GROUP ASSIGNMENT

TECHNOLOGY PARK MALAYSIA

CT044-3-1-IOOP

INTRODUCTION TO OBJECT ORIENTED PROGRAMMING

APU1F2109/APD1F2109SE/CS(IS)

LECTURER'S NAME: Au Yit Wah

HAND OUT DATE: 1ST APRIL 2022

HAND IN DATE: 27TH JUNE 2022

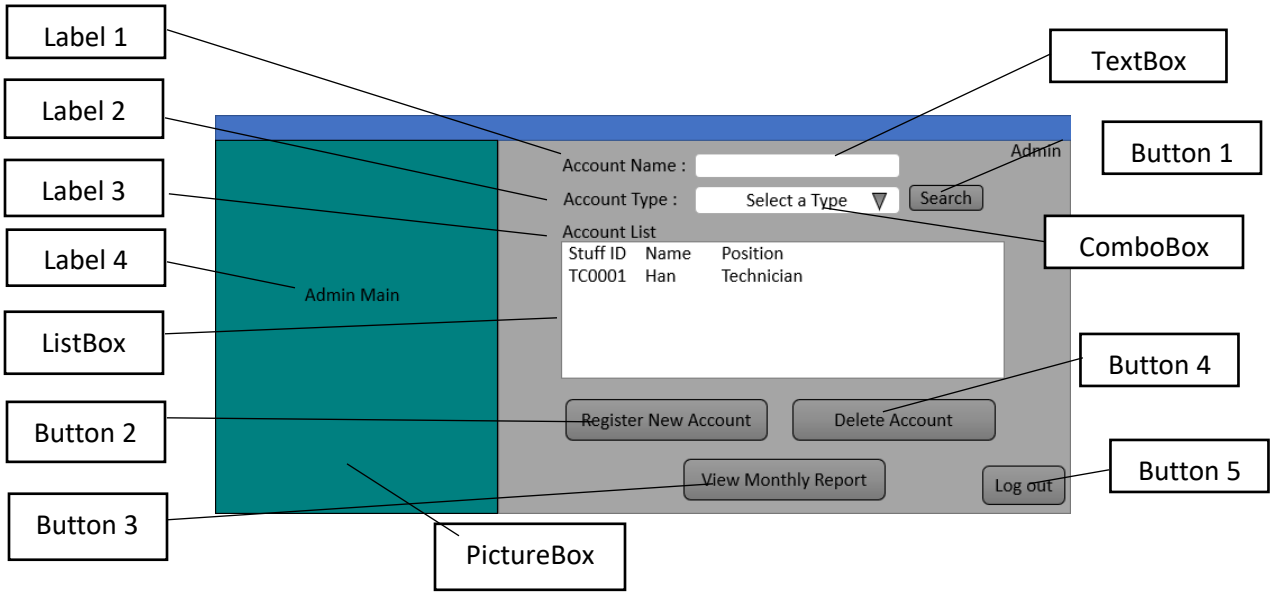
Group Members:

1. **Hasan Akram Abdullah Zaid – TP066635**
2. **Chai Chean Han - TP064801**
3. **Abdallah Zakaria Elkhatal - TP064837**

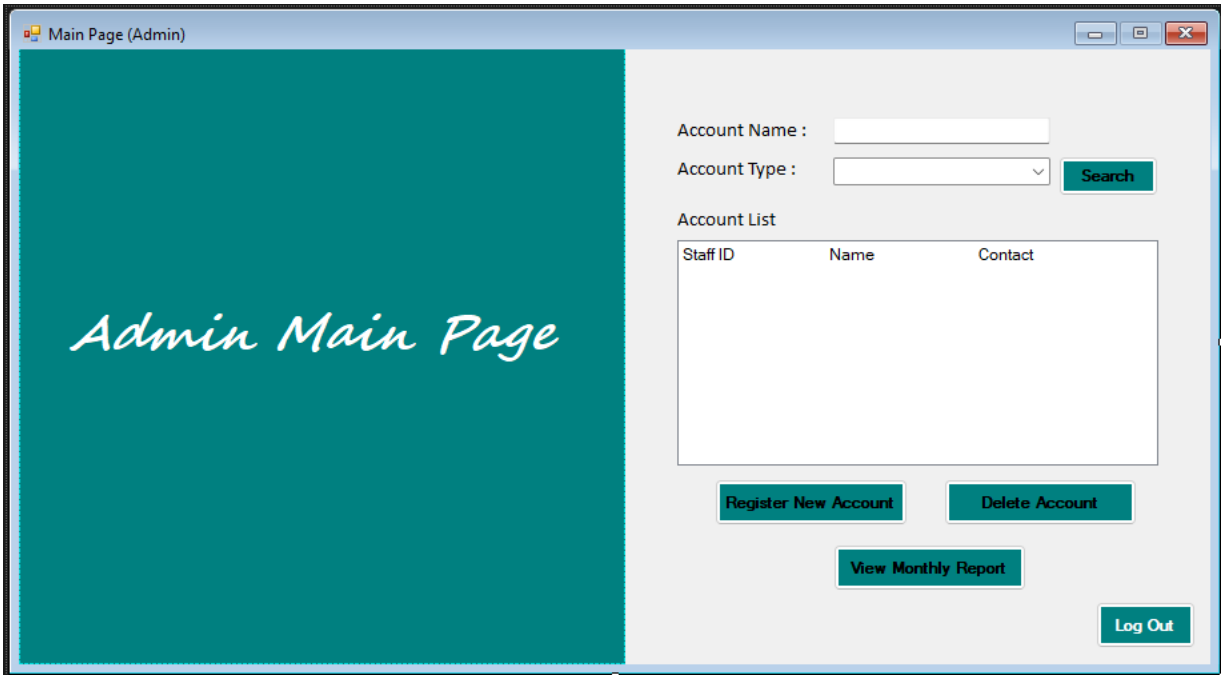
Table of Contents

Storyboard

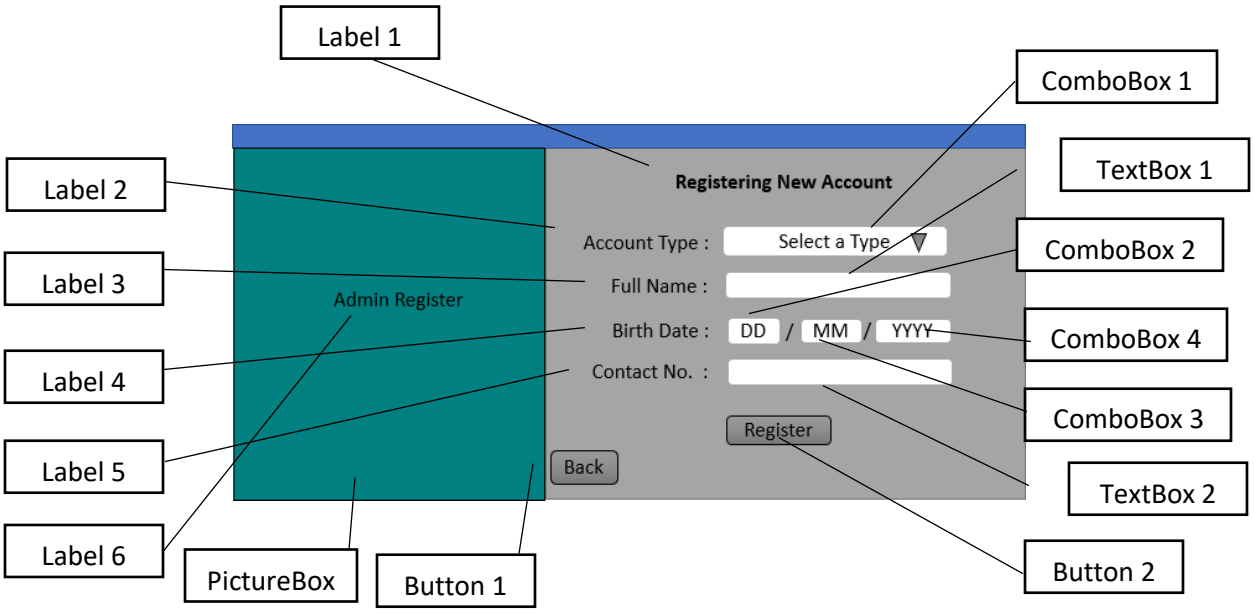
AdminMain Form



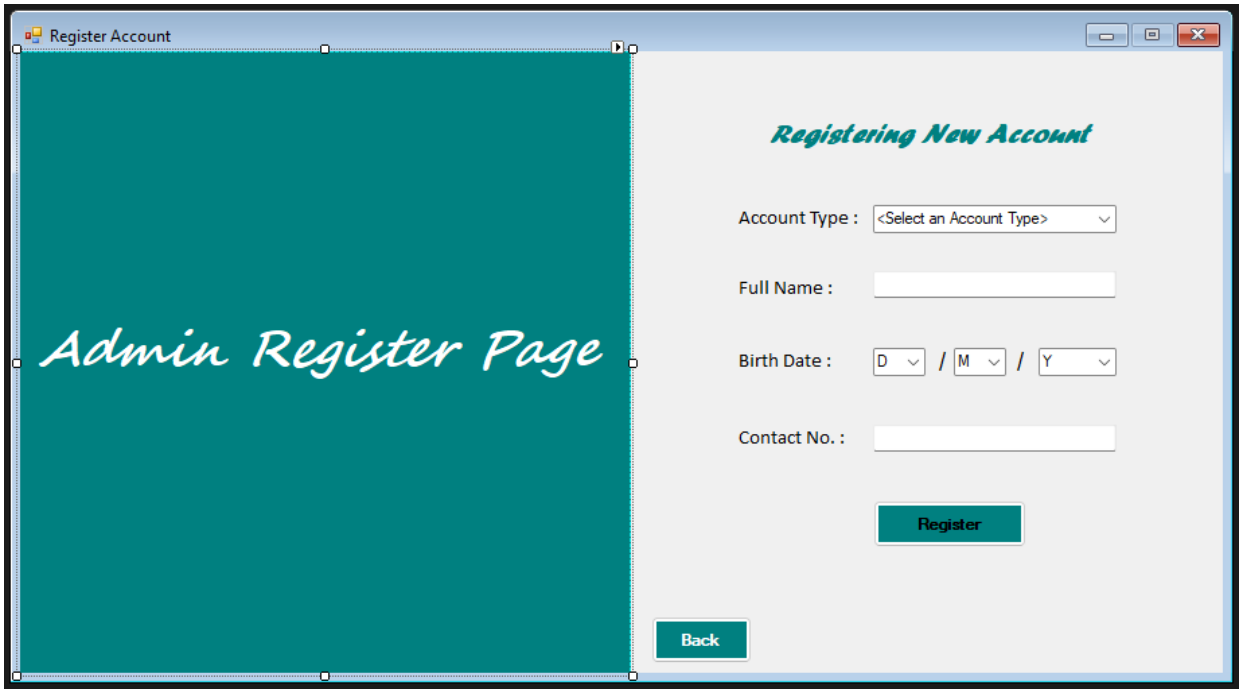
Control	Control Name	Description
Label 1	label1	To label the related controls
Label 2	label2	
Label 3	label3	
Label 4	label4	Show what page currently is
PictureBox	pictureBox1	Decoration
ListBox	listAcc	Show the list of Account
TextBox	txtSchAccName	Allow entering Name search (Optional)
ComboBox	cboxSchAccType	Choose the type of account searching
Button 1	btnSearch	Enable searching account needed
Button 2	btnRegisterAcc	Enable registering new account
Button 3	btnViewRep	Enable viewing monthly report
Button 4	btnDeleteAcc	Enable deleting account selected in list
Button 5	btnLogout	Enable logging out


