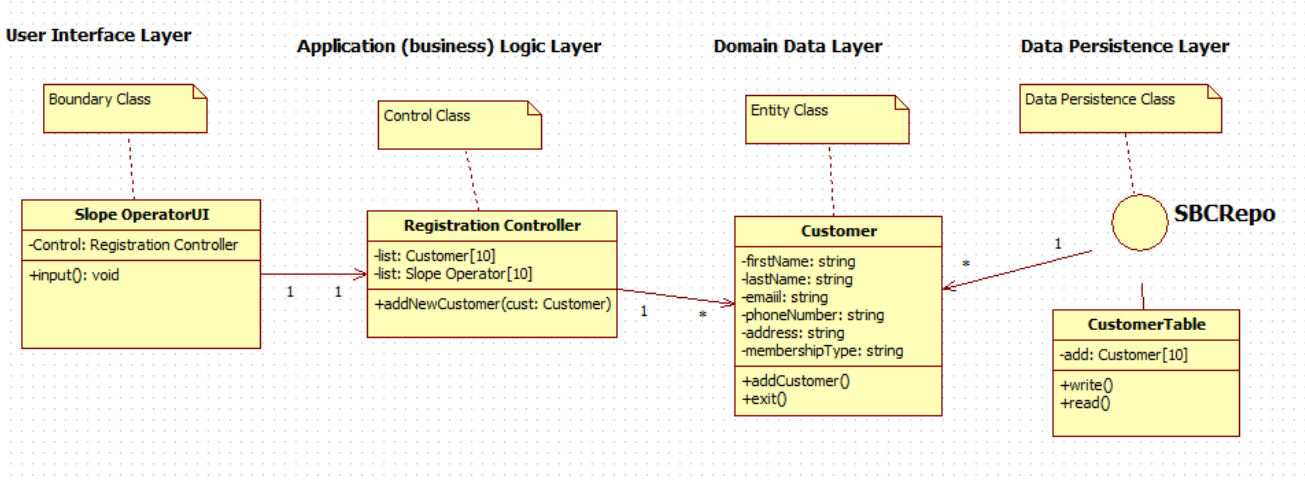
Student Name: Asis Rai	**SID: 6528683**
Task 1: To complete design specification, implementation, testing and integration – evidence with a report supported by research with references in the CU Harvard referencing style in a maximum of 1,000 words	

Use case description and diagram

Name of use case: Registration
Pre-conditions: The receptionist selects 'Registration' and is given a window to register a new customer
Post-conditions: A customer has been registered into the system
Purpose: To register customer/customers into the system
Description: The receptionist selects 'Registration' and is given a window to register a new customer. The customer details then will be stored in the system and can be sent to booking section.
Main Success Scenario: 1. The receptionist selects the 'Make Registration' option from the screen 2. The receptionist enters details given by the customer. 3. The system checks if the user is already registered into the system, if they then the system informs the receptionist. If the customer is new, then the system stores this new detail in the system.
Alternatives: 1. Customer is already registered. 2. Customer forgets that he/she was registered before. 3. The system displays a message saying customer is already registered.

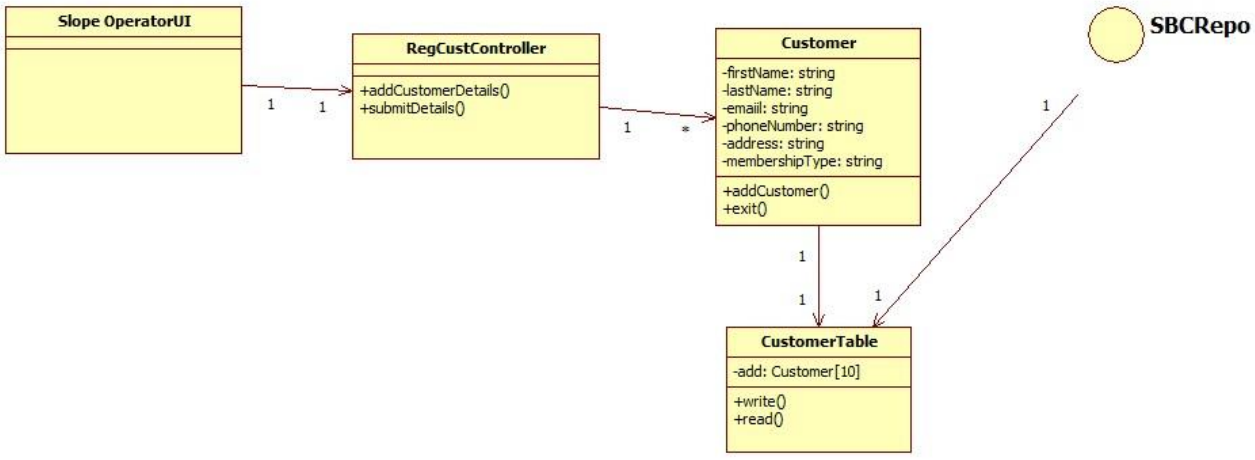


The final design class diagram (produced in a CASE tool) for your own functionality only



This is a four-layered model of how **Registration Functionality** works, it shows how the components will communicate and work with each other to store data into the database and shows the data flow the slope Operator UI is the boundary class and Registration form in the system. It provides an interface for the operator where input is entered of the customer details and passed onto the Registration controller. Registration controller is the Control class and application logic layer, Registration controller takes the input from Slope Operator and validates that the data set provided are correct against the data requirements set on the database. Registration controller does this by passing the data sets into the Domain data layer which is the customer data requirements. This Entity class holds all the data requirements (data type validation) which must be matched with data sets provided by the Registration controller. If the data sets are matched with data requirements, the data set is passed into the Data persistence layer to be stored and used by other Functionalities. The data persistence layer holds the database, for Registration functionality Microsoft SQL server database was decided to use.

Necessary design details based on the GRASP use case controller pattern

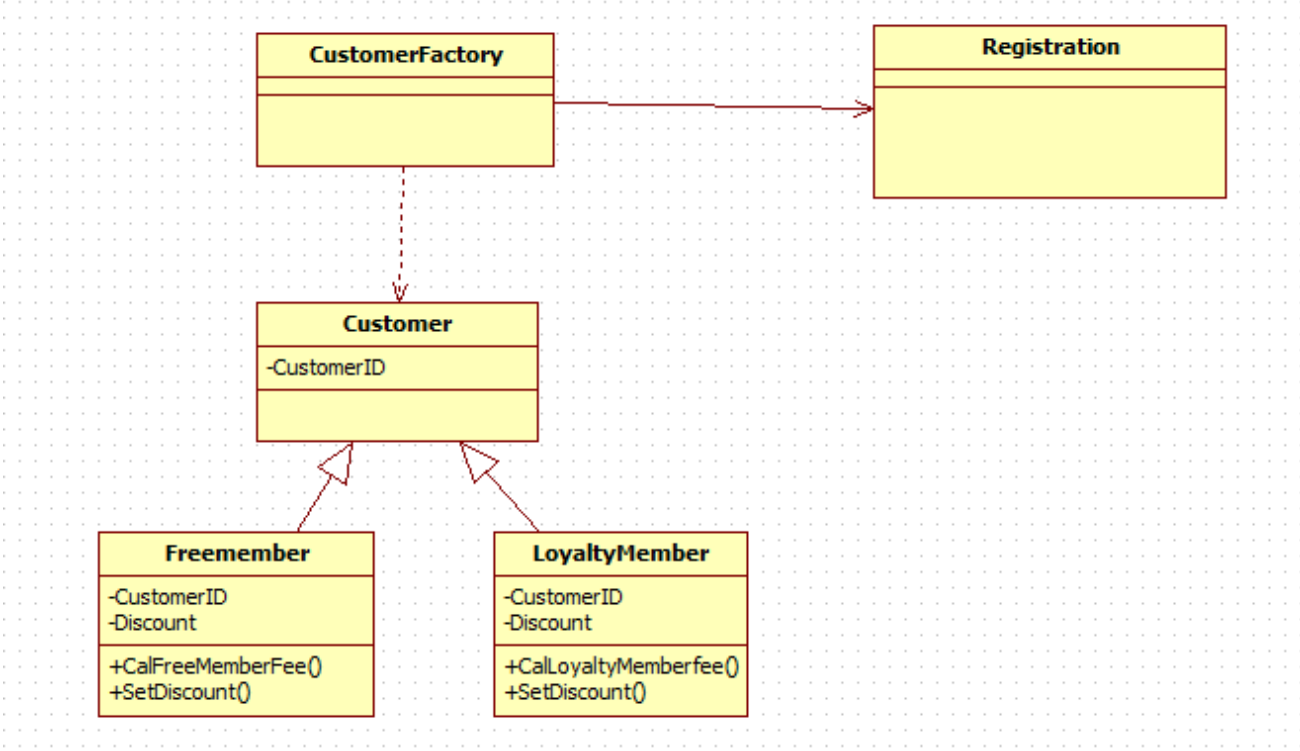


The GRASP use case controller is focused on creating control class for the Registration functionality and how the system interacts with registration and around it.

