GROUP ASSIGNMENT

CT044-3-1-IOOP

Introduction to Object Oriented Programming

MODULE CODE : CT044-3-1-IOOP

ASSIGNMENT TITLE : Documentation Group Assignment Report

INTAKE : APD1F2106CS(CYB), APU/APD1F2106CGD

DATE ASSIGNED : 2nd December 2021

DATE COMPLETED : 16th February 2022

GROUP NO. : Group 35

LECTURER : Ms. Sathiapriya A/P Ramiah

No.	Student Name	TP Number
1	Kazim Chin Shen Bin Ismail (Group Leader)	TP064473
2	Soh Wei Zheng	TP063631
3	Tan Ming Hui	TP064987
4	Teoh Will Shen	TP063735

Table of Contents

1.0 Contents 4
1.1 Storyboard 4
1.1.1 Login Page - Teoh Will Shen TP063735 4
1.1.1.1Login Page (Storyboard) 4
1.1.2 Receptionist - Teoh Will Shen TP063735 6
1.1.2.1 Receptionist Page (Storyboard) 6
1.1.2.2 Edit Receptionist Profile Page (Storyboard) 9
1.1.2.3 Payment Page (Storyboard) 11
1.1.2.4 Receipt Page (Storyboard) 13
1.1.3 Technician - Soh Wei Zheng TP063631 15
1.1.3.1 Home Form (Storyboard) 15
1.1.3.2 View Requested Service Form (Storyboard) 16
1.1.3.3 Add Description and Laptop Collection Date Form (Storyboard) 18
1.1.3.4 Edit Profile Form (Storyboard) 20
1.1.4 Customer - Tan Ming Hui TP064987 21
1.1.4.1 View ongoing service information page (storyboard) 21
1.1.4.2 Edit customer profile (storyboard) 22
1.1.4.3 Change requested service (storyboard) 23
1.1.5 Admin - Kazim Chin Shen Bin Ismail TP064473 25
1.1.5.1 Home Form (Storyboard) 25
1.1.5.2 Income Report Form (Storyboard) 27
1.1.5.4 Admin Edit Profile Form (Storyboard) 29
1.2 Use-Case Diagram 31
1.3 Class Diagram 32
1.3.1 Login 32
1.3.2 Receptionist 32
1.3.3 Admin 33
1.3.4 Customer 34
1.3.5 Technician 36
1.4 Object-Oriented Programming Concepts 38
1.4.1 Login - Teoh Will Shen TP063735 38
1.4.1.1 Class 38
1.4.1.2 Methods 39
1.4.1.3 Objects 40

1.4.1.3 Events 41
1.4.2 Receptionist - Teoh Will Shen TP063735 44
1.4.2.1 Class 44
1.4.2.2 Methods 50
1.4.2.3 Objects 57
1.4.2.4 Events 59
1.4.3 Admin 71
1.4.3.1 Class 71
1.4.3.2 Methods 72
1.4.3.3 Objects 75
1.4.3.4 Events 76
1.4.4 Customer 82
1.4.4.1 Class 82
1.4.4.2 Methods 84
1.4.4.3 Objects 91
1.4.4.4 Events 93
1.4.5 Technician - Soh Wei Zheng TP063631 99
1.4.5.1 Class 99
1.4.5.2 Methods and Object 103
1.5 Test Plan and Test Cases 124

2.0 Conclusion 130

3.0 References 132

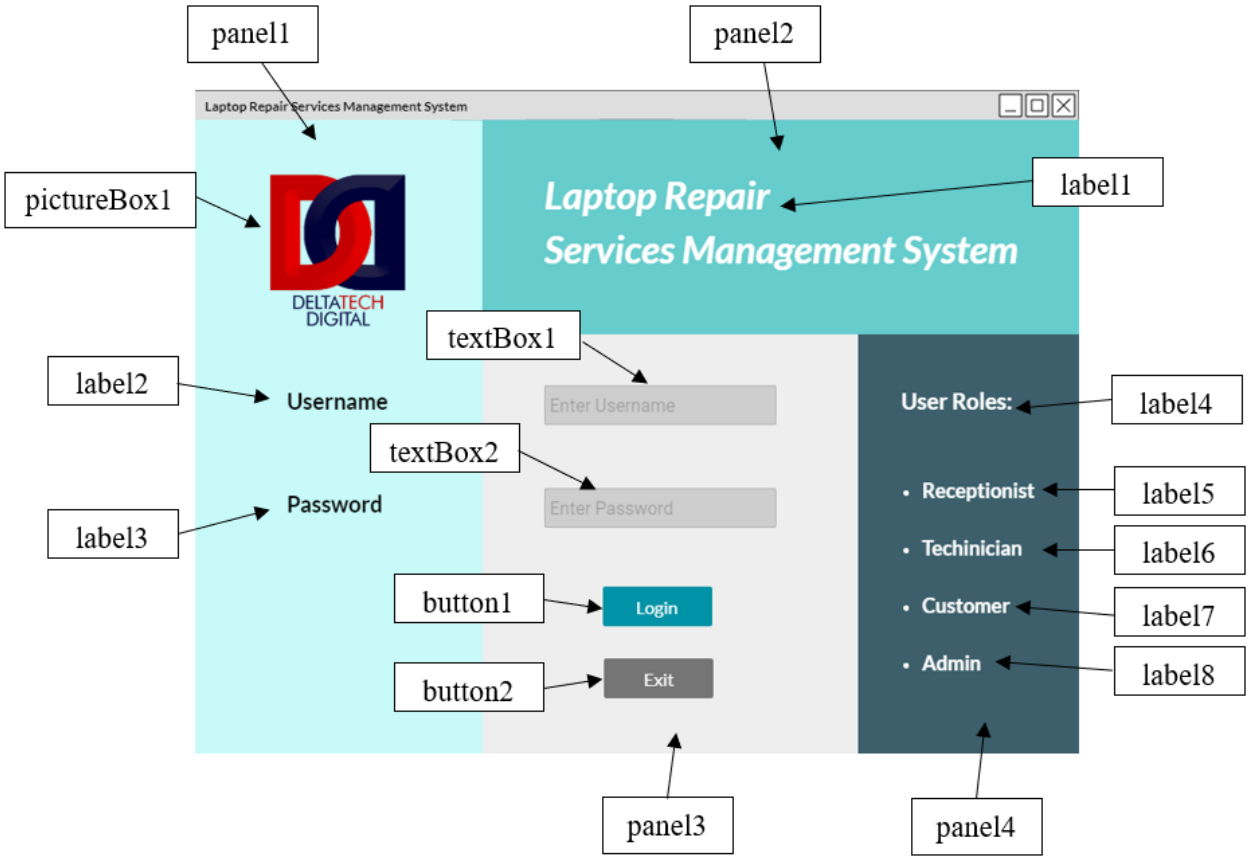
4.0 Workload Matrix 133

1.0 Contents

1.1 Storyboard

1.1.1 Login Page - Teoh Will Shen TP063735

1.1.1.1Login Page (Storyboard)

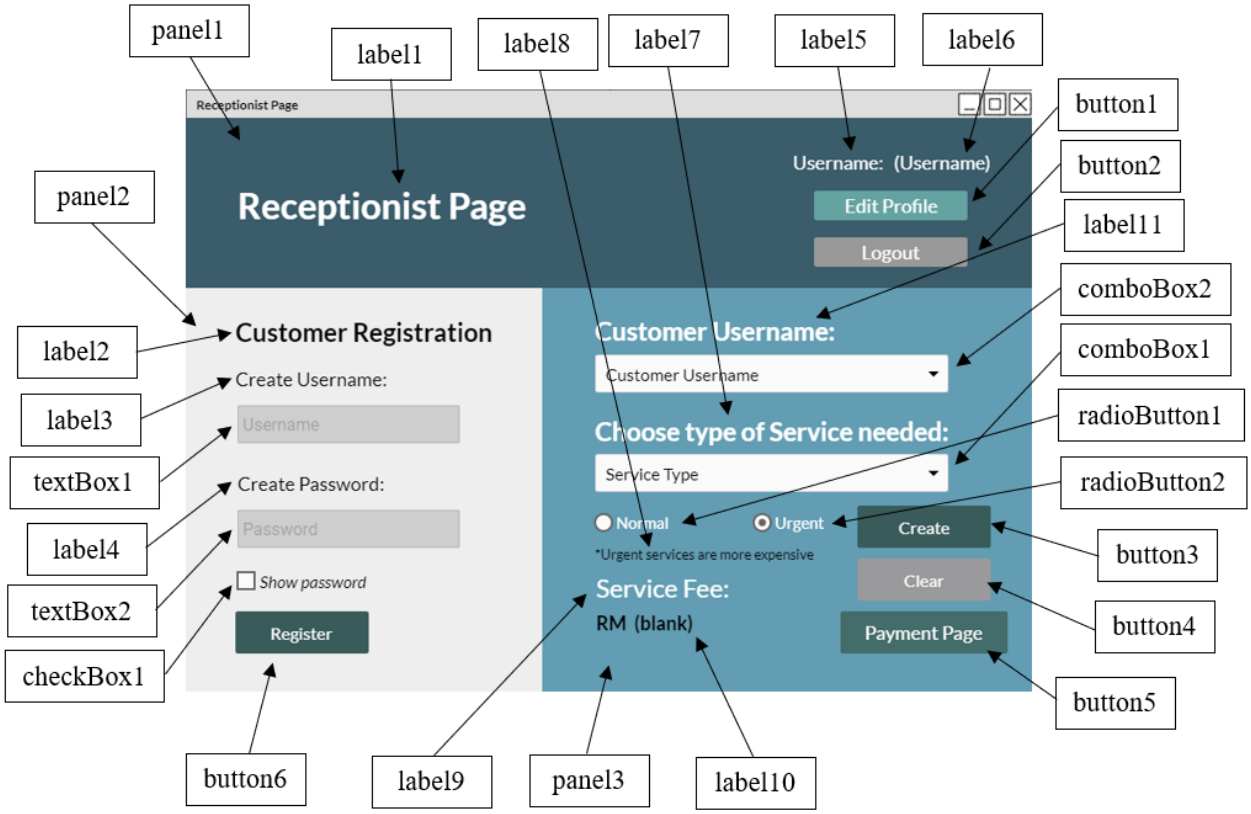


Control	Control Name	Description
panel1	pnlLogo	To separate each part of the interface
panel2	pnlLogTitle	

panel3	pnlLogInput	
panel4	pnlLogDes	
label1	lblLogTitle	To label the Login Page Title
label2	lblLogUsnmTag	To label the related controls
label3	lblLogPwdTag	
label4	lblLogRoleTitle	
label5	lblLogRcptn	
label6	lblLogTechn	
label7	lblLogCust	
label8	lblLogAdmin	
textBox1	txtLogUsnm	Let user enter username to login
textBox2	txtLogPwd	Let user enter password to login
button1	btnLogLogin	Press to login to system
button2	btnLogExit	Press to exit the system
pictureBox1	picLogLogo	Show the system logo

1.1.2 Receptionist - Teoh Will Shen TP063735

1.1.2.1 Receptionist Page (Storyboard)

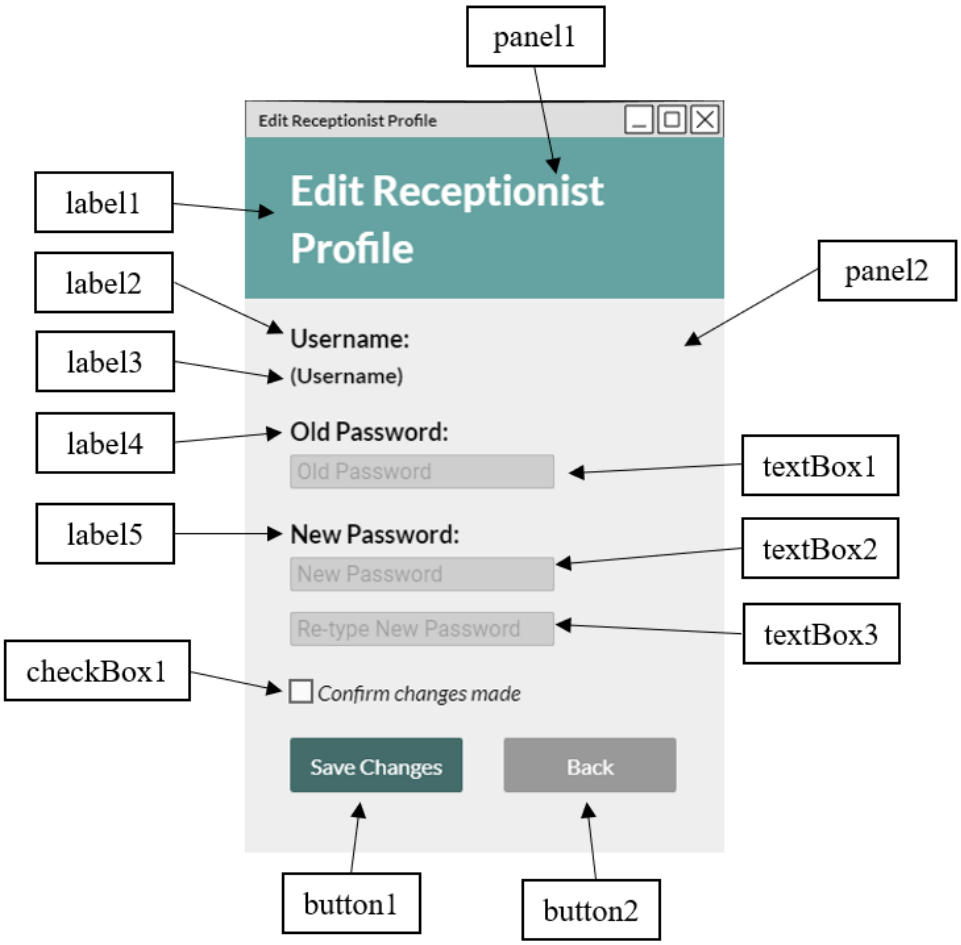


Control	Control Name	Description
panel1	pnlRcptnTitle	To separate each part of the interface
panel2	pnlRcptnCustReg	
panel3	pnlRcptnSer	
label1	lblRcptnPg	The title of the Receptionist Page
label2	lblRcptnCustReg	The title of Customer Registration
label3	lblRcptnCustUsnmTag	Guide the user what to type

label4	lblRcptnCustPwdTag	on the textbox under it
label5	lblRcptnUsnmTag	Explain what is the label beside
label6	lblRcptnDetsUsnmSh	Show the current logged in receptionist
label7	lblRcptnTosTag	The title of Choose type of service
label8	lblRcptnSerMes	Display the message saying urgent service is more expensive
label9	lblRcptnSerFeeTag	The title of Service Fee
label10	lblRcptnSerFeeSh	Show the service fee
label11	lblRcptnCustUsnmCboTag	The title of customer username combo box
textBox1	txtRcptnRegCustUsnm	Let user to enter the customer username to be created in the system
textBox2	txtRcptnRegCustPwd	Let user to enter the customer password to be created in the system
button1	btnRcptnEditPrf	Press to go to the update receptionist profile page
button2	btnRcptnLogout	Press to exit the receptionist page
button3	btnRcptnCreCust	Press to create the customer and details, and save it into the system
button4	btnRcptnClear	Clear all inputs of the receptionist page
button5	btnRcptnPymtPg	Press to go to the payment

		page
button6	btnRcptnRegCust	To register the customer
checkBox1	chkRcptnShPwd	Tick to show the password entered on the textbox above
radioButton1	radRcptnNormSer	Press to choose the normal service
radioButton2	radRcptnUrgSer	Press to choose the urgent service
comboBox1	cboRcptnSerType	Enable the user to choose a list of service
comboBox2	cboRcptnCustUsnm	Choose the Customer username to create the service for them

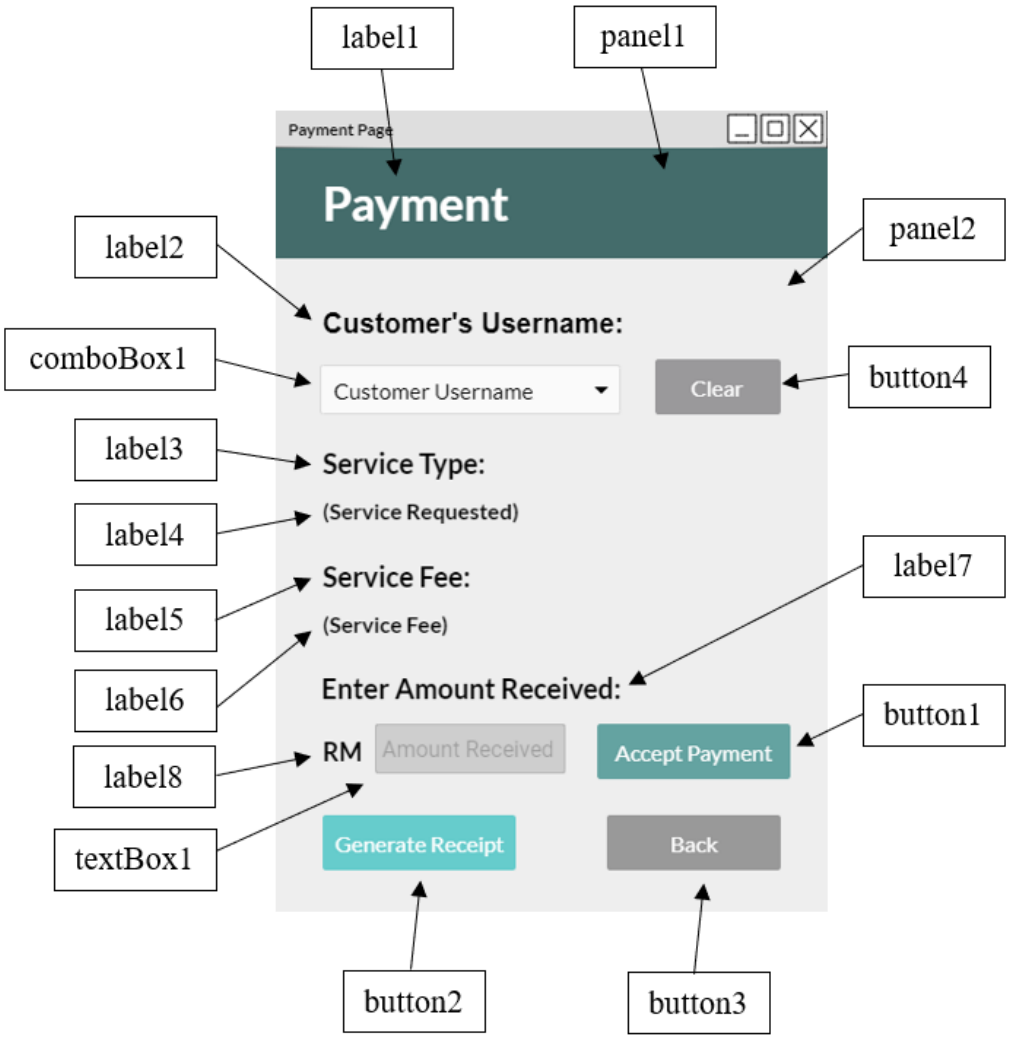
1.1.2.2 Edit Receptionist Profile Page (Storyboard)



Control	Control Name	Description
panel1	pnlRcptnEditPrf	To separate each part of the interface
panel2	pnlRcptnEditDets	
label1	lblRcptnEditPrf	The title of the Update Receptionist Profile Page
label2	lblRcptnEditUsnmTag	To label the related controls
label3	lblRcptnEditUsnmSh	To show the username of the current logged in receptionist
label4	lblRcptnEditOldPwdTag	To label the related controls

label5	lblRcptnEditNewPwdTag	
textBox1	txtRcptnEditOldPwd	Allow the receptionist to enter its old password
textBox2	txtRcptnEditNewPwd	Allow the receptionist to enter its new password
textBox3	txtRcptnEditReNewPwd	Allow the receptionist to re-type its new password
button1	btnRcptnEditPrfSave	Press to update and save the receptionist profile
button2	btnRcptnEditPrfBack	Press to go back to the receptionist page
checkBox1	chkRcptnEditConf	Check to confirm the changes made

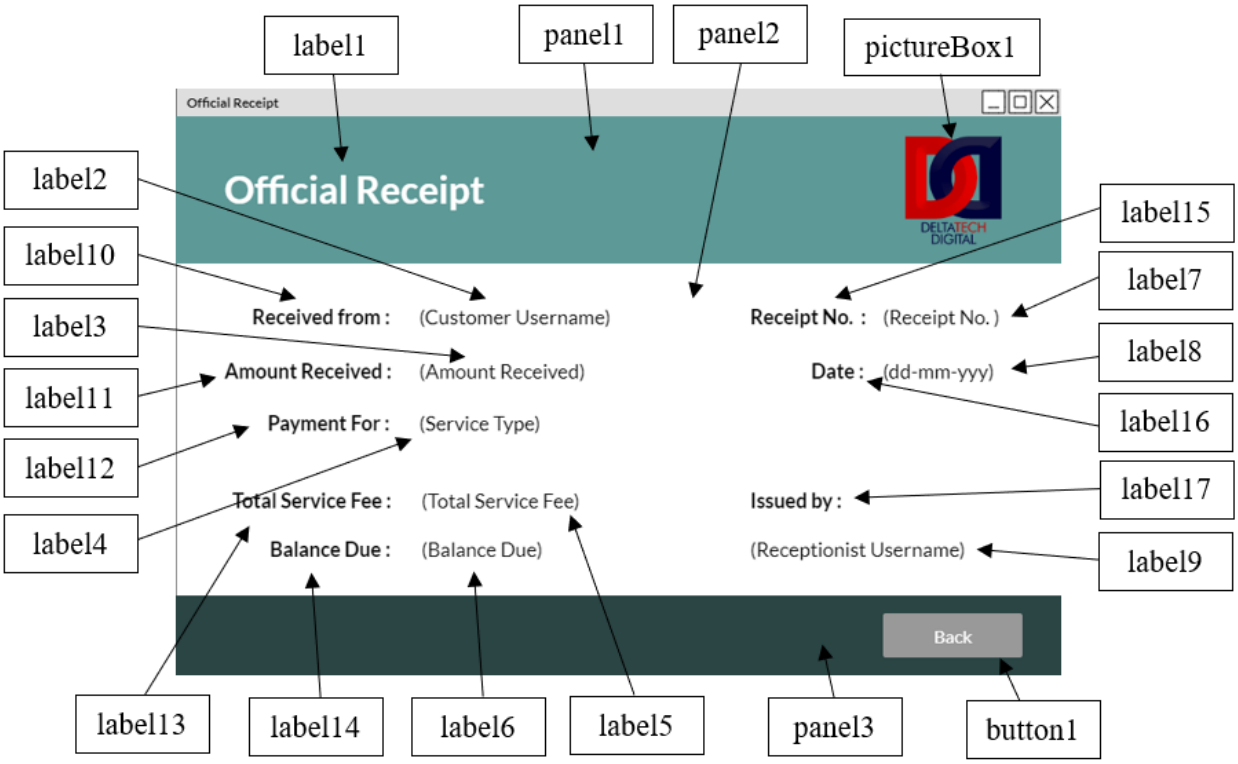
1.1.2.3 Payment Page (Storyboard)