Main Page (Admin)
Admin Main Page
Account Name :
Account Type :
Search
Account List
Staff ID
Name
Contact
Register New Account
Delete Account
View Monthly Report
Log Out

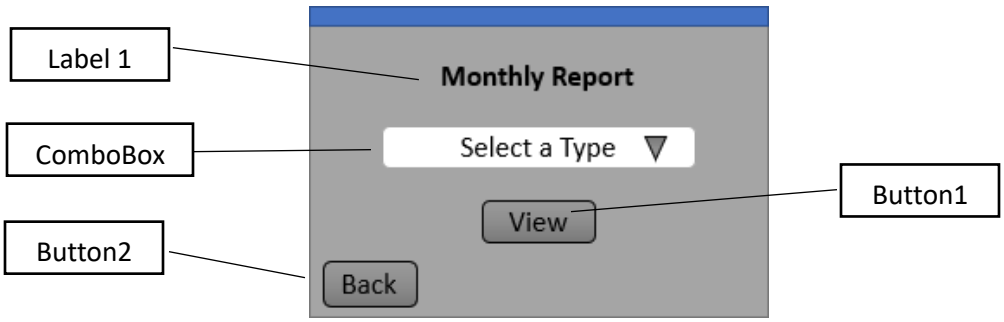
AdminRegister Form



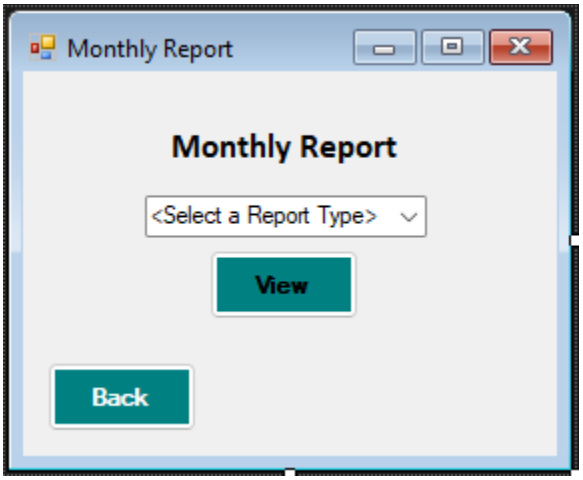
Control	Control Name	Description
Label 1	label1	To label the related controls
Label 2	label2	
Label 3	label3	
Label 4	label4	
Label 5	label5	
Label 6	label6	To show what page currently is
PictureBox	pictureBox1	Decoration
TextBox 1	txtRegName	Allow entering user's name
TextBox 2	txtRegContact	Allow entering user's contact
ComboBox 1	cboxRegAccType	Allow selecting type of account
ComboBox 2	cboxRegBDay	Allow selecting day of birth
ComboBox 3	cboxRegBMonth	Allow selecting month of birth
ComboBox 4	cboxRegBYear	Allow selecting year of birth
Button 1	btnBack	Enable returning to main page
Button 2	btnRegister	Enable registering account


Register Account
Admin Register Page
Registering New Account
Account Type :
<Select an Account Type>
Full Name :
Birth Date :
D
/
M
/
Y
Contact No. :
Register
Back

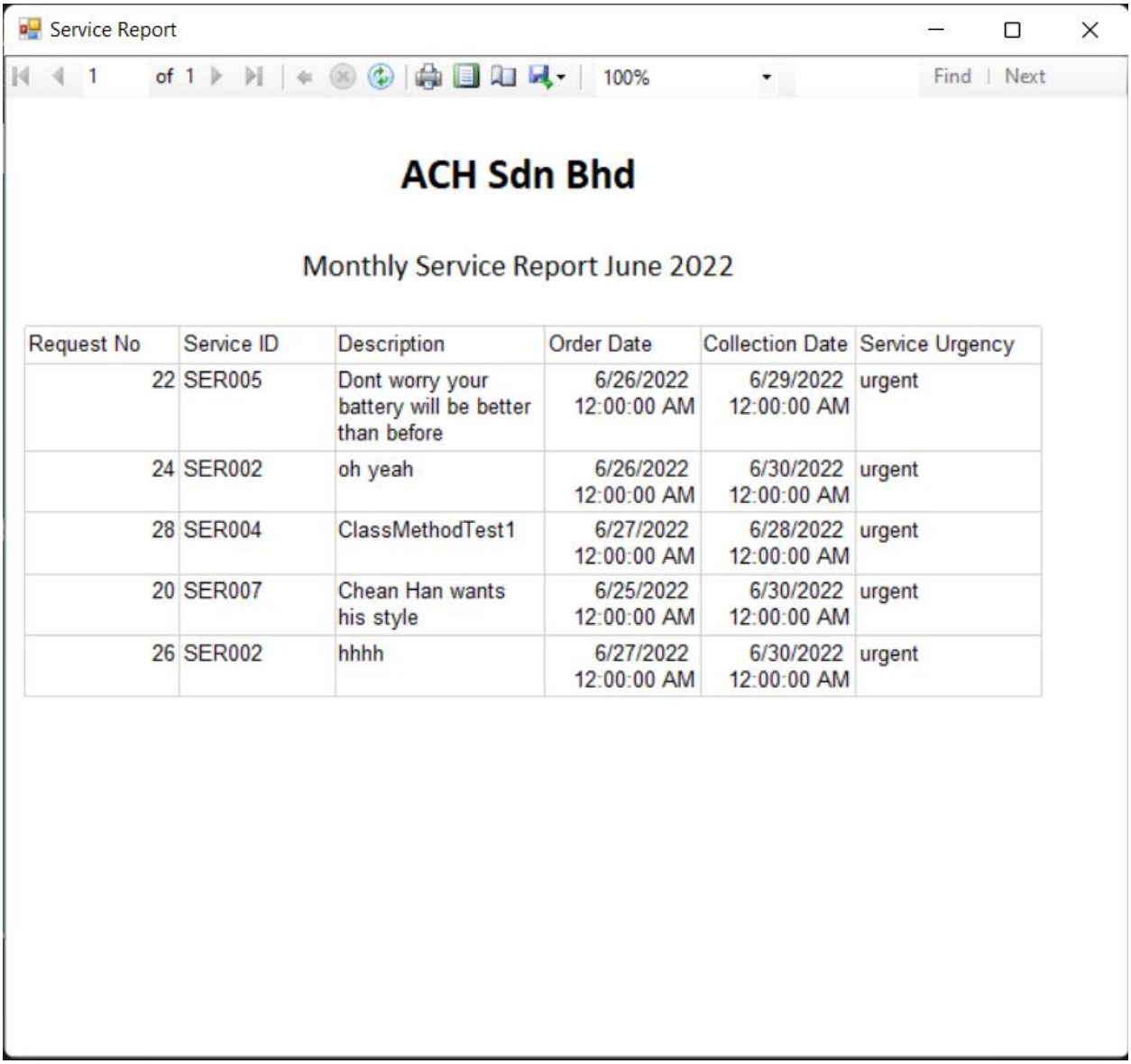
AdminReport Form



Control	Control Name	Description
Label	label1	To label the related controls
ComboBox	cboxReportType	Enable selecting type of report needed
Button 1	btnView	Enable viewing report
Button 2	btnBack	Enable going back to main page

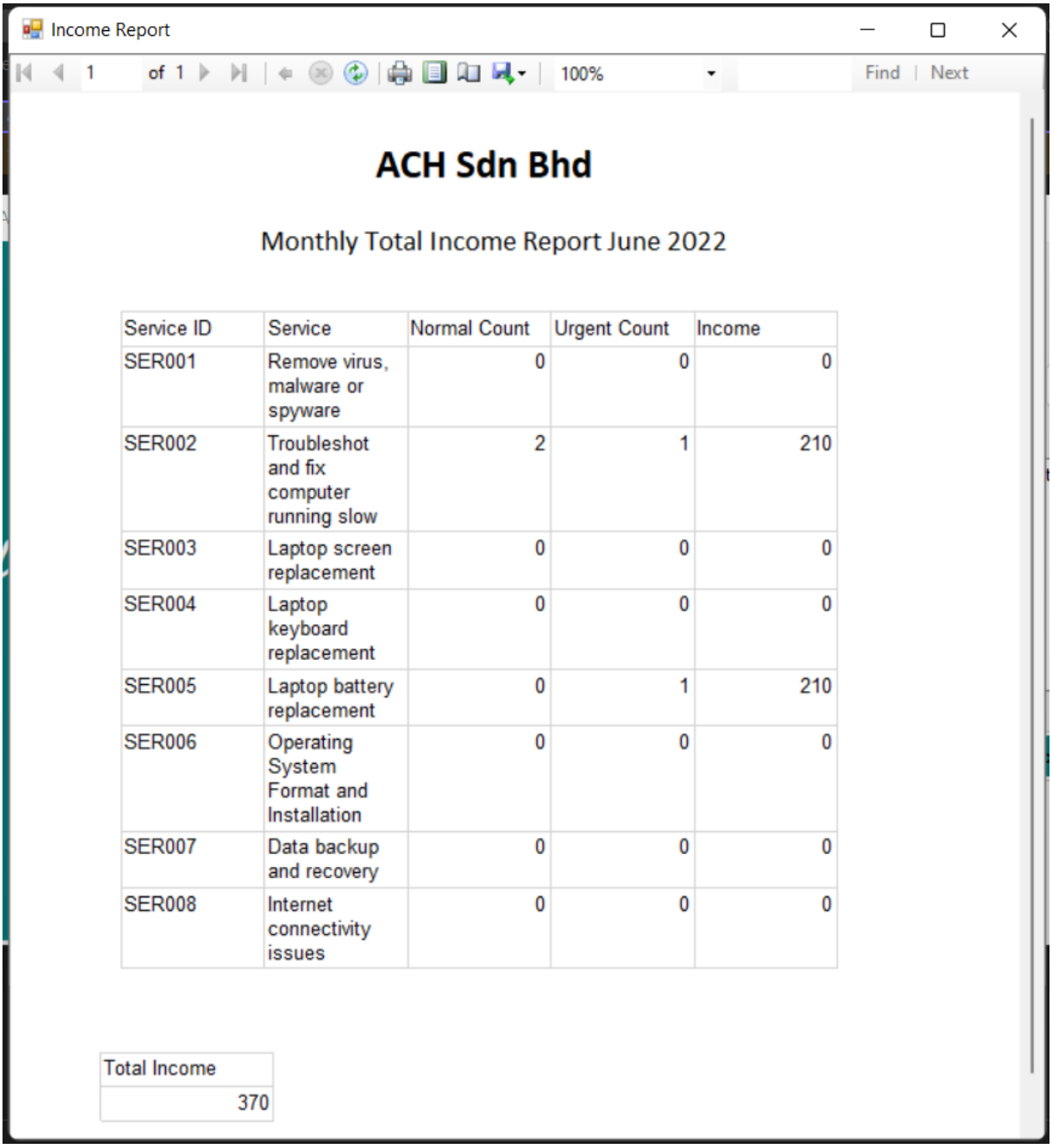


ServiceRepForm Form



Request No	Service ID	Description	Order Date	Collection Date	Service Urgency
22	SER005	Dont worry your battery will be better than before	6/26/2022 12:00:00 AM	6/29/2022 12:00:00 AM	urgent
24	SER002	oh yeah	6/26/2022 12:00:00 AM	6/30/2022 12:00:00 AM	urgent
28	SER004	ClassMethodTest1	6/27/2022 12:00:00 AM	6/28/2022 12:00:00 AM	urgent
20	SER007	Chean Han wants his style	6/25/2022 12:00:00 AM	6/30/2022 12:00:00 AM	urgent
26	SER002	hhhh	6/27/2022 12:00:00 AM	6/30/2022 12:00:00 AM	urgent