The design of the user interface is the SlopOperatorUI which is called Registration form in the system. The application layer form will give the slopeOprator(user) option to put in necessary customer details. When this information is entered, the information is passed onto the RegCustController and this is the application logic layer. The RegCustController have the operations (+addCustomerDetails()) and (+submitDetails()), these operations are self-explanatory. +addCustomerDetails() will pack the data set to be summited to the next layer and +submitDetails() will submit the data set to the next layer. The next layer is called data domain layer which is named customer. This layer holds the data requirements for the data sets, which means the data sets

received from RegCustController controller must be matched in-order for it to be summited to the database. The benefit of the GRASP use case controller is that as it clearly shows the methods in the system and how each layer communicates with other layers making it easier for me to create the Registration Functionality.

at least ONE Gang of Four (GoF) pattern of your choice applied to the design

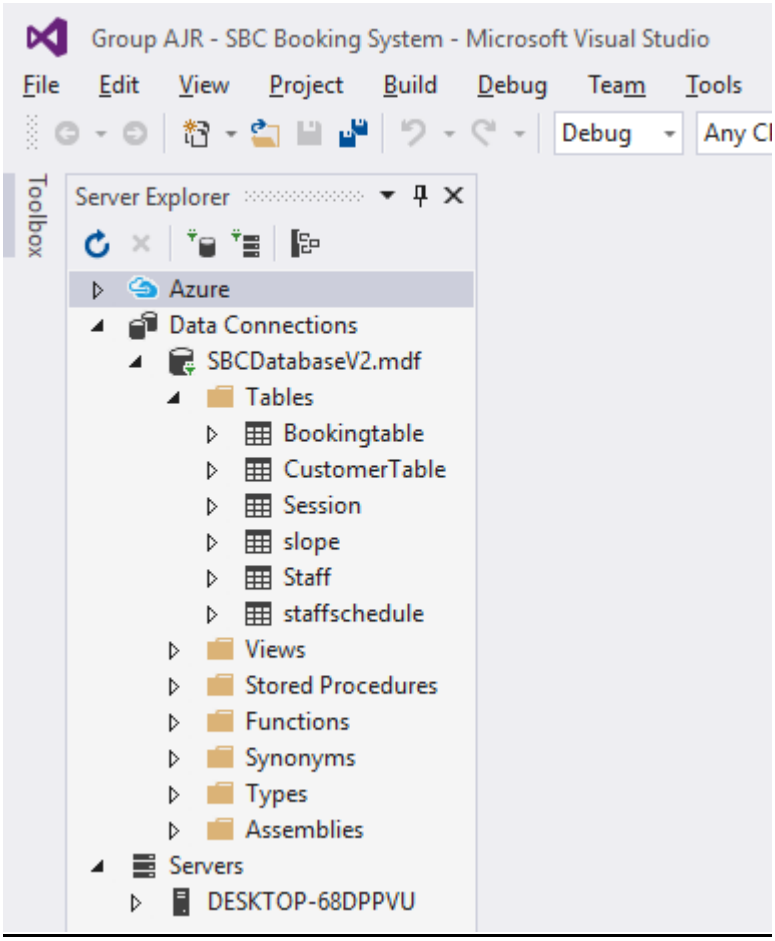


Singleton pattern - In software engineering, the singleton pattern is a design pattern that restricts the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system. The concept is sometimes generalized to systems that operate more efficiently when only one object exists, or that restrict the instantiation to a certain number of objects. The term comes from the mathematical concept of a singleton.

The idea here is that when the type of user is selected in the Registration form, the user factory will handle the data type chose i.e 0 for free member and 1 for LoyaltyMember(Premium Member), therefore restricting the installation of two class free member and loyalty member into one object.

Commented source code for your own functionality only

Database Tables



Registration.cs

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.ComponentModel;
4.  using System.Data;
5.  using System.Drawing;
6.  using System.Linq;
7.  using System.Text;
8.  using System.Threading.Tasks;
9.  using System.Windows.Forms;
10. using System.Data.SqlClient;
11.
12. namespace Group_AJR___SBC_Booking_System.Registration
13. {
14.     public partial class Registration : Form
15.     {
16.         public Registration()
17.         {
18.             InitializeComponent();
19.         }
20.
21.         private void Registration_Load(object sender, EventArgs e)
22.         {
23.
24.         }
25.
26.         private void button1_Click(object sender, EventArgs e) //Register button
27.         { //when register button is clicked
```

```
28.             using (SqlConnection Connection = new SqlConnection(@"Data Source=(Loc
    alDB)\MSSQLLocalDB;AttachDBFilename=|DataDirectory|\SBCDatabaseV2.mdf;Integrated S
    ecurity=True;Connect Timeout=30")) //setting database connection path
29.             { //validation to make sure none fields in the form are left empty
30.                 if (textBox1.Text == String.Empty || textBox2.Text == String.Empty
     || textBox3.Text == String.Empty || textBox4.Text == String.Empty || textBox5.Tex
    t == String.Empty) //fields - firstname, lastname etc.
31.                 {
32.                     MessageBox.Show("Error, All Customer details must be filled",
    "Value Error", MessageBoxButtons.OK, MessageBoxIcon.Error); //displays a message b
    ox with this error if the fields are left empty
33.                 }
34.
35.                 else //if the fields are not empty
36.                 {
37.                     try
38.                     {
39.                         Connection.Open(); //open the sql connection //sql command
    s to add value got from the fields into their places in the sql database
40.                         SqlCommand cmd = new SqlCommand(@"INSERT INTO CustomerTabl
    e ([firstname], [lastname], [email], [phonenumber], [address], [premember]) VALUES
     (@firstname, @lastname, @email, @phonenumber, @address, @premember);", Connection
    );
41.                         cmd.Parameters.AddWithValue("@firstname", textBox1.Text);

42.                         cmd.Parameters.AddWithValue("@lastname", textBox2.Text);
43.                         cmd.Parameters.AddWithValue("@email", textBox3.Text);
44.                         cmd.Parameters.AddWithValue("@phonenumber", textBox4.Text)
    ;
45.                         cmd.Parameters.AddWithValue("@address", textBox5.Text);
46.                         cmd.Parameters.AddWithValue("@premember", checkBox1.Checke
    d);
47.
48.
49.                         int i = cmd.ExecuteNonQuery();
50.                         Connection.Close();//close the sql connection
51.
52.                         if (i == 1) //if data is successfully added
53.                         {
54.                             MessageBox.Show("Customer has been registered"); //dis
    play this message
55.                             getPrimaryKey(); //start this function
56.                             //clear the fields
57.                             textBox1.Clear();
58.                             textBox2.Clear();
59.                             textBox3.Clear();
60.                             textBox4.Clear();
61.                             textBox5.Clear();
62.                         }
63.                         else
64.                         {
65.                             MessageBox.Show("Customer couldn't be added, Please Tr
    y again");//if the data were not succesfully added
66.                         }
67.                     }
68.                     catch (Exception ex)
69.                     {
70.                         MessageBox.Show("Unexpected Error has occured:" + ex.Messa
    ge); //catches any other errors
71.
72.                     }
```

```
                }
            }
        }

        private void getPrimaryKey() //function to set the next data into a new ro
m
        {
            using (SqlConnection Connection = new SqlConnection(@"Data Source=(Loc
alDB)\MSSQLLocalDB;AttachDBFilename=|DataDirectory|\SBCDatabaseV2;Integrated Secur
ity=True;Connect Timeout=30")) //setting database path
            {
                try
                {
                    Connection.Open();//opens sql connection
                    SqlCommand cmd = new SqlCommand(@"SELECT MAX(Id)+1 FROM Custom
erTable;", Connection); //sets the next data(customer) into a new row, so that the
re are no duplicates
                    Connection.Close();

                }

                catch (Exception ex)
                {
                    MessageBox.Show("Unexpected Error has occured: " + ex.Message)
; //catches errors
                }
            }
        }


        private void button2_Click(object sender, EventArgs e) //exit button
        {
                    this.Close(); //closes sql connection
                    MainUIform main = new MainUIform();// when form is exited, Main
UI form is opened.
                    main.Show();//this opens the main form
                }
            }
        }
```

Registration controller

```
using System;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace Group_AJR___SBC_Booking_System
{
    class Controller
    {
        private TextBox textBox1, textBox2, textBox3, textBox4, textBox5;
        private CheckBox checkBox1;

        public Controller(TextBox textBox1, textBox2, textBox3, textBox4, textBox5
, CheckBox checkBox1)
        {
            firstname = textBox1;
            lastname = textBox2;
            email = textBox3;
            Phonenumber = textBox4;
            address = textBox5;
```

```
19.             premember = checkBox1;
20.
21.         }
22.
23.         public Controller()
24.         {
25.
26.         }
27.
28.         public bool Update()
29.         {
30.             Bookingtable type = Factory.getType(b.Checked);
31.             bool getType = type.type(b.Checked);
32.
33.             Registration reg = new Registration(int.Parse(firstname.Text.ToString(
    )), int.Parse(lastname.Text.ToString()), int.Parse(email.Text.ToString()), int.Par
    se(Phonenumber.Text.ToString()), int.Parse(address.Text.ToString()), int.Parse(pre
    member.Text.ToString()), getType);
34.             bool i = reg.Update();
35.             return i;
36.         }
37.
38.         public bool getRegistration()
39.         {
40.             Registration Registration = new Registration();
41.             bool i = Registration.getRegistration(int.Parse(CustomerID.Text.ToStri
    ng()));
42.
43.             if (i)
44.             {
45.                 firstname.Text = Registration.firstname.ToString();
46.                 lastname.Text = Registration.lastname.ToString();
47.                 email.Text = Registration.email.ToString();
48.                 Phonenumber.Text = Registration.Phonenumber.ToString();
49.                 address.Text = Registration.address.ToString();
50.                 premember = Registration.ToString;
51.             }
52.             return i;
53.         }
54.
55.         public bool Save()
56.         {
57.             Bookingtable type = Factory.getType(b.Checked);
58.             bool getType = type.type(b.Checked);
59.
60.             Registration reg = new Registration(int.Parse(firstname.Text.ToString(
    )), int.Parse(lastname.Text.ToString()), int.Parse(email.Text.ToString()), int.Par
    se(Phonenumber.Text.ToString()), int.Parse(address.Text.ToString()), int.Parse(pre
    member.Text.ToString()), getType);
61.             bool i = reg.Update();
62.             return i;
63.         }
64.
65.         public bool manualTest()
66.         {
67.             Console.WriteLine("----------------BEGINING TEST----------------");
68.             using (SqlConnection Connection = new SqlConnection(@"Data Source=(Loc
    alDB)\MSSQLLocalDB;AttachDBFilename=|DataDirectory|\SBCDatabaseV2.mdf;Integrated S
    ecurity=True;Connect Timeout=30"))
69.             {
70.                 try
```