Control	Control Name	Description
panel1	pnlRcptnPymt	To separate each part of the interface
panel2	pnlRcptnPymtDets	
label1	lblRcptnPymt	The title of the payment page
label2	lblRcptnPymtCustUsnmTag	To let the user know what to type in the textbox below

label3	lblRcptnPymtSerTypTag	To show what the label below shows
label4	lblRcptnPymtSerSh	To show the service chosen by the customer
label5	lblRcptnPymtSerFeeTag	To label the related controls
label6	lblRcptnPymtSerFeeSh	To show the service fee
label7	lblRcptnPymtRcvdTag	To show what to type in the textbox below
label8	lblRcptnPymtRcvdRM	To show the currency
textBox1	txtRcptnPymtRcvd	Allow the receptionist to enter the amount received from customer
button1	btnRcptnPymtAcc	Press to save the payment made by the customer
button2	btnRcptnPymtGenRcpt	Press to generate the receipt of the payment made
button3	btnRcptnPymtBack	Press to go back to the receptionist page
button4	btnRcptnPymtClear	Clear all controls in payment page
comboBox1	cboRcptnPymtCustUsnm	Choose the Customer username to create the payment for them

1.1.2.4 Receipt Page (Storyboard)

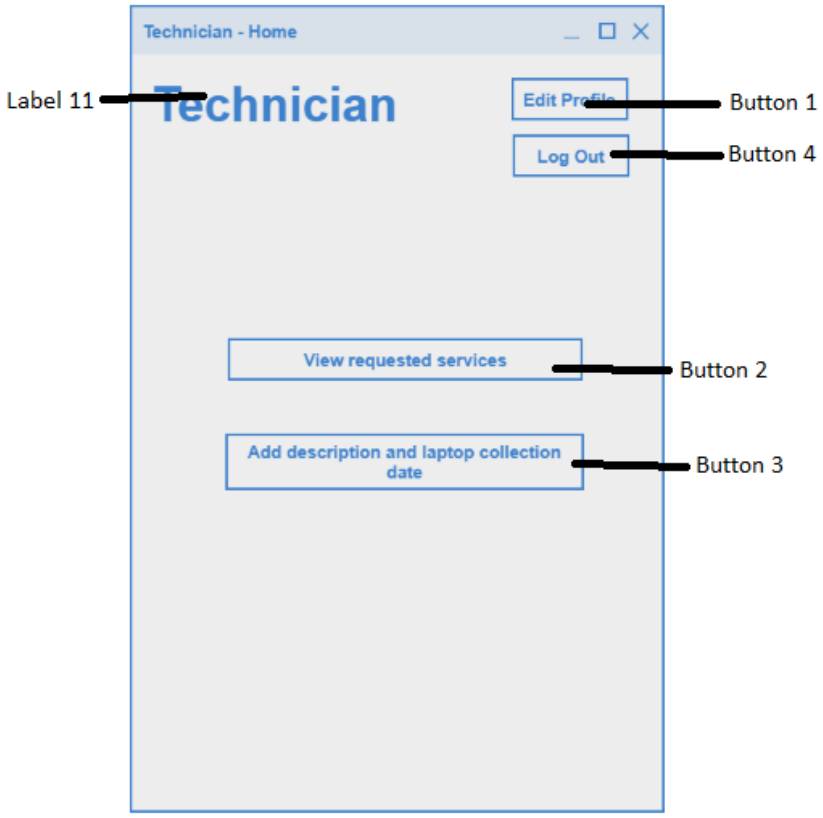


Control	Control Name	Description
panel1	pnlRcptnOfflRec	To separate each part of the interface
panel2	pnlRcptnRecDets	
panel3	pnlRcptnRecBack	
label1	lblRcptnRecTitle	The title of the form
label2	lblRcptnRecCustUsnm	To show the customer of the receipt
label3	lblRcptnRecAmtRcvd	To show the amount received by the customer
label4	lblRcptnRecSerType	To show the service chosen by the customer
label5	lblRcptnRecSerFee	To show the service fee

label6	lblRcptnRecBalDue	To show the balance due
label7	lblRcptnRecNo	To show the receipt number
label8	lblRcptnRecDate	To show the date when the receipt is issued
label9	lblRcptnRecRcptnUsnm	To show the receptionist who issues the receipt
label10	lblRcptnRecCustUsnmTag	To describe the corresponding label
label11	lblRcptnRecAmtRcvdTag	
label12	lblRcptnRecSerTypeTag	
label13	lblRcptnRecSerFeeTag	
label14	lblRcptnRecBalDueTag	
label15	lblRcptnRecNoTag	
label16	lblRcptnRecDateTag	
label17	lblRcptnRecRcptnUsnmTag	
button1	lblRcptnRecBack	Bring back to the payment page
pictureBox1	picRcptnRecLogo	This is the system logo

1.1.3 Technician - Soh Wei Zheng TP063631

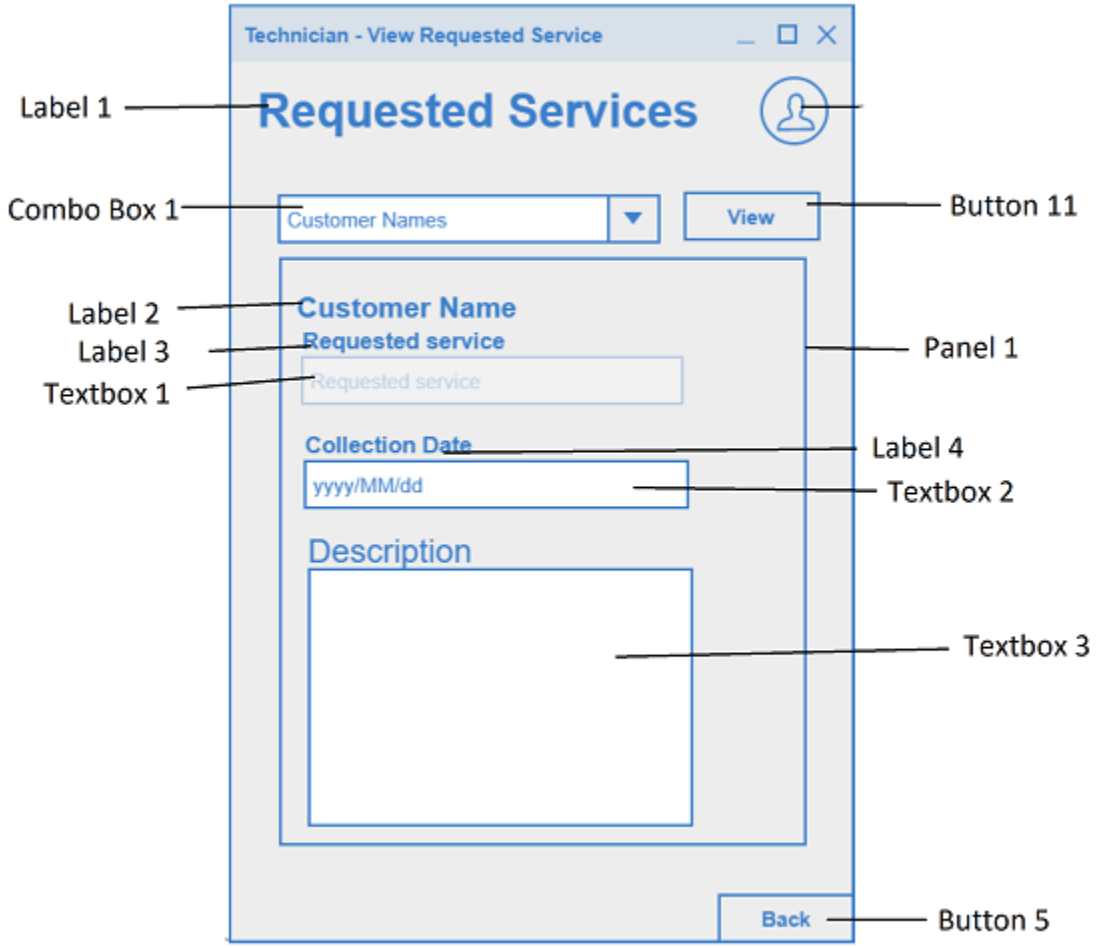
1.1.3.1 Home Form (Storyboard)



Control	Control Name	Description
Label 11	lblTechnician	To label the form header
Button 1	btnTechEditProfile	To enable user to edit the profile
Button 2	btnTechViewService	To enable user to view the customer requested service
Button 3	btnTechEditDescDate	To enable user to add description and laptop

		collection date
Button 4	btnTechLogOut	To enable user to sign out

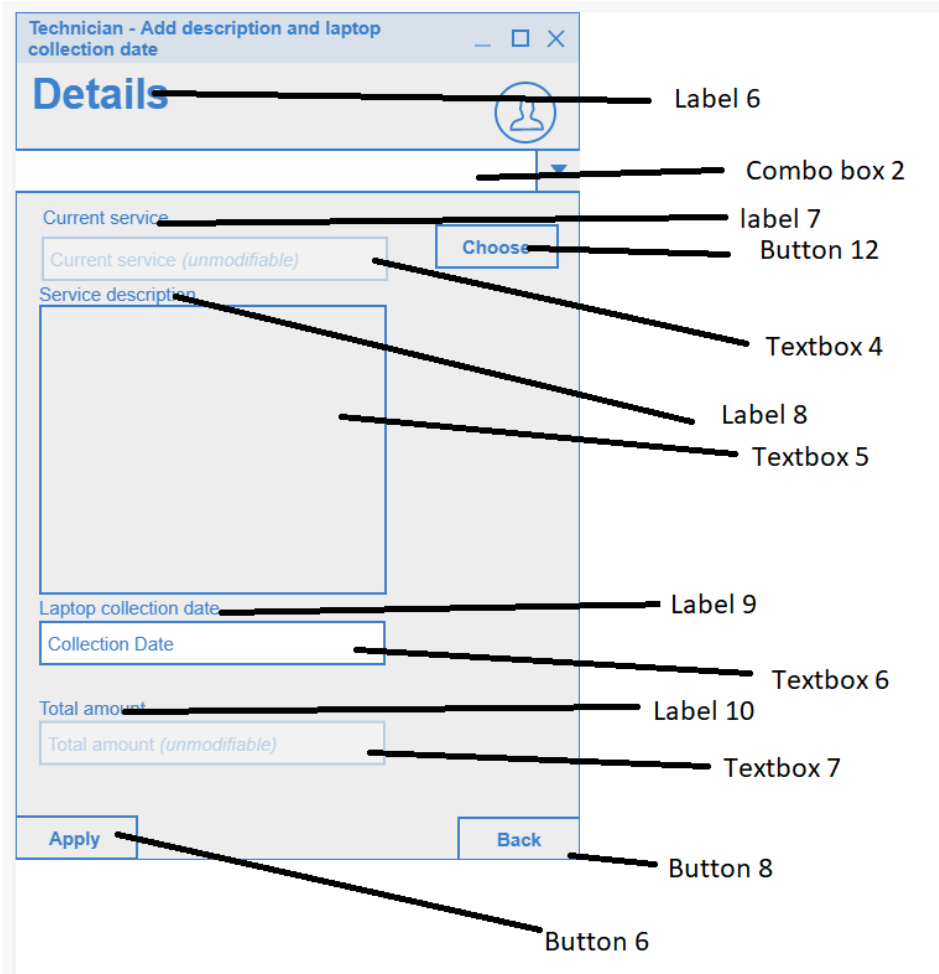
1.1.3.2 View Requested Service Form (Storyboard)



Control	Control Name	Description
Label 1	lblTechRequestedServices	To label the form header
Label 2	lblTechCustomerName	To label the panel header
Label 3	lblTechRequestedService	To label the related control

Label 4	lblTechCollectionDate	
Label 5	lblTechDescription	
Textbox 1	txtTechRequestedService	To let technicians know the customer's requested service
Textbox 2	txtTechCollectionDate	To let technicians enter the laptop collection date
Textbox 3	txtTechDescription	To let technicians enter the service's description
Button 5	btnTechBack	To enable technicians return to menu
Button 7	btnTechEdit	To enable technicians move to Edit form
Button 11	btnTechView	To enable technician view customer's details
Combo Box 1	cboTechCustomerNames	To enable technicians choose the customers
Panel 1	pnlTechCustomerName	To group the customer details

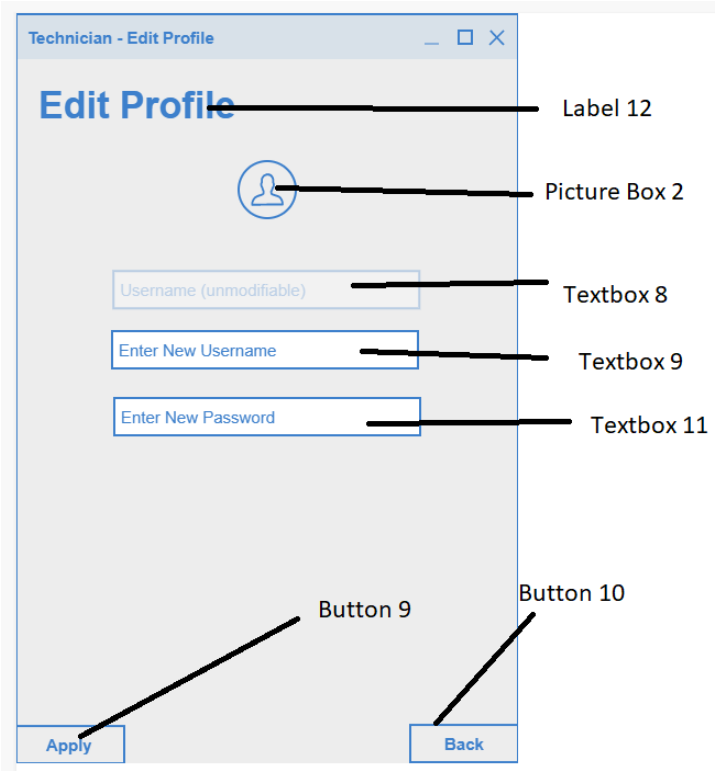
1.1.3.3 Add Description and Laptop Collection Date Form (Storyboard)



Control	Control Name	Description
Label 6	lblTechDetails	To label form header
Label 7	lblTechCurrentService	To label related controls
Label 8	lblTechServiceDesc	
Label 9	lblTechCollectionDate	
Label 10	lblTechTotalAmount	
Textbox 4	txtTechCurrentService	To let technicians know the

		customer's current service.
Textbox 5	txtTechServiceDesc	To let technicians add the service description of the service.
Textbox 6	txtTechCollectionDate	To let technicians add the laptop collection date
Textbox 7	txtTechTotalAmount	To let technicians know the total amount
Button 6	btnTechApply	To enable user to store the description and collection date
Button 8	btnTechBack	To enable user back to the menu
Button 12	btnTechCHoose	To enable user choose the customer
Combo box 2	cboTechUsername	To let technician select a customer

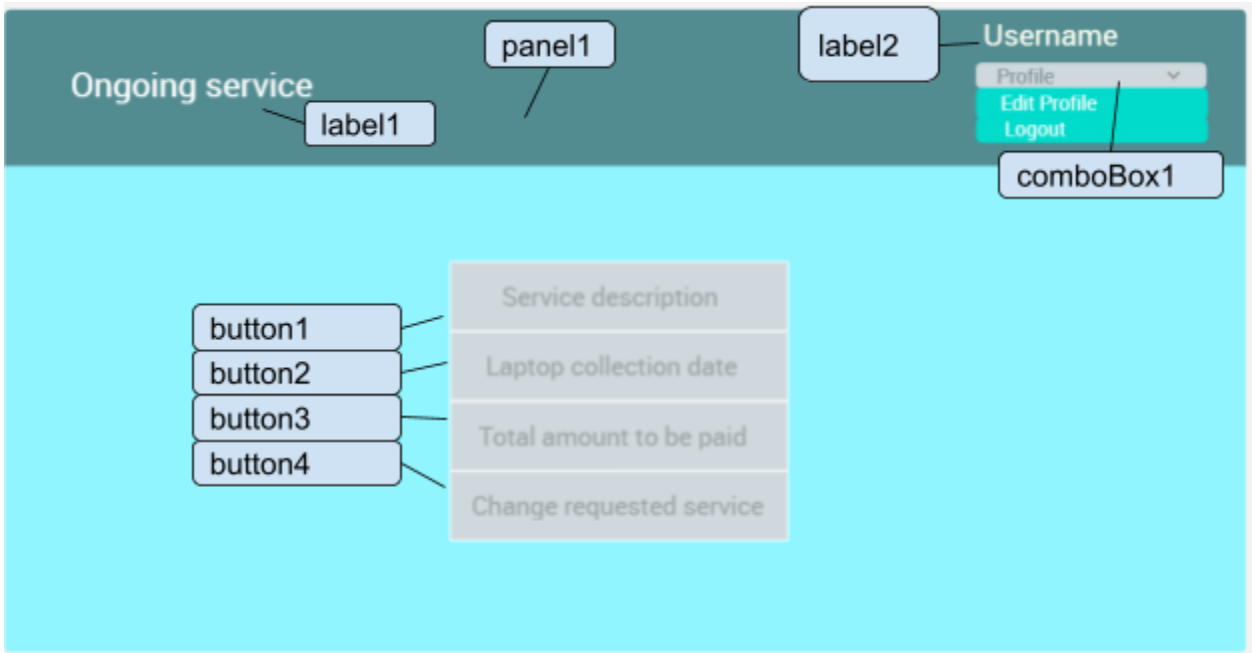
1.1.3.4 Edit Profile Form (Storyboard)



Control	Control Name	Description
Label 12	lblTechEditProfile	To label the form header
Picture Box 2	pcbTechEditProfile	To display logo
Textbox 8	txtTechUsername	To display the username
Textbox 9	txtTechName	To let technician enter new name
Textbox 11	txtNewPassword	To let technician enter the new password
Button 9	btnTechApply	To enable technician change all the information
Button 10	btnTechBack	To enable technician back to the previous page

1.1.4 Customer - Tan Ming Hui TP064987

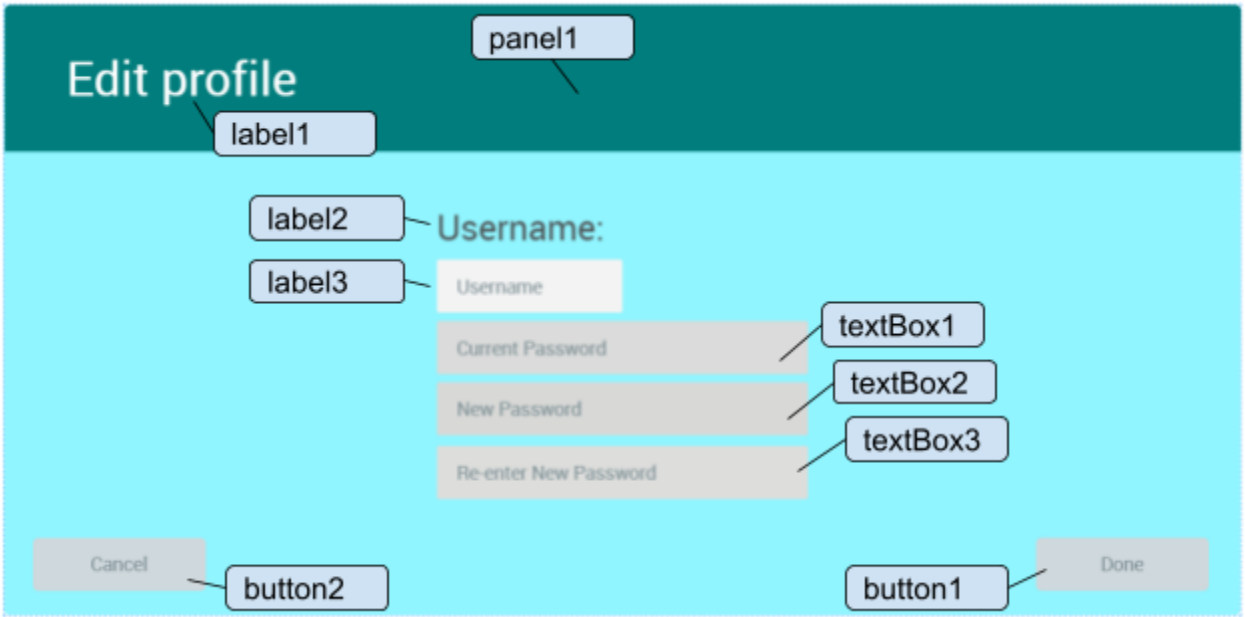
1.1.4.1 View ongoing service information page (storyboard)



Control	Control Name	Description
panel1	pnlOnServ	To separate the parts of interface
label1	lblOnServ	Page title
label2	lblUsername	To display username
comboBox1	cboProfileMenu	To let customer select profile related activity(edit or logout)
button1	btnServiceDesc	To let customer select the information to show
button2	btnCusCollectionDate	
button3	btnCusTotalAmount	

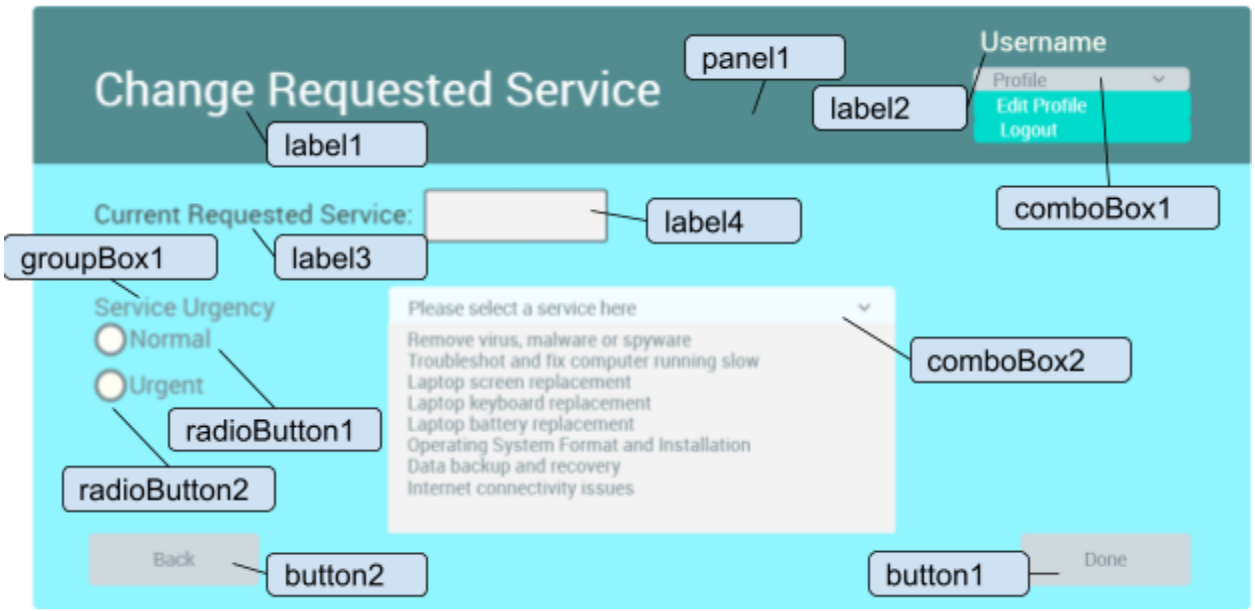
button4	btnServiceChg	To let customer change requested service