IncomeRepForm Form



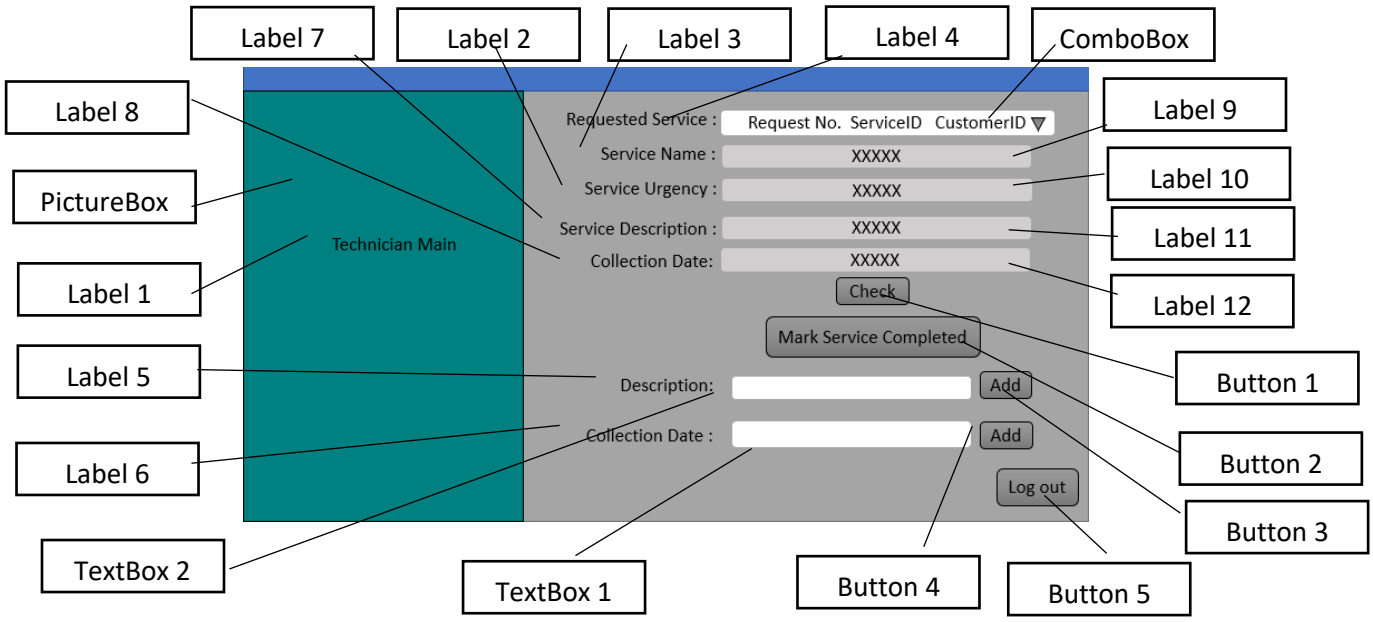
ACH Sdn Bhd

Monthly Total Income Report June 2022

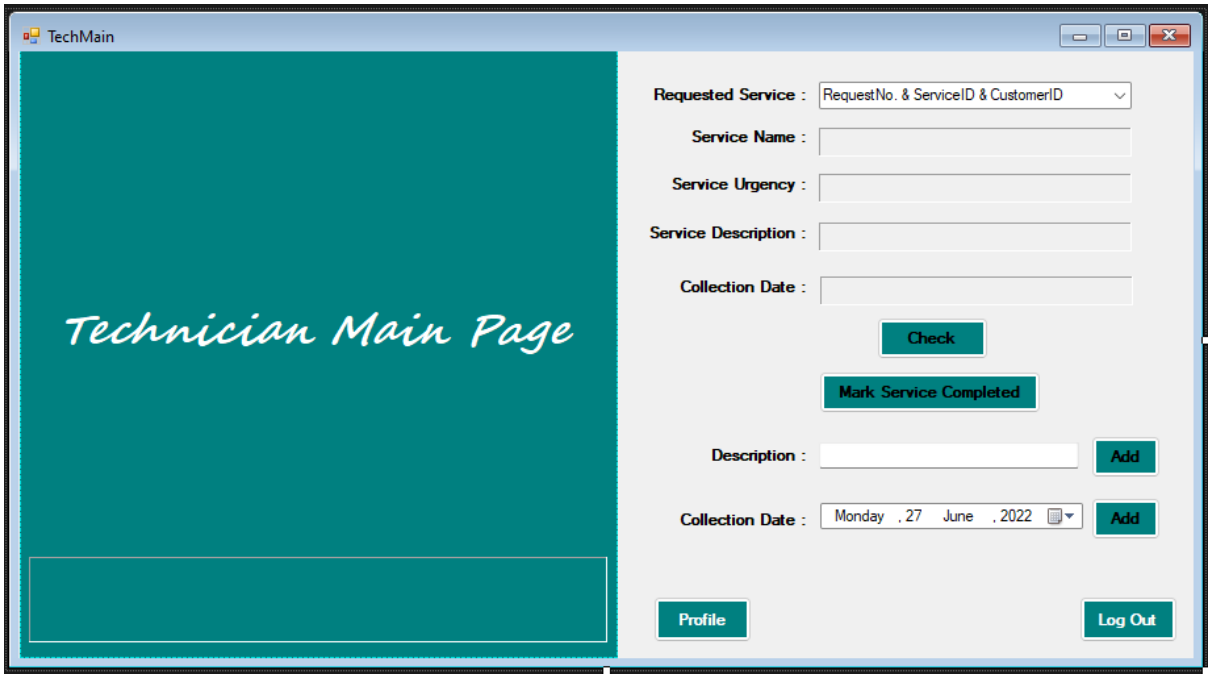
Service ID	Service	Normal Count	Urgent Count	Income
SER001	Remove virus, malware or spyware	0	0	0
SER002	Troubleshot and fix computer running slow	2	1	210
SER003	Laptop screen replacement	0	0	0
SER004	Laptop keyboard replacement	0	0	0
SER005	Laptop battery replacement	0	1	210
SER006	Operating System Format and Installation	0	0	0
SER007	Data backup and recovery	0	0	0
SER008	Internet connectivity issues	0	0	0

Total Income
370

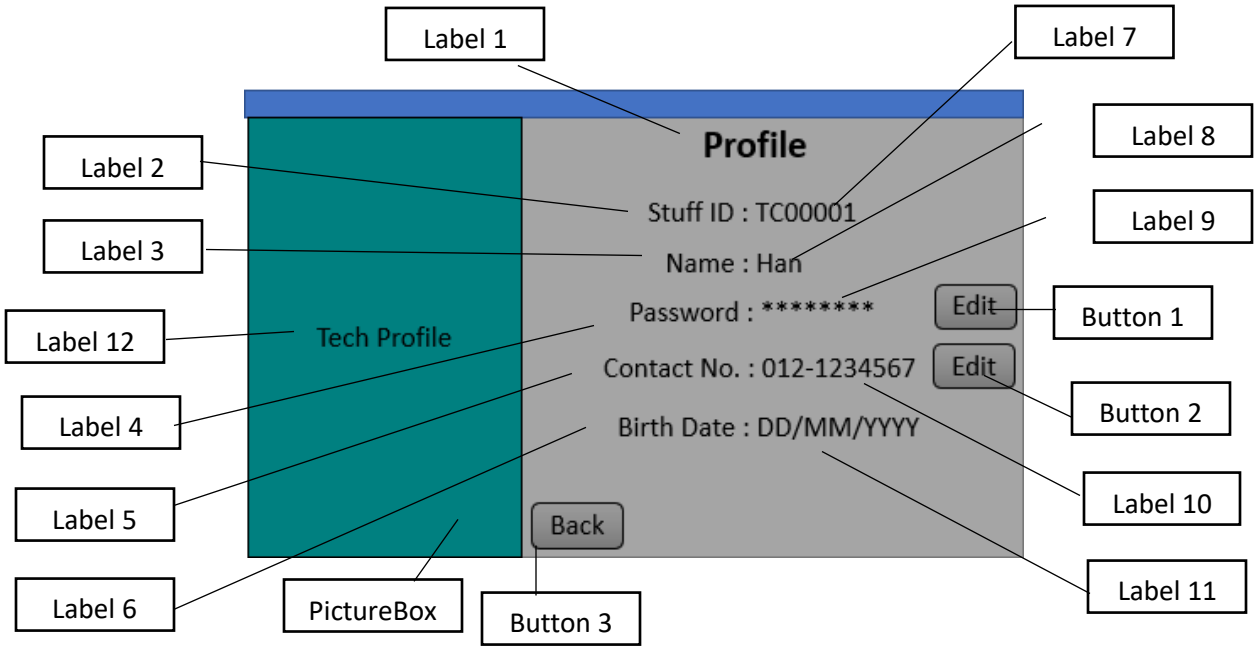
TechMain Form



Control	Control Name	Description
Label 1	label1	To label the related controls
Label 2	label2	
Label 3	label3	
Label 4	label4	
Label 5	label5	
Label 6	label6	
Label 7	label7	
Label 8	label8	
Label 9	lblSName	To show service name
Label 10	lblUrgency	To show service urgency
Label 11	lblDescription	To show service description
Label 12	lblCollectDate	To show collection date
PictureBox	pictureBox1	Decoration
TextBox 1	txtColDate	To input the collection date
TextBox 2	txtDesc	To input the description
ComboBox	cboxService	Allow selecting service
Button 1	btnCheck	Check the information of service
Button 2	btnMarkCmp	Mark the completed service
Button 3	btnAddDesc	To add the description
Button 4	btnAddColDate	To add the collection date
Button 5	btnLogOut	To log out

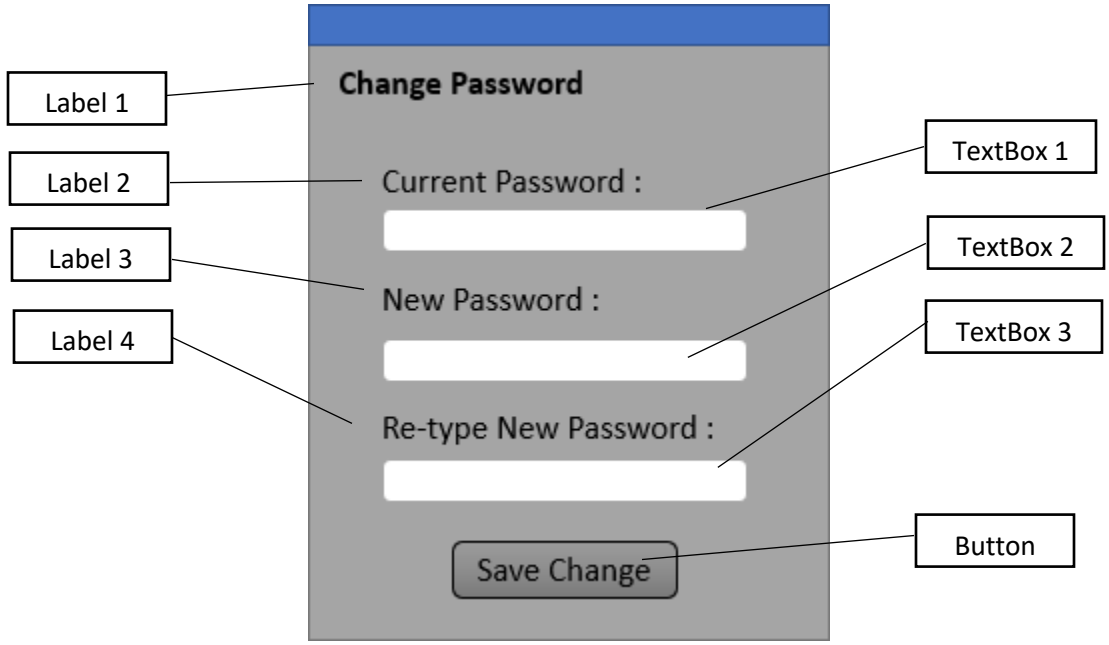

TechMain
Technician Main Page
Requested Service :
RequestNo. & ServiceID & CustomerID
Service Name :
Service Urgency :
Service Description :
Collection Date :
Check
Mark Service Completed
Description :
Add
Collection Date :
Monday , 27 June , 2022
Add
Profile
Log Out