```
                {
                    Connection.Open();
                    SqlCommand cmd = new SqlCommand("UPDATE CustomerTable SET firs
tname=Asis, lastname=Rai, email=asis@gmai.com, Phonenumber=07588259523, address=7
Arliss Way premember=('1'), WHERE Id = 1", Connection);
                    int i = cmd.ExecuteNonQuery();

                    if (i > 0)
                    {
                        Console.WriteLine("--------UPDATE PASS--------");
                        cmd = new SqlCommand("SELECT Count(*) FROM CustomerTable W
HERE Id=18 AND firstname=Asis, lastname=Rai, email=asis@gmai.com, Phonenumber=0758
8259523, address=7 Arliss Way premember=('1')", Connection);
                        i = (int)cmd.ExecuteScalar();

                        if (i > 0)
                        {
                            Console.WriteLine("--------VERIFY PASS--------");
                            Console.WriteLine("----------------ALL PASS----------
-----");
                            Connection.Close();
                            return true;
                        }
                        else
                        {
                            Console.WriteLine("--------VERIFY FAIL--------");
                            Connection.Close();
                            return false;
                        }
                    }
                    else
                    {
                        Console.WriteLine("--------UPDATE FAIL--------");
                        Connection.Close();
                               return false;
                           }
                       }
                       catch (Exception ex)
                       {
                           Connection.Close();
                           MessageBox.Show(ex.ToString());
                           Console.WriteLine("--------UNKNOWN FAIL--------");
                           return false;
                       }
                   }
               }
           }
       }
```

Gang of Four

```
namespace Group_AJR___SBC_Booking_System.gof
{
    internal class RegistrationController
    {
        public static Registration[] Fetchpremember(int MembershipId)
        {
            //creates request from the sql server and prints reponse
            Dictionary<string, object> Registration = new Dictionary<string, objec
t>();
            booking.Add("MembershipId", MembershipId);
```

```
10.             DatabaseReply reply = DatabaseManager.DatabaseRequest("MembershipDispl
    ay.sql", Registration);
11.
12.             //takes array of members from the resposne
13.             return Prememeber.FromSql(reply.ReplyRows);
14.         }
15.
16.         public static CustomerwithLoyalty[] GetCustomer(Registration premember)
17.         {
18.             DatabaseReply reply = DatabaseManager.DatabaseRequest("MembershipDispl
    ay.sql",
19.                 new Dictionary<string, object>() { { "MembershipId", Registration.
    premember } });
20.             Customer[] customers = Customer.FromSql(reply.ReplyRows);
21.
22.             Customer[] customerClone = CloneFactory.getCustomerClone(customers);
23.
24.             Registration.Customers = customers;
25.             CustomerwithLoyalty[] CustomerwithLoyalty = new CustomerwithLoyalty[cu
    stomers.Length];
26.             for (int i = 0; i < customers.Length; i++)
27.                 CustomerwithLoyalty[i] = new CustomerwithLoyalty(customers[i], rep
    ly.ReplyRows[i]["CustomerId"].ToObject<int>());
28.
29.             return CustomerwithLoyalty;
30.         }
31.
32.         public static CustomerwithFree[] GetCustomer(Registration premember)
33.         {
34.             DatabaseReply reply = DatabaseManager.DatabaseRequest("MembershipDispl
    ay.sql",
35.                 new Dictionary<string, object>() { { "MembershipId", Registration.
    premember } });
36.             Customer[] customers = Customer.FromSql(reply.ReplyRows);
37.
38.             Customer[] customerClone = CloneFactory.getCustomerClone(customers);
39.
40.             Registration.Customers = customers;
41.             CustomerwithFree[] CustomerwithFree = new CustomerwithFree[customers.L
    ength];
42.             for (int i = 0; i < customers.Length; i++)
43.                 CustomerwithFree[i] = new CustomerwithFree(customers[i], reply.Rep
    lyRows[i]["CustomerId"].ToObject<int>());
44.
45.             return CustomerwithFree;
46.         }
47.     }
48.
49.     internal class CustomerwithLoyalty
50.     {
51.         public readonly Customer Customer;
52.         public readonly int prememebrID;
53.
54.         public CustomerwithLoyalty(Customer customer, int CustomerId)
55.         {
56.             this.Customer = customer;
57.             this.prememberID = prememberID;
58.         }
59.     }
60.
61.     internal class CustomerwithFree
```

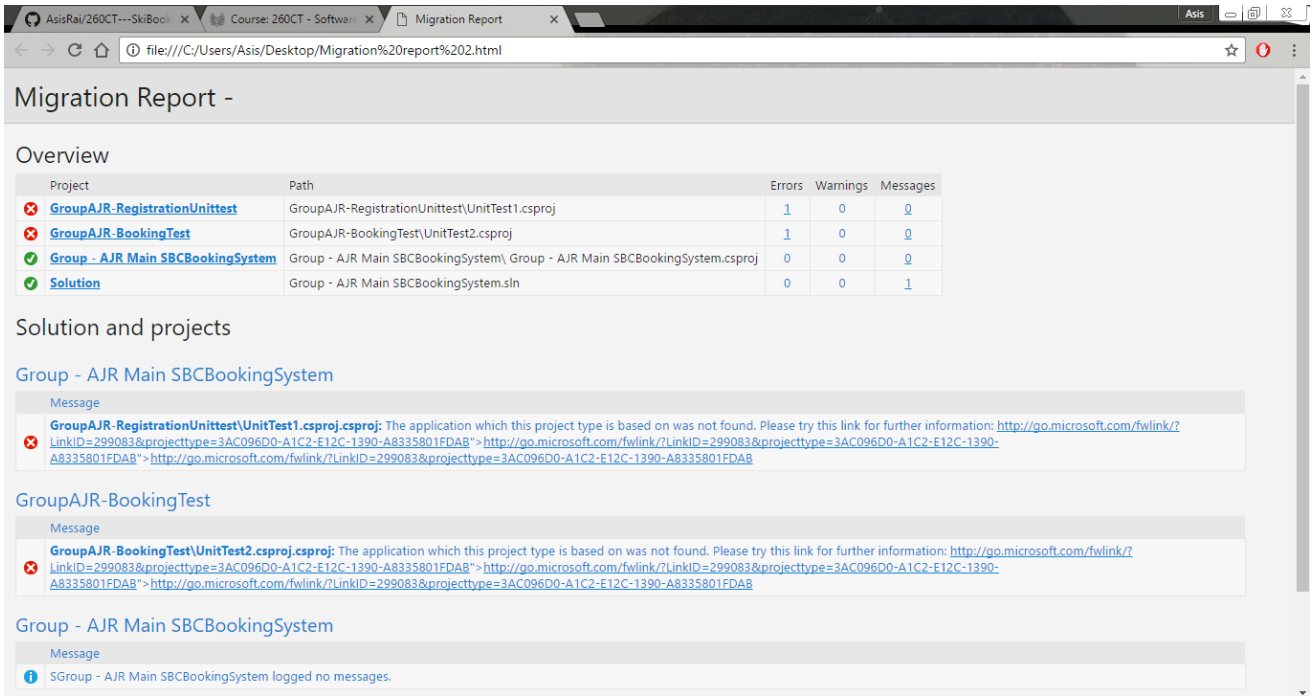
```
62.     {
63.         public readonly Customer Customer;
64.         public readonly int prememebrID;
65.
66.         public CustomerwithFree(Customer customer, int CustomerId)
67.         {
68.             this.Customer = customer;
69.             this.prememberID = prememberID;
70.         }
71.     }
72. }
```

Automated unit testing test script

Tool used – Microsoft Visual Studio Self automated test

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Text.RegularExpressions;
4.  using System.Windows.Input;
5.  using System.Windows.Forms;
6.  using System.Drawing;
7.  using Microsoft.VisualStudio.TestTools.UITesting;
8.  using Microsoft.VisualStudio.TestTools.UnitTesting;
9.  using Microsoft.VisualStudio.TestTools.UITest.Extension;
10. using Keyboard = Microsoft.VisualStudio.TestTools.UITesting.Keyboard;
11. using System.Data.SqlClient;
12. using Microsoft.VisualStudio.TestTools.UnitTesting;
13.
14. namespace GroupAJR_RegistrationUnittest
15. {
16.     [TestClass]
17.     public class UnitTest1
18.     {
19.         [TestMethod]
20.         public void TestMethod1()
21.         {
22.         }
23.
24.         [TestMethod]
25.         public void CodedUIRegistrationTest()
26.         {
27.             //Perform automated test
28.             this.UIMap.RecordedMethodUpdate();
29.
30.             string rtxt = UIMap.UIUpdateRegistrationWindow.UITxtfirstnameWindow.te
    xtBox1.Text;
31.             string rbox = UIMap.UIUpdateRegistrationWindow.UICheckboxWindow.checkB
    ox1.ToString();
32.             string rbutt = UIMap.UIUpdateRegistrationWindow.UIButtonWindow.button1
    .ToString();
33.
34.             this.UIMap.RecordedMethodSubmit();
35.
36.             //Validation that the data is there
37.             Verify verify = new Verify();
38.             bool i = verify.getRegistration(rtxt, rbox, rbutt);
39.             if (i == false)
40.             {
41.                 Assert.Fail();
```