1.1.4.2 Edit customer profile (storyboard)



Control	Control Name	Description
panel1	pnlEditProfile	To separate the parts of interface
label1	lblEditProfile	Page title
label2	lblname	To indicate the label below is username
label3	lblUsername	To display username
textBox1	txtCurrentPw	To enter the current password
textBox2	txtNewPw	To enter the new password
textBox3	txtRePw	To re-enter the new password
button1	btnDone	To confirm the new changes
button2	btnCancel	To cancel the new changes

1.1.4.3 Change requested service (storyboard)

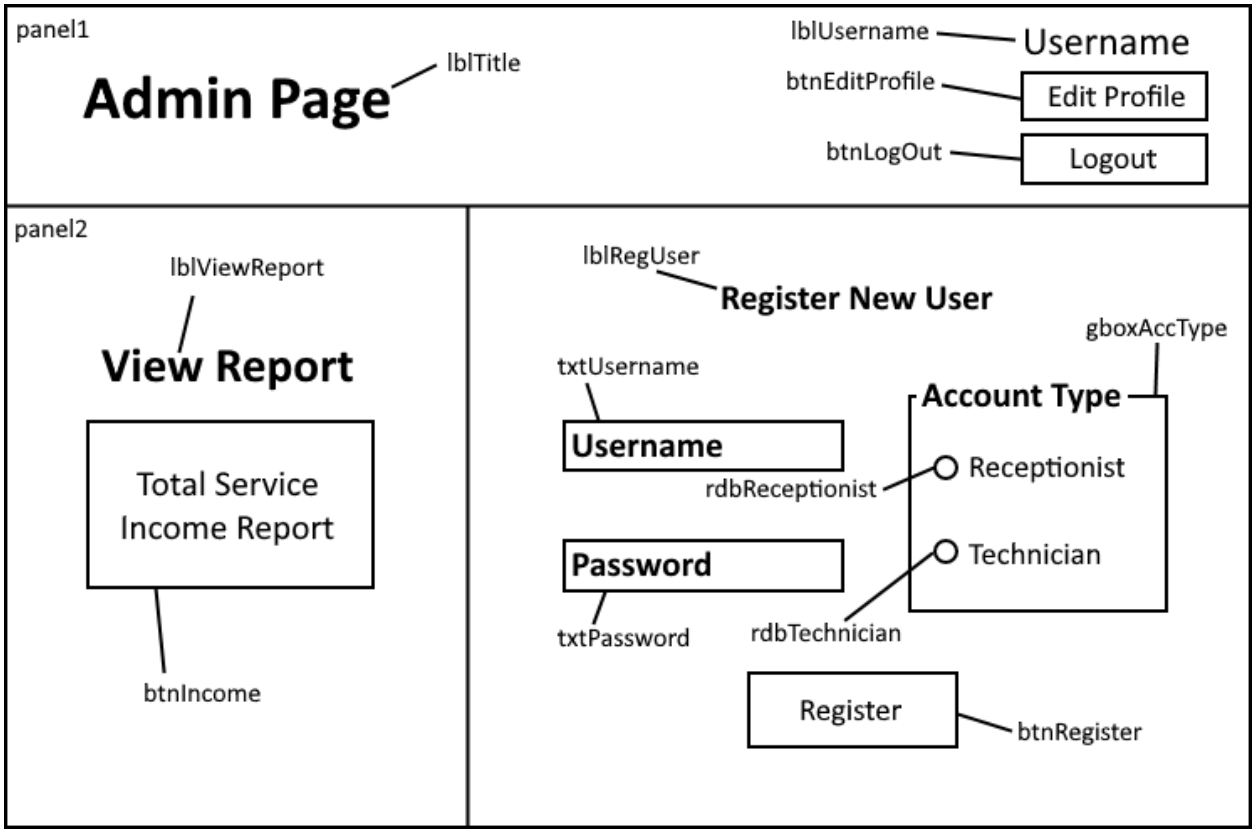


Control	Control Name	Description
panel1	pnlChgReqServ	To separate the parts of interface
label1	lblTitle	Page title
label2	lblUsername	To display username
label3	lblCurrentReqServ	To indicate the label on the right is current requested service
label4	lblReqServ	To display current requested service
comboBox1	cboProfileMenu	To let customer select profile related activity(edit or logout)

comboBox2	cboServChg	To show the selections for service change
groupBox1	grpBoxServUrg	To group up radio buttons
radioButton1	rdbNormal	To let customer select service urgency
radioButton2	rdbUrgent	
button1	btnDone	To confirm the new changes
button2	btnBack	To go back last page

1.1.5 Admin - Kazim Chin Shen Bin Ismail TP064473

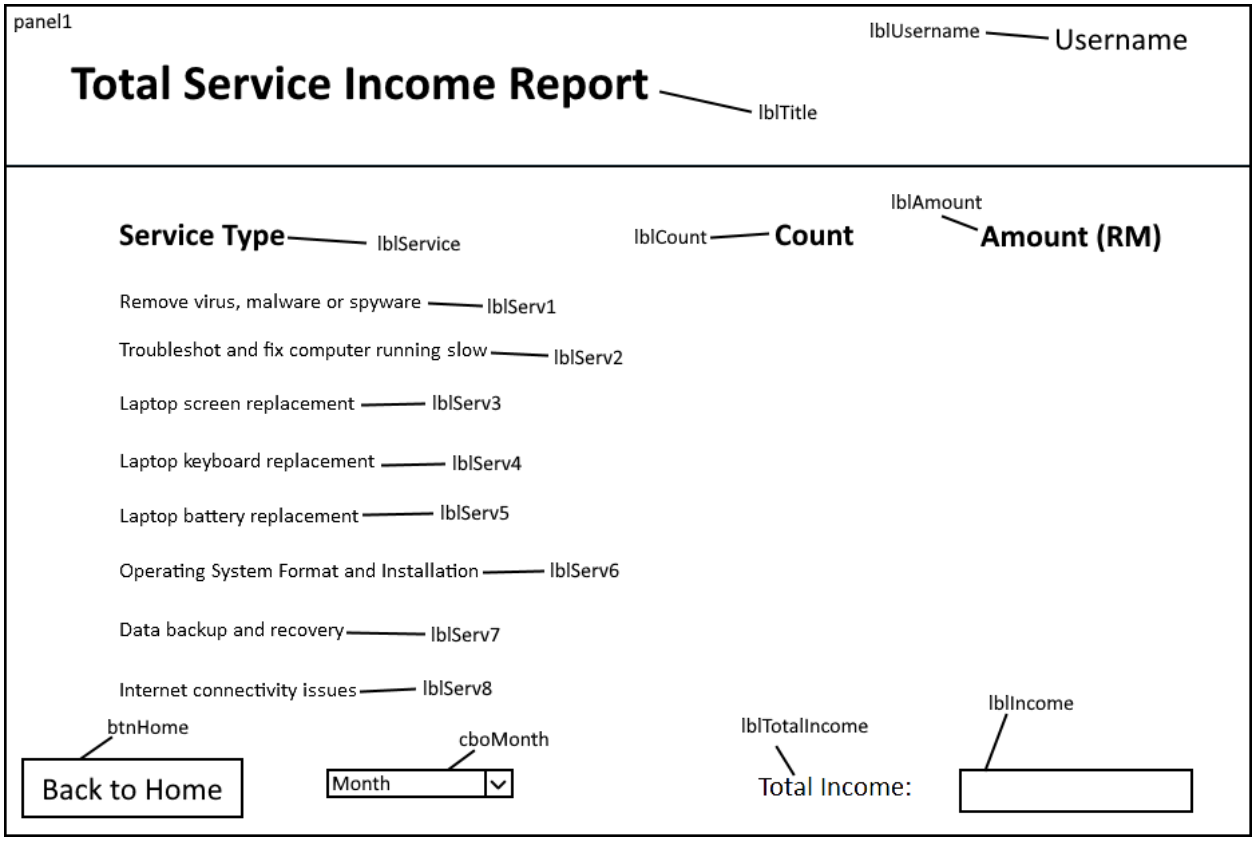
1.1.5.1 Home Form (Storyboard)



Control	Control Name	Description
panel1	pnlHeader	To separate the parts of interface
panel2	pnlReport	To separate the parts of interface
label1	lblTitle	To display current form title
label2	lblUsername	Displays Username
label3	lblRegUser	For interfacing
label4	lblViewReport	For interfacing

button1	btnEditProfile	To edit user account
button2	btnLogOut	To enable user to sign out
button3	btnIncome	To view report
button4	btnRegister	To register new user
textbox1	txtUsername	To enter new username
textbox2	txtPassword	To enter new password
groupBox1	gboxAccType	To group radio buttons
radioButton1	rdbReceptionist	To select receptionist account type
radioButton2	rdbTechnician	To select technician account type

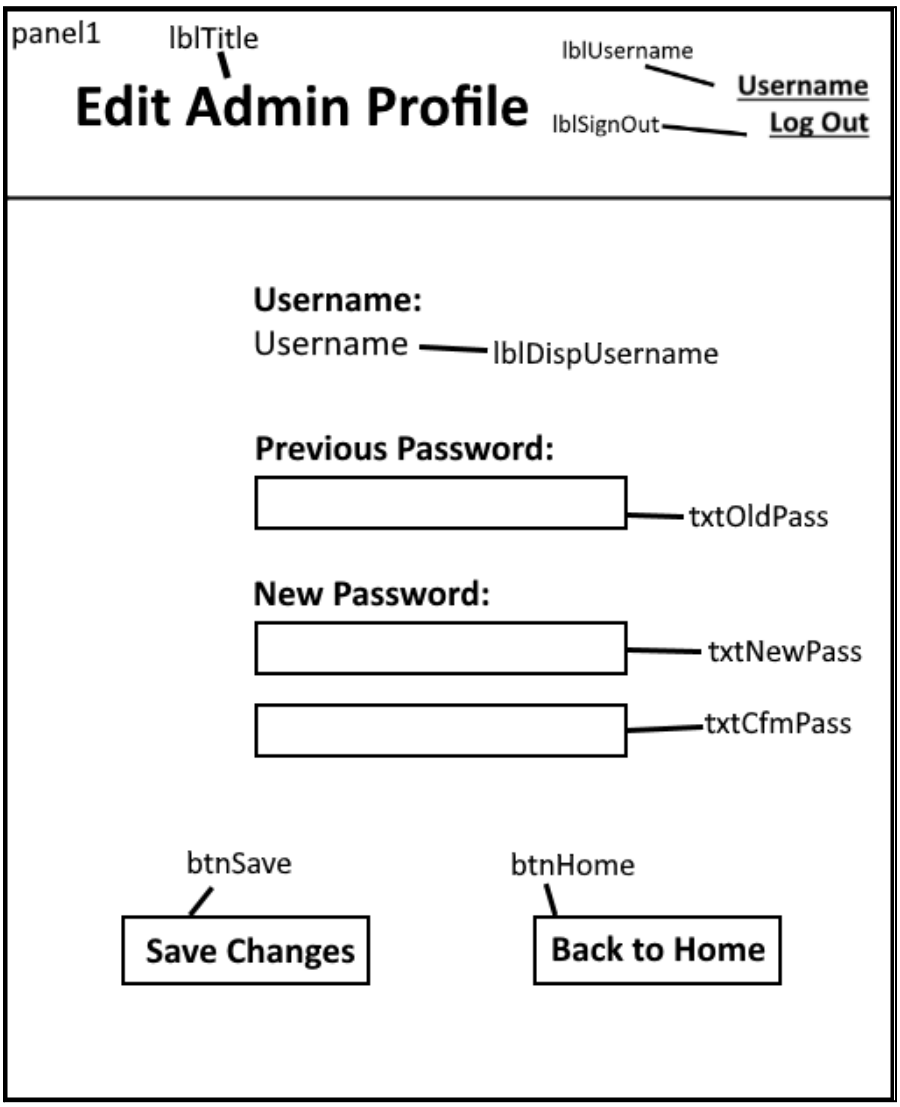
1.1.5.2 Income Report Form (Storyboard)



Control	Control Name	Description
panel1	pnlHeader	To separate the parts of interface
label1	lblTitle	To display current form title
label2	lblUsername	Displays Username
label4	lblTotalIncome	For Interfacing
label5	lblIncome	For displaying calculated total income
label6	lblService	For Interfacing
label7	lblServ1	For interfacing

label8	lblServ2	For Interfacing
label9	lblServ3	For interfacing
label10	lblServ4	For Interfacing
label11	lblServ5	For interfacing
label12	lblServ6	For Interfacing
label13	lblServ7	For interfacing
label14	lblServ8	For interfacing
comboBox1	cboMonth	To select month of income to view
button1	btnHome	Redirects back to home

1.1.5.4 Admin Edit Profile Form (Storyboard)



Control	Control Name	Description
panel1	pnlHeader	To separate the parts of interface
label1	lblTitle	To display current form title
label2	lblUsername	Displays Username
label3	lblDispUsername	Displays Username at form
button1	btnLogOut	Exits to login page
button2	btnHome	Redirects back to home
button3	btnEditProfile	Redirect to edit profile page
button4	btnSave	Save changes to profile
checkBox1	cboxConfirm	Confirm changes to profile
textBox1	txtOldPass	Input for old password
textBox2	txtNewPass	Input for new password
textBox3	txtCfmPass	Repeat new password

1.2 Use-Case Diagram

1.3 Class Diagram

1.3.1 Login

User_Login
private string username private string password
public User_Login(string, string) public string login(string)

1.3.2 Receptionist

Receptionist_Cust
private string custName private string custPwd private int serTypeIndex private string service_selected private string urgency private int serFee
public Receptionist_Cust() public Receptionist_Cust(string) public Receptionist_Cust(string, string) public Receptionist_Cust(int, string) public Receptionist_Cust(string, string, string, int) public string addCust() public static ArrayList addCustCbo() public int showPrice()

public string addService() public ArrayList showPayment() public string addPaidAmount(string) public double calBal(int, double)

Receptionist_Prof
private string rcptnName private string oldPass private string newPass private string reNewPass
public Receptionist_Prof(string, string, string, string) public string editRcptnProf()

1.3.3 Admin

Admin
private string username private string oldPass private string newPass private string cfmPass
public string edit() public string register()

Admin_Report
-
public static int dispCount1(int) public static int dispCount2(int) public static int dispCount3(int) public static int dispCount4(int) public static int dispCount5(int) public static int dispCount6(int) public static int dispCount7(int) public static int dispCount8(int) public static int dispSum1(int) public static int dispSum2(int) public static int dispSum3(int) public static int dispSum4(int) public static int dispSum5(int) public static int dispSum6(int) public static int dispSum7(int) public static int dispSum8(int) public static int addIncome(int)

1.3.4 Customer

CustOnServ
private string username; private int custUserId;

private string serviceDesc; private string laptopCollectDate; private int serviceFee;
public string ServiceDesc { get => serviceDesc; set => serviceDesc = value; } public string LaptopCollectDate { get => laptopCollectDate; set => laptopCollectDate = value; } public int ServiceFee { get => serviceFee; set => serviceFee = value; }
public CustOnServ(string) public static void viewServiceDesc(object) public static void viewLaptopCollectionDate(object) public static void viewServiceFee(object)

CustEditProfile
private string username; private string currentPw; private string newPw; private string rePw;
public CustEditProfile(string, string, string, string)
public string editProfile()

CustServChg

private string username; private int custUserId; private string serviceType; private string changedServ;
public string ServiceType { get => serviceType; set => serviceType = value; } public CustServChg(string)
public static void currentReqServ(object) public string updateServChg(string, string)

1.3.5 Technician

Technician
private static string username private static string password
public Technician() public Technician(string) public Technician(string, string)
public ArrayList Get_matched_customer_username() public string view_requested_service(string) public string view_requested_service(string) public string view_collection_date(string) public string view_description(string) public string edit_total_amount(string) public void update_detalis(string, string ,string)

```
private string return_user_id(string)
private void update_new_password_username(string,
string, string)
public void edit_profile(string, string, string)
```

1.4 Object-Oriented Programming Concepts

1.4.1 Login - Teoh Will Shen TP063735

1.4.1.1 Class

User_Login

For the login function of the program, a class named **User_Login** has been created for every user role to be able to login to their respective forms. Therefore, this class needs to know about the username and the password of the user in order for the user to login.

```
3 references
internal class User_Login
{
    private string username;
    private string password;
```

For the class **User_Login,** it has a private member field **username** and **password**. username and password is private because it is only wanted in the **User_Login** class and should not be accessed by any class.

There is only one constructor for class **User_Login,** which is shown in the figure below:

```
1 reference
public User_Login(string usnm, string pwd)
{
    username = usnm;
    password = pwd;
}
```

This is a parameterized constructor which accepts two arguments with both having 'string' as the data type. It will be executed once the object for login has been created.

frmUserLogin

```
3 references
public partial class frmUserLogin : Form
{
    1 reference
    public frmUserLogin()
    {
        InitializeComponent();
    }
```

This is the class for the form **frmUserLogin**.

1.4.1.2 Methods

User_Login member methods

login

There is only one member method for the class User_Login, which has the login functions, called login. The whole code is presented in the figure below:

```
1 reference
public string login(string un)
{
    string status = null;
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["myCS"].ToString());
    con.Open();

    //SqlCommand for counting the total number of account with the username and password enetered (should be only one if have)
    SqlCommand cmd = new SqlCommand("select count (*) from Account where username = '" + username + "' and password = '" + password + "'", con);

    int count = Convert.ToInt32(cmd.ExecuteScalar().ToString());

    if (count > 0)
    {
        //SqlCommand to identify the role of the user
        SqlCommand cmd2 = new SqlCommand("select role from Account where username = '" + username + "' and password = '" + password + "'", con);
        string role = cmd2.ExecuteScalar().ToString();

        //Bringing the user to their pages according to their roles
        if (role == "Admin")
        {
            frmAdminHome a = new frmAdminHome(un);
            a.ShowDialog();
        }
        else if (role == "Receptionist")
        {
            frmRcptnPage r = new frmRcptnPage(un);
            r.ShowDialog();
        }
        else if (role == "Technician")
        {
            frmTechHome t = new frmTechHome(un);
            t.ShowDialog();
        }
        else if (role == "Customer")
        {
            frmCustOnServ r = new frmCustOnServ(un);
            r.ShowDialog();
        }
    }
    else
        status = "Incorrect username/password";
    con.Close();

    return status;
}
```

The method login will need to have one parameter which accepts the username of the user as an argument. The code started off by having a string called status to indicate the status of the login session. After that will be creating a new connection, "**con**", for the SqlCommand and passing a connection string called "**myCS**" to the database as the parameter. The connection is then opened using the **Open()** method.

The first SqlCommand, "**cmd**", is for counting the number of rows in the table "**Account**" with the username and password entered which should have only one row if it is in the table. An integer "**count**" is to assign the number of rows counted.

Therefore, if "**count**" is larger than 0, it means the username with the password entered is an existing user. If the user exists, the following SqlCommand, "**cmd2**", will obtain its role from the "**Account**" table. The role of the user obtained will be assigned to a string called "**role**". Each row will bring the user to their respective forms according to their role. "Admin" will be brought to "frmAdminHome"; "Receptionist" will be brought to "frmRcptnPage"; "Technician" will be brought to "frmTechHome"; "Customer" will be brought to "frmCustOnServ".

Else means that "**count**" is smaller than or equal to 0, it means the username with the password entered does not have an existing user. Therefore the status will be set as "Incorrect username/password" to indicate the user has entered a wrong username or password.

The code will end by having a **Close()** method called for the connection, "**con**", to close the connection and also return status.

1.4.1.3 Objects

User_Login objects

frmUserLogin

```
string stat;
User_Login obj1 = new User_Login(txtLogUsnm.Text, txtLogPwd.Text);
stat = obj1.login(txtLogUsnm.Text);
if (stat != null)
{
    MessageBox.Show(stat);
}
```

An object of class **User_Login** named "**obj1**" has been created, and creating a new instance of class **User_Login and** passing down the text from "**txtLogUsnm.Text**" and "**txtLogPwd.Text**" as an argument into the constructor.

Once "**obj1**" is created, the following code is about accessing the member method "**login**" in class **User_Login**. The string "**status**" will have the value returned by the member method, if it is null means that the user has been successfully loginned to their intended forms, if the status is not null means it is stated as "Incorrect username/password".

1.4.1.3 Events

frmUserLogin

```
1 reference
private void btnLogLogin_Click(object sender, EventArgs e)
{
    this.Hide();
    string stat;
    User_Login obj1 = new User_Login(txtLogUsnm.Text, txtLogPwd.Text);
    stat = obj1.login(txtLogUsnm.Text);
    if (stat != null)
    {
        MessageBox.Show(stat);
    }

    txtLogUsnm.ForeColor = Color.Gray;
    txtLogPwd.ForeColor = Color.Gray;
    txtLogUsnm.Text = "Enter Username";
    txtLogPwd.PasswordChar = '\0';
    txtLogPwd.Text = "Enter Password";
    this.Show();
}
```

This is the code for the click event when the user clicked the Login Button in **frmUserLogin**. Firstly, the code is using "**this.Hide()**" using the "**this**" keyword and the **Hide()** method to hide the login form and will open up the intended form with the following code by creating an object of class User_Login and calling the member method "**login**". If the login is not successful a message box will pop up, stating "Incorrect username/password", instead of the intended form. After that, the codes are for changing the **frmUserLogin** form into its default form. Lastly, "**this.Show()**" will show back **frmUserLogin** form once the intended form or message box is closed.

```
1 reference
private void btnLogExit_Click(object sender, EventArgs e)
{
    this.Close();
}
```

This is the code for the click event when the user clicked the Exit Button in **frmUserLogin**. The code "**this.Close()**" is used by having the "**this**" keyword and **Close()** method.

```
1 reference
private void txtLogUsnm_Enter(object sender, EventArgs e)
{
    if(txtLogUsnm.Text == "Enter Username")
    {
        txtLogUsnm.Text = "";

        txtLogUsnm.ForeColor = Color.Black;
    }
}
```

```
1 reference
private void txtLogPwd_Enter(object sender, EventArgs e)
{
    if (txtLogPwd.Text == "Enter Password")
    {
        txtLogPwd.PasswordChar = '•';

        txtLogPwd.Text = "";

        txtLogPwd.ForeColor = Color.Black;
    }
}
```

These are the codes for the event "Enter" when textbox "**txtLogUsnm**" and "**txtLogPwd**" is entered. This allows us to have a placeholder or so called watermarking for the textbox to show the user what to write in the textbox. For textbox "**txtLogUsnm**" the placeholder is "Enter Username", before the user enter the textbox it will be written as "Enter Username" as default. The code's logic is to have the placeholder cleared up and allow the user to type their intended text. Therefore, when entering the textbox if the text in the textbox is "Enter Username" then the text will be cleared up and change the colour of the text to black as the colour for the placeholder

is gray. The code is the same for textbox "**txtLogPwd**", just that the placeholder is set as "Enter Password" and when user enters the textbox the password character for it will set as '•' so that the password entered is not viewable. The password character, '•', is not set as default because the placeholder "Enter Password" needs to be viewable, so it only can be set once the user enters the textbox.

```
1 reference
private void txtLogUsnm_Leave(object sender, EventArgs e)
{
    if (txtLogUsnm.Text == "")
    {
        txtLogUsnm.Text = "Enter Username";

        txtLogUsnm.ForeColor = Color.Gray;
    }
}
```

```
1 reference
private void txtLogPwd_Leave(object sender, EventArgs e)
{
    if (txtLogPwd.Text == "")
    {
        txtLogPwd.PasswordChar = '\0';

        txtLogPwd.Text = "Enter Password";

        txtLogPwd.ForeColor = Color.Gray;
    }
}
```

These are the codes for the event "Leave" when the user left the textbox "**txtLogUsnm**" and "**txtLogPwd**". It is a part of code to allow the placeholders to work. These codes will bring the textboxes to their default form, displaying the placeholders, if the textbox is empty. If the textboxes are empty means the user has not typed anything or has deleted all the texts on the textboxes. So once the user deselected the textboxes, the placeholders "Enter Username" and "Enter Password" will show up in their respective textboxes and are in gray colour. For textbox "**txtLogPwd**", there is a additional line of code, " txtLogPwd.PasswordChar = '\0' ", to remove the password character, as we need to view the placeholder in text form and not in password character form.