TechProfile Form



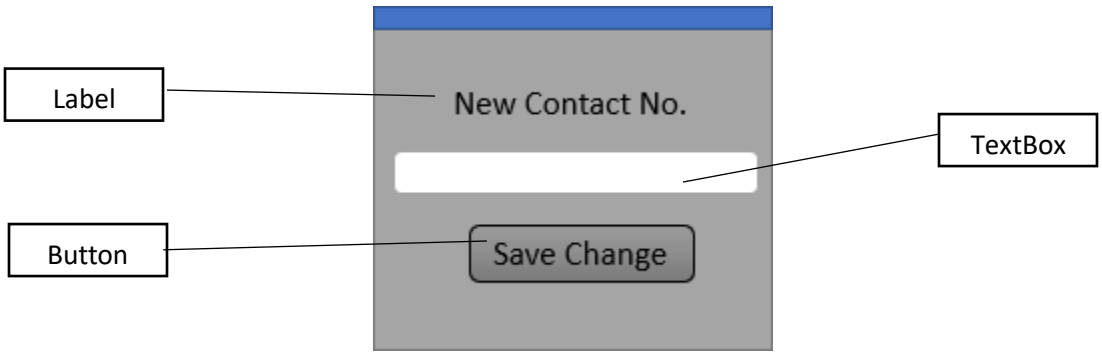
Control	Control Name	Description
Label 1	label1	To label the related controls
Label 2	label2	
Label 3	lable3	
Label 4	label4	
Label 5	label5	
Label 6	label6	
Label 7	lblStaffID	To show the information of stuff
Label 8	lblName	
Label 9	lblPassword	
Label 10	lblContact	
Label 11	lblBDate	
Label 12	label12	To show what page currently is
PictureBox	pictureBox1	Decoration
Button 1	btnEditPwd	Enable editing password
Button 2	btnEditContact	Enable editing contact
Button 3	btnBack	Enable returning to main page

TechChgPwd Form

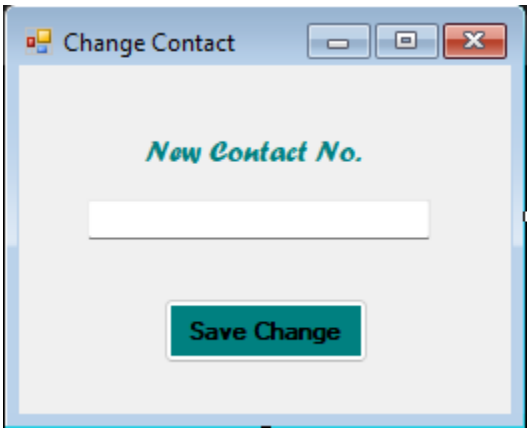


Control	Control Name	Description
Label 1	Label 1	To label the related controls
Label 2	Label 2	
Label 3	Label 3	
Label 4	Label 4	
TextBox 1	txtCurrentPw	Allow entering current password
TextBox 2	txtNewPw1	Allow entering new password
TextBox 3	txtNewPw2	Allow re-entering new password
Button	btnSavePw	Enable saving new password

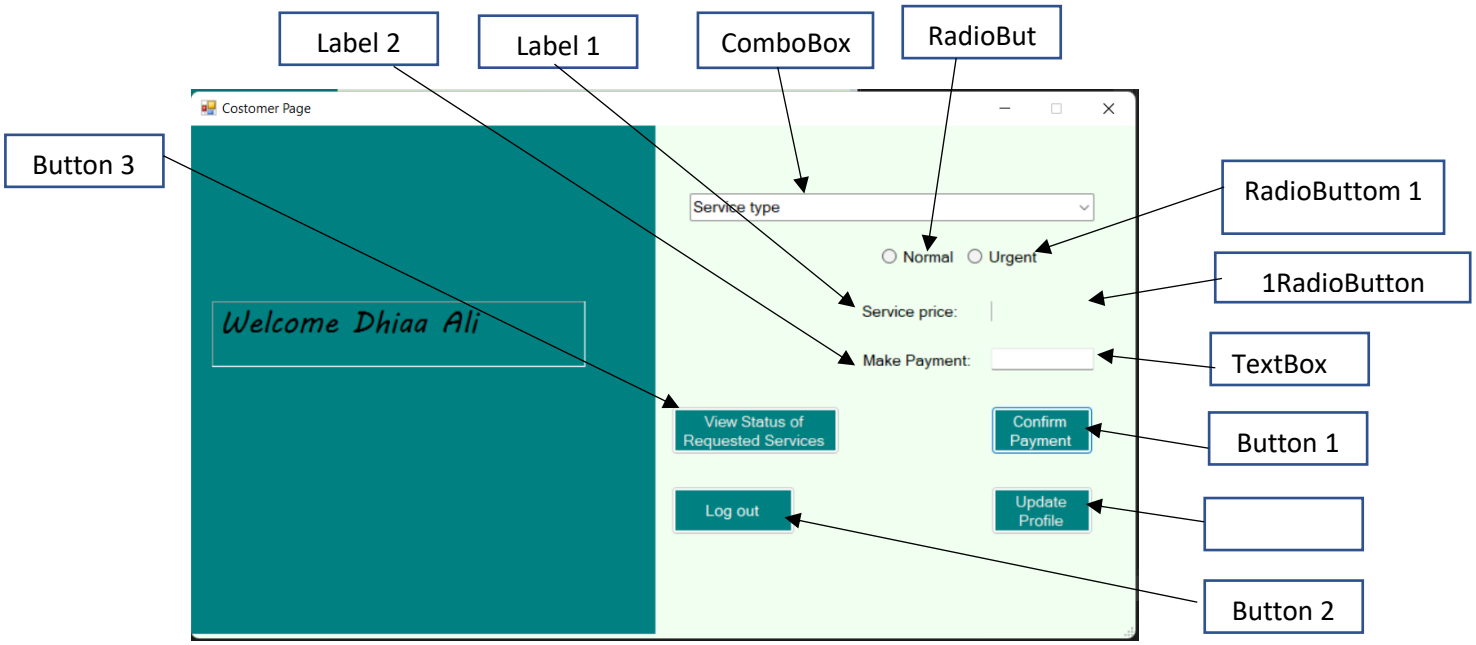
TechChgContact Form



Control	Control Name	Description
Label	label1	To label the related controls
TextBox	txtNewContact	Allow entering new contact
Button	txtSaveContact	Enable saving new contact



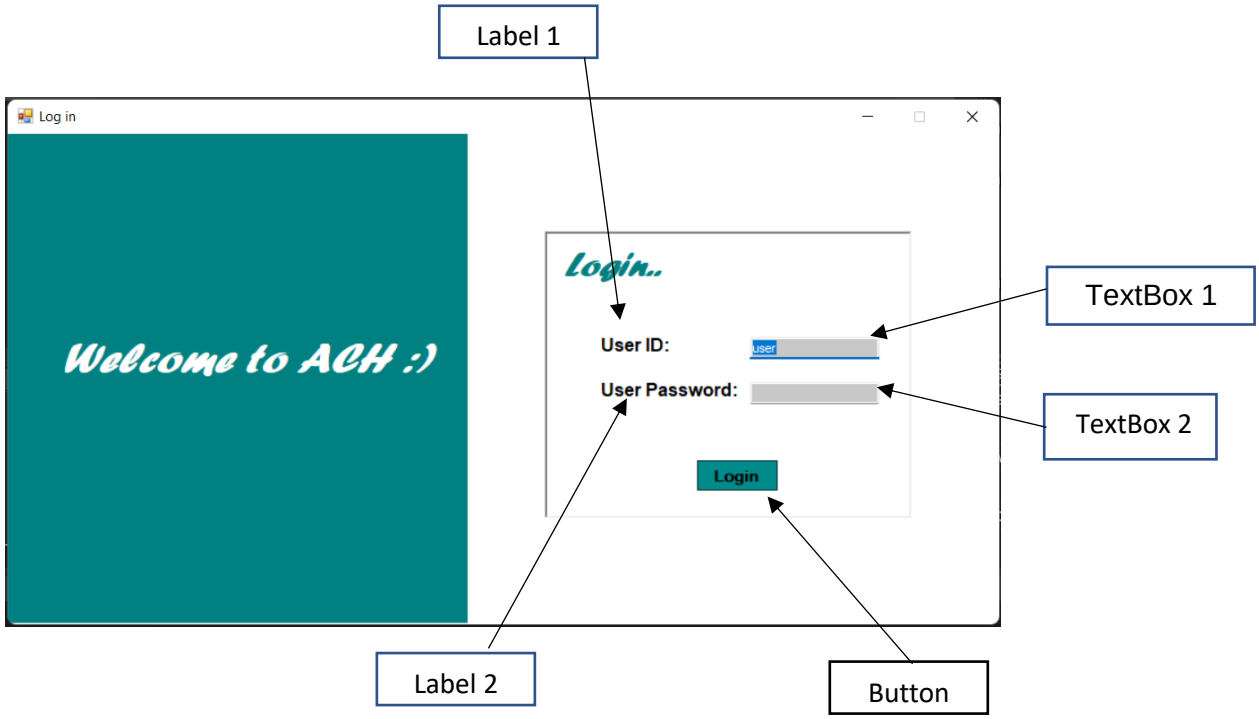
Customer_Main Form



Control	Control Name	Description
Label 1	Label 1	To label the related controls to the right
Label 2	Label 2	To label the related controls to the right
ComboBox	cboProduct	To allow selection of service from a list
TextBox	txtPayment	To allow entering the payment
Button 1	btnConfirm	To enable confirm payment
Button 2	btnLogout	To enable logout
Button 3	Button 3	To enable view status of requested services