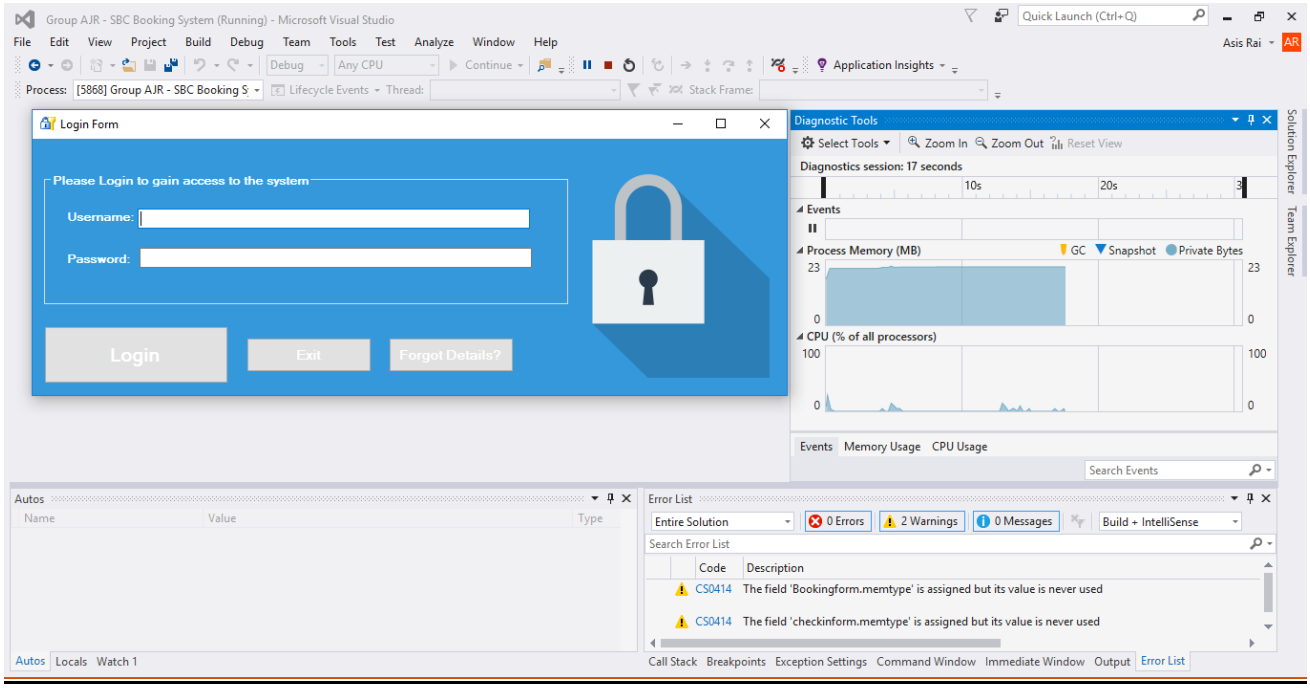
```
            }
        }

        #region Additional test attributes

        // You can use the following additional attributes as you write your tests
    :

        ////Use TestInitialize to run code before running each test
        //[TestInitialize()]
        //public void MyTestInitialize()
        //{
        //    // To generate code for this test, select "Generate Code for Coded U
    I Test" from the shortcut menu and select one of the menu items.
        //}

        ////Use TestCleanup to run code after each test has run
        //[TestCleanup()]
        //public void MyTestCleanup()
        //{
        //    // To generate code for this test, select "Generate Code for Coded U
    I Test" from the shortcut menu and select one of the menu items.
        //}

        #endregion

        /// <summary>
        ///Gets or sets the test context which provides
        ///information about and functionality for the current test run.
        ///</summary>
        public TestContext TestContext
        {
            get
            {
                return testContextInstance;
            }
            set
            {
                testContextInstance = value;
            }
        }
        private TestContext testContextInstance;

        public UIMap UIMap
        {
            get
            {
                if ((this.map == null))
                {
                    this.map = new UIMap();
                }

                return this.map;
            }
        }

        private UIMap map;
    }
}
```



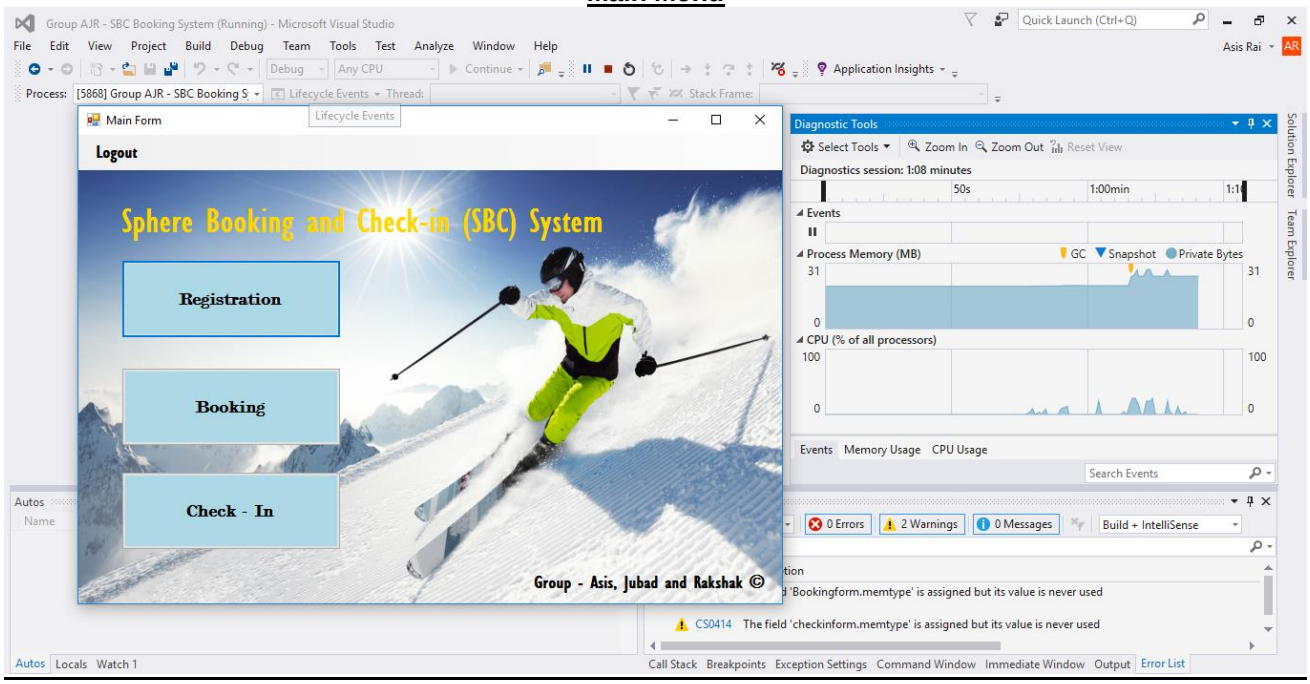
Screen shot demo of at least **TWO prototypes for the group integrated program**