1.4.2 Receptionist - Teoh Will Shen TP063735

1.4.2.1 Class

Receptionist_Cust

This is one of the classes for the receptionist named **Receptionist_Cust**. This class includes the functions and methods that are able to be used by the receptionist such as creating new customers, creating requested services, creating payment, and generating receipts for the customers. The class **Receptionist_Cust** needs to have details about the customer for most of its member methods to work such as customer username, customer password, service chosen, the urgency of the service, service fee, etc.

```
22 references
internal class Receptionist_Cust
{
    private string custName;
    private string custPwd;
    private int serTypeIndex;
    private string service_selected;
    private string urgency;
    private int serFee;
```

All the member fields in **Receptionist_Cust** are set as private because we will not let them be accessed or altered by other classes.

The constructor in this class is overloaded because different parameters are needed for different objects. The overloaded constructor are shown in the figure below:

```
2 references
public Receptionist_Cust(string cn)
{
    custName = cn;
}

1 reference
public Receptionist_Cust(string cn, string cp)
{
    custName = cn;
    custPwd = cp;
}

2 references
public Receptionist_Cust(int sti, string urg)
{
    serTypeIndex = sti;
    urgency = urg;
}

1 reference
public Receptionist_Cust(string cn, string ss, string urg, int sf)
{
    custName = cn;
    service_selected = ss;
    urgency = urg;
    serFee = sf;
}

1 reference
public Receptionist_Cust()
{

}
```

There are four parameterized constructor and one parameterless constructor. There are parameterized constructor which accept one string, two strings, one integer and one string, three strings and one integer.

```
22 references
internal class Receptionist_Cust
{
    private string custName;
    private string custPwd;
    private int serTypeIndex;
    private string service_selected;
    private string urgency;
    private int serFee;

    static SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["myCS"].ToString());
```

The connection to the database is done here because most of the member methods have to use SqlCommands. If the connection is done here the connection would not have to be done in all methods one by one.

Receptionist_Prof

This is another class for the receptionist called **Receptionist_Prof**. This class is for the receptionist to make some changes for their profile such as their password. This class needs to have the receptionist's username, id, and password in order to work.

```
3 references
internal class Receptionist_Prof
{
    private string rcptnName;
    private string oldPass;
    private string newPass;
    private string reNewPass;
```

All the member fields for class **Receptionist_Prof** are also private.

The figure below is the constructor for class **Receptionist_Prof**:

```
1 reference
public Receptionist_Prof(string n, string op, string np, string rnp)
{
    rcptnName = n;
    oldPass = op;
    newPass = np;
    reNewPass = rnp;
}
```

This constructor is a parameterized constructor which accepts four strings. It accepts the receptionist username, the old password entered, the new password entered, and the retype new password entered.

```
3 references
internal class Receptionist_Prof
{
    private string rcptnName;
    private string oldPass;
    private string newPass;
    private string reNewPass;

    static SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["myCS"].ToString());

    1 reference
```

The connection to the database for this class is also done here.

frmRcptnPage

```
5 references
public partial class frmRcptnPage : Form
{
    public static string name;

    1 reference
    public frmRcptnPage(string n)
    {
        InitializeComponent();
        name = n;
    }

    0 references
    public frmRcptnPage()
    {
        InitializeComponent();
    }
```

This is the class for the form **frmRcptnPage**. It has a member field “public static string name” as this name may be accessed by other classes. It has overloaded constructors, one of them has a string parameter which accepts the username of the user as an argument.

frmRcptnPymtPage

```
5 references
public partial class frmRcptnPymtPage : Form
{
    public bool new_load = true;
    public static double balance;
    public static string rec_no;
    public static string collect_date;
    public static string name;

    0 references
    public frmRcptnPymtPage()
    {
        InitializeComponent();
    }

    1 reference
    public frmRcptnPymtPage(string n)
    {
        InitializeComponent();
        name = n;
    }
```

This is the class for the **frmRcptnPymtPage.** All of the member fields are public as may be accessed by other classes. Its overloaded constructor is similar to the **frmRcptnPage** constructor.

frmRcptnOfflRec

```
5 references
public partial class frmRcptnOfflRec : Form
{
    public static ArrayList rec_info;

    0 references
    public frmRcptnOfflRec()
    {
        InitializeComponent();
    }

    1 reference
    public frmRcptnOfflRec(ArrayList ri)
    {
        InitializeComponent();
        rec_info = ri;
    }
```

This is the class of the form **frmRcptnOfflRec**. It has the array list **rec_info** as a public static member field. It also has overloaded constructors, one of them accepts array list as an argument.

frmRcptnEditPrf

```
5 references
public partial class frmRcptnEditPrf : Form
{
    public static string name;

    0 references
    public frmRcptnEditPrf()
    {
        InitializeComponent();
    }

    1 reference
    public frmRcptnEditPrf(string n)
    {
        InitializeComponent();
        name = n;
    }
```

This is the class for **frmRcptnEditPrf**. The member field and constructor is similar to the **frmRcptnPage**.

1.4.2.2 Methods

Receptionist_Cust member methods

addCust

```
1 reference
public string addCust()
{
    string status;
    con.Open();

    //count the existing customer with the username entered
    SqlCommand cmd = new SqlCommand("select count(*) from Account where username = '" + custName + "'", con);

    int count = Convert.ToInt32(cmd.ExecuteScalar().ToString());

    if (count > 0)
    {
        //one username only belongs to one customer
        status = "This Customer already exist!\nPlease create a different Customer username!!!";
    }
    else
    {
        //if the username is available then it will be put into the database
        SqlCommand cmd2 = new SqlCommand("insert into Account (username, password, role) values(@name, @password, 'Customer')", con);
        cmd2.Parameters.AddWithValue("@name", custName);
        cmd2.Parameters.AddWithValue("@password", custPwd);

        int i = cmd2.ExecuteNonQuery();
        if (i != 0)
            status = "Registration Successful.";
        else
            status = "Unable to register.";
    }
    con.Close();
    return status;
}
```

This is method **addCust,** it is used for creating a new customer account. It doesn't have any parameters. Same as before, string "**status**" is used to display whether the function is successful or not. The code will start by opening the connection with the **Open()** method. At first the code will check whether the customer username has existed or not and will determine will the customer be created or need another username. The first SqlCommand, **cmd**, is to count the number of customer usernames in the **Account** table; if there is a customer in the **Account** table with the username entered then the integer **count** should be 1, and if not it should be 0. The following if else statement is to determine if the customer username entered can be created or not. If **count** is larger than 0 means it is 1 and the customer username with its password should not be created. Else **count** is equal to or less than 0 and the customer account can be created, therefore the second SqlCommand, **cmd2**, will be used to add the customer username and

password into the **Account** table of the database. The code will end by closing the connection and returning the **status**.

addCustCbo

```
3 references
public static ArrayList addCustCbo()
{
    ArrayList cust_usnm = new ArrayList();
    con.Open();
    SqlCommand cmd = new SqlCommand("select username from Account where role = 'Customer'", con);
    SqlDataReader rd = cmd.ExecuteReader();
    while (rd.Read())
    {
        cust_usnm.Add(rd.GetString(0));
    }
    con.Close();
    return cust_usnm;
}
```

The **addCusCbo** method is for adding all the existing customer username into the combo box in the forms. Firstly, an array list is created to put all the customer username from the **Account** table. The connection will then open with **Open()**, and the SqlCommand, **cmd**, is used to get all the customer username from the table. The SqlDataReader, **rd**, is used to read the data from the SqlCommand and with the while loop it will store every customer username into the array list. The connection will be closed with **Close()** and the array list, **cust_usnm**, will be returned.

showPrice

```
2 references
public int showPrice()
{
    int price = 0;

    if (urgency == "Normal")
    {
        if (serTypeIndex == 0)
        {
            price = 50;
        }
        else if (serTypeIndex == 1)
        {
            price = 60;
        }
        else if (serTypeIndex == 2)
        {
            price = 380;
        }
        else if (serTypeIndex == 3)
        {
            price = 160;
        }
        else if (serTypeIndex == 4)
        {
            price = 180;
        }
        else if (serTypeIndex == 5)
        {
            price = 100;
        }
        else if (serTypeIndex == 6)
        {
            price = 80;
        }
        else if (serTypeIndex == 7)
        {
            price = 70;
        }
    }
    else if (urgency == "Urgent")
    {
        if (serTypeIndex == 0)
        {
            price = 80;
        }
        else if (serTypeIndex == 1)
        {
            price = 90;
        }
        else if (serTypeIndex == 2)
        {
            price = 430;
        }
        else if (serTypeIndex == 3)
        {
            price = 200;
        }
        else if (serTypeIndex == 4)
        {
            price = 210;
        }
        else if (serTypeIndex == 5)
        {
            price = 150;
        }
        else if (serTypeIndex == 6)
        {
            price = 130;
        }
        else if (serTypeIndex == 7)
        {
            price = 100;
        }
    }

    return price;
}
```

This **showPrice** method is for displaying the price of the service and urgency level selected by the customer. Initially, an integer called **price** is created and set to 0; **price** is used to store the price of service and urgency level selected by the customer. If else statement is used to determine the service and urgency level and assign the proper price to **price**. First the if else statement is for determining the urgency level, which is “Normal” or “Urgent”, because each has a different price. Both **urgency** and **serTypeIndex** is from the member field. **serTypeIndex** is for determining which type of service is chosen and make it into an index number instead of the

whole service text to better manage the if else statement. The proper price will be assigned to **price** and **price** will be returned.

addService

```
1 reference
public string addService()
{
    string status;
    DateTime reqDate = DateTime.Now;
    con.Open();

    SqlCommand cmd = new SqlCommand("select user_id from Account where username = '" + custName +"'", con);
    int user_id = int.Parse(cmd.ExecuteScalar().ToString());

    SqlCommand check = new SqlCommand("select count(*) from Service where user_id = '" + user_id + "' and paid_amount is null", con);

    int count = Convert.ToInt32(check.ExecuteScalar().ToString());

    if (count > 0)
    {
        status = "Please wait until your current service is ended";
    }
    else
    {
        SqlCommand cmd2 = new SqlCommand("insert into Service (user_id, service_type, service_urgency, request_date, service_fee) values (@id, @st, @urg, @reqDate, @sf)", con);
        cmd2.Parameters.AddWithValue("@id", user_id);
        cmd2.Parameters.AddWithValue("@st", service_selected);
        cmd2.Parameters.AddWithValue("@urg", urgency);
        cmd2.Parameters.AddWithValue("@reqDate", reqDate);
        cmd2.Parameters.AddWithValue("sf", serFee);

        cmd.ExecuteNonQuery();
        int i = cmd2.ExecuteNonQuery();
        if (i != 0)
            status = "Service is successfully created!";
        else
            status = "Unable to create service.";
    }
    con.Close();
    return status;
}
```

This **addService** method is for creating the service requested by the customer. As usual string **status** is used for knowing whether the service creation is successful or not. “DateTime.Now” is used to get the system date from the computer and put it in **reqDate**, this date is the date which the customer makes the service request (Microsoft, n.d.). Firstly the SqlCommand, **cmd**, is used to get the user_id of the customer selected from the **Account** table. The user_id is then used in SqlCommand **check** to see if the customer has already had an ongoing service or not, this can be fulfill by checking whether the **paid_amount** in the **Service** is null or not, if it is null means payment has not been made and the service is still ongoing. The integer **count** is used to store the number of rows counted from the SqlCommand **check**. If count is larger than 0 means there is still a service with **paid_amount** is null and the service is still ongoing, therefore **status** will be put as “Please wait until your current service is ended”. Else there is no ongoing service and the customer can request another service. SqlCommand, **cmd2**, is used to insert the service information into the **Service** table. Once ended the connection will close and return **status**.

showPayment

```
1 reference
public ArrayList showPayment()
{
    ArrayList service_info = new ArrayList();

    con.Open();
    SqlCommand cmd = new SqlCommand("select user_id from Account where username = '" + custName + "'", con);
    int user_id = int.Parse(cmd.ExecuteScalar().ToString());

    SqlCommand cmd2 = new SqlCommand("select service_type, service_urgency, service_fee, receipt_no, " +
        "convert(varchar, laptop_collection_date, 23) as laptop_collection_date from Service where user_id = '" + user_id + "' and paid_amount is null", con);
    SqlDataReader rd = cmd2.ExecuteReader();
    while (rd.Read())
    {
        service_info.Add(rd["service_type"].ToString());
        service_info.Add(rd["service_urgency"].ToString());
        service_info.Add(rd["service_fee"].ToString());
        service_info.Add(rd["receipt_no"].ToString());
        service_info.Add(rd["laptop_collection_date"].ToString());
    }

    con.Close();
    return service_info;
}
```

The **showPayment** method is for displaying the payment information into the payment form. An array list called **ser_info** is created to store the service information needed in the payment page. There is the same code as before to get the user_id. The SqlCommand, **cmd2**, is used to select the information needed from the **Service** table. In **cmd2** "convert(varchar, laptop_collection_date, 23)" is used to change the date format into "yyyy-MM-dd" (Edgewood Solutions, 2021). **cmd2** needs to make sure that the selected row the **paid_amount** is null, the reason is stated before. SqlDataReader **rd** is used to read **cmd2** and using a while loop to store the information needed into the array list **service_info**. The connection is close and the array list **service_info is** returned.

addPaidAmount

```
1 reference
public string addPaidAmount(string payFee)
{
    string status;
    con.Open();
    SqlCommand cmd = new SqlCommand("select user_id from Account where username = '" + custName + "'", con);
    int user_id = int.Parse(cmd.ExecuteScalar().ToString());

    SqlCommand cmd2 = new SqlCommand("update Service set paid_amount = '" + payFee + "' where user_id = '" + user_id + "' and paid_amount is null", con);

    int i = cmd2.ExecuteNonQuery();
    if (i != 0)
        status = "Payment Successfully.";
    else
        status = "Unable to make payment.";
    con.Close();

    return status;
}
```

addPaidAmount method is used to add the paid amount entered by the user and save it into the **Service** table. This method is parameterized to accept paid amount entered in a string data type as an argument. The same code is used for getting the customer user_id. SqlCommand **cmd2** is used for updating the **paid_amount** in **Service** table from null to the paid amount entered, **payFee**.

calBal

```
1 reference
public double calBal(int sf, double pa)
{
    int ser_fee = sf;
    double paid_amount = pa;

    double balance = paid_amount - ser_fee;

    return balance;
}
```

calBal is a method used to calculate the balance by subtracting the service fee from the paid amount.

Receptionist_Prof member methods

editRcptnProf

```
1 reference
public string editRcptnProf()
{
    string status;
    string password = null;
    con.Open();

    SqlCommand cmd = new SqlCommand("select password from Account where username = '" + rcptnName + "'", con);
    SqlDataReader rd = cmd.ExecuteReader();
    while (rd.Read())
    {
        password = rd.GetString(0);
    }
    rd.Close();

    if (password != oldPass)
    {
        status = "You've entered the wrong old password!";
    }
    else
    {
        if (newPass != reNewPass)
        {
            status = "The new password entered in both textbox does not match.";
        }
        else
        {
            SqlCommand cmd2 = new SqlCommand("update Account set password = '" + reNewPass + "' where username = '" + rcptnName + "'", con);

            int i = cmd2.ExecuteNonQuery();
            if (i != 0)
                status = "Password updated Successfully.";
            else
                status = "Unable to update password.";
        }
    }

    con.Close();
    return status;
}
```

The **editRcptnProf** method is for the receptionist to edit their own profile. This method validates the old password entered and allows the new password update if it is valid. Update new password would be denied if the old password entered is incorrect. SqlCommand **cmd** is for getting the password set before from the **Account** table. A string **password** is created to store the password obtained from the **Account** table. In the if else statement, if **password** is not equal to the old password entered, the **status** will have a string stored stating that the user entered the wrong old password. Else they are equal and the new password and retype new password text needs to be checked if they match, if they are not match then the **status** will notify that error, ele SqlCommand **cmd2** will update the password with the retype new password text from the textbox of the form. The **status** will be returned showing that the process is successful or not.

1.4.2.3 Objects

Receptionist_Cust objects

frmRcptnPage

```
Receptionist_Cust cust1 = new Receptionist_Cust(txtRcptnRegCustUsnm.Text, txtRcptnRegCustPwd.Text);
MessageBox.Show(cust1.addCust());
```

"**cust1**" is the object created for creating a new customer, and creating a new instance of class **Receptionist_Cust** and passing down the text from "**txtRcptnRegCustUsnm.Text**" and "**txtRcptnRegCustPwd.Text**" as an argument into the constructor. **cust1** is then used to call the **addCust** method inside a message box to display the **status** returned.

```
Receptionist_Cust ser_price1 = new Receptionist_Cust(index, urgency);

price = ser_price1.showPrice();
```

ser_price1 is an object created for the service chosen in a "Normal" urgency level. **Ser_price1** is used to call the **showPrice** method so that the price returned will be stored inside the **price** string.

```
Receptionist_Cust ser_price2 = new Receptionist_Cust(index, urgency);

price = ser_price2.showPrice();
```

ser_price2 is an object created with the same reason as **service_price1** just that the urgency level is "Urgent".

```
Receptionist_Cust ser = new Receptionist_Cust(cboRcptnCustUsnm.Text, cboRcptnSerType.Text, urgency, serFee);

MessageBox.Show(ser.addService());
```

ser is an object used to have the service request created. The customer username, service chosen, service urgency, and service fee will be passed down into the **Receptionist_Cust**. **ser** is then

used to call the **addService** method to create the service and the status returned will be displayed in a message box.

frmRcptnPymtPage

```
ArrayList pymt_info = new ArrayList();

Receptionist_Cust pymt = new Receptionist_Cust(cboRcptnPymtCustUsnm.SelectedItem.ToString());

pymt_info = pymt.showPayment();
```

pymt is the object for showing the payment details. The selected customer username is passed down into the class. **pymt** is then used to call the **showPayment** method and **pymt_info** is an array list created to store the array list from the **showPayment** method**.**

```
Receptionist_Cust bal = new Receptionist_Cust();
balance = bal.calBal(ser_fee, paid_amount);
```

bal is the created object for calling the **calBal** method to calculate the balance. The **ser_fee** and the **paid_amount** is received by the method as a n argument.

```
Receptionist_Cust cust_pymt = new Receptionist_Cust(cboRcptnPymtCustUsnm.SelectedItem.ToString());
MessageBox.Show(cust_pymt.addPaidAmount(txtRcptnPymtRcvd.Text));
```

cust_pymt is an object for passing down the customer username selected and calls the **addPaidAmount** method which receives the received amount from the **txtRcptnPymtRcvd** textbox as an argument to store it into the database.

Receptionist_Prof objects

frmRcptnEditPrf

```
Receptionist_Prof rcptnProf = new Receptionist_Prof(name, txtRcptnEditOldPwd.Text, txtRcptnEditNewPwd.Text, txtRcptnEditReNewPwd.Text);
MessageBox.Show(rcptnProf.editRcptnProf());
```

The **rcptnProf** is created to pass down the receptionist username, old password, new password, and retype password into the constructor of **Receptionist_Prof** and calls the **editRcptnProf** method. A message box will display the **status** returned by the **editRcptnProf** method.