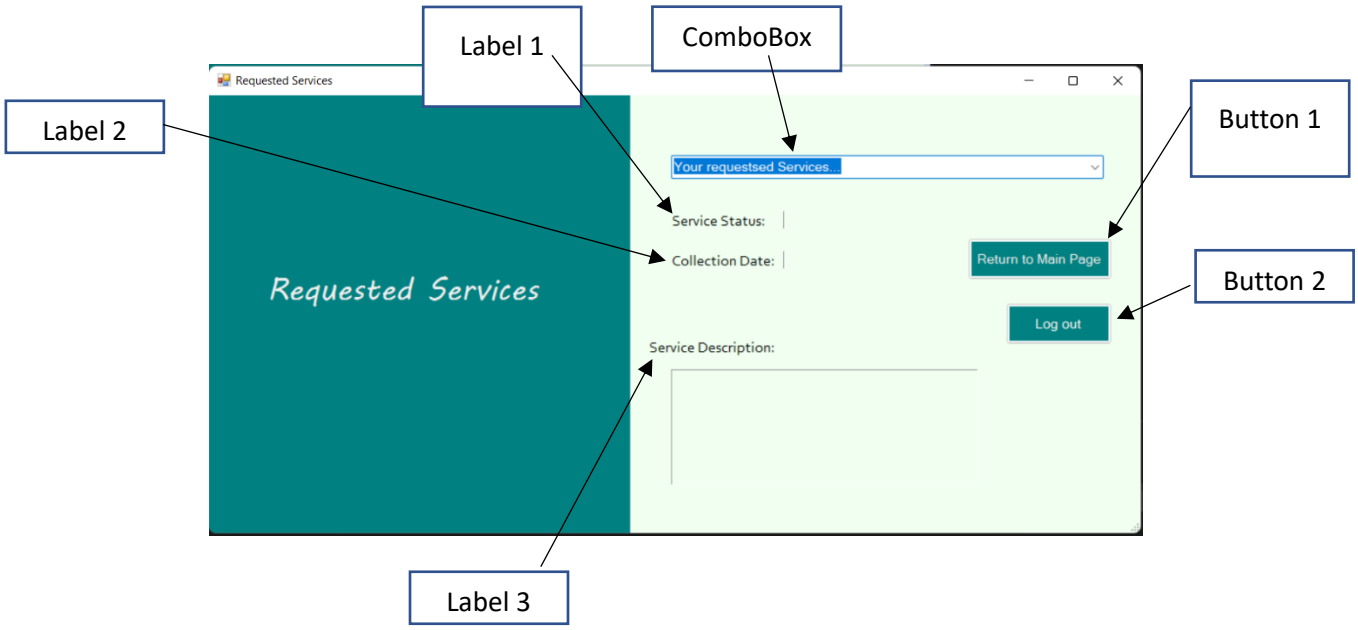
Button 4	Button 4	To enable update profile
RadioButton 1	RadioButton 1	To display urgent prices
RadioButton 2	RadioButton 2	To display normal prices

LoginForm



Control	Control Name	Description
Label 1	Label 1	To label the related controls to the right
Label 2	Label 2	To label the related controls to the right
TextBox 1	TextBox 1	To allow entering user ID
TextBox 2	TextBox 2	To allow entering user password
Botton 1	bntLogin	To enable login account

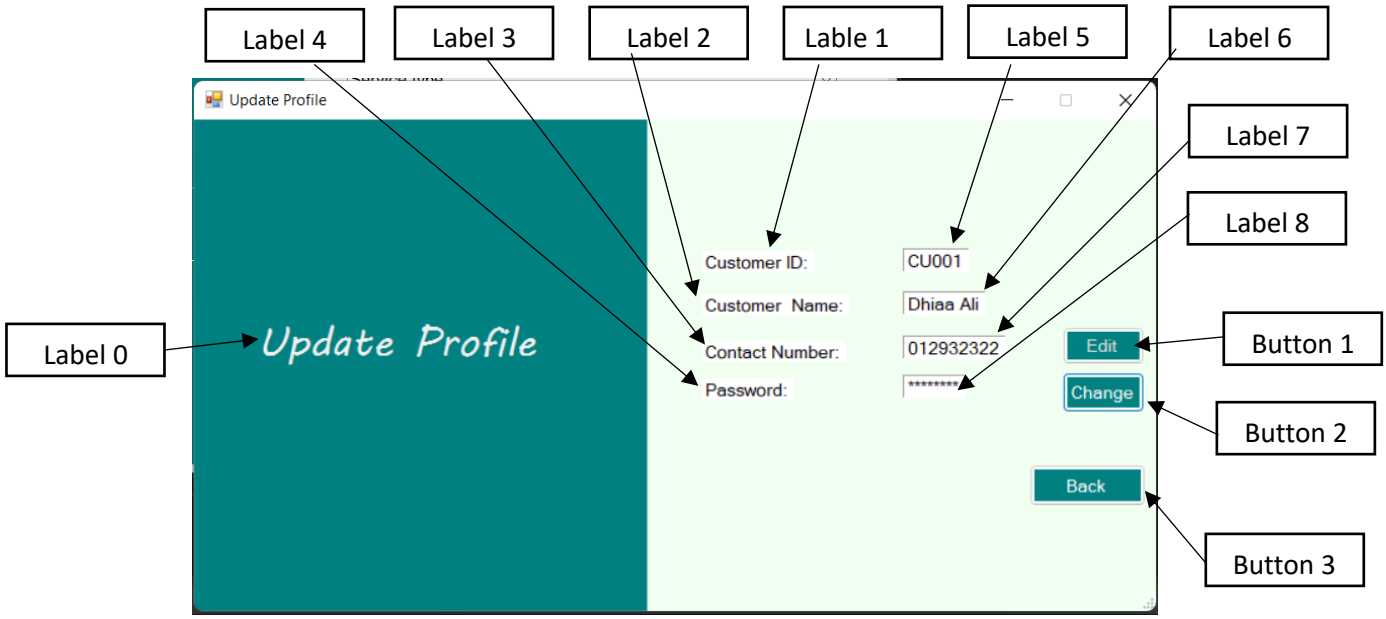
CustomersRequetedServices Form



Control	Control Name	Description
Label 1	Label 1	To label the related controls to the right
Label 2	Label 2	To label the related controls to the right
Label 3	Label 3	To label the related controls to the right
ComboBox	ComboBox	To allow selection of requested services from a list
Button 1	Button 1	To enable return to main form

Button 2	Button 2	To enable log out

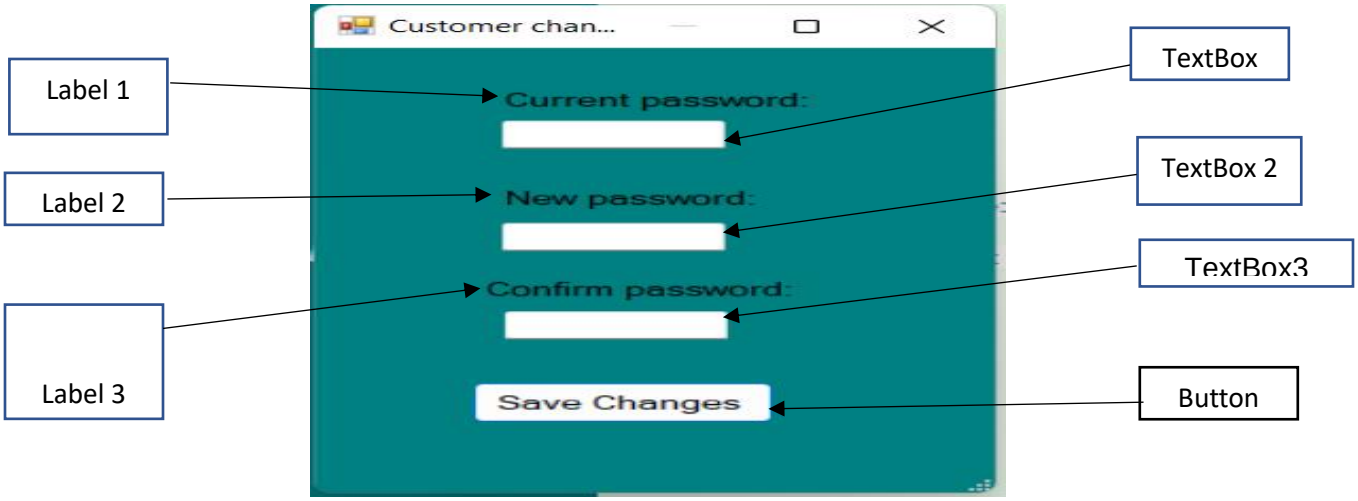
Cus_Update Form



Control	Control Name	Description
Label 1	Label 1	To label the related controls to the right
Label 2	Label 2	
Label 3	Label 3	
Label 4	Label 4	To label the related controls to the right
Label 5	LblID	To display the customer ID
Label 6	lablName	To display the customer name
Label 7	lblContactNumber	To display the customer contact number
Label 8	lablPasswoed	To display the customer password

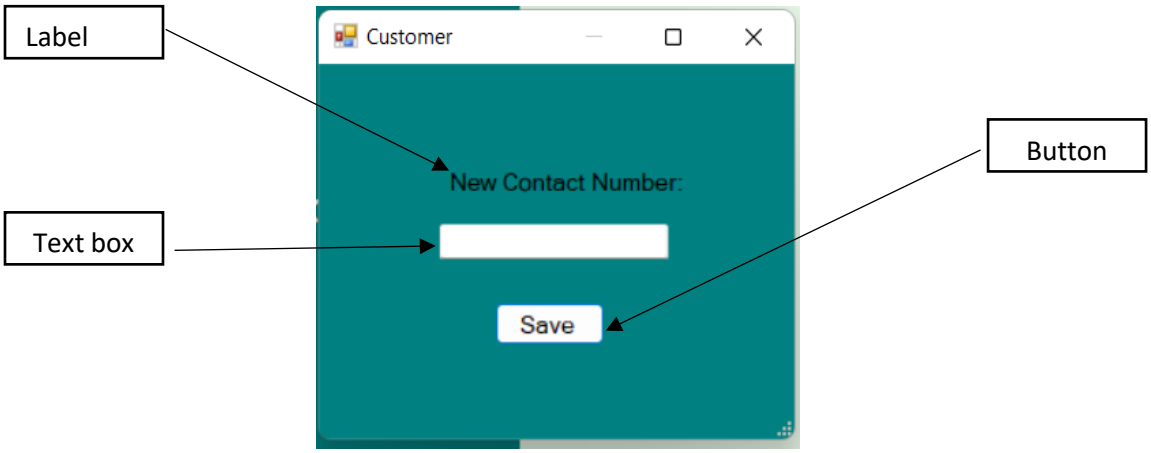
Button 1	bntEdit	To enable Edit contact number
Button 2	bntChange	To enable change customer password
Button 3	bntSaveChanges	To enable save changes

Cus_change_Password Form



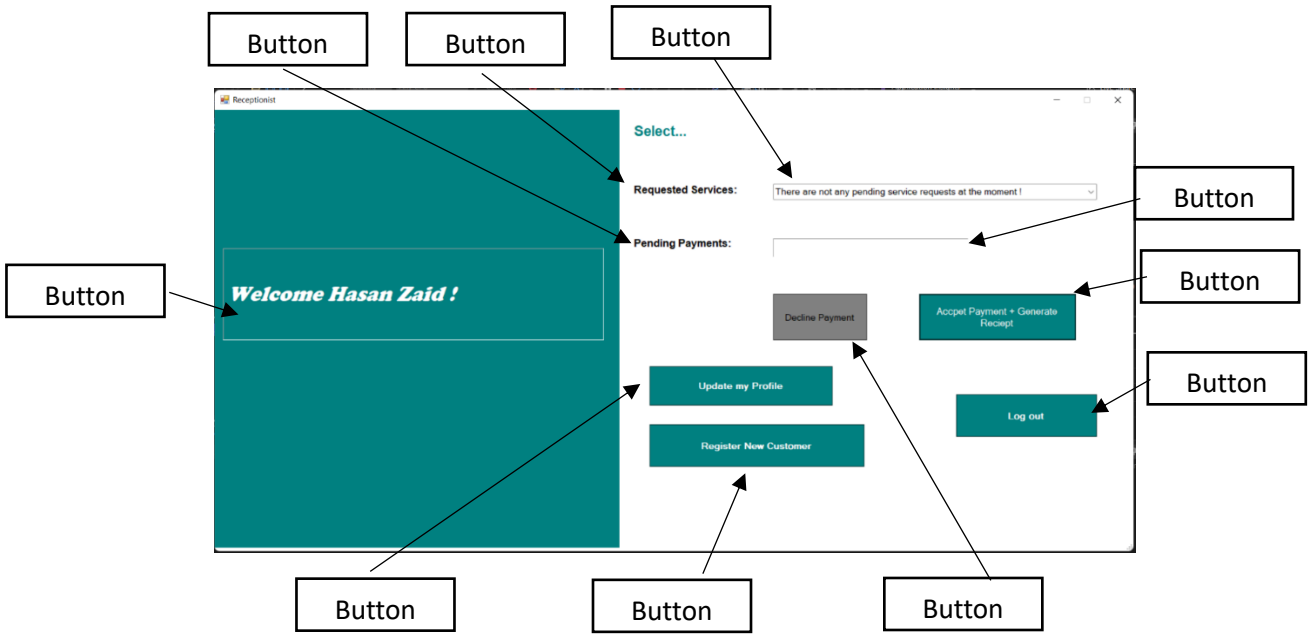
Control	Control Name	Description
Label 1	lblCurrentPassword	To label the related controls
Label 2	lblNewPassword	To label the related controls
Label 3	lblConfirmPassword	To label the related controls
TextBox 1	txtCurrentPassword	Allow entering current password
TextBox 2	txtNewPassword	Allow entering new password
TextBox 3	txtConfirmPassword	Allow re-entering new password
Botton 1	bntSaveChange	Enable saving new password