First Prototype

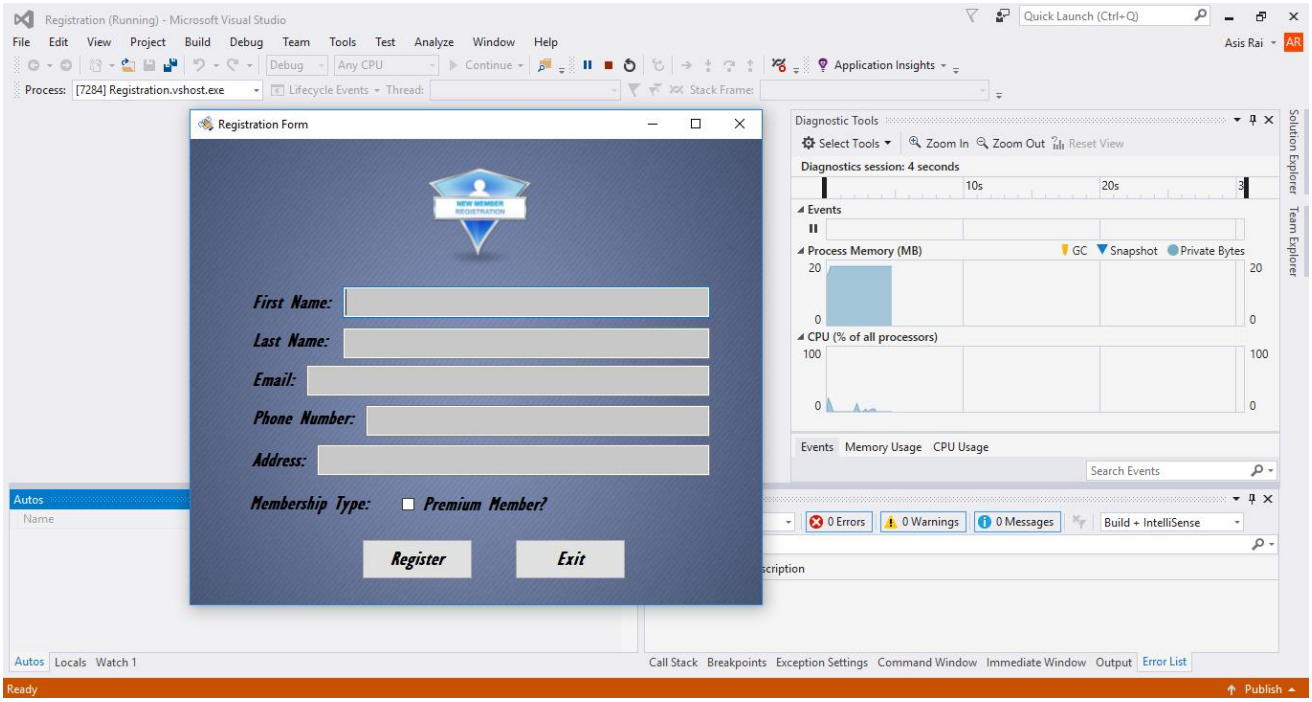
Login Screen



Main Menu



Registration



Tutor client's feedback

Registration – This is my functionality and first prototype. This is the first process of the system where the customer has to be registered to make a booking and then checked-in.

The feed back I received from the tutor was:

Front End

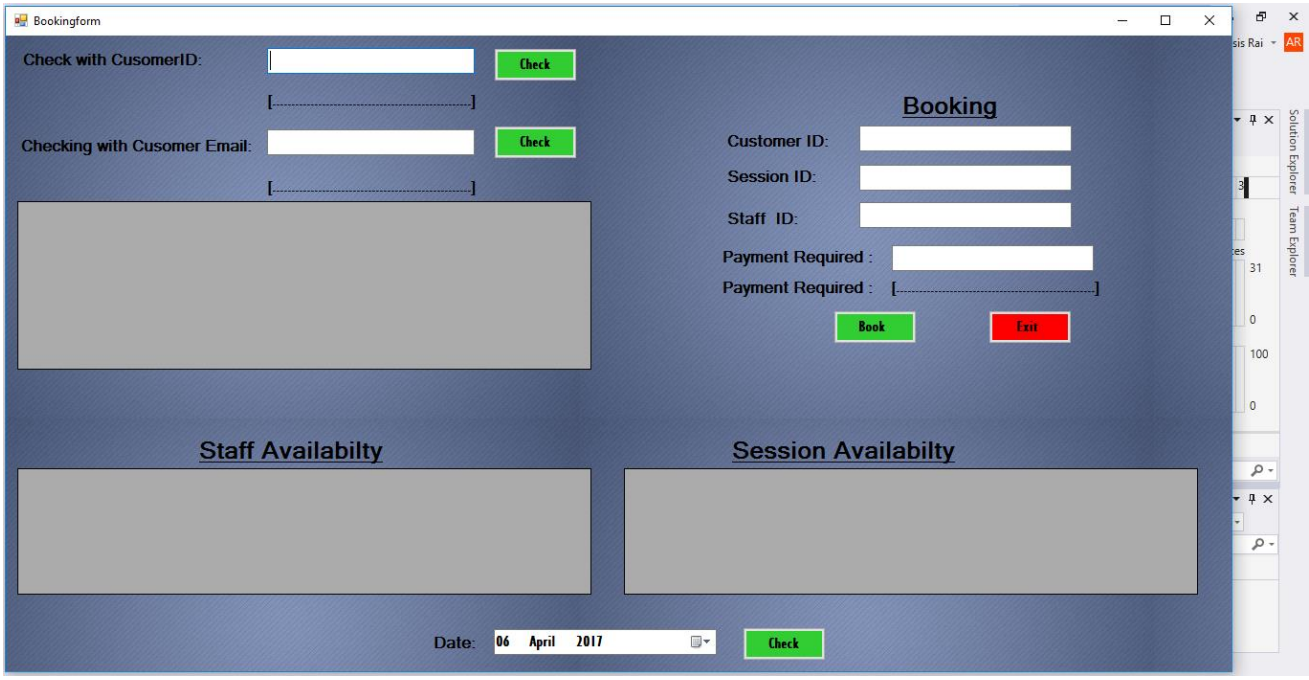
- Make the logo more visible – Text in the logo is not very clear to see
- Align the text boxes, make them consistent with each other

Back-End

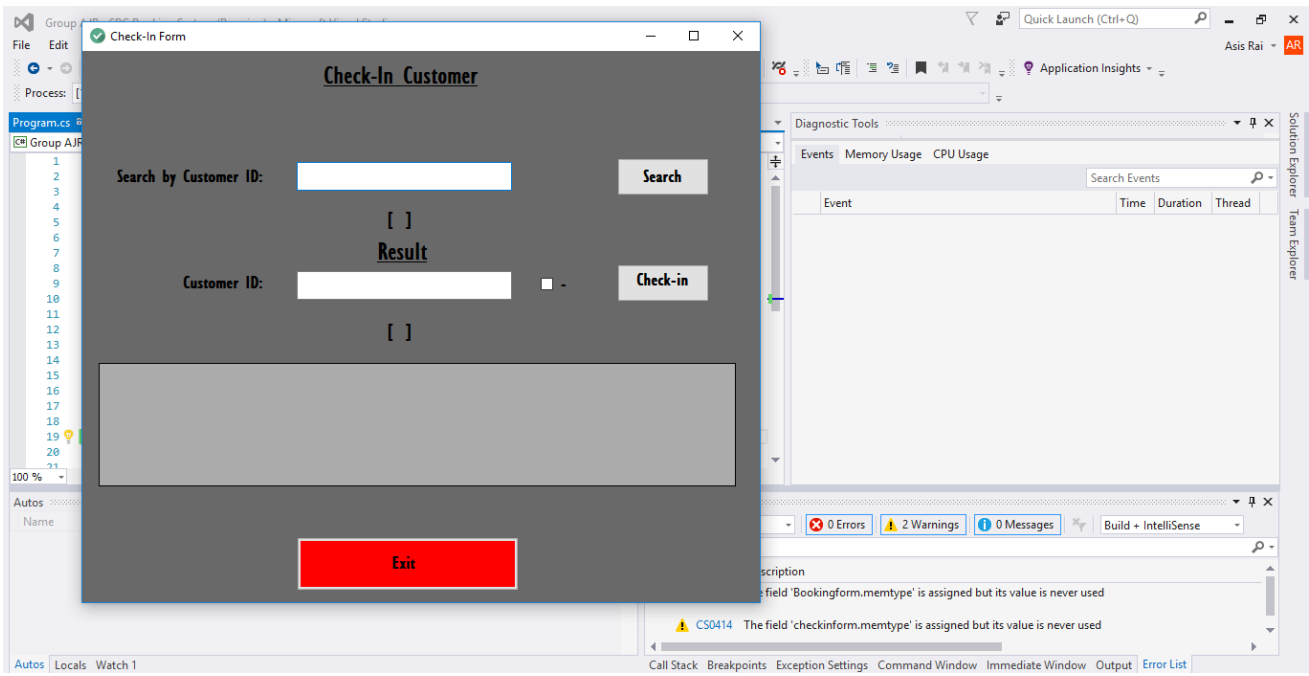
- Add some validation to the text boxes – such as email should check '@'.

The feedback was taken into consideration and implemented successfully in the second prototype, this is shown below in screenshots of Prototype 2.