1.4.2.4 Events

frmRcptnPage

```
1 reference
private void frmRcptnPage_Load(object sender, EventArgs e)
{
    lblRcptnDetsUsnmSh.Text = name;

    cboRcptnCustUsnm.DataSource = Receptionist_Cust.addCustCbo();
    cboRcptnCustUsnm.Text = "Customer Username";
}
```

This is the code when the form is loaded which sets the form into its default way. The data source of the combo box **cboRcptnCustUsnm** will be obtained by calling the **addCustCbo** method from the Receptionist_Cust class. An object is not needed because the method is static.

```
1 reference
private void btnRcptnRegCust_Click(object sender, EventArgs e)
{
    Receptionist_Cust cust1 = new Receptionist_Cust(txtRcptnRegCustUsnm.Text, txtRcptnRegCustPwd.Text);
    MessageBox.Show(cust1.addCust());

    txtRcptnRegCustUsnm.ForeColor = Color.Gray;
    txtRcptnRegCustPwd.ForeColor = Color.Gray;
    txtRcptnRegCustUsnm.Text = "Username";
    txtRcptnRegCustPwd.PasswordChar = '\0';
    txtRcptnRegCustPwd.Text = "Password";
    chkRcptnShPwd.Checked = false;

    cboRcptnCustUsnm.DataSource = Receptionist_Cust.addCustCbo();
    cboRcptnCustUsnm.Text = "Customer Username";
}
```

This is the code once the button **btnRcptnRegCust** is clicked. The object **cust1** is created here and the code below is to set it back to its default form once a customer has been registered.

```
1 reference
private void txtRcptnRegCustUsnm_Enter(object sender, EventArgs e)
{
    if (txtRcptnRegCustUsnm.Text == "Username")
    {
        txtRcptnRegCustUsnm.Text = "";

        txtRcptnRegCustUsnm.ForeColor = Color.Black;
    }
}

1 reference
private void txtRcptnRegCustUsnm_Leave(object sender, EventArgs e)
{
    if (txtRcptnRegCustUsnm.Text == "")
    {
        txtRcptnRegCustUsnm.Text = "Username";

        txtRcptnRegCustUsnm.ForeColor = Color.Gray;
    }
}
```

```
1 reference
private void txtRcptnRegCustPwd_Enter(object sender, EventArgs e)
{
    if (txtRcptnRegCustPwd.Text == "Password")
    {
        txtRcptnRegCustPwd.Text = "";

        txtRcptnRegCustPwd.ForeColor = Color.Black;

        if (chkRcptnShPwd.Checked == true)
        {
            txtRcptnRegCustPwd.PasswordChar = '\0';
        }
        else
        {
            txtRcptnRegCustPwd.PasswordChar = '•';
        }
    }
}

1 reference
private void txtRcptnRegCustPwd_Leave(object sender, EventArgs e)
{
    if (txtRcptnRegCustPwd.Text == "")
    {
        txtRcptnRegCustPwd.PasswordChar = '\0';

        txtRcptnRegCustPwd.Text = "Password";

        txtRcptnRegCustPwd.ForeColor = Color.Gray;
    }
}
```

This is the code for making a placeholder just like at the login form.

```
1 reference
private void radRcptnNormSer_CheckedChanged(object sender, EventArgs e)
{
    string urgency = radRcptnNormSer.Text;
    int price;
    int index = cboRcptnSerType.SelectedIndex;

    if (cboRcptnSerType.Text != "Service Type")
    {
        Receptionist_Cust ser_price1 = new Receptionist_Cust(index, urgency);

        price = ser_price1.showPrice();

        lblRcptnSerFeeSh.Text = "RM " + price.ToString();
        }
    }

1 reference
private void radRcptnUrgSer_CheckedChanged(object sender, EventArgs e)
{
    string urgency = radRcptnUrgSer.Text;
    int price;
    int index = cboRcptnSerType.SelectedIndex;

    if (cboRcptnSerType.Text != "Service Type")
    {
        Receptionist_Cust ser_price2 = new Receptionist_Cust(index, urgency);

        price = ser_price2.showPrice();

        lblRcptnSerFeeSh.Text = "RM " + price.ToString();
    }
}
```

This is the code when one of the radio buttons is checked, either **radRcptnNormSer** or **radRcptnUrgSer**. They both have an object which is mentioned in the object part of the documentation for calling the **showPrice** method. Once the price is obtained from the method it is displayed in **lblRcptnSerFeeSh** with "RM" as the currency.

```
1 reference
private void cboRcptnSerType_SelectedIndexChanged(object sender, EventArgs e)
{
    radRcptnNormSer.Checked = false;
    radRcptnUrgSer.Checked = false;
    lblRcptnSerFeeSh.Text = String.Empty;
}

1 reference
private void cboRcptnCustUsnm_SelectedIndexChanged(object sender, EventArgs e)
{
    cboRcptnSerType.Text = "Service Type";
    radRcptnNormSer.Checked = false;
    radRcptnUrgSer.Checked = false;
    lblRcptnSerFeeSh.Text = String.Empty;
}
```

This is the code for what will happen if the user selects another customer or service type from their respective combo box. Once the combo box is selected and has been changed, all of the other things below it will be set to their default ways to prevent errors.

```
1 reference
private void btnRcptnCreCust_Click(object sender, EventArgs e)
{
    string urgency = null;
    int serFee;

    if (cboRcptnCustUsnm.Text == "Customer Username" || cboRcptnSerType.Text == "Service Type" || (radRcptnNormSer.Checked == false && radRcptnUrgSer.Checked == false)
        || cboRcptnCustUsnm.SelectedIndex < 0 || cboRcptnSerType.SelectedIndex < 0)
    {
        cboRcptnCustUsnm.Text = "Customer Username";
        cboRcptnSerType.Text = "Service Type";
        radRcptnNormSer.Checked = false;
        radRcptnUrgSer.Checked = false;
        lblRcptnSerFeeSh.Text = String.Empty;
        MessageBox.Show("Please fill in all required details to request a service");
    }
    else
    {
        string[] price = lblRcptnSerFeeSh.Text.Split(" ");
        serFee = int.Parse(price[1]);

        if (radRcptnNormSer.Checked == true)
        {
            urgency = radRcptnNormSer.Text;
        }
        else if (radRcptnUrgSer.Checked == true)
        {
            urgency = radRcptnUrgSer.Text;
        }

        Receptionist_Cust ser = new Receptionist_Cust(cboRcptnCustUsnm.Text, cboRcptnSerType.Text, urgency, serFee);

        MessageBox.Show(ser.addService());

        cboRcptnCustUsnm.Text = "Customer Username";
        cboRcptnSerType.Text = "Service Type";
        radRcptnNormSer.Checked = false;
        radRcptnUrgSer.Checked = false;
        lblRcptnSerFeeSh.Text = String.Empty;
    }
}
```

This is the code once the button Create, **btnRcptnCreCust**, is clicked. A string **urgency** and int **serFee** is declared for later to store the urgency level and the service fee. The if else statement is used to ensure that the user enters all the needed information and the correctness of the information to prevent errors. If the error occurs a message box will pop up, else the code will obtain the information from textboxes, radio buttons, labels, combobox, and pass it down to the object **ser**. A message box will pop up to show the status and afterwards everything will be set back to default.

```
string[] price = lblRcptnSerFeeSh.Text.Split(" ");
serFee = int.Parse(price[1]);
```

This piece of code is used to separate the currency from the service fee, so that only the numbers are obtained and stored into **serFee** as an integer.

```
1 reference
private void btnRcptnClear_Click(object sender, EventArgs e)
{
    cboRcptnCustUsnm.Text = "Customer Username";
    cboRcptnSerType.Text = "Service Type";
    radRcptnNormSer.Checked = false;
    radRcptnUrgSer.Checked = false;
    lblRcptnSerFeeSh.Text = String.Empty;
}
```

This code is for once the button Clear, **btnRcptnClear**, is clicked then everything will be cleared and set back to default.

```
1 reference
private void btnRcptnLogout_Click(object sender, EventArgs e)
{
    this.Close();
}
```

This button logout is for closing the form.

```
1 reference
private void btnRcptnPymtPg_Click(object sender, EventArgs e)
{
    this.Hide();

    frmRcptnPymtPage p = new frmRcptnPymtPage(name);
    p.ShowDialog();

    this.Show();
}
```

```
1 reference
private void btnRcptnEditPrf_Click(object sender, EventArgs e)
{
    this.Hide();

    frmRcptnEditPrf ep = new frmRcptnEditPrf(name);
    ep.ShowDialog();

    this.Show();
}
```

These two buttons, once clicked, will bring the user to their intended page which is either the **frmRcptnPymtPage** or **frmRcptnEditPrf**.

frmRcptnPymtPage

```
1 reference
private void frmRcptnPymtPage_Load(object sender, EventArgs e)
{
    cboRcptnPymtCustUsnm.DataSource = Receptionist_Cust.addCustCbo();
    btnRcptnPymtGenRcpt.Enabled = false;
}
```

When the form is loaded the combo box, **cboRcptnPymtCustUsnm**, will get its source just like at **frmRcptnPage**, using the **addCustCbo** method from the **Receptionist_Cust** class. The generate receipt button, **btnRcptnPymtGenRcpt**, will be disabled.

```
1 reference
private void cboRcptnPymtCustUsnm_SelectedIndexChanged(object sender, EventArgs e)
{
    if (new_load == true)
    {
        cboRcptnPymtCustUsnm.Text = "Customer Username";
        new_load = false;
    }
    else
    {
        ArrayList pymt_info = new ArrayList();

        Receptionist_Cust pymt = new Receptionist_Cust(cboRcptnPymtCustUsnm.SelectedItem.ToString());

        pymt_info = pymt.showPayment();

        if (pymt_info.Count == 5)
        {
            lblRcptnPymtSerSh.Text = pymt_info[0].ToString() + " (" + pymt_info[1].ToString() + ")";
            lblRcptnPymtSerFeeSh.Text = pymt_info[2].ToString();
            rec_no = pymt_info[3].ToString();
            collect_date = pymt_info[4].ToString();

            if (collect_date == "")
            {
                lblRcptnPymtSerSh.Text = String.Empty;
                lblRcptnPymtSerFeeSh.Text = String.Empty;
                MessageBox.Show("Please pay on laptop collection date!");
            }
        }
        else
        {
            lblRcptnPymtSerSh.Text = String.Empty;
            lblRcptnPymtSerFeeSh.Text = String.Empty;
            MessageBox.Show("This customer has not made any service request yet!");
        }
    }
}
```

The **new_load** is for changing the combo box to its default form when the form is initially loaded, this can prevent the combo box from selecting the first item itself. If the **new_load** is done then it will be set as false. As mentioned in the objects part, the array list will be obtained from the **showPayment** method and stored in array list **pymt_info**. The following if else statement is to check whether the customer has an ongoing service, if the **pymt_info** has 5 elements means the payment info is enough to make a payment and generate receipt, but the collection date needs to be checked also, if the collection date is empty means the service has not ended yet and the payment cannot be made and all the labels will be emptied. Else **pymt_info** if

it does not have 5 elements means it is empty indicating that the customer has no ongoing service.

```
1 reference
private void btnRcptnPymtClear_Click(object sender, EventArgs e)
{
    cboRcptnPymtCustUsnm.Text = "Customer Username";
    lblRcptnPymtSerSh.Text = String.Empty;
    lblRcptnPymtSerFeeSh.Text= String.Empty;
    txtRcptnPymtRcvd.Text = String.Empty;
}
```

This is the clear button, everything will be cleared and set to its default way.

```
1 reference
private void btnRcptnPymtAcc_Click(object sender, EventArgs e)
{
    int ser_fee;
    double paid_amount;

    if (cboRcptnPymtCustUsnm.Text == "Customer Username" || cboRcptnPymtCustUsnm.Text == String.Empty || cboRcptnPymtCustUsnm.SelectedIndex < 0 || lblRcptnPymtSerSh.Text == "")
    {
        MessageBox.Show("Please select a customer!");
        cboRcptnPymtCustUsnm.Text = "Customer Username";
        lblRcptnPymtSerSh.Text = String.Empty;
        lblRcptnPymtSerFeeSh.Text = String.Empty;
        txtRcptnPymtRcvd.Text = String.Empty;
    }
    else if (txtRcptnPymtRcvd.Text == "")
    {
        MessageBox.Show("Please enter paid amount!");
    }
    else if (double.TryParse(txtRcptnPymtRcvd.Text, out paid_amount))
    {
        ser_fee = int.Parse(lblRcptnPymtSerFeeSh.Text);
        paid_amount = double.Parse(txtRcptnPymtRcvd.Text);

        Receptionist_Cust bal = new Receptionist_Cust();
        balance = bal.calBal(ser_fee, paid_amount);

        if (balance < 0)
        {
            MessageBox.Show("Please pay at least the amount of the service fee!");
        }
        else
        {
            Receptionist_Cust cust_pymt = new Receptionist_Cust(cboRcptnPymtCustUsnm.SelectedItem.ToString());
            MessageBox.Show(cust_pymt.addPaidAmount(txtRcptnPymtRcvd.Text));

            btnRcptnPymtClear.Enabled = false;
            btnRcptnPymtBack.Enabled = false;
            cboRcptnPymtCustUsnm.Enabled = false;
            txtRcptnPymtRcvd.Enabled = false;
            btnRcptnPymtAcc.Enabled = false;
            btnRcptnPymtGenRcpt.Enabled = true;
        }
    }
    else
    {
        MessageBox.Show("Please enter correct format for paid amount!");
    }
}
```

This is the code once the accept payment button,**btnRcptnPymtAcc** , is clicked. The integer ser_fee is used to store the text from **lblRcptnPymtSerFeeSh**, the double **paid_amount** is used to store the amount of money entered in **txtRcptnPymtRcvd**. The if else statement used here is

to ensure everything selected and typed is correct before the process of accepting the payment using the **addPaidAmount** method.

```
else if (double.TryParse(txtRcptnPymtRcvd.Text, out paid_amount))
{
    ser_fee = int.Parse(lblRcptnPymtSerFeeSh.Text);
    paid_amount = double.Parse(txtRcptnPymtRcvd.Text);
```

```
else
{
    MessageBox.Show("Please enter correct format for paid amount!");
}
```

This is used to check if the amount entered in the textbox is a number or not.

```
Receptionist_Cust bal = new Receptionist_Cust();
balance = bal.calBal(ser_fee, paid_amount);

if (balance < 0)
{
    MessageBox.Show("Please pay at least the amount of the service fee!");
}
```

This is used to check if the paid amount entered is sufficient for the service fee or not.

```
else
{
    Receptionist_Cust cust_pymt = new Receptionist_Cust(cboRcptnPymtCustUsnm.SelectedItem.ToString());
    MessageBox.Show(cust_pymt.addPaidAmount(txtRcptnPymtRcvd.Text));

    btnRcptnPymtClear.Enabled = false;
    btnRcptnPymtBack.Enabled = false;
    cboRcptnPymtCustUsnm.Enabled = false;
    txtRcptnPymtRcvd.Enabled = false;
    btnRcptnPymtAcc.Enabled = false;
    btnRcptnPymtGenRcpt.Enabled = true;
```

If everything is fine, then the payment is accepted, and a message box will pop up showing the payment is successful or not. Once the payment is successfully done, all the buttons, text boxes, combo boxes, are disabled, only the generate receipt button is enabled to generate the receipt. This is because once the payment is done the receptionist is not allowed to change anything to ensure the receipt is correct.

```
1 reference
private void btnRcptnPymtGenRcpt_Click(object sender, EventArgs e)
{
    this.Hide();
    ArrayList rec_info = new ArrayList();
    rec_info.Add(cboRcptnPymtCustUsnm.SelectedItem.ToString());
    rec_info.Add(txtRcptnPymtRcvd.Text);
    rec_info.Add(lblRcptnPymtSerSh.Text);
    rec_info.Add(lblRcptnPymtSerFeeSh.Text);
    rec_info.Add(balance.ToString());
    rec_info.Add(rec_no);
    rec_info.Add(collect_date);
    rec_info.Add(name);

    frmRcptnOfflRec or = new frmRcptnOfflRec(rec_info);
    or.ShowDialog();
    btnRcptnPymtClear.Enabled = true;
    btnRcptnPymtBack.Enabled = true;
    cboRcptnPymtCustUsnm.Enabled = true;
    txtRcptnPymtRcvd.Enabled = true;
    btnRcptnPymtAcc.Enabled = true;
    btnRcptnPymtGenRcpt.Enabled = false;
    cboRcptnPymtCustUsnm.Text = "Customer Username";
    lblRcptnPymtSerSh.Text = String.Empty;
    lblRcptnPymtSerFeeSh.Text = String.Empty;
    txtRcptnPymtRcvd.Text = String.Empty;
    this.Show();
}
```

If the generate receipt button is clicked, the payment page is hidden, and an array list **rec_info** is created to pass it down to the receipt form, **frmRcptnOfflRec**, then the **frmRcptnOfflRec** is shown. Once the receipt page is closed, all buttons, text boxes, combo boxes, are enabled back, only the generate receipt button is disabled, and the **frmRcptnPymtPage** is shown back.

```
1 reference
private void btnRcptnPymtBack_Click(object sender, EventArgs e)
{
    this.Close();
}
```

This is the back button used to close the **frmRcptnPymtPage**.

frmRcptnEditPrf

```
1 reference
private void frmRcptnOfflRec_Load(object sender, EventArgs e)
{
    lblRcptnRecCustUsnm.Text = rec_info[0].ToString();
    lblRcptnRecAmtRcvd.Text = "RM " + rec_info[1].ToString();
    lblRcptnRecSerType.Text = rec_info[2].ToString();
    lblRcptnRecSerFee.Text = "RM " + rec_info[3].ToString();
    lblRcptnRecBalDue.Text = "RM " + rec_info[4].ToString();
    lblRcptnRecNo.Text = rec_info[5].ToString();
    lblRcptnRecDate.Text = rec_info[6].ToString();
    lblRcptnRecRcptnUsnm.Text = rec_info[7].ToString();
}
```

Once the **frmRcptnEditPrf** is loaded all the elements in the array list rec_info, passed down from **frmRcptnPymtPage**, are displayed in their respective labels.

```
1 reference
private void lblRcptnRecBack_Click(object sender, EventArgs e)
{
    this.Close();
}
```

This is the back button used to close the **frmRcptnEditPrf**.

frmRcptnEditProf

```
1 reference
private void frmRcptnEditPrf_Load(object sender, EventArgs e)
{
    lblRcptnEditUsnmSh.Text = name;
    btnRcptnEditPrfSave.Enabled = false;
}
```

Once the form is loaded the current receptionist username is displayed in the label **lblRcptnEditUsnmSh** and the button save button, **btnRcptnEditPrfSave**, is disabled.

```
1 reference
private void chkRcptnEditConf_CheckedChanged(object sender, EventArgs e)
{
    if (chkRcptnEditConf.Checked == true)
    {
        if (txtRcptnEditOldPwd.Text == "Old Password" || txtRcptnEditOldPwd.Text == "" || txtRcptnEditNewPwd.Text == "New Password" ||
            txtRcptnEditNewPwd.Text == "" || txtRcptnEditReNewPwd.Text == "Re-type New Password" || txtRcptnEditReNewPwd.Text == "")
        {
            chkRcptnEditConf.Checked = false;
            MessageBox.Show("Please fill in all details!");
        }
        else
        {
            txtRcptnEditOldPwd.Enabled = false;
            txtRcptnEditNewPwd.Enabled = false;
            txtRcptnEditReNewPwd.Enabled = false;
            btnRcptnEditPrfSave.Enabled = true;
        }
    }
    else if (chkRcptnEditConf.Checked == false)
    {
        txtRcptnEditOldPwd.Enabled = true;
        txtRcptnEditNewPwd.Enabled = true;
        txtRcptnEditReNewPwd.Enabled = true;
        btnRcptnEditPrfSave.Enabled = false;
    }
}
```

If the checkbox **chkRcptnEditConf** is checked there will be the if else statement to check whether all information needed is entered or not. If all is entered then all text boxes are disabled as this check box is for confirming changes and should not alter the text boxes once confirmed. If the checkbox is not checked then all text boxes are allowed to be typed.

```
1 reference
private void btnRcptnEditPrfSave_Click(object sender, EventArgs e)
{
    Receptionist_Prof rcptnProf = new Receptionist_Prof(name, txtRcptnEditOldPwd.Text, txtRcptnEditNewPwd.Text, txtRcptnEditReNewPwd.Text);
    MessageBox.Show(rcptnProf.editRcptnProf());

    btnRcptnEditPrfSave.Enabled = false;
    txtRcptnEditOldPwd.Enabled = true;
    txtRcptnEditNewPwd.Enabled = true;
    txtRcptnEditReNewPwd.Enabled = true;
    chkRcptnEditConf.Checked = false;
    txtRcptnEditOldPwd.PasswordChar = '\0';
    txtRcptnEditNewPwd.PasswordChar = '\0';
    txtRcptnEditReNewPwd.PasswordChar = '\0';
    txtRcptnEditOldPwd.ForeColor = Color.Gray;
    txtRcptnEditNewPwd.ForeColor = Color.Gray;
    txtRcptnEditReNewPwd.ForeColor = Color.Gray;
    txtRcptnEditOldPwd.Text = "Old Password";
    txtRcptnEditNewPwd.Text = "New Password";
    txtRcptnEditReNewPwd.Text = "Re-type New Password";
}
```

The Save button, **btnRcptnEditPrfSave**, is about updating the password into the database. The **rcptnProf** is created here. All the code below is to set the form into its default way.

```
1 reference
private void txtRcptnEditOldPwd_Enter(object sender, EventArgs e)
{
    if (txtRcptnEditOldPwd.Text == "Old Password")
    {
        txtRcptnEditOldPwd.PasswordChar = '•';
        txtRcptnEditOldPwd.Text = "";

        txtRcptnEditOldPwd.ForeColor = Color.Black;
    }
}

1 reference
private void txtRcptnEditOldPwd_Leave(object sender, EventArgs e)
{
    if (txtRcptnEditOldPwd.Text == "")
    {
        txtRcptnEditOldPwd.PasswordChar = '\0';
        txtRcptnEditOldPwd.Text = "Old Password";

        txtRcptnEditOldPwd.ForeColor = Color.Gray;
    }
}
```

```
1 reference
private void txtRcptnEditNewPwd_Enter(object sender, EventArgs e)
{
    if (txtRcptnEditNewPwd.Text == "New Password")
    {
        txtRcptnEditNewPwd.PasswordChar = '•';
        txtRcptnEditNewPwd.Text = "";

        txtRcptnEditNewPwd.ForeColor = Color.Black;
    }
}

1 reference
private void txtRcptnEditNewPwd_Leave(object sender, EventArgs e)
{
    if (txtRcptnEditNewPwd.Text == "")
    {
        txtRcptnEditNewPwd.PasswordChar = '\0';
        txtRcptnEditNewPwd.Text = "New Password";

        txtRcptnEditNewPwd.ForeColor = Color.Gray;
    }
}
```

```
1 reference
private void txtRcptnEditReNewPwd_Enter(object sender, EventArgs e)
{
    if (txtRcptnEditReNewPwd.Text == "Re-type New Password")
    {
        txtRcptnEditReNewPwd.PasswordChar = '•';
        txtRcptnEditReNewPwd.Text = "";

        txtRcptnEditReNewPwd.ForeColor = Color.Black;
    }
}

1 reference
private void txtRcptnEditReNewPwd_Leave(object sender, EventArgs e)
{
    if (txtRcptnEditReNewPwd.Text == "")
    {
        txtRcptnEditReNewPwd.PasswordChar = '\0';
        txtRcptnEditReNewPwd.Text = "Re-type New Password";

        txtRcptnEditReNewPwd.ForeColor = Color.Gray;
    }
}
```

These are the same code logic for the placeholder to work in the text boxes.

1.4.3 Admin

1.4.3.1 Class

Admin

This is a class for the administrators named **Admin**. This class includes the basic methods for the admin to register a new account, or edit their own account. This class requires details including username, registered password, role, old password, new password and password confirmation.

```
internal class Admin
{
    private string username;
    private string password;
    private string role;
    private string oldPass;
    private string newPass;
    private string cfmPass;
```

Member fields in **Admin** are set as private to prevent cross access by other classes.

```
static SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["myCS"].ToString());

0 references
public string OldPass { get => oldPass; set => oldPass = value; }
0 references
public string NewPass { get => newPass; set => newPass = value; }
0 references
public string ReNewPass { get => cfmPass; set => cfmPass = value; }

1 reference
public Admin(string usnm, string Pwd, string r)
{
    username = usnm;
    password = Pwd;
    role = r;
}

1 reference
public Admin(string usnm, string oPwd, string nPwd, string cPwd)
{
    username = usnm;
    oldPass = oPwd;
    newPass = nPwd;
    cfmPass = cPwd;
}
```

The Sql Connection is first configured and set up to allow connections to the database to be opened and closed when needed. Furthermore, below that are two overloaded constructors with different parameters for different uses.

Admin_Report

This is the second class under the Admin usage. Unlike the first Admin.cs class, this class does not have any predeclared variables as everything inside this class are methods that does not require any specific information to run. The methods in this class are used to generate, calculate and count the number of services as well as the total income depending on the month selected.

```
internal class Admin_Report
{
    static SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["myCS"].ToString());
```

The only two non-method lines of code used to create a Sql Connection to the database for the methods to operate correctly.