Cus_change_Number Form



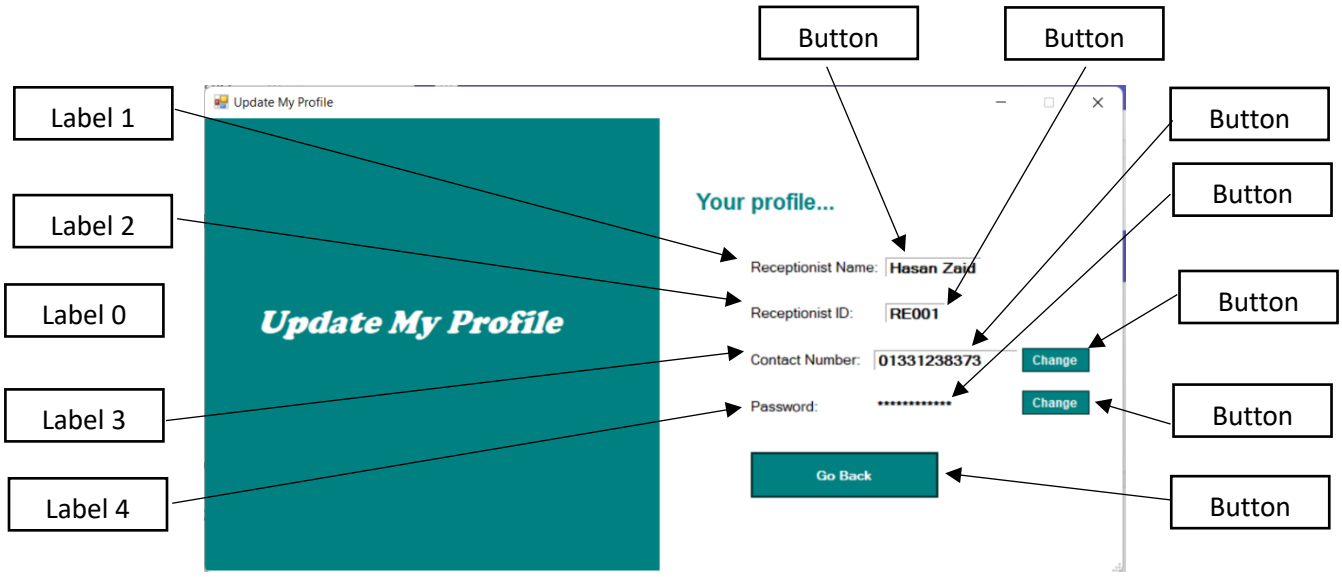
Control	Control Name	Description
Label	Label 1	To label the related controls
TextBox	txtNewContact	Allow entering new contact
Button	bntSave	Enable saving new contact

ReceptionistForm



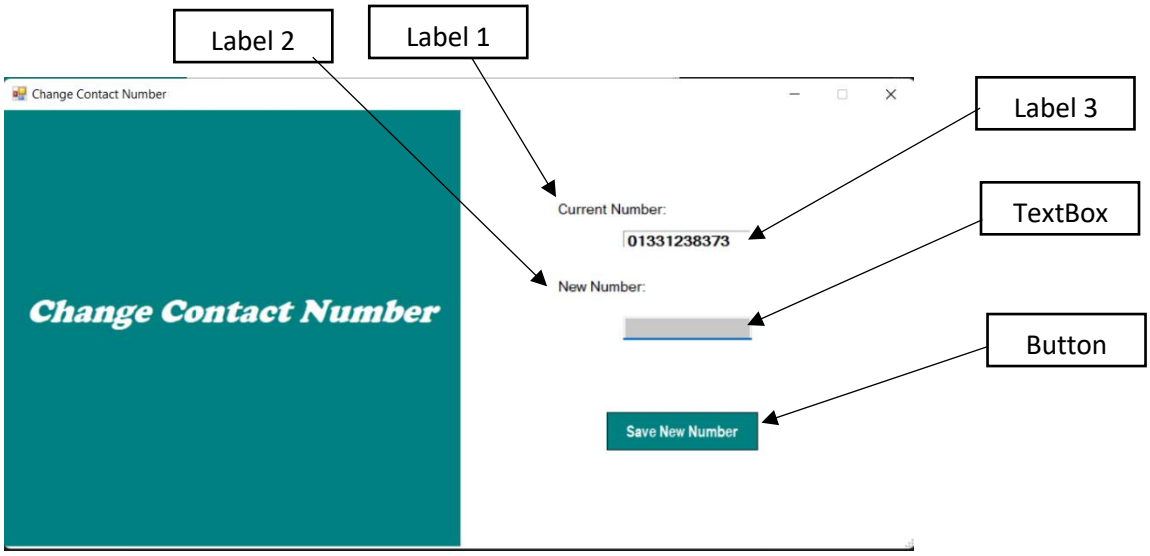
Control	Control Name	Description
Label 1	Label 1	To label the related controls to the right
ComboBox	cboProduct	To allow selection of service from a list
TextBox	txtPayment	To allow entering pending payment
Button 1	btnConfirm	To enable confirm order
Button 2	btnLogout	To enable logout
Button 3	btnUpdateProfile	To open update profile form
Button 4	BtnRegister	To open register customer form

RecUpdateProfileForm



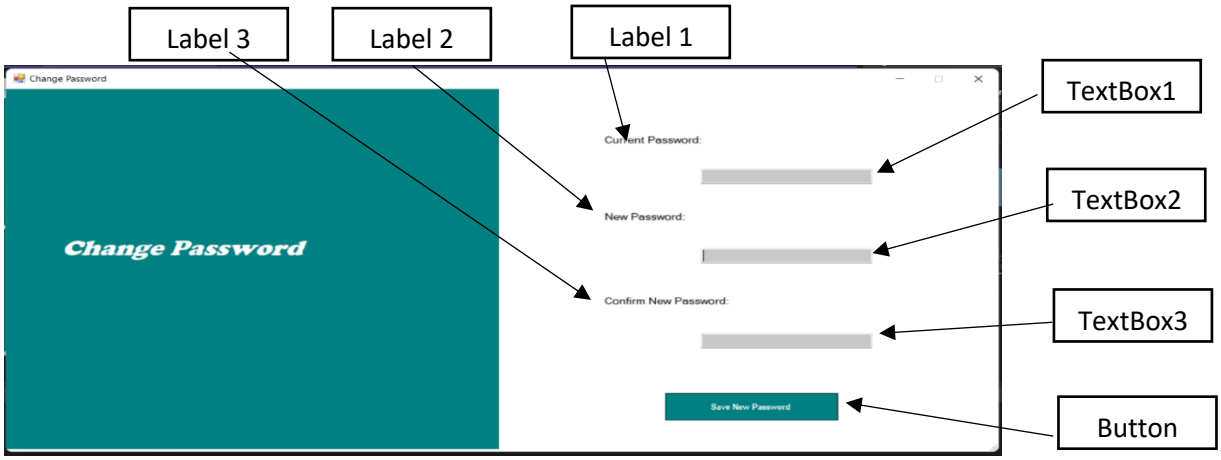
Control	Control Name	Description
Label 1	Label 1	To label the related controls to the right
Label 2	Label 2	To label the related controls to the right
Label 3	Label 3	
Label 4	Label 4	
Label 5	receptionistNameLbl	To display the receptionist ID
Label 6	receptionistIDLbl	To display the receptionist name
Label 7	receptionistContactNumberLbl	To display the receptionist contact number
Label 8	receptionistPasswordLbl	To display the receptionist password
Button 1	changeContactNumberBtn	To enable Edit contact number
Button 2	changePasswordBtn	To enable change receptionist password
Button 3	saveChangesBtn	To enable save changes

Rec_Change_Number Form



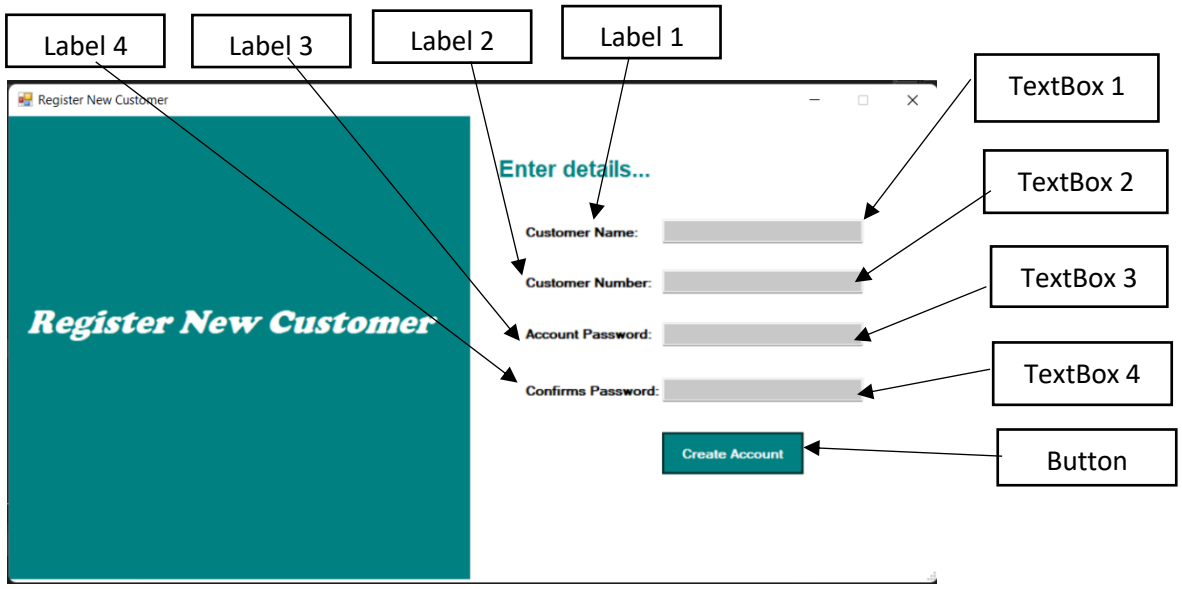
Control	Control Name	Description
Label 1	Label 1	To label the related controls
Label 2	Label 2	
Label 3	currentNumberLbl	Show current contact number
TextBox	newNumberTextBox	Allow entering new contact
Button	saveNewNumberBtn	Enable saving new contact