Booking

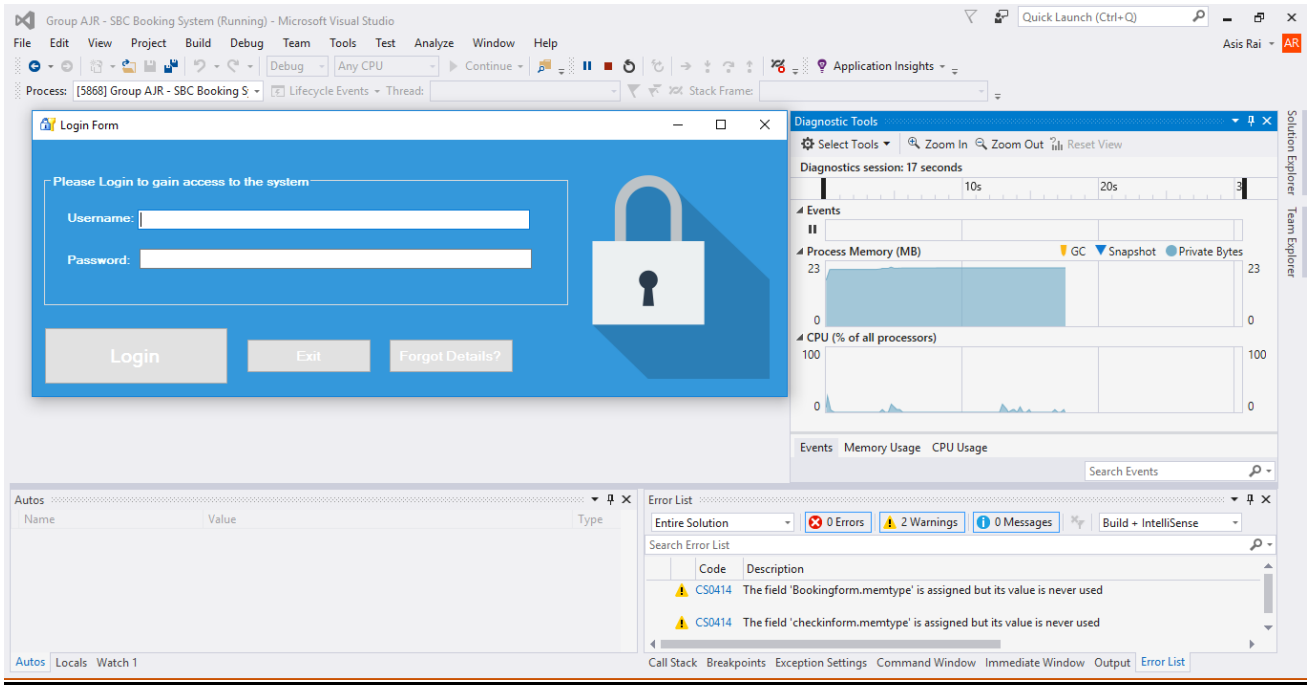


Check-In

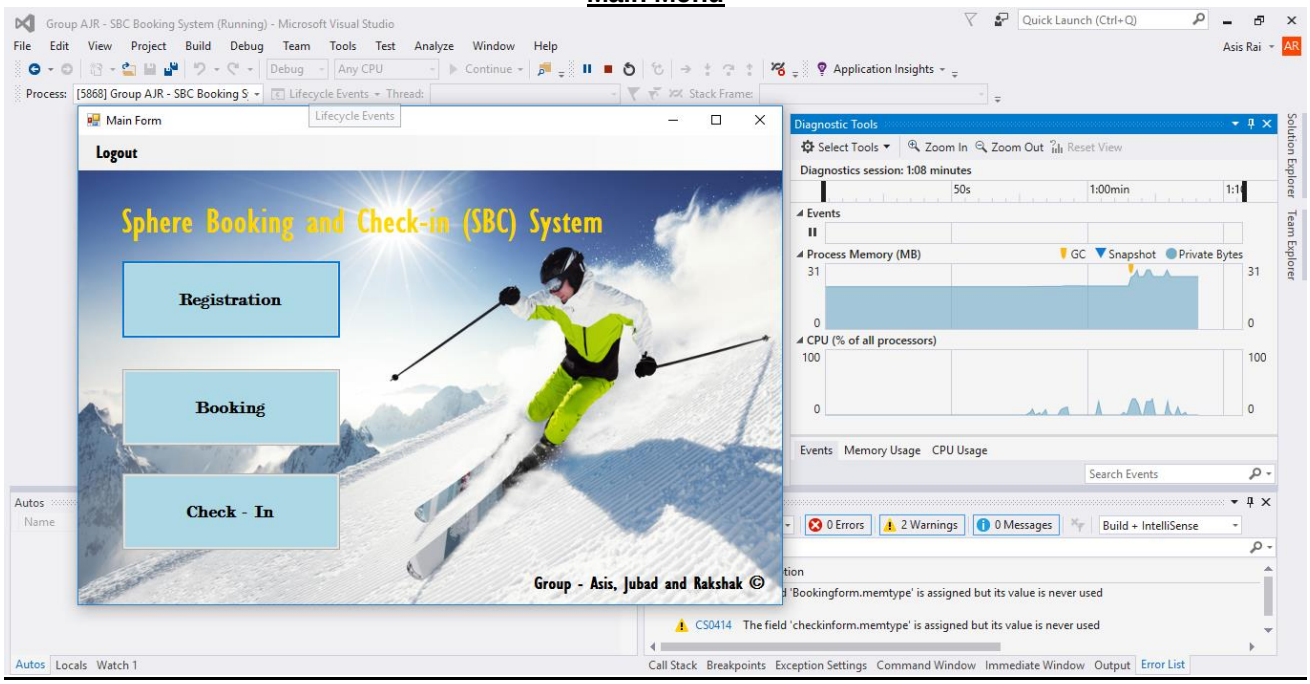


Second Prototype

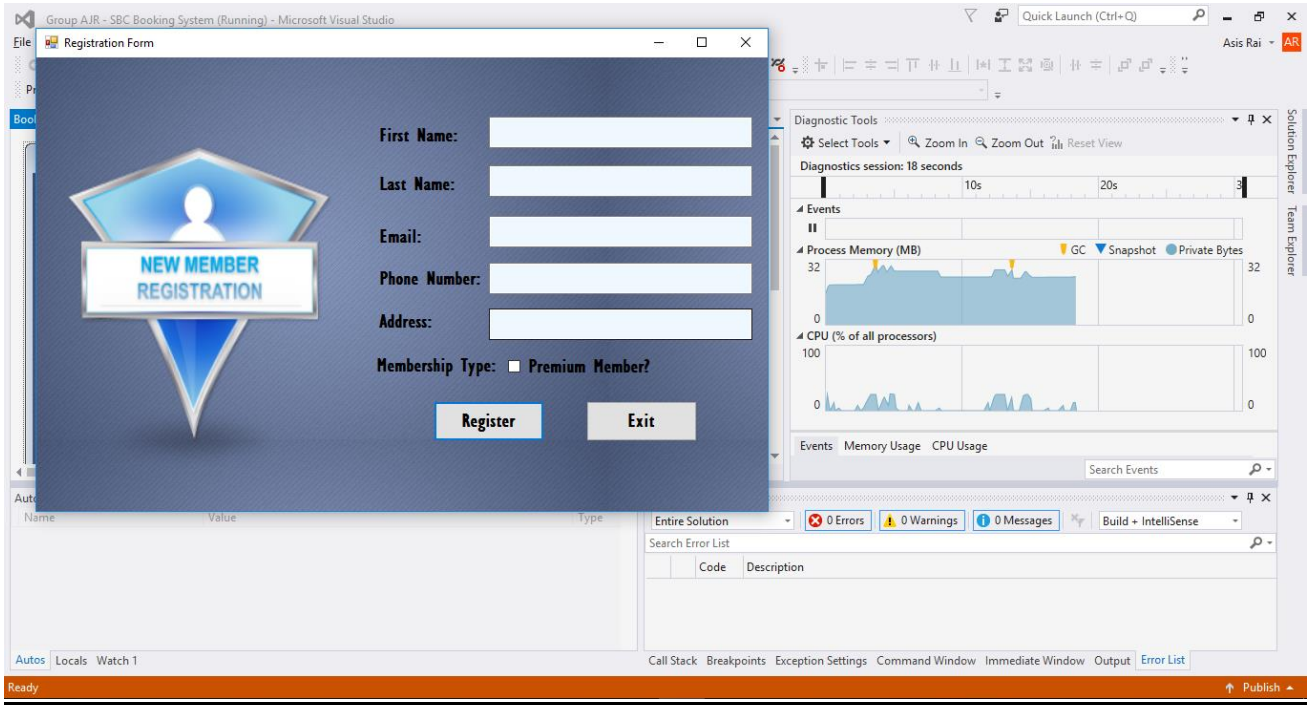
Login Screen



Main Menu



Registration



The feed back I received from the tutor was:

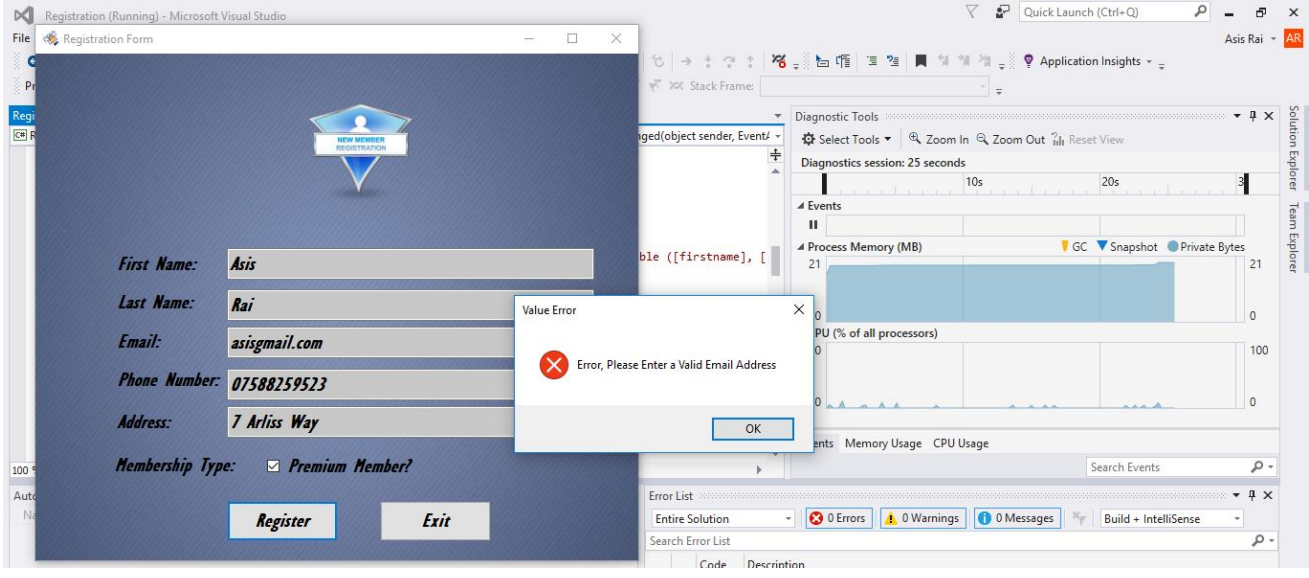
Front End

- Make the logo more visible – Text in the logo is not very clear to see
  = As you can see I have made the logo bigger, it is now more visible and the text is clear and visible to read.