1.4.3.2 Methods

Admin.cs admin method

edit()

```
public string edit()
{
    string status;
    string password = null;
    con.Open();

    SqlCommand cmd = new SqlCommand("update Account set password = '" + cfmPass + "' where username = '" + username + "'", con);

    int i = cmd.ExecuteNonQuery();
    if (i != 0)
        status = "Password updated Successfully.";
    else
        status = "Unable to update password.";

    con.Close();
    return status;
}
```

This is the method **edit(),** it is used to edit the Admin's profile password. It does not require a parameter. The code starts off declaring a few strings and information, then accessing the connection with the **Open()** method. The first SqlCommand, **cmd**, is to select and update the password on the account where the username matches with the current logged in Administrator. It then uses ExecuteNonQuery to detect how many lines have been affected. If there are no changed records, the status will be set to "Unable to update password". The method then closes the connection and returns the status.

register()

```
public string register()
{
    string status = null;
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["myCS"].ToString());
    con.Open();

    //SqlCommand objectName = new Constructor(sqlQuery, connectionString);
    SqlCommand cmd = new SqlCommand("select count(*) from Account where username='" + username + "'", con);
    int count = Convert.ToInt32(cmd.ExecuteScalar().ToString());


    if (count == 0)
    {
        SqlCommand cmd2 = new SqlCommand("insert into Account(username, password, role) values (@un, @p, @role)", con);
        cmd2.Parameters.AddWithValue("@un", username);
        cmd2.Parameters.AddWithValue("@p", password);
        cmd2.Parameters.AddWithValue("@role", role);
        cmd2.ExecuteNonQuery();

        status = "Successfully registered";
    }
    else
        status = "Username is already registered";
    con.Close();

    return status;
}
```

This is the method **register(),** it is used to register a new receptionist or technician into the account database. It first creates a connection and opens the connection. Then, a new SqlCommand is created that counts the number of accounts that matches with the registering username. If there are no matches found, the information of the new account is registered into the Accounts database, and the status is set to “Successfully registered.” However, if there are accounts that have the same username, it will not register and set the status to “Username is already registered.” Finally, the connection is closed and the status is returned.

Admin_Report.cs admin method dispCount#()

```
public static int dispCount3(int m)
{
    int month = m;
    con.Open();
    SqlCommand cmdCount3 = new SqlCommand("select count (*) from Service where MONTH(request_date) = '" + month + "'AND service_type = 'Laptop screen replacement'", con);
    int count3 = Convert.ToInt32(cmdCount3.ExecuteScalar().ToString());
    con.Close();
    return count3;
}

1 reference
public static int dispCount4(int m)
{
    int month = m;
    con.Open();
    SqlCommand cmdCount4 = new SqlCommand("select count (*) from Service where MONTH(request_date) = '" + month + "'AND service_type = 'Laptop keyboard replacement'", con);
    int count4 = Convert.ToInt32(cmdCount4.ExecuteScalar().ToString());
    con.Close();
    return count4;
}

1 reference
public static int dispCount5(int m)
{
    int month = m;
    con.Open();
    SqlCommand cmdCount5 = new SqlCommand("select count (*) from Service where MONTH(request_date) = '" + month + "'AND service_type = 'Laptop battery replacement'", con);
    int count5 = Convert.ToInt32(cmdCount5.ExecuteScalar().ToString());
    con.Close();
    return count5;
}
```

The above image shows three out of 8 recurring methods named dispCount#, with the # replaced with the service number. These methods are replicated and repeated because a method is not able to return multiple variables without grouping into a tuple or list. The connection is first opened and make an SqlCommand that counts the amount of times a certain service has been done in the selected month. It then parses and converts the result of the command into an integer before closing the connection and returning the value.

dispSum#()

```
public static int dispSum4(int m)
{
    {
        int sum4 = 0;
        int month = m;
        con.Open();
        SqlCommand cmd = new SqlCommand("select SUM(service_fee) from Service where MONTH(request_date) = '" + month + "'AND service_type = 'Laptop keyboard replacement'", con);
        SqlCommand cmd2 = new SqlCommand("select count (*) from Service where MONTH(request_date) = '" + month + "'AND service_type = 'Laptop keyboard replacement'", con);
        int count = Convert.ToInt32(cmd2.ExecuteScalar().ToString());

        if (count > 0)
        {
            sum4 = int.Parse(cmd.ExecuteScalar().ToString());
        }
        else
            sum4 = 0;

        con.Close();
        return sum4;
    }
}
```

This is one of the dispSum# method. Similarly to the dispCount# method, it is repeated and the # is replaced with the service number. It first opens the connection after declaring a few variables, then creates and runs two SqlCommands. The first command adds the total income from the same service, while the second one checks if there are any records of the service. The second command is important because it will be used to prevent an exception error of attempting to convert null into an integer. If the record does not exist, it will automatically skip to setting the sum as 0. Finally, it closes the connection and returns the calculated sum.

addIncome()

```
public static int addIncome(int m)
{
    int sum = 0;
    int month = m;
    con.Open();
    SqlCommand cmd = new SqlCommand("select SUM(service_fee) from Service where MONTH(request_date) = '" + month + "'", con);
    SqlCommand cmd2 = new SqlCommand("select count (*) from Service where MONTH(request_date) = '" + month + "'", con);
    int count = Convert.ToInt32(cmd2.ExecuteScalar().ToString());

    if (count > 0)
    {
        sum = int.Parse(cmd.ExecuteScalar().ToString());
    }
    else
        sum = 0;

    con.Close();
    return sum;
}
```

addIncome() is a method that will add every single service income in the selected month. It first declares two variables before opening the connection. The base of the code is very similar to the dispSum#() method, with the exception that the filter for the service type has been removed. This will collect and sum all the service income in the selected month before closing and returning the value.

1.4.3.3 Objects

Admin objects

frmAdminEdit

```
private void btnSave_Click(object sender, EventArgs e)
{
    if (cboxConfirm.Checked == true)
        if (txtOldPass.Text == pass)
            if (txtNewPass.Text == txtCfmPass.Text)
            {
                Admin adminProf = new Admin(name, txtOldPass.Text, txtNewPass.Text, txtCfmPass.Text);
                MessageBox.Show(adminProf.edit());
            }
            else
                MessageBox.Show("Confirm Password does not match.");
        else
            MessageBox.Show("Old Password Incorrect.");
    else
        MessageBox.Show("Please confirm your changes");
}
```

"adminProf" is an object created as an instance of the class **"Admin"** with the information inside the textboxes such as the user's input of their old password, their new password and their

repeat confirmation password. This object is used to run the **Admin.edit()** method to change the password of the user.

frmAdminHome

```
private void btnRegister_Click(object sender, EventArgs e)
{
    if (rdbReceptionist.Checked == true)
        role = "Receptionist";
    else if (rdbTechnician.Checked == true)
        role = "Technician";
    if (txtUsername.Text != "" && txtPassword.Text != "" && role != "null")
    {
        Admin obj1 = new Group_35_Assignment.Admin(txtUsername.Text, txtPassword.Text, role);
        MessageBox.Show(obj1.register());
    }
}
```

"obj1" is another created instance of the class **"Admin"**, which uses the other overloaded constructor. This object is used to run the **Admin.register()** method to register a new technician or receptionist.

1.4.3.4 Events

frmAdminHome

```
private void frmAdminHome_Load(object sender, EventArgs e)
{
    lblUsername.Text = name;
}
```

This code changes the label displaying the current logged in user's username.

```
private void txtUsername_Enter(object sender, EventArgs e)
{
    if (txtUsername.Text == "Username")
    {
        txtUsername.Text = "";

        txtUsername.ForeColor = Color.Black;
    }
}

1 reference
private void txtUsername_Leave(object sender, EventArgs e)
{
    if (txtUsername.Text == "")
    {
        txtUsername.Text = "Username";

        txtUsername.ForeColor = Color.Gray;
    }
}

1 reference
private void txtPassword_Enter(object sender, EventArgs e)
{
    if (txtPassword.Text == "Password")
    {
        txtPassword.PasswordChar = '•';
        txtPassword.Text = "";

        txtPassword.ForeColor = Color.Black;
    }
}

1 reference
private void txtPassword_Leave(object sender, EventArgs e)
{
    if (txtPassword.Text == "")
    {
        txtPassword.PasswordChar = '\0';
        txtPassword.Text = "Password";

        txtPassword.ForeColor = Color.Gray;
    }
}
```

This section of the code automates the placeholder text on the textbox by switching the forecolor and replacing the placeholder text whenever the user leaves the textboxes empty.

```
private void btnEditProfile_Click(object sender, EventArgs e)
{
    frmAdminEdit f1 = new frmAdminEdit(name,pass);
    this.Hide();
    f1.ShowDialog();
    txtUsername.ForeColor = Color.Gray;
    txtPassword.ForeColor = Color.Gray;
    txtUsername.Text = "Username";
    txtPassword.PasswordChar = '\0';
    txtPassword.Text = "Password";
    this.Show();
}
```

This section of the code replaces all of the placeholder text back onto the textbox while it opens the edit profile form **"frmAdminEdit"**. It also hides the current admin homepage until the edit profile form is closed.

```
private void btnIncome_Click(object sender, EventArgs e)
{
    frmAdminIncome f3 = new frmAdminIncome(name,pass);
    this.Hide();
    f3.ShowDialog();
    this.Show();
}
```

This event handler runs and opens the **"frmAdminIncome"** and hides the main home form until the admin income form is closed.

```
private void btnLogOut_Click(object sender, EventArgs e)
{
    this.Close();
}
```

This event simply closes the Admin Home form and exits to the main login form when the log out button is pressed.

```
private void btnRegister_Click(object sender, EventArgs e)
{
    if (rdbReceptionist.Checked == true)
        role = "Receptionist";
    else if (rdbTechnician.Checked == true)
        role = "Technician";
    if (txtUsername.Text != "" && txtPassword.Text != "" && role != "null")
    {
        Admin obj1 = new Group_35_Assignment.Admin(txtUsername.Text, txtPassword.Text, role);
        MessageBox.Show(obj1.register());
    }
}
```

When the register button is pressed, it first checks which radio button is checked to select the correct role, and assigns the said role into the variable **"role"**. Then, it checks if all three username, password and roles are given enough information, and proceeds with creating a new instance object under the class **"Admin"**. This class is used to run the method register, and the status of the registration is returned at the end of the method to be displayed in a message box.

frmAdminEdit

```
private void frmAdminEdit_Load(object sender, EventArgs e)
{
    lblDispUsername.Text = name;
}
```

Upon the event of **"frmAdminEdit"** being loaded, the form automatically changes the text of the label displaying the username to show the correct username.

```
private void txtOldPass_Enter(object sender, EventArgs e)
{
    {
        if (txtOldPass.Text == "Old Password")
        {
            txtOldPass.PasswordChar = '•';
            txtOldPass.Text = "";

            txtOldPass.ForeColor = Color.Black;
        }
    }
}

1 reference
private void txtOldPass_Leave(object sender, EventArgs e)
{
    if (txtOldPass.Text == "")
    {
        txtOldPass.PasswordChar = '\0';
        txtOldPass.Text = "Old Password";

        txtOldPass.ForeColor = Color.Gray;
    }
}

1 reference
private void txtNewPass_Enter(object sender, EventArgs e)
{
    {
        if (txtNewPass.Text == "New Password")
        {
            txtNewPass.PasswordChar = '•';
            txtNewPass.Text = "";

            txtNewPass.ForeColor = Color.Black;
        }
    }
}

1 reference
private void txtNewPass_Leave(object sender, EventArgs e)
{
    if (txtNewPass.Text == "")
    {
        txtNewPass.PasswordChar = '\0';
        txtNewPass.Text = "New Password";

        txtNewPass.ForeColor = Color.Gray;
    }
}
```

Similar to the code in **"frmAdminHome"**, this code automates the placeholder text on the textbox by switching the forecolor and replacing the placeholder text whenever the user leaves the textboxes empty.

```
private void btnSave_Click(object sender, EventArgs e)
{
    if (cboxConfirm.Checked == true)
        if (txtOldPass.Text == pass)
            if (txtNewPass.Text == txtCfmPass.Text)
            {
                Admin adminProf = new Admin(name, txtOldPass.Text, txtNewPass.Text, txtCfmPass.Text);
                MessageBox.Show(adminProf.edit());
            }
            else
                MessageBox.Show("Confirm Password does not match.");
        else
            MessageBox.Show("Old Password Incorrect.");
    else
        MessageBox.Show("Please confirm your changes");
}
```

Upon clicking **"btnSave"**, the code confirms if **"cboxConfirm"** has been checked before checking if the old password the admin has inputted matches with the correct current password. Next, it checks if the new password and the confirmation password are the same before finally creating a new instance of the class **"Admin"** to run the edit method. Finally, the status in the method is returned to be displayed inside a message box.

```
private void btnHome_Click(object sender, EventArgs e)
{
    this.Close();
}
```

When the button labeled "Back to Home" is pressed, it exits the current form and returns back to the Admin Home form.

frmAdminIncome

```
private void btnHome_Click(object sender, EventArgs e)
{
    this.Close();
}
```

When the button labeled "Back to Home" is pressed, it exits the current form and returns back to the Admin Home form.

```
private void frmAdminIncome_Load(object sender, EventArgs e)
{
    DateTime dt = DateTime.Now;
    int currentMonth = dt.Month;
    cboMonth.SelectedIndex = currentMonth - 1;
}
```

Upon the loading of the form, today's date and time is collected and the month is set into the variable currentMonth. This variable is used to set the default month of the combo box **"cboMonth"** by subtracting the current month by one and using that as an index since the index starts from 0.

```
private void cboMonth_SelectedIndexChanged(object sender, EventArgs e)
{
    lblServ1Count.Text = Admin_Report.dispCount1(cboMonth.SelectedIndex + 1).ToString();
    lblServ2Count.Text = Admin_Report.dispCount2(cboMonth.SelectedIndex + 1).ToString();
    lblServ3Count.Text = Admin_Report.dispCount3(cboMonth.SelectedIndex + 1).ToString();
    lblServ4Count.Text = Admin_Report.dispCount4(cboMonth.SelectedIndex + 1).ToString();
    lblServ5Count.Text = Admin_Report.dispCount5(cboMonth.SelectedIndex + 1).ToString();
    lblServ6Count.Text = Admin_Report.dispCount6(cboMonth.SelectedIndex + 1).ToString();
    lblServ7Count.Text = Admin_Report.dispCount7(cboMonth.SelectedIndex + 1).ToString();
    lblServ8Count.Text = Admin_Report.dispCount8(cboMonth.SelectedIndex + 1).ToString();

    lblServ1Sum.Text = Admin_Report.dispSum1(cboMonth.SelectedIndex + 1).ToString();
    lblServ2Sum.Text = Admin_Report.dispSum2(cboMonth.SelectedIndex + 1).ToString();
    lblServ3Sum.Text = Admin_Report.dispSum3(cboMonth.SelectedIndex + 1).ToString();
    lblServ4Sum.Text = Admin_Report.dispSum4(cboMonth.SelectedIndex + 1).ToString();
    lblServ5Sum.Text = Admin_Report.dispSum5(cboMonth.SelectedIndex + 1).ToString();
    lblServ6Sum.Text = Admin_Report.dispSum6(cboMonth.SelectedIndex + 1).ToString();
    lblServ7Sum.Text = Admin_Report.dispSum7(cboMonth.SelectedIndex + 1).ToString();
    lblServ8Sum.Text = Admin_Report.dispSum8(cboMonth.SelectedIndex + 1).ToString();

    lblIncome.Text = "RM "+Admin_Report.addIncome(cboMonth.SelectedIndex + 1).ToString();
}
```

Lastly, whenever the index of the combo box **"cboMonth"** is changed, it automatically updates the count and income for each of the services using the methods in the class **"Admin_Report"**.

1.4.4 Customer

1.4.4.1 Class

CustOnServ

This is a class for the customer named **CustOnServ**. This class includes the functions for customer to view service description, view laptop collection date and view service fee. This class requires details including username, customer user id, service description, the laptop collection date and service fee.

```
class CustOnServ
{
    private string username;
    private int custUserId;
    private string serviceDesc;
    private string laptopCollectDate;
    private int serviceFee;
```

Member fields in **CustOnServ** are set as private to prevent cross access by other classes.

```
static SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["myCS"].ToString());

1 reference
public string ServiceDesc { get => serviceDesc; set => serviceDesc = value; }
1 reference
public string LaptopCollectDate { get => laptopCollectDate; set => laptopCollectDate = value; }
1 reference
public int ServiceFee { get => serviceFee; set => serviceFee = value; }

3 references
public CustOnServ(string un)
{
    username = un;
}
```

The connection to the database is applied here to access the SqlCommands. There are also three public backing fields that serve as value passing outside the class. Then, there is a parameterized constructor that accepts one string argument.

CustEditProfile

This is a class for the customer named **CustEditProfile**. This class includes the functions for a customer to edit a profile. This class requires details including username, current password, new password and re-enter password.

```
class CustEditProfile
{
    private string username;
    private string currentPw;
    private string newPw;
    private string rePw;

    static SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["myCS"].ToString());

    1 reference
    public CustEditProfile(string un, string cp, string np, string rp)
    {
        username = un;
        currentPw = cp;
        newPw = np;
        rePw = rp;
    }
```

The connection to the database is applied here to access the SqlCommands. There is a parameterized constructor that accepts four string arguments.

CustServChg

This is a class for the customer named **CustServChg**. This class includes the functions for a customer to change requested service. This class requires details including username, current password, new password and re-enter password.

```
class CustServChg
{
    private string username;
    private int custUserId;
    private string serviceType;
    private string changedServ;

    static SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["myCS"].ToString());

    1 reference
    public string ServiceType { get => serviceType; set => serviceType = value; }

    2 references
    public CustServChg(string un)
    {
        username = un;
    }
```

The connection to the database is applied here to access the SqlCommands. There is one public backing field that serves as value passing outside the class. Then, there is a parameterized constructor that accepts one string arguments.

1.4.4.2 Methods

CustOnServ member method
viewServiceDesc

```
public static void viewServiceDesc(CustOnServ o1)
{
    con.Open();
    SqlCommand cmd = new SqlCommand("select user_id from Account where username = '" + o1.username + "'", con);
    SqlDataReader rd = cmd.ExecuteReader();
    while (rd.Read())
    {
        o1.custUserId = rd.GetInt32(0);
    }
    rd.Close();
    SqlCommand cmd2 = new SqlCommand("select service_description from Service where user_id = '" + o1.custUserId + "' and paid_amount is null", con)
    SqlDataReader rd2 = cmd2.ExecuteReader();
    while (rd2.Read())
    {
        o1.serviceDesc = rd2["service_description"].ToString();
    }
    con.Close();
}
```

This is the method **viewServiceDesc,** it is used to view service description. It requires an object parameter. The code starts off with accessing the connection with the **Open()** method. The first SqlCommand, **cmd**, is to check for the user id that matches the username in the **Account** table. Then, the user id will be stored in a variable and passed down to the second SqlCommand. The second SqlCommand, **cmd2**, is to check the service description with the user id from the **Service** table of the database. The code will end by closing the connection.

viewLaptopCollectionDate

```
public static void viewLaptopCollectionDate(CustOnServ o2)
{
    con.Open();
    SqlCommand cmd = new SqlCommand("select user_id from Account where username = '" + o2.username + "'", con);
    SqlDataReader rd = cmd.ExecuteReader();
    while (rd.Read())
    {
        o2.custUserId = rd.GetInt32(0);
    }
    rd.Close();
    SqlCommand cmd2 = new SqlCommand("select laptop_collection_date from Service where user_id = '" + o2.custUserId + "'and paid_amount is null", con);
    SqlDataReader rd2 = cmd2.ExecuteReader();
    while (rd2.Read())
    {
        o2.laptopCollectDate = rd2["laptop_collection_date"].ToString();
    }
    con.Close();
}
```

This is the method **viewLaptopCollectionDate,** it is used to view laptop collection date for customers. It requires an object parameter. The code starts off with accessing the connection with the **Open()** method. The first SqlCommand, **cmd**, is to check for the user id that matches the username in the **Account** table. Then, the user id will be stored in a variable and passed down to the second SqlCommand. The second SqlCommand, **cmd2**, is to check the laptop collection date with the user id from the **Service** table of the database. The code will end by closing the connection.

viewServiceFee

```
public static void viewServiceFee(CustOnServ o3)
{
    con.Open();
    SqlCommand cmd = new SqlCommand("select user_id from Account where username = '" + o3.username + "'", con);
    SqlDataReader rd = cmd.ExecuteReader();
    while (rd.Read())
    {
        o3.custUserId = rd.GetInt32(0);
    }
    rd.Close();
    SqlCommand cmd2 = new SqlCommand("select service_fee from Service where user_id = '" + o3.custUserId + "'and paid_amount is null", con);
    SqlDataReader rd2 = cmd2.ExecuteReader();
    while (rd2.Read())
    {
        o3.serviceFee = int.Parse(rd2["service_fee"].ToString());
    }
    con.Close();
}
```

This is the method **viewServiceFee,** it is used to view service fee for customers. It requires an object parameter. The code starts off with accessing the connection with the **Open()** method. The first SqlCommand, **cmd**, is to check for the user id that matches the username in the **Account** table. Then, the user id will be stored in a variable and passed down to the second SqlCommand. The second SqlCommand, **cmd2**, is to check the service fee with the user id from the **Service** table of the database. The code will end by closing the connection.

CustEditProfile member method

editProfile

```
public string editProfile()
{
    string status;
    string password;
    password = null;
    con.Open();

    SqlCommand cmd = new SqlCommand("select password from Account where username = '" + username + "'", con);
    SqlDataReader rd = cmd.ExecuteReader();
    while (rd.Read())
    {
        password = rd.GetString(0);
    }
    rd.Close();
```

This is the method **editProfile,** it is used to edit profile for customers. It requires no parameter. The string variable **status** is for return statement at the end of the code while string variable password is as a variable to store the password data from the SqlCommand. The first SqlCommand, **cmd**, is to check for the password that matches the username in the **Account** table.

```
    if (password != currentPw)
    {
        status = "You have entered the wrong current password";
    }
    else
    {
        if (newPw != rePw)
        {
            status = "Your new password and re-enter password does not match";
        }
        else
        {
            SqlCommand cmd2 = new SqlCommand("update Account set password = '" + rePw + "' where username = '" + username + "'", con);
            int i = cmd2.ExecuteNonQuery();
            if (i != 0)
                status = "Update Successfully.";
            else
                status = "Unable to update";
        }
    }

    con.Close();
    return status;
}
```

After getting the password from the database, it will authenticate the current password recorded from the customer's input. If it is different, that means the customer had input the wrong current password and it will show a message. If it is correct, then it will check if the new password input is the same with the re-entered password in order to pass it to the next SqlCommand, **cmd2**, to update the re-entered password into the **Account** table according to the username. Then, it will check if the update is done successfully and store it into **status**, which will then be returned at the end of the code.