Rec_Change_Password Form



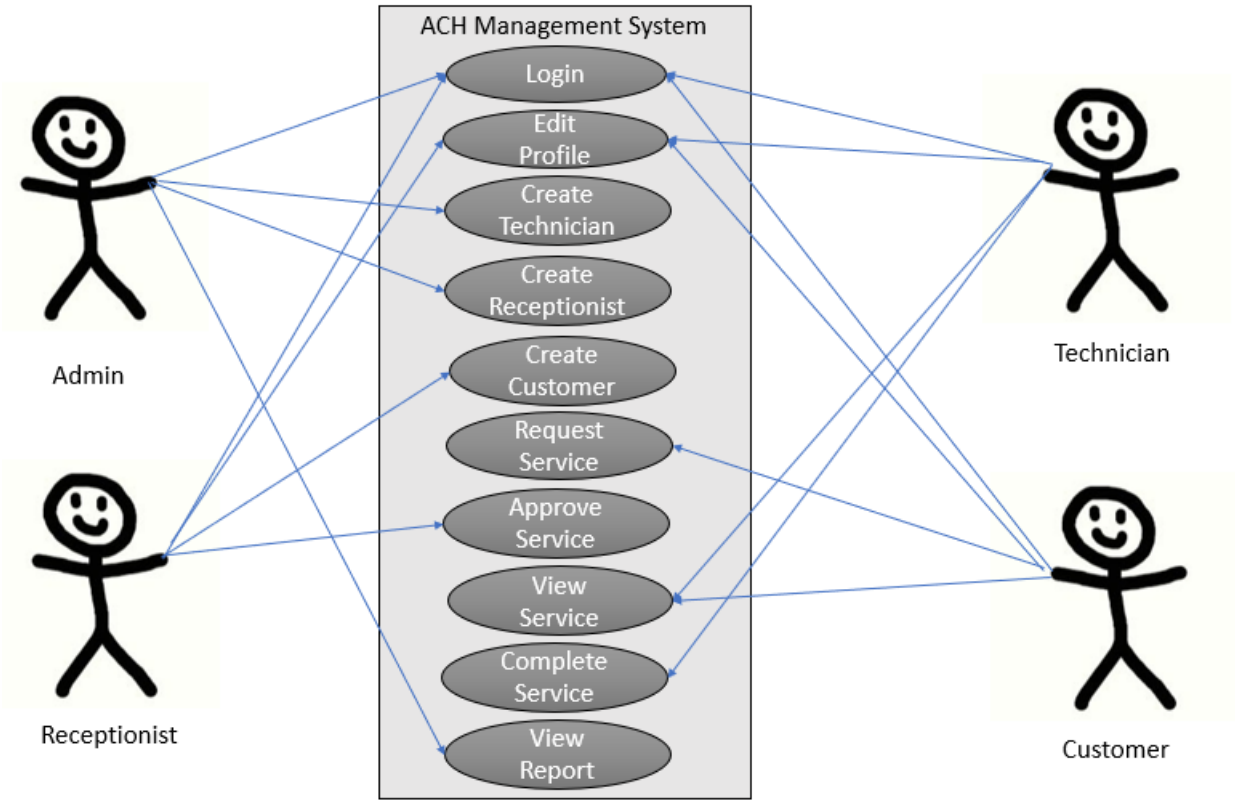
Control	Control Name	Description
Label 1	lblCurrentPassword	To label the related controls
Label 2	lblNewPassword	
Label 3	lblS	
TextBox 1	currentPasswordTextBox	Allow entering current password
TextBox 2	newPasswordTextBox	Allow entering new password
TextBox 3	confirmNewPasswordTextBox	Allow re-entering new password
Button	saveNewPasswordBtn	Enable saving new password

RegisterCustomersForm

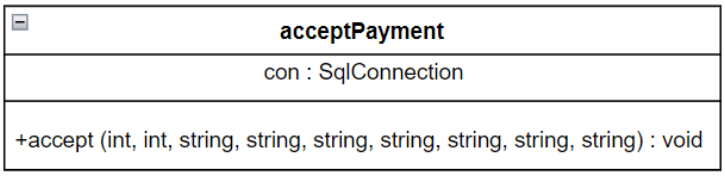


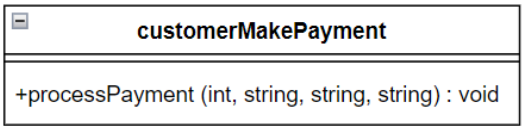
Control	Control Name	Description
Label 1	Label 1	To label the related controls
Label 2	Label 2	To label the related controls
Label 3	Label 3	To label the related controls
Label 4	Label 4	To label the related controls
TextBox 1	customerNameTextBox	Allow entering customers name
TextBox 2	customerNumberTextBox	Allow entering customers number
TextBox 3	accountPasswordTextBox	Allow entering new password
TextBox 4	confirmPasswordTextBox	Allow re-entering new password
Button	createAccountBtn	Enable saving new account

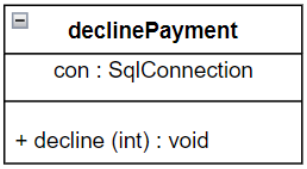
Use-case Diagram:

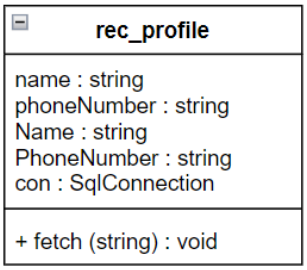


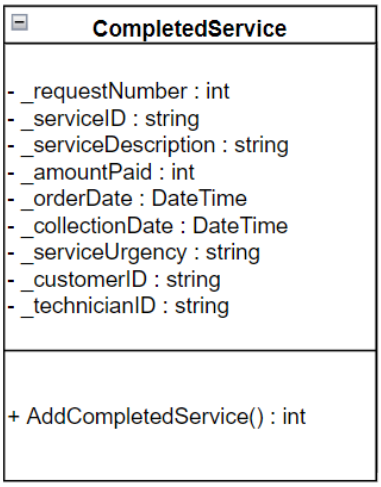
Class Diagram:

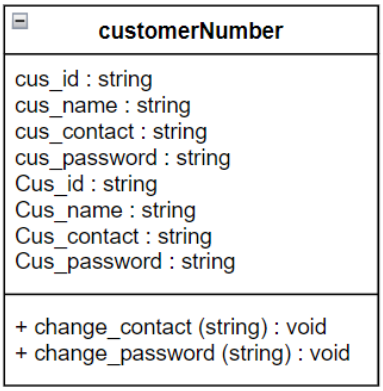


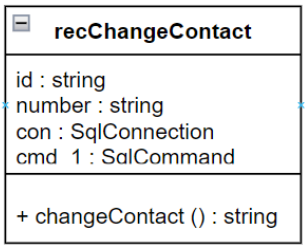


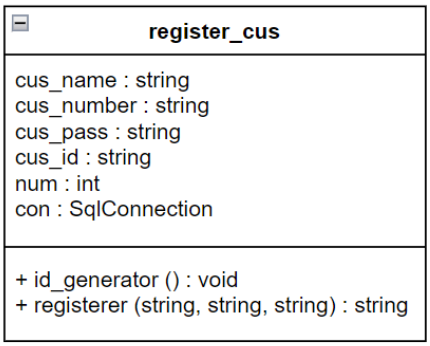


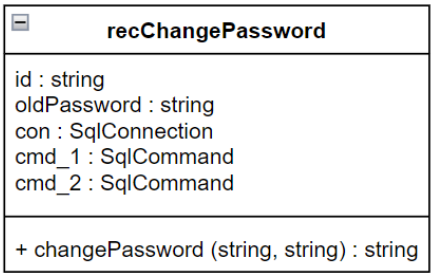


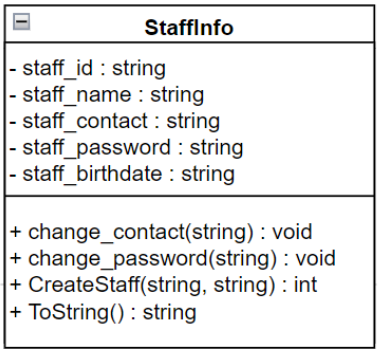





recChangeContact
id : string
number : string
con : SqlConnection
cmd_1 : SqlCommand
+ changeContact () : string


register_cus
cus_name : string
cus_number : string
cus_pass : string
cus_id : string
num : int
con : SqlConnection
+ id_generator () : void
+ registerer (string, string, string) : string


recChangePassword
id : string
oldPassword : string
con : SqlConnection
cmd_1 : SqlCommand
cmd_2 : SqlCommand
+ changePassword (string, string) : string


StaffInfo
- staff_id : string
- staff_name : string
- staff_contact : string
- staff_password : string
- staff_birthdate : string
+ change_contact(string) : void
+ change_password(string) : void
+ CreateStaff(string, string) : int
+ ToString() : string

Explanation of Codes:

Database Properties

Name: “ACH-DB”

Number of Table: 11 Tables

Names of Tables:

- admins
- completed_services
- customers
- income_report
- receipts
- receptionists
- requested_services
- services
- technicians
- Total_Income
- users_login

Table used for login: “users_login”

Login as Customer:

- ID: CU001
- Password: customer

Login as Admin:

- ID: AD001
- Password: admin

Login as Technician:

- ID: TC001
- Password: technician

Login as Receptionist:

- ID: RE001
- Password: receptionist

CompletedService Class

```
private int _requestNumber;
private string _serviceID;
private string _serviceDescription;
private int _amountPaid;
private DateTime _orderDate;
private DateTime _collectionDate;
private string _serviceUrgency;
private string _customerID;
private string _technicianID;
```

These are the backing fields of CompletedService Class. This class is used for completing the service and inserting data of completed service into completed_service table in database for the completion of monthly service report.

```
public CompletedService (int requestNumber, string serviceID, string serviceDescription, int amountPaid,
DateTime orderDate, DateTime collectionDate, string serviceUrgency, string customerID, string technicianID)
{
  _requestNumber = requestNumber;
  _serviceID = serviceID;
  _serviceDescription = serviceDescription;
  _amountPaid = amountPaid;
  _orderDate = orderDate;
  _collectionDate = collectionDate;
  _serviceUrgency = serviceUrgency;
  _customerID = customerID;
  _technicianID = technicianID;
}
```

This is the only constructor of CompletedService Class. This class doesn't have an empty constructor because it is not needed for this project. This constructor takes 9 arguments and assigns each of them to the corresponding backing field. This action will allow the program to run method inside the class using the properties of the class.

```
public int AddCompletedService()
```

```
{
    SqlConnection con = new SqlConnection
                (ConfigurationManager.ConnectionStrings["ACHdb"].ToString());
    con.Open();
    SqlCommand cmd = new SqlCommand("INSERT INTO completed_services
                            VALUES(@requestNo, @serviceId, @serviceDesc, @amountPaid, @orderDate,
                           @collectionDate, @serviceUrgency, @cusID, @techID);", con);
    cmd.Parameters.AddWithValue("@requestNo", RequestNumber);
    cmd.Parameters.AddWithValue("@serviceID", ServiceID);
    cmd.Parameters.AddWithValue("@serviceDesc", ServiceDescription);
    cmd.Parameters.AddWithValue("@amountPaid", AmountPaid);
    cmd.Parameters.AddWithValue("@orderDate", OrderDate);
    cmd.Parameters.AddWithValue("@collectionDate", CollectionDate);
    cmd.Parameters.AddWithValue("@serviceUrgency", ServiceUrgency);
    cmd.Parameters.AddWithValue("@cusID", CustomerID);
    cmd.Parameters.AddWithValue("@techID", TechnicianID);
    int x = cmd.ExecuteNonQuery();
    con.Close();
    if (x != 0)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

Method above is the only method of CompletedService Class. In order to run this method, an object of CompletedService should be created first. When the constructor is called and the value has assigned to the backing fields through the argument passed by constructor when creating an object, this method will then be usable. This method doesn't have any parameter, so it will not take any arguments from outside. This method will first form a connection to database, and then create a SQL query string that have parameters which will bring in the value of the backing field accordingly. This part has implemented basic Object-Oriented Programming Concept into it. The method will then execute the query, if changes were made in database, the method will return 1, else will return 0 for the validation purpose.