- Align the text boxes, make them consistent with each other
  = I have aligned the text boxes into the same margin and have made them consistent.

Back-End

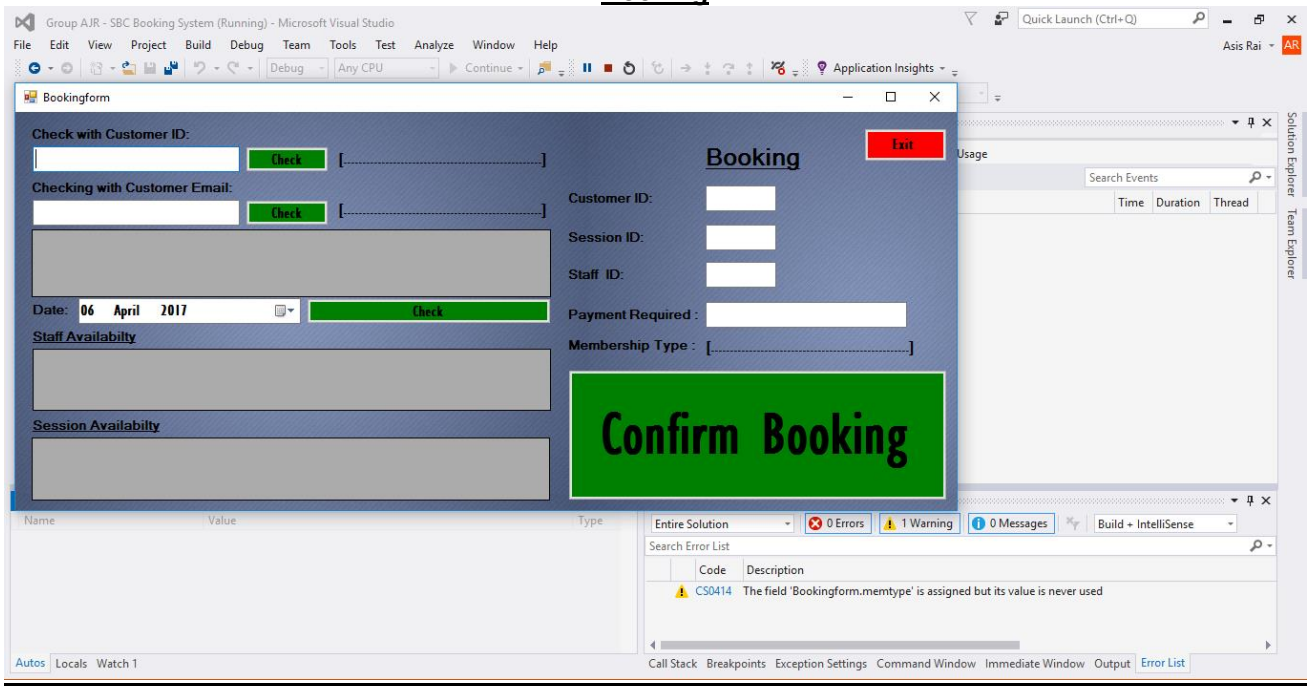
- Add some validation to the text boxes – such as email should check '@'.



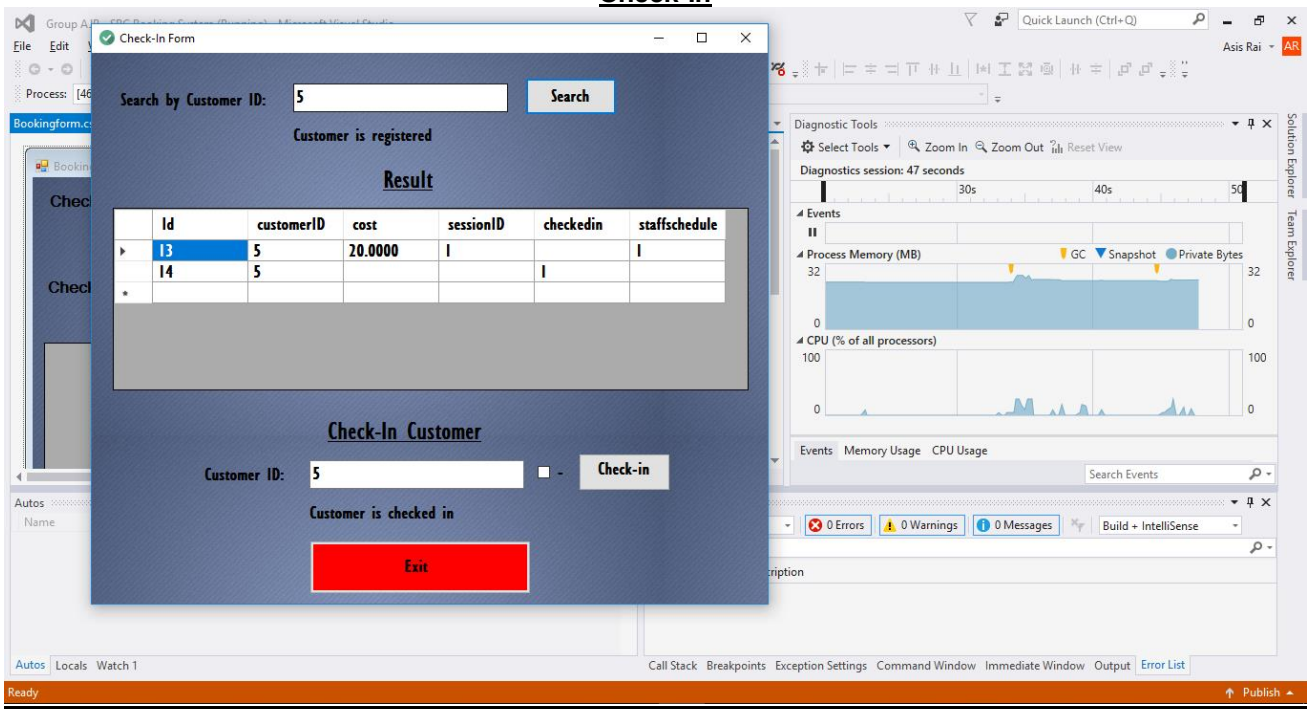

```
37
38          string email = textBox3.Text;
39          if (email.IndexOf('@') == -1 || email.IndexOf(".") == -1)
40          {
41              MessageBox.Show("Error, Please Enter a Valid Email Address", "Value Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
42          }
```

=As you can see I have made changes to the intial prototype code which throws an error message if there is an '@' missing.

Booking



Check-In



Task 2: To evaluate the management of the project – evidence with a report supported by research with references in the CU Harvard referencing style in a maximum of 1,000 words

group meeting records **(as a group)**

Weekly Report

Week 1

- Group Introductions and agreeing on the software to use, and assigning

	the functionalities to the group members and exchanging contact information.
Week 2	• We have agreed on using the C# in visual studio. Jubad has been assigned booking, Asis has taken the responsibility to create login screen and registration forms. Rakshak has been assigned the check in Form and Harvey has been assigned Manager User interface. • Designing use case diagrams for the project • We also started working on producing the use case diagrams and use case description for our individual functionality. We are also researching about how to use C# language. • We have been introduced to the layered models, therefore, we have started working on understanding how to come up with a layered model for our own functionality.
Week 3	• We have started learning about grasp patterns where we had to learn about different types of controllers and how they function. • Made a start on programming our functionalities in c#. We've also started working on designing the database for the system and the front end look of the system.
Week4 and Week5	• Still continuing to work on the creating the system, since C# is a new language to all four of us, we are starting from the basics, and working our way up. We have completed use case diagrams and use case descriptions. • At this point we have finished creating the database and tables. We are further researching on the connectivity of the database to the forms created. • This week we met up for a group meeting outside of lessons to complete the 4 layered model for one of the system functionality.
Week 6	• This week we are learning about gang of four design patterns. • We have managed to complete system analysis and how the user interface of the system should look. • Some of the issues we had are going forward with database connectivity with forms. Therefore, we have decided to use different external methods to create database and connecting it by importing it. • Rakshak has started working on the SQLite studio, whereas Jubad was trying to use Microsoft SQL server software to create database. • On the other hand, the Asis and harvinder have been working on getting their layered models done as well as their functionalities. Asis has finished

	his login screen
Week 7 and 8	• In this week we have managed to design all the forms should look, but the booking and check in systems are quite hard to implement on code, as they are quite complex. • We have also been attending some of the support sessions to get more help on the layered models, Rakshak, jubad and Asis have managed to complete the finalised version of their layered models. Whereas, harvinder is still yet to complete his model. • We have also started preparing for the in class test.
Week 9	• Asis has finished his 4 layered model and we have all started working on the user case controller pattern. • Rakshak and jubad and harvinder are still working on their functionalities. Asis has managed to figure out how to update a table inside a database, through windows form. • Booking functionality has also been partly done, and in order to do checking in functionality, booking functionality needs to be fully completed. • Harvinder is still working on the four layered model and his Manager Tools functionality.
Week 10	• In this week we have completed the booking and registration completely and we are working on checking in functionality. • Asis and jubad have received feedback on their functionalities and have started making the appropriate changes based on the feedback received. • This week we have created GitHub accounts and started making commits with all the work we have produced. • Rakshak is working his checking in functionality. On the other hand we have been working on implementing the gang of four patterns, which we have been struggling for a while. • We have also been writing our evaluation reports on our system and started to put all the work we did into our portfolios. • We have also started working on the unit testing and assignment tasks. • The groups is still awaiting on Harvinder's layered model and functionality
Week	• This week Rakshak has completed his checking in system, and has