CustServChg member method

currentReqServ

```
public static void currentReqServ(CustServChg o1)
{
    con.Open();
    SqlCommand cmd = new SqlCommand("select user_id from Account where username = '" + o1.username + "'", con);
    SqlDataReader rd = cmd.ExecuteReader();
    while (rd.Read())
    {
        o1.custUserId = rd.GetInt32(0);
    }
    rd.Close();
    SqlCommand cmd2 = new SqlCommand("select service_type from Service where user_id = '" + o1.custUserId + "'and paid_amount is null", con);
    SqlDataReader rd2 = cmd2.ExecuteReader();
    while (rd2.Read())
    {
        o1.serviceType = rd2["service_type"].ToString();
    }
    con.Close();
}
```

This is the method **currentReqServ,** it is used to view current requested service for customers. It requires an object parameter. The code starts off with accessing the connection with the **Open()** method. The first SqlCommand, **cmd**, is to check for the user id that matches the username in the **Account** table. Then, the user id will be stored in a variable and passed down to the second SqlCommand. The second SqlCommand, **cmd2**, is to check the service type with the user id from the **Service** table of the database. The code will end by closing the connection.

updateServChg

```
public string updateServChg(string su, string cs)
{
    string status;
    string servUrgency = su;
    con.Open();
    changedServ = cs;
    SqlCommand cmd = new SqlCommand("select user_id from Account where username = '" + username + "'", con);
    SqlDataReader rd = cmd.ExecuteReader();
    while (rd.Read())
    {
        custUserId = rd.GetInt32(0);
    }
    rd.Close();

    SqlCommand cmd2 = new SqlCommand("update service set service_type = '" + changedServ + "' where user_id = '" + custUserId +"' and paid_amount is null", con);
    int i = cmd2.ExecuteNonQuery();
    if (i != 0)
    {
        status = "Service change update succesfully. ";
    }
    else
    {
        status = "Service change unable to update. ";
    }
    con.Close();
```

This is the method **updateServChg,** it is used to change requested service for customers. It requires two string parameter. The string variable **status** is for the return statement at the end of the code, string variable **servUrgency** is to store the service urgency from the class argument

while **changedServ** is the variable to store the requested service change from the class argument. The first SqlCommand, **cmd**, is to check for the user id that matches the username in the **Account** table. After getting the user id from the database, it will pass it to the next SqlCommand, **cmd2**, to update the **changedServ** into service_type in the **Service** table according to the user id. Then, it will check if the update is done successfully and store it into **status**.

```
con.Open();
SqlCommand cmd3 = new SqlCommand("update service set service_urgency = '" + servUrgency + "' where user_id = '" + custUserId + "' and paid_amount is null", con);
int j = cmd3.ExecuteNonQuery();
if (j != 0)
{
    status = status + "Service urgency update succesfully. ";
}
else
{
    status = status + "Service urgency unable to update. ";
}
con.Close();
```

After that, it will pass it to the next SqlCommand, **cmd3**, to update the **servUrgency** into service_urgency in the **Service** table according to the user id. Then, it will check if the update is done successfully and store it into **status**.

```
int price = 0;

if (servUrgency == "Normal")
{
    if (changedServ == "Remove virus, malware or spyware")
    {
        price = 50;
    }
    else if (changedServ == "Troubleshot and fix computer running slow")
    {
        price = 60;
    }
    else if (changedServ == "Laptop screen replacement")
    {
        price = 380;
    }
    else if (changedServ == "Laptop keyboard replacement")
    {
        price = 160;
    }
    else if (changedServ == "Laptop battery replacement")
    {
        price = 180;
    }
    else if (changedServ == "Operating System Format and Installation")
    {
        price = 100;
    }
    else if (changedServ == "Data backup and recovery")
    {
        price = 80;
    }
    else if (changedServ == "Internet connectivity issues")
    {
        price = 70;
    }
}
```

```
else if (servUrgency == "Urgent")
{
    if (changedServ == "Remove virus, malware or spyware")
    {
        price = 80;
    }
    else if (changedServ == "Troubleshot and fix computer running slow")
    {
        price = 90;
    }
    else if (changedServ == "Laptop screen replacement")
    {
        price = 430;
    }
    else if (changedServ == "Laptop keyboard replacement")
    {
        price = 200;
    }
    else if (changedServ == "Laptop battery replacement")
    {
        price = 210;
    }
    else if (changedServ == "Operating System Format and Installation")
    {
        price = 150;
    }
    else if (changedServ == "Data backup and recovery")
    {
        price = 130;
    }
    else if (changedServ == "Internet connectivity issues")
    {
        price = 100;
    }
}
```

The code continues with creating an integer variable price and depending on the **servUrgency** and **changedServ**, the price varies.

```
con.Open();
SqlCommand cmd4 = new SqlCommand("update Service set service_fee = '" + price + "' where user_id = '" + custUserId + "' and paid_amount is null", con);
int n = cmd4.ExecuteNonQuery();
if (n != 0)
{
    status = status + "Service fee update succesfully. ";
}
else
{
    status = status + "Service fee unable to update. ";
}
con.Close();

return status;
```

Then, another SqlCommand, **cmd4,** is to update the **price** into service_fee in the **Service** table. Then, it will check if the update is done successfully and store it into **status** that will be returned at the end of the code.

1.4.4.3 Objects

CustOnServ objects

frmCustOnServ

```
private void btnServiceDesc_Click(object sender, EventArgs e)
{
    CustOnServ COS = new CustOnServ(name);
    CustOnServ.viewServiceDesc(COS);
    MessageBox.Show("Service Description: " + COS.ServiceDesc);
}
```

"**COS**" is the object created for customers to view service description, and creating a new instance of class **CustOnServ** and passing down the name as an argument into the constructor. **COS** is then used as an object argument to call the **viewServiceDesc** method and show a message box to display the service description.

```
private void btnCusCollectionDate_Click(object sender, EventArgs e)
{
    CustOnServ COS = new CustOnServ(name);
    CustOnServ.viewLaptopCollectionDate(COS);
    MessageBox.Show("Laptop Collection Date: " + COS.LaptopCollectDate);
}

1 reference
private void btnCusTotalAmount_Click(object sender, EventArgs e)
{
    CustOnServ COS = new CustOnServ(name);
    CustOnServ.viewServiceFee(COS);
    MessageBox.Show("Total Amount To Be Paid: " + COS.ServiceFee);
}
```

Same process is executed for the customer to view the laptop collection date and total amount to be paid.

CustServChg objects

frmCustServChange

```
private void btnDone_Click(object sender, EventArgs e)
{
    if (rdbNormal.Checked)
    {
        if (cboServChg.SelectedItem.ToString() != lblReqServ.Text)
        {
            CustServChg obj1 = new CustServChg(name);
            MessageBox.Show(obj1.updateServChg(rdbNormal.Text, cboServChg.SelectedItem.ToString()));
        }
        else
        {
            MessageBox.Show("Please choose a different service from the current one to change");
        }
    }
    else if(rdbUrgent.Checked)
    {
        if (cboServChg.SelectedItem.ToString() != lblReqServ.Text)
        {
            CustServChg obj1 = new CustServChg(name);
            MessageBox.Show(obj1.updateServChg(rdbUrgent.Text, cboServChg.SelectedItem.ToString()));
        }
```

```
        else
        {
            MessageBox.Show("Please choose a different service from the current one to change");
        }
    }
    else
    {
        MessageBox.Show("Please choose the service urgency");
    }
}
```

"**obj1**" is the object created for customers to update service change, and creating a new instance of class **CustServChg** and passing down the name as an argument into the constructor. **obj1** is used to call the **updateServChg** method with the appropriate radio button text and combobox selection argument and show a message box to display the returned status.

CustEditProfile objects

frmCustEditProfile

```
private void btnDone_Click(object sender, EventArgs e)
{
    CustEditProfile CEP = new CustEditProfile(name, txtCurrentPw.Text, txtNewPw.Text, txtRePw.Text);
    MessageBox.Show(CEP.editProfile());
}
```

"**CEP**" is the object created for customers to edit profile, and creating a new instance of class **CustEditProfile** and passing down the name, current password, new password and re-entered password as argument into the constructor. **CEP** is then used to call the **editProfile** method and show a message box to display the returned status.

1.4.4.4 Events

frmCustOnServ

```
private void frmCustOnServ_Load(object sender, EventArgs e)
{
    lblUsername.Text = name;
}
```

This is the code when the form is loaded. It will take the name passed down from the login page to the username label to show the customer's username on the form.

```
private void cboProfileMenu_SelectedIndexChanged(object sender, EventArgs e)
{
    if (cboProfileMenu.SelectedIndex == 0)
    {
        frmCustEditProfile a = new frmCustEditProfile(name);
        a.ShowDialog();
    }
    else
    {
        this.Hide();
        frmUserLogin b = new frmUserLogin();
        b.ShowDialog();
        this.Close();
    }
}
```

When the selection of combobox for profile menu is selected, it will open the customer edit profile form if edit profile is selected or open the user login form and close the current form if logout is selected.

```
private void btnServiceDesc_Click(object sender, EventArgs e)
{
    CustOnServ COS = new CustOnServ(name);
    CustOnServ.viewServiceDesc(COS);
    MessageBox.Show("Service Description: " + COS.ServiceDesc);
}
1 reference
private void btnCusCollectionDate_Click(object sender, EventArgs e)
{
    CustOnServ COS = new CustOnServ(name);
    CustOnServ.viewLaptopCollectionDate(COS);
    MessageBox.Show("Laptop Collection Date: " + COS.LaptopCollectDate);
}

1 reference
private void btnCusTotalAmount_Click(object sender, EventArgs e)
{
    CustOnServ COS = new CustOnServ(name);
    CustOnServ.viewServiceFee(COS);
    MessageBox.Show("Total Amount To Be Paid: " + COS.ServiceFee);
}
```

These three actions occur when the respective button is clicked and will create an object of class **CustOnServ.** Then, it will show a messageBox for respective event including service description, laptop collection date and service fee.

```
private void btnServiceChg_Click(object sender, EventArgs e)
{
    this.Hide();
    frmCustServChange c = new frmCustServChange(name);
    c.ShowDialog();
}
```

This event takes place when the change requested service button is clicked, and it will hide the current form then show the change requested service form.

frmCustEditProfile

```
private void frmCustEditProfile_Load(object sender, EventArgs e)
{
    lblUsername.Text = name;
}
```

This is the code when the form is loaded. It will take the name passed down from the last form to the username label to show the customer's username on the form.

```
private void btnDone_Click(object sender, EventArgs e)
{
    CustEditProfile CEP = new CustEditProfile(name, txtCurrentPw.Text, txtNewPw.Text, txtRePw.Text);
    MessageBox.Show(CEP.editProfile());
}
```

This event occurs when the done button is clicked, and it will create an object of class CustEditProfile with the arguments including name, current password, new password and re-entered password. Then it will show the status on the message box with the editProfile method called.

```
private void btnCancel_Click(object sender, EventArgs e)
{
    this.Close();
}
```

When the cancel button is clicked, the customer edit profile form will close.

frmCustServChg

```
private void frmCustServChange_Load(object sender, EventArgs e)
{
    lblUsername.Text = name;
    CustServChg CSC = new CustServChg(name);
    CustServChg.currentReqServ(CSC);
    lblReqServ.Text = CSC.ServiceType;
}
```

This is the code when the change requested service form is loaded, it will take the name passed down from the last opened form to show it on the username label. Then, it will call the currentReqServ method and to show the service type on the lblReqServ.

```
private void btnDone_Click(object sender, EventArgs e)
{
    if (rdbNormal.Checked)
    {
        if (cboServChg.SelectedItem.ToString() != lblReqServ.Text)
        {
            CustServChg obj1 = new CustServChg(name);
            MessageBox.Show(obj1.updateServChg(rdbNormal.Text, cboServChg.SelectedItem.ToString()));
        }
        else
        {
            MessageBox.Show("Please choose a different service from the current one to change");
        }
    }
    else if(rdbUrgent.Checked)
    {
        if (cboServChg.SelectedItem.ToString() != lblReqServ.Text)
        {
            CustServChg obj1 = new CustServChg(name);
            MessageBox.Show(obj1.updateServChg(rdbUrgent.Text, cboServChg.SelectedItem.ToString()));
        }
        else
        {
            MessageBox.Show("Please choose a different service from the current one to change");
        }
    }
    else
    {
        MessageBox.Show("Please choose the service urgency");
    }
}
```

This is the code when the done button is clicked. First of all, it will check if any of the radio buttons are checked, or else it will show a message to ask for a selection for service urgency. If one of the radio button is checked, then it will check if the selection of combobox for service change is same as the current required service label text. Only if it is different then it will call the updateServChg method with the arguments, or else it will show a message to ask for a different selection in combobox.

```
private void btnCancel_Click(object sender, EventArgs e)
{
    this.Hide();
    frmCustOnServ s = new frmCustOnServ();
    s.ShowDialog();
    this.Close();
}
```

When the cancel button is clicked, the current form will hide and close while showing the customer ongoing service form.

```
private void cboProfileMenu_SelectedIndexChanged(object sender, EventArgs e)
{
    if (cboProfileMenu.SelectedIndex == 0)
    {
        frmCustEditProfile a = new frmCustEditProfile(name);
        a.ShowDialog();
    }
    else
    {
        this.Hide();
        frmUserLogin b = new frmUserLogin();
        b.ShowDialog();
        this.Close();
    }
}
```

When the selection of combobox for profile menu is selected, it will open the customer edit profile form if edit profile is selected or open the user login form and close the current form if logout is selected.

1.4.5 Technician - Soh Wei Zheng TP063631

1.4.5.1 Class

Technician.cs

After technicians log in into their technician account, technician will be led to 'Technician – Home' and the Technician.cs class will be utilized.

```
29 references
class Technician
{

    static SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["myCS"].ToString());
    private static string username;
    private static string password;
    public string customer_status;
    12 references
    public Technician()
    {

    }
    1 reference
    public Technician(string un)
    {
        username = un;
    }

    0 references
    public Technician(string un, string p)
    {
        username = un;
        password = p;
    }
```

This is the '**Technician.cs**' class. There are overloaded constructors created. The first constructor does not require any parameter but the second and third constructors require two strings parameters to store their username and password. The concept of class has been used.

frmTechHome.cs

```
5 references
public partial class frmTechHome : Form
{
    FormCollection forms = Application.OpenForms;

    private string username;
    private string password;

    0 references
    public frmTechHome(string un, string p)
    {
        InitializeComponent();
        username = un;
        password = p;
    }

    1 reference
    public frmTechHome(string un)
    {
        InitializeComponent();
        username = un;
    }

    1 reference
    private void frmTechHome_Load(object sender, EventArgs e)
    {
        lblTechUsername.Text = username;

    }
```

This class called '**frmTechHome.cs**'. There are overloaded constructors. The first constructor requires username and password as parameters. The second constructor only requires a username as a parameter. As the Technician Home Page is loaded, the username will be shown at the top. The concept of class has been used.

frmTechViewRequestedService.cs

```
{
    9 references
    public partial class frmTechViewRequestedService : Form
    {
        public static string SetValueFor_RequestedService;
        public static string SetValueFor_SelectedUsername;
        public static string SetValueFor_ServiceDescription;
        public static string SetValueFor_CollectionDate;

        public string current_collection_date;
        1 reference
        public frmTechViewRequestedService()
        {
            InitializeComponent();

        }
        1 reference
        private void frmTechViewRequestedService_Load(object sender, EventArgs e)
        {
            ArrayList username = new ArrayList();
            Technician tech = new Technician();
            username = tech.Get_matched_customer_username();

            foreach(string un in username)
            {
                cboTechCustomerNames.Items.Add(un);
            }

        }
```

This is '**View Requested Service.cs**' form and class. Firstly, there are four public strings created to store requested service, selected username, service description, collection date. Therefore, the technician can go 'Add Description and Collection Date' form directly with filled customer's data. As the form is loaded, it will create an array list to store customers' usernames. The concept of class, object, and method have been used.

frmTechEditProfile.cs

```
6 references
public partial class frmTechEditProfile : Form
{
    private string username;
    private bool apply_or_back;

                                                                                  – references
    public bool Apply_or_back { get => apply_or_back; set => apply_or_back = value; }

    0 references
    public frmTechEditProfile()
    {
        InitializeComponent();
    }

    1 reference
    public frmTechEditProfile(string un)
    {
        InitializeComponent();
        username = un;

    }
```

There are overloaded constructors. The first constructor requires nothing and the second constructor requires a string as a parameter. When the technician clicks on 'Edit Profile' at the home page, it create an object with the technician's username that is used to login. The username will be the parameter of the second constructor. There is a boolean created to store what the technician clicks on. Besides, the private boolean will be converted to public boolean through the concept of Get and Set. The concept of class has been used.

1.4.5.2 Methods and Object

Technician.cs

```
//view requested service
2 references
public ArrayList Get_matched_customer_username()
{
    con.Open();
    ArrayList matched_username_Account_arraylist = new ArrayList();
    SqlCommand cmd_matched_username_Account = new SqlCommand("select username from Account where role = 'Customer'", con);
    SqlDataReader rd_matched_username_Account = cmd_matched_username_Account.ExecuteReader();

    while (rd_matched_username_Account.Read())
    {
        matched_username_Account_arraylist.Add(rd_matched_username_Account.GetString(0));
    }

    rd_matched_username_Account.Close();
    con.Close();

    return matched_username_Account_arraylist;
}
```

This function in Technician class is **Get_matched_customer_username()**. Firstly, the connection function of the imported SQL library is used so the program can be connected to the database. After that, an array list object has been created to store those usernames which are customer and store each of the usernames from SQL command into array list. In the end, the array list, 'matched_username_Account_arraylist' will be returned to the function, so the combo box can list out all customers username for technician to select them. The concept of class and method have been used.

```
2 references
public string view_requested_service(string un)
{

    con.Open();
    string user_id;
    string requested_service;
    SqlCommand cmd_user_id = new SqlCommand("select user_id from Account where username ='" + un + "'", con);
    user_id = cmd_user_id.ExecuteScalar().ToString();
    con.Close();

    con.Open();
    SqlCommand cmd_requested_service = new SqlCommand("select service_type from Service where user_id='"+Int16.Parse(user_id)+"'",con);
    requested_service = cmd_requested_service.ExecuteScalar().ToString();
    con.Close();

    return requested_service;

}
```

This method called '**view_requested_service**'. Its main purpose is to return a selected customer's requested service. Firstly, the method needs to use SQL command select the user id

which is matched with the username that technician selected because there is no username in our Service database and store it as a created string, 'user_id'. After selecting customers' user id, the SQL command select the requested service of the selected user id and store the requested service as a string. In the end, the string of requested service will be returned. The concept of method and object have been used.

```
2 references
public string view_collection_date(string un)
{
    con.Open();
    string user_id;
    string collection_date;
    int count_status;
    SqlCommand cmd_user_id = new SqlCommand("select user_id from Account where username ='" + un + "'", con);
    user_id = cmd_user_id.ExecuteScalar().ToString();


    SqlCommand check = new SqlCommand("select count(*) from Service where user_id = '" + user_id + "' and paid_amount is null", con);
    count_status = Int16.Parse(check.ExecuteScalar().ToString());
    if (count_status > 0)
    {
        SqlCommand cmd_collection_date = new SqlCommand("select convert(varchar,laptop_collection_date,23) from Service where user_id='" + Int16.Parse(user_id) + "' AND paid_amount IS NULL", con);
        collection_date = cmd_collection_date.ExecuteScalar().ToString();
    }
    else
    {
        collection_date = "No service.";
    }

    con.Close();

    return collection_date;
}
```

'**view_collection_date**' will return the collection date based on the selected customer. The process is similar to the previous method. Firstly, the program connects to the database. After that, the program creates a string to store the user id and then use the user id to select the date based on the returned user id by creating an object to implement SQL command. The concept of method and object have been used.

```
2 references
public string view_description(string un)
{
    con.Open();
    string user_id;
    string service_description;
    SqlCommand cmd_user_id = new SqlCommand("select user_id from Account where username ='" + un + "'", con);
    user_id = cmd_user_id.ExecuteScalar().ToString();
    con.Close();

    con.Open();
    SqlCommand cmd_service_description = new SqlCommand("select service_description from Service where user_id='" + Int16.Parse(user_id) + "'", con);
    service_description = cmd_service_description.ExecuteScalar().ToString();
    con.Close();

    return service_description;
}
```

'**view_description**' returns the selected customer's service_description. The process is same as the previous method as well. Firstly, it creates two strings, one is for storing user id and another string is for storing the service description that returned by SQL command. After that, a

SQLCommand object is created to implement SQL command to select user id. Then, the returned user id will be used to select the service description from Service database by creating another SQLCommand object and the return of 'ExecuteScalar()' will be converted into string and stored as a string. In the end, the string will be returned. The concept of method and object have been used.

```
//Add Description and Collection Date
2 references
public string edit_total_amount(string un)
{
    con.Open();
    string user_id;
    string total_amount;
    SqlCommand cmd_user_id = new SqlCommand("select user_id from Account where username ='" + un + "'", con);
    user_id = cmd_user_id.ExecuteScalar().ToString();
    con.Close();

    con.Open();
    SqlCommand cmd_total_amount = new SqlCommand("select service_fee from Service where user_id='"+Int16.Parse(user_id)+"'",con);
    total_amount = "RM "+cmd_total_amount.ExecuteScalar().ToString();
    con.Close();

    return total_amount;
}
```

'edit_total_amount' is similar to previous method as well. Its main purpose is to select the total amount that customer need to pay. In the end, the total amount will be stored as string and the string will be returned. The concept of method and object have been used.

```
1 reference
public void update_detalis(string un,string s,string c)
{
    string user_id;

    con.Open();
    SqlCommand cmd_user_id = new SqlCommand("select user_id from Account where username='" + un + "'", con);
    user_id = cmd_user_id.ExecuteScalar().ToString();
    con.Close();
    con.Open();

    SqlCommand cmd_update_details = new SqlCommand("update Service SET laptop_collection_date='"+c+"','+" service_description='"+s+"' where user_id="+Int16.Parse(user_id)+" AND paid_amount IS NULL",con);
    cmd_update_details.ExecuteScalar();
    con.Close();

}
```

'**update_details**' main purpose is to update customers' collection date and service description. It will be used on '*frmTechAddDescriptionAndLaptopCollectionDate.cs*'. This method is similar to previous method as well. However, there is a different part, which is the updating part. A SQLCommand object will be created to implement SQL command to update the collection date and service description in the database. The concept of method and object have been used.

```
//Edit Profile
1 reference
private string return_user_id(string un)
{
    con.Open();
    SqlCommand cmd_user_id = new SqlCommand("select user_id from Account where username='" + username + "'",con);
    string user_id = cmd_user_id.ExecuteScalar().ToString();
    con.Close();
    return user_id;
}
```

This method called **'return_current_password**()'. Its main purpose is to return user id in Account database which is matched to the user's username. For example, if an user login with username 'Ali', then 'Ali' will be stored as a private string and the string will be input into this method to get his user id from Account database. The concept of method and object have been used.

```
1 reference
private void update_new_password_username(string current_username, string new_username, string new_password)
{

    string user_id = return_user_id(current_username);
    con.Open();
    SqlCommand cmd_update_password = new SqlCommand("update Account SET password='"+new_password+"', username ='"+new_username+"' where (user_id='"+user_id+"')",con);
    cmd_update_password.ExecuteNonQuery();
    con.Close();
}
```

'**update_new_password_username**'s main purpose is to update the details of user. User can update their username and password. There are three parameters in this method. Firstly, a string will be created to store a return of a method, 'return_current_password()' to get the user id of the first parameter, 'current_username' which is the username that user used to login. Then, the second parameter will be the string of new username, and the third parameter will be new password. After user enter their new username and password, a SQLCommand object will be created to update their details. To make it more secure, this method will be private and called in another public method, 'edit_profile'. The concept of method and object have been used.

```
1 reference
public void edit_profile(string current_username,string new_username, string new_password)
{
    //username = username;
    update_new_password_username(current_username, new_username, new_password);


}
```

This method called **'edit_profile'**. It is a public method and it calls the previous method. It will be called on *'frmTechEditProfile.cs'*. The concept of method has been used.

```
//Check if technicians click on apply or back button
2 references
public bool IsApply(string apply_or_back)
{
    if (apply_or_back == "Apply")
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

This method called '**IsApply**' and it returns what the technician clicks on. If the technician clicks on 'Apply' button, it returns true. Otherwise, it returns false.

frmTechHome.cs

```
1 reference
private void btnTechViewReqServ_Click(object sender, EventArgs e)
{
    this.Hide();
    frmTechViewRequestedService viewService = new frmTechViewRequestedService();
    viewService.ShowDialog();
    this.Show();
}
```

When the 'View Requested Service' button is clicked, it will create an object of a form. After that, the form will be shown. The concept of method and object have been used.