```
SqlCommand cmd = new SqlCommand("SELECT request_no, service_id, cus_id, tech_id,
service_description, collection_date, amount_paid, service_urgency, order_date FROM
requested_services WHERE request_no = " + requestNo, con);
```

```
SqlDataReader reader = cmd.ExecuteReader();
reader.Read();
//Created an Object of CompletedService
CompletedService Service = new CompletedService(reader.GetInt32(0),
                        reader.GetString(1), reader.GetString(4), reader.GetInt32(6),
                        Convert.ToDateTime(reader["order_date"]), Convert.ToDateTime(reader["collection_date"]),
                        reader.GetString(7), reader.GetString(2), technician_id);
reader.Close();
            //Called the Method of CompletedService
int validate = Service.AddCompletedService();

            if (validate != 0)

            {
                        MessageBox.Show("Service Completed!");
                        cboxService.Items.Remove(cboxService.SelectedItem);
                        SqlCommand cmd2 = new SqlCommand("UPDATE requested_services SET
                        service_completion = 1, tech_id = " + "'" + technician_id + "'" + " WHERE request_no = " +
                        requestNo, con);
            }
            else
            {
                        MessageBox.Show("Opps! Error Occur!");
            }
```

Codes above shows example of usage of CompletedService Class in TechMain.cs. The codes will first create a SQL query, and then execute the query to read the information of specific service request inside database. An object is created to store the information of the specific service request by calling the constructer and passing the arguments into it. The reader is closed after that to prevent error from happening. The variable validate which has datatype of integer will then call the method of CompletedService Class using the object created and store its return value. It will then do validation to decide whether to continue or to stop the process.

StaffInfo Class

```
        private string staff_id;
```

```
        private string staff_name;
        private string staff_contact;
        private string staff_password;
        private string staff_birthdate;
```

These are the backing fields of StaffInfo Class. This class will be used to do some staff-related process such as registering new staff, extracting the information of staff or changing the detail of staff.

```
        public StaffInfo(string Staff_id, string Staff_name, string Staff_contact, string Staff_password, string
        Staff_birthdate)
    {
       this.staff_id = Staff_id;
       this.staff_name = Staff_name;
       this.staff_contact = Staff_contact;
       this.staff_password = Staff_password;
       this.staff_birthdate = Staff_birthdate;
    }

    public StaffInfo(string Staff_id)
    {
       this.staff_id = Staff_id;
    }

    public StaffInfo() { }
```

These are the constructors of StaffInfo Class. This class contain three different constructers which will be used base on different purpose of usage. For example, the constructer with the most parameters will be used when creating staffs and the constructer with single parameter will be used when we want to create an object with existing staff id for further usage.

```
    public void change_contact(string contact)
    {
       this.staff_contact = contact;
    }

    public void change_password(string password)
```

```
{
    this.staff_password = password;
}

public override string ToString()
{
    return $"{staff_id}\t\t{staff_name}\t\t\t{staff_contact}";
}

public int CreateStaff(string query, string query2)
{
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["ACHdb"].ToString());
    con.Open();

    SqlCommand cmd = new SqlCommand(query + query2, con);
    cmd.Parameters.AddWithValue("@id", Staff_id);
    cmd.Parameters.AddWithValue("@name", Staff_name);
    cmd.Parameters.AddWithValue("@contact", Staff_contact);
    cmd.Parameters.AddWithValue("@password", Staff_password);
    cmd.Parameters.AddWithValue("@bdate", Staff_birthdate);
    int x = cmd.ExecuteNonQuery();
    con.Close();
    if (x != 0)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

These are the methods of StaffInfo Class. There are total of 4 methods inside this class. For the first and second method, change_contact () and change_password (), its purpose is to bring the data into an object, assigning value to backing field to perform further processing such as changing that detail of staffs by using the properties of an object. For the third method, ToString (), it overrides the original ToString () method and returns its own unique value such as properties of an object when this method is called by an object.

For the fourth method, CreateStaff (), it establish a connection with database first, and then it create a SQL query which will contain the two arguments passed into this method. The parameters inside the SQL query will then be added with the values of properties accordingly. The method then execute the query and do validation.

```
//Created an Object of StaffInfo
StaffInfo techData = new StaffInfo(TechID, txtRegName.Text, txtRegContact.Text, TechID + "@" +
cboxRegBMonth.Text + cboxRegBDay.Text, cboxRegBYear.Text + "-" + cboxRegBMonth.Text + "-" +
cboxRegBDay.Text);
query = "INSERT INTO technicians VALUES (@id, @name, @contact, @password,
        @bdate);";
query2 = "INSERT INTO users_login VALUES (@id, @password, 'technician',
```

```
            @name);";
validate = techData.CreateStaff(query, query2);
```

Codes above shows example of usage of StaffInfo Class in AdminRegister.cs. The object of StaffInfo, techData, is created to store information of new technician by calling the constructor and passing arguments into it. The variable validate which has datatype of integer will then call the method of StaffInfo Class, CreateStaff(query, query2), using the object created and store its return value. The validation will be done after this.

```
List<StaffInfo> staffs = new List<StaffInfo>();

while (reader.Read())
{
//Deleted Accounts Have ID like "[DEL]RE001", Exclude Those Accounts
            if (!reader[accid].ToString().StartsWith("[DEL]"))
            {
//Add an Object to the List of Objects (staffs) using Object Initialiser
                        staffs.Add(new StaffInfo()
                        {
                                    Staff_id = reader[accid].ToString(),
                                    Staff_name = reader[accname].ToString(),
                                    Staff_contact = reader[accphone].ToString()
                        });
            }
}
reader.Close();
//Show All Objects Inside List of Objects (staffs) to the ListBox
for (int i = 0; i < staffs.Count; i++)
{
            listAcc.Items.Add(staffs[i].ToString());
}
```

Codes above shows another example of usage of StaffInfo Class in AdminMain.cs. The list of objects of StaffInfo, staffs, is created first. Inside the while loop, if the staff id did not have "[DEL]" in front of it, then we will create and add an object into the list staffs using object initialiser. When all the staff in the database has been read and looped through this while loop, we close the reader. After that, base on the number of objects inside the list staffs, we created a for loop to add all objects inside the staffs into list box which will display the staffs to admin.

acceptPayment Class

- this class has only one method.

```
SqlConnection con;
```

- Above is a string that initializes an instance of SqlConnection class.

```
public void accept(int number,int money, string id, string name, string service, string rec_id, string rec_name, string
urgency, string service_id)
      {
         con = new      SqlConnection(ConfigurationManager.ConnectionStrings["ACHdb"].ToString());
         con.Open();

         string command_2 = "update requested_services set payment_accepted = 1, reviewed = 1 where request_no =
@ID";
         SqlCommand cmd_2 = new SqlCommand(command_2, con);
         cmd_2.Parameters.AddWithValue("@ID", number);
         cmd_2.ExecuteNonQuery();

         string command_3 = "insert into receipts values ('" + id + " : " + name + " attempted payement was accepted by
receptionist: " + rec_id + " : " + rec_name + "','" + DateTime.Now.ToString("yyyy-MM-dd hh:mm:ss") + "', ' " + service +
"', " + number + ", " + money + "); ";
         SqlCommand cmd_3 = new SqlCommand(command_3, con);
         cmd_3.ExecuteNonQuery();


         if (urgency == "urgent")
         {
            string command_4 = "select urgent_count, income from income_report where service_id = @service_id ;";
            SqlCommand cmd_4 = new SqlCommand(command_4, con);
            cmd_4.Parameters.AddWithValue("@service_id", service_id);


            string comand = "select Total_Income from Total_Income;";
            SqlCommand cd = new SqlCommand(comand, con);
            SqlDataReader reader = cd.ExecuteReader();
            int total_income = 0;
            while (reader.Read())
            {
               total_income = reader.GetInt32(0) + money;

            }
            reader.Close();
            string comand_2 = "update Total_Income set Total_Income = " + total_income + ";";
            SqlCommand cd_2 = new SqlCommand(comand_2, con);
            cd_2.ExecuteNonQuery();


            SqlDataReader read = cmd_4.ExecuteReader();
            if (read.Read())
            {
               int urgent_count = read.GetInt32(0) + 1;
               int income = read.GetInt32(1) + money;
               read.Close();
               string command_5 = "update income_report set urgent_count = @urgent_count, income = @income where
service_id = @id ;";
               SqlCommand cmd_5 = new SqlCommand(command_5, con);
               cmd_5.Parameters.AddWithValue("@urgent_count", urgent_count);
               cmd_5.Parameters.AddWithValue("@income", income);
               cmd_5.Parameters.AddWithValue("@id", service_id);
               cmd_5.ExecuteNonQuery();
            }
```

```
        }
        else if (urgency == "normal")
        {
            string command_6 = "select normal_count, income from income_report where service_id = @service_id ;";
            SqlCommand cmd_6 = new SqlCommand(command_6, con);
            cmd_6.Parameters.AddWithValue("@service_id", service_id);


            string comand = "select Total_Income from Total_Income;";
            SqlCommand cd = new SqlCommand(comand, con);
            SqlDataReader reader = cd.ExecuteReader();
            int total_income = 0;
            while (reader.Read())
            {
                total_income = reader.GetInt32(0) + money;

            }
            reader.Close();
            string comand_2 = "update Total_Income set Total_Income = " + total_income + ";";
            SqlCommand cd_2 = new SqlCommand(comand_2, con);
            cd_2.ExecuteNonQuery();


            SqlDataReader read_2 = cmd_6.ExecuteReader();
            if (read_2.Read())
            {
                int normal_count = read_2.GetInt32(0) + 1;
                int income = read_2.GetInt32(1) + money;
                read_2.Close();
                string command_7 = "update income_report set normal_count = @normal_count, income = @income where
service_id = @id ;";
                SqlCommand cmd_7 = new SqlCommand(command_7, con);
                cmd_7.Parameters.AddWithValue("@normal_count", normal_count);
                cmd_7.Parameters.AddWithValue("@income", income);
                cmd_7.Parameters.AddWithValue("@id", service_id);
                cmd_7.ExecuteNonQuery();
            }
        }
    }
```

- Above method accepts 9 arguments and uses them to access the database and complete the payment process.
- This method is used to update the payment status of the requested_services and update the income report.
- Above: the cmd_2 initiates an SQL command that will run the query in command_2 after adding the values to the parameters.
- After executing cmd_2 and updating the requested_services table, the command ‘cd’ and ‘cd_2’ are executed to add the paid amount to the ‘Total_Income’ table.
- An if statement is then used to check whether the service urgency is urgent or normal. If found urgent, the program will update the urgent table in the income report and increment the urgent service requests by one + it will add the new payment to the total income value in the database.

- However, if found that the request is normal, the program will increment the normal service requests by one and will add the new payment to the total income.

customerMakePayment Class

- This class has only one method.

```
public void processPayment(int amount, string customer_id, string service_id, string urgency)
    {

        SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["ACHdb"].ToString());
        con.Open();

        string command_1 = "insert into requested_services(service_id, cus_id, service_completion, reviewed,
amount_paid, service_urgency, order_date) values ( @service_id , @customer_id , 0, 0, @amount , @urgency , @date );";
        SqlCommand cmd_1 = new SqlCommand(command_1, con);

        string uppercase = customer_id.ToUpper();
```

```
        DateTime myDateTime = DateTime.Now;
        string sqlFormattedDate = myDateTime.ToString("yyyy-MM-dd");

        cmd_1.Parameters.AddWithValue("@date", sqlFormattedDate);
        cmd_1.Parameters.AddWithValue("@service_id", service_