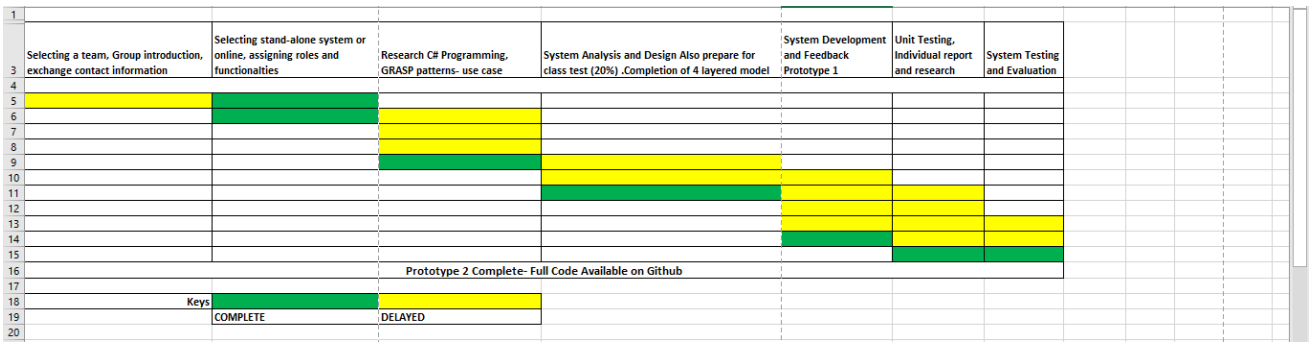
11

received feedback. He has completed his feedback functionality and has started working on this assignment tasks.

- Whilst Rakshak has been working on his feedback, Asis and jubad have been on completing their program based on the feedback given and completing the assignment submission.

- Harvinder is still yet to finish his layered model, use case diagram and functionality.

project planning (e.g. Gnatt Chart) (**as a group)**

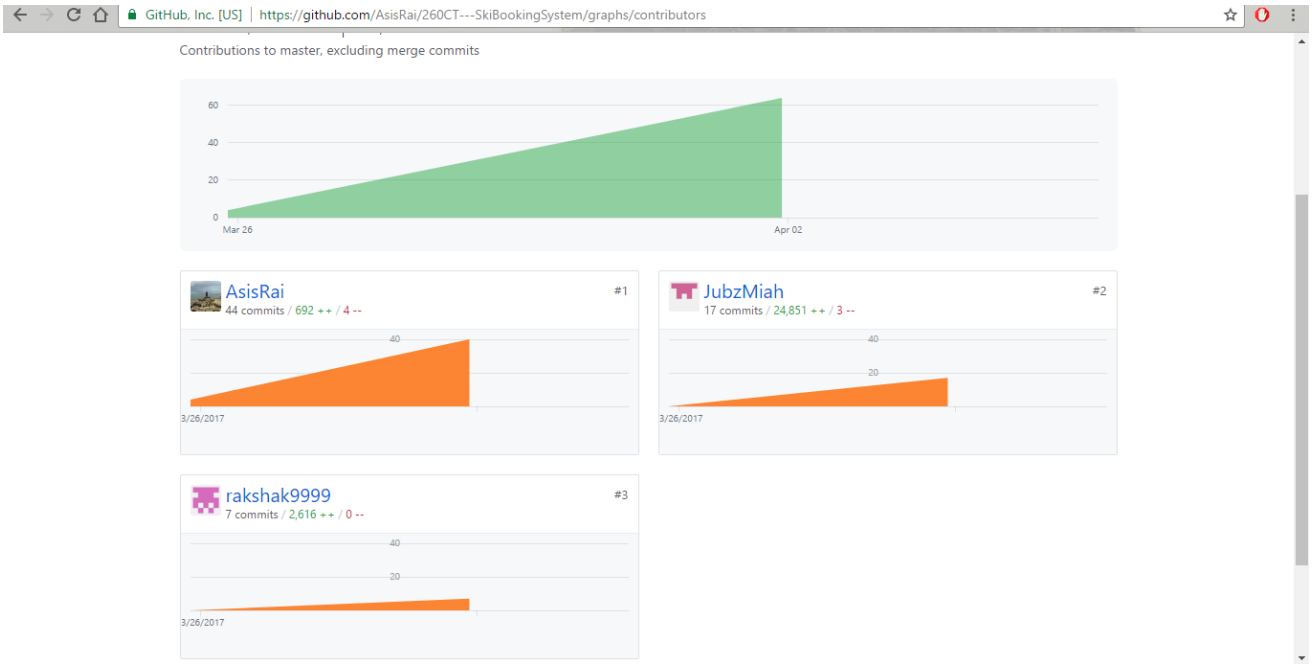


a summary of feedback from the tutor (acting as the client) (**as an individual)**

The feedback from the tutor was helpful as he tested the whole system. The feedback he gave me for my functionality only also helped as It made the form look better and the text more visible. The validation feedback was helpful as it made sure that there was '@' in the email text box which I should have thought about as in the real world this is applied. Overall the feedback was helpful not just for my functionality but for the whole system as it made sure that the system work without any errors and any errors that we could have missed was picked up by the tutor so that we would work on it as a group and made the code was not obsolete.

evidence of group working via GitHub regular commits in screen shots (**Individual commits contributing to the group prototype)**

GitHub link - https://github.com/AsisRai/260CT---SkiBookingSystem



Evaluation of the prototyping method

I used The incremental prototyping method**.** The incremental model method is a method of software development where the model is designed first, then created and tested incrementally (a little more is added each time) until the product is finished. The steps in this model are Initial-Planning →Planning →Requirements→Analysis & Design→Implementation→Deployment→Testing. This model involves both development and maintenance. This model combines the elements of the waterfall model with the iterative philosophy of prototyping.

I followed this model method to create my registration functionality and our whole group followed this model as well to create their own respective models. The registration functionality is defined as a finished product as I satisfied all its requirements. We used this method by creating our components separately, all of which were designed and built separately. Each component was derived to the tutor when they were completed for feedback. This helped me by avoiding long development time as I could see how our first prototypes worked and therefore partly utilising all our prototypes. This helped me by showing that for booking functionality to work I needed to add a membership type in my functionality, free and premium membership. This saved me and the group development time as this would have influenced how the whole system worked and saved me from doing extra work.

It specifically helped me in my functionality by generating working software quickly and early as I was creating my functionality separately to the others and it helped me test my code numerous times and change it without influencing other functionalities. I could test the code as many times I wanted and I could de-bug the errors every time there was an error, this helped me by giving me a plan on what to code, ways to code so that the form will connect to the SQL database and store the values.

However, there was also downside to using this model. Each time a new prototype was created, it would have to be implemented with other functionalities and the code would have to be adjusted accordingly either in the new prototype or in the other functionality where the new prototype is being implemented. Although we discussed we would use the same software and database to create the system, there are several ways where you can code which allows you to interact between the main software and the database. This created a problem for me as I had one way to connect to SQL and my way of saving and getting the data from SQL database did not match with the one in Booking functionality and this created a debate to which one of us would have to change their code. This created problems as this delayed the development of the system, because of this method turned out to be a Build-and-fix model couple of times.

Other problems I faced was linking the database together. As I created my functionality separately I had to create an independent SQL database to make sure that my functionality was working. This became problematic because once again the database table were different to Booking functionality's database table names. This meant that the destination of the database was also different and so was the code as the database data table and data types were different. This made it difficult during iteration of the main system because this meant the iteration was rigid and did not overlap each other because of different table records.

I researched several prototyping methods to be applied to the project. One of them was the agile scrum method and it is based on iterative and incremental method, where requirements and solutions evolve through collaboration between cross-functional teams. Agile methods or Agile processes generally promote a disciplined project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices intended to allow for rapid delivery of high-quality software. I did not like this method as this required a Group leader to be elected and it also meant that I would have frequent inspection to my functionality and I would also have to inspect other functionalities which would slow down the development process because I would have to keep adapting the registration functionality code to the code of the other functionalities, therefore I and the team voted not to use this prototyping method.

In conclusion, I think that following the incremental prototyping method really helped as an individual and as team member. It gave me flexibility to plan, design and implement my registration functionality as we agreed to present all our functionalities after we finished it, this gave me 2 weeks to plan and design the 2 more weeks to implement the design into an actual working registration functionality. This helped me make the functionality quicker and earlier than others because I had little pressure and more time to focus on designing the interface and planning the code and therefore I ended up with littles mistakes with code and that meant I had little de-bugging to do. This allowed me to get an early feedback from the tutor and improve my functionality even more and made sure that I met all the requirements for my functionality.

References

Ghahrai, Amir. "Incremental Model - Advantages And Disadvantages". *Testing Excellence*. N.p., 2017. Web. 6 Apr. 2017.

Moodle, Coventry. "Design Patterns". *https://cumoodle.coventry.ac.uk*. N.p., 2017. Web. 6 Apr. 2017.

"What Is Agile? What Is Scrum?". *cPrime*. N.p., 2017. Web. 6 Apr. 2017.