```
1 reference
private void lblTechAddDescAndCollecDate_Click(object sender, EventArgs e)
{
    this.Hide();
    frmTechAddDescriptionAndLaptopCollectionDate editService = new frmTechAddDescriptionAndLaptopCollectionDate();
    editService.ShowDialog();
    this.Show();
}
```

When the 'Add Description and Collection Date' button is clicked, it will create an object of a form. After that, the form will be shown. The concept of method and object have been used.

```
1 reference
private void btnEditProfile_Click(object sender, EventArgs e)
{
    this.Hide();
    frmTechEditProfile editProfile = new frmTechEditProfile(username);
    editProfile.ShowDialog();
    if (editProfile.Apply_or_back==true)
    {
        this.Close();
    }
    else
    {
        this.Show();
    }
}
```

When the ‘Edit Profile’ button is clicked, it will create an object of a form with the username that the user used to login and check if the technician clicks on ‘Apply’ button or ‘Back’ button. After that, the form will be shown or closed based on the button technician clicked. The concept of method and object have been used.

```
1 reference
private void btnTechSignOut_Click(object sender, EventArgs e)
{
    this.Close();

}
```

When the ‘Sign Out’ button is clicked, it will exit to the login page. The concept of method object has been used.

frmTechAddDescriptionAndLaptopCollectionDate.cs

```
1 reference
private void current_service()
{
    string selected_username;
    string c;
    selected_username = cboTechCustomerNames.Text;
    Technician tech = new Technician();
    c= tech.view_requested_service(selected_username);
    if(current_collection_date=="No service")
    {
        txtTechCurrentService.Text = "";

    }
    else
    {
        txtTechCurrentService.Text = c;
    }
}
```

'**current_service**'s main purpose is to show the selected customer's current service. Firstly, a string will be created to store the selected username from combo box. After that, a Technician object will be created to use the Technician class's method. Then, the 'view_requested_service()' will be called with the string which is stored selected customer's username. The concept of method and object have been used.

```
1 reference
private void service_description()
{
    string selected_username;
    string edit_service_description;
    selected_username = cboTechCustomerNames.Text;
    Technician tech = new Technician();
    edit_service_description = tech.view_description(selected_username);

    if(current_collection_date=="No service")
    {
        txtTechServiceDescription.Text = "";
    }
    else
    {
        if (edit_service_description == "")
        {
            txtTechServiceDescription.Text = "None.";
        }
        else
        {
            txtTechServiceDescription.Text = edit_service_description;
        }
    }


}
```

'**service_description**'s main purpose it to show the selected customer's service description. The process is similar to previous methods. Firstly, it creates a string and stores the selected customer username. Then, it creates an object to call the 'view_description()' function with a string of selected customer's username as a parameter. If the service description is nothing, then the service description will show "None". Instead, the whole service description will be shown. The concept of method and object have been used.

```
1 reference
private void collection_date()
{
    string selected_username;
    selected_username = cboTechCustomerNames.Text;
    string edit_collection_date;
    Technician tech = new Technician();
    edit_collection_date = tech.view_collection_date(selected_username);

    if (edit_collection_date == "No service.")
    {
        current_collection_date = edit_collection_date;
        MessageBox.Show("No service.");
    }
    else
    {
        txtTechLaptopCollectionDate.Text = edit_collection_date;
        current_collection_date = edit_collection_date;
    }

}
```

'**collection_date**'s main purpose is to show collection date. This method is almost the same as previous method. The concept of method and object have been used.

```
private void selected_total_amount(string un)
{
    string edit_total_amount;
    Technician tech = new Technician();
    edit_total_amount = tech.edit_total_amount(un);
    txtTechTotalAmount.Text = edit_total_amount;

    if(current_collection_date=="No service.")
    {
        txtTechTotalAmount.Text = "";
    }
    else
    {
        txtTechTotalAmount.Text = edit_total_amount;
    }
}
```

This method will be called when the technician firstly go to the '*View Requested Service*' and select customer to view. There is a button called 'Edit'. If the technician wants to edit the details, he can click on it and it will open a 'Add Description and Laptop Collection Date' form to allow technician edit. At this moment, this method will be called and the selected customer's details will be copied to the new form by using global variable. The process is similar to the previous method. The concept of method and object have been used.

```
1 reference
private void unselected_total_amount()
{
    string edit_total_amount;
    string selected_username = cboTechCustomerNames.Text;
    Technician tech = new Technician();
    edit_total_amount = tech.edit_total_amount(selected_username);

    if(current_collection_date=="No service.")
    {
        txtTechTotalAmount.Text = "";
    }
    else
    {
        txtTechTotalAmount.Text = edit_total_amount;
    }
}
```

When the technician go to the '*Add Description and Laptop Collection Date*' directly from the Home Page, there is no any pre-selected customer. Therefore, this method will be called after the technician click 'Choose' button. The concept of method and object have been used.

```
1 reference
private void frmTechAddDescriptionAndLaptopCollectionDate_Load(object sender, EventArgs e)
{
    ArrayList username = new ArrayList();
    Technician tech = new Technician();
    username = tech.Get_matched_customer_username();

    foreach(string un in username)
    {
        cboTechCustomerNames.Items.Add(un);
    }

    if (frmTechViewRequestedService.SetValueFor_SelectedUsername != null)
    {
        lblTechSelectedUsername.Text = frmTechViewRequestedService.SetValueFor_SelectedUsername;
        txtTechCurrentService.Text = frmTechViewRequestedService.SetValueFor_RequestedService;
        txtTechServiceDescription.Text = frmTechViewRequestedService.SetValueFor_ServiceDescription;
        txtTechLaptopCollectionDate.Text = frmTechViewRequestedService.SetValueFor_CollectionDate;
        selected_total_amount(lblTechSelectedUsername.Text);
    }
    else
    {
        lblTechSelectedUsername.Text = "Please pick a customer.";
    }
}
```

When the form is loaded, the program create an array list object to store the customers' username for technician to choose at combo box. After that, the program add each of the name in array to the array list. Then, it will check if the technician comes from the 'View Requested Service' with a selected customer or from the 'Home Page' without selected username. If there is selected customer, the customer's name will be shown at the top. Instead, it will show 'Please pick a customer'. The concept of method and object have been used.

```
1 reference
private void btnTechChoose_Click(object sender, EventArgs e)
{
    if (cboTechCustomerNames.Text == "")
    {
        MessageBox.Show("Please pick a customer.");
    }
    else
    {
        collection_date();
        if (current_collection_date=="No service.")
        {
            MessageBox.Show("This customer has no service. Please choose other customer.");
        }
        else
        {
            lblTechSelectedUsername.Text = cboTechCustomerNames.Text;
            current_service();
            service_description();

            unselected_total_amount();
        }

    }
}
```

When the technician clicks the 'Choose' button, it will check if the technician has selected customer. If there is selected customer, those mentioned methods will be called and the customer's name will be shown at the top. The concept of method and object have been used.

```
1 reference
private void btnTechApply_Click(object sender, EventArgs e)
{
    Technician tech = new Technician();

    string collection_date = txtTechLaptopCollectionDate.Text;
    string service_description = txtTechServiceDescription.Text;
    string username;
    if(lblTechSelectedUsername.Text!="Please pick a customer.")
    {
        username = lblTechSelectedUsername.Text;
    }
    else
    {
        username = cboTechCustomerNames.Text;
    }
    try
    {
        tech.update_detalis(username, service_description, collection_date);
        MessageBox.Show("Detail Has Been Updated Sucessfully.","Detail Update",MessageBoxButtons.OK,MessageBoxIcon.Information);
    }
    catch
    {
        MessageBox.Show("Please make sure your computer format is yyyy/MM/dd.","Error",MessageBoxButtons.OK,MessageBoxIcon.Error);
    }

}
```

When the technician clicks the ‘Apply’ button, it will firstly create a ‘Technician’ object, strings for collection date, service description and username. After that, it will check if the label that shown at the top is not ‘Please pick a customer’. If the label is not ‘Please pick a customer’, it means there is a selected username and the username will be stored as ‘username’ string. Instead, the username will be stored as the selected customer at the combo box. Then, it will call ‘update_details()’ with the username that just stored , service description, and the collection date. After that, the message box will be shown to inform the technicians if it is updated successfully. The concept of method and object have been used.

frmTechViewRequestedService.cs

```
1 reference
private void view_requested_service()
{
    string v;
    string selected_username;
    selected_username = cboTechCustomerNames.Text;
    Technician tech = new Technician();
    v = tech.view_requested_service(selected_username);
    if (current_collection_date=="No service.")
    {
        txtTechRequestedService.Text = "";
    }
    else
    {
        txtTechRequestedService.Text = v;

    }

}
```

'**view_requested_service**'s main purpose is to get customer's requested service and show it on the textbox. The concept of method and object have been used.

```
1 reference
private void view_collection_date()
{
    string selected_username;
    selected_username = cboTechCustomerNames.Text;
    string collection_date;
    Technician tech = new Technician();
    collection_date = tech.view_collection_date(selected_username);
    if (collection_date=="No service.")
    {
        current_collection_date = collection_date;
        MessageBox.Show("No service.");
    }
    else
    {
        txtCollectionDate.Text = collection_date;
        current_collection_date = collection_date;
    }
}
```

'**view_collection_date**'s main purpose is to get customer's collection date and show it on textbox. It will show 'None', if there is no collection date. The concept of method and object have been used.

```
1 reference
private void view_service_description()
{
    string selected_username;
    selected_username = cboTechCustomerNames.Text;
    string service_description;
    Technician tech = new Technician();
    service_description = tech.view_description(selected_username);

    if (current_collection_date == "No service")
    {
        txtTechDescription.Text = "";
    }
    else
    {
        if (service_description == "")
        {
            txtTechDescription.Text = "None";
        }
        else
        {
            txtTechDescription.Text = service_description;
        }
    }
    SetValueFor_ServiceDescription = txtTechDescription.Text;
}
```

'**view_service_description**' is used to get customer's service description and show it on textbox. It will show 'None' if there is no service description. The concept of method and object have been used.

```
1 reference
private void btnTechView_Click(object sender, EventArgs e)
{
    if (cboTechCustomerNames.Text == "")
    {
        MessageBox.Show("Please pick a customer.");
    }
    else

    {
        view_collection_date();
        if (current_collection_date=="No service.")
        {
            MessageBox.Show("This customer has no service. Please choose other customer.");
        }
        else
        {
            lblTechCustomerName.Text = cboTechCustomerNames.Text;
            view_requested_service();
            view_service_description();

        }
    }
}
```

When the technician clicks ‘View’ button, it will firstly check if the technician selected customer name at combo box. If technician selected customer, it will call those mentioned method. Instead, it will inform the technician to pick a customer. The concept of method and object have been used.

```
1 reference
private void btnTechEdit_Click(object sender, EventArgs e)
{
    frmTechAddDescriptionAndLaptopCollectionDate editCustomer = new frmTechAddDescriptionAndLaptopCollectionDate();
    SetValueFor_SelectedUsername = cboTechCustomerNames.Text;
    SetValueFor_RequestedService = txtTechRequestedService.Text;
    SetValueFor_CollectionDate = txtCollectionDate.Text;
    SetValueFor_ServiceDescription = txtTechDescription.Text;

    if (SetValueFor_SelectedUsername == ""||SetValueFor_SelectedUsername == ""||SetValueFor_RequestedService == ""||SetValueFor_ServiceDescription=="") {
        MessageBox.Show("Please select a customer.");
    }else
    {

        editCustomer.Show();
        this.Close();
    }
}
```

When the technician clicks ‘Edit’ button, it will link to new form and all the selected username, requested service, collection date, and service description will be stored as strings, so the program does not have to select these details from database again. Then, the ‘Add Description

and Collection Date' will be shown. If those strings are empty, it means the technician does not choose a customer and he will be informed. The concept of method and object have been used.

```
1 reference
private void btnTechBack_Click(object sender, EventArgs e)
{
    this.Close();

}
```

When the technician clicks on 'Back' button, it will close this form. The concept of method has been used.

frmTechEditProfile

```
1 reference
private void btnTechApply_Click(object sender, EventArgs e)
{
    string new_username = txtNewUsername.Text;

    string new_password=txtNewPassword.Text;


    Technician tech = new Technician(username);
    if ((new_username != "Enter New Username")&&(new_password !="Enter New Password"))
    {
        try
        {
            tech.edit_profile(username, new_username, new_password);
            apply_or_back = tech.IsApply("Apply");
            MessageBox.Show("Detail Has Been Updated Sucessfully.", "Edit Profile");
            this.Close();
            MessageBox.Show("Please Login Again.");
        }
        catch
        {
            MessageBox.Show("Failed To Update.", "Error");
            MessageBox.Show("Please Make Sure You Insert Properly.", "Tips");
        }
    }
    else
    {
        MessageBox.Show("Username and Password Must Not Be Blank.","Error");
    }
}
```

When the user click on 'Apply' button, the program creates two strings to store the new username and new password. After that, it creates a new 'Technician' object with the technician's username that just stored. Then, it calls the 'edit_profile()' with stored technician's username, new username, and the new password. In the end, it will back to the log in page. The concept of method and object have been used.

1.5 Test Plan and Test Cases

Test Case	Function name	Test Objective	Expected Result	Actual Result	Remarks
Login					
1	Login	Able to redirect to the intended page based on the user role	The login form is hidden, and the intended page is shown Display error message if the username and password is incorrect	The login form is hidden, and the intended page is shown The login form is hidden, and a message box pops up saying incorrect username / password	This function is successful
Admin					
1	Register receptionist or technician	Successfully add and register a new entry into the database based on what role is selected	A new record is added into the database and the new account can be logged in.	Registration is successful and the new accounts can log in without any issues.	There are no visual cues on if the registration has failed or succeeded.
2	Edit Admin Profile	Successfully change the login password for the current account.	The password should be changed and updated in the database, and login should have no problems	Logging in after changing the password using the new password was successful.	None

3	Displaying the number of certain services completed (Service Report)	All services in the database are filtered and processed to obtain the number of times a certain service has been completed in a selected month	The labels displaying the number of times each service has been completed in the selected month.	The count labels are displaying the correct numbers and changes based on the current selected month and service.	None
4	Calculating and summing the income and total income (Total Income)	The income (payable and paid) are collected from the database and displayed on the count labels based on the selected month.	The income labels are displaying the correct sum of income for each service at the selected month	The income labels are displaying the correct numbers and changes based on the current selected month and service.	None
Receptionist					
1	Register customer	Create a customer and store into the database	A message box pops up telling whether the registration is successful or not	Registration is done correctly.	None

2	Show password	Display the text of the password needed to be created	Able to see the text of the password.	At first the password text is shown. But if checking the checkbox multiple times it remains as the password character.	The code needs to be checked again
3	Showing the service fee when creating the service	The service fee are able to be shown according to the service selected	The service fee is successfully displayed according to the service selected	The label only shows the service fee of the first service in the combo box.	The code in the class needs to be checked again
4	Accept payment	Able to accept payment from customer	The paid amount is successfully stored in the database	Error message pops up because the customer has not made any request yet	Customer with no service request cannot be selected
5	Generate Receipt	Able to generate receipt	The receipt is generated with all labels are displayed correctly	The receipt is generated successfully with all the information displayed correctly	None

6	Edit Receptionist Profile	Able to change the password of the receptionist.	The password is successfully changed	Password is changed correctly	None
Customer					
1	View Service Description	Able to show customer's service description	Shows a message box regarding the customer's service description.	A message box is shown but service description is null because no service description is input from technician yet.	Needs a technician to provide an input to show the proper result.
2	View Laptop Collection Date	Able to show customer's laptop collection date	Shows a message box regarding the customer's laptop collection date.	A message box is shown but service description is null because no service description is input from technician yet.	Needs a technician to provide an input to show the proper result.
3	View Total Amount To Be Paid	Able to show customer's service fee	Shows a message box regarding the customer's service fee.	A message box is shown with the correct service fee .	None

4	Change Requested Service	Able to change requested service with updated service urgency and service fee	The service type, service urgency and service fee are successfully updated based on selection.	The service type, service urgency and service fee are successfully updated with a message box shown.	None
5	Edit Profile	Able to change customer's password	The current password is changed into the new password.	Customer's password is successfully updated to new password.	None
Technician					
1	Check and Update Technician Profile	To update user's details.	The username and password will be updated.	The username and password have been changed.	None
2	View Requested Service	To view requested service.	The selected customer's requested service, description, and collection date will be shown.	The customer's requested service, description, and collection date have been shown. If the technician does not select	None

				customer and click ‘View’ button, a message box will be shown.	
3	Add Description and Collection Date	To add or edit selected customer’s collection date and description.	The selected customer’s collection date and description will be updated.	The selected customer’s collection date and description will be updated. If the technician does not select a customer and apply it, a message box will be shown.	The date format must be yyyy-MM-dd.

2.0 Conclusion

This prototype of a laptop repair management system is done using Visual Studio using C# and the data is stored in a database management system, Microsoft SQL Server. There are a total of four types of users which are able to login to this system, they are the admin, receptionist, technician, and customer. Each of the user roles have their respective page to go to. The receptionists are in charge of registering new customer accounts and have them pick a service. Once the service is done, the customer should come on the laptop collection date to pick up their laptop and make a payment, a receipt will also be generated. Each customer can make only one service at a time, which means that there should not have two ongoing services for one customer in this system. The customer also can only make the payment once the laptop collection date is set. Once a service is created, the technician can view the requested service and repair the laptop. Once repairment is done, the technician can set the laptop collection date. Moreover, the customer is allowed to change their requested service if the service is not done. They can also review their service description, laptop collection date, and total amount that needs to be paid if any of these are set. The admin can register new technicians and receptionists. They can also view the monthly service report and the total income gained. Lastly, all of the users are able to change their passwords if necessary.

The strength of our designed system is that it is user-friendly as everything is perfectly designed and states clearly what to do in each part of the system. The system automatically obtained the system's date and does not have to type manually when creating a service for the customer. It is very convenient for the customer to have their own account to view the service requested and some related information, this idea could make it into an online system. The weaknesses of this prototype system created is there is too little information to store for the user account. The customer can only have one service at a time makes the system very inconvenient to use. The enhancements for this system could have every user to include more personal details like a resume or photo, this system could be made online so that the possible user can access anywhere at any time. The admin should be able to view more types of reports like customers

and attendance of the employee, etc. Overall this is still a prototype but have the main features of a laptop repair management system, more innovative ideas should be implemented in the system.

3.0 References

Edgewood Solutions. (2021). *Date and Time Conversions Using SQL Server*. MSSQLTips.com. https://www.mssqltips.com/sqlservertip/1145/date-and-time-conversions-using-sql-server/

Microsoft. (n.d.). *DateTime.Now Property*. Microsoft. https://docs.microsoft.com/en-us/dotnet/api/system.datetime.now?view=net-6.0

4.0 Workload Matrix

No.	Assigned Task & Brief Description	Assigned Member Name	Completion Status / Comment
1.	Login Storyboard	Teoh Will Shen	Done
2.	Receptionist Storyboard	Teoh Will Shen	Done
3.	Design Login form in Microsoft Visual Studio	Teoh Will Shen	Done
4.	Design Receptionist in Microsoft Visual Studio	Teoh Will Shen	Done
5.	Write the codes for the login function	Teoh Will Shen	Done
6.	Write the codes for all the functions of receptionist	Teoh Will Shen	Done
7.	Create the documentation template	Teoh Will Shen	Done
8.	Login Class Diagram	Teoh Will Shen	Done
9.	Receptionist Class Diagram	Teoh Will Shen	Done
10.	Explain the codes written for the all functions of receptionist in Object-Oriented Programming Concepts	Teoh Will Shen	Done
10.	Test Plan and Test Cases for receptionist	Teoh Will Shen	Done
11.	Conclusion	Teoh Will Shen	Done
12.	Use-Case Diagram	Teoh Will Shen & Soh Wei Zheng	Done
13.	Technician Storyboard	Soh Wei Zheng	Done

14.	Design Technician in Microsoft Visual Studio	Soh Wei Zheng	Done
15.	Write the codes for all the function of technician	Soh Wei Zheng	Done
16.	Technician Class Diagram	Soh Wei Zheng	Done
17.	Explain the codes written for the all functions of technician in Object-Oriented Programming Concepts	Soh Wei Zheng	Done
18.	Test Plan and Test Cases for technician	Soh Wei Zheng	Done
19.	Administrator Storyboard	Kazim Chin Shen	Done
20.	Design Administration form in Microsoft Visual Studio	Kazim Chin Shen	Done
21.	Design Logo for the program	Kazim Chin Shen	Done
22.	Write the codes for the admin related classes	Kazim Chin Shen	Done
23.	Admin Class Diagram	Kazim Chin Shen	Done
24.	Explain the codes written for the all functions of Admin in Object-Oriented Programming Concepts	Kazim Chin Shen	Done
25.	Test Plan and Test Cases for Admin	Kazim Chin Shen	Done
26.	Compilation and Bug Fixing	Kazim Chin Shen	Done
27.	Customer Storyboard	Tan Ming Hui	Done
28.	Design Customer form in Microsoft Visual Studio	Tan Ming Hui	Done
29.	Write the codes for customer related classes	Tan Ming Hui	Done
30.	Customer Class Diagram	Tan Ming Hui	Done

31.	Explain the codes written for the all functions of Admin in Object-Oriented Programming Concepts	Tan Ming Hui	Done
32	Test Plan and Test Cases for Customer	Tan Ming Hui	Done
33.	Create Database	Tan Ming Hui	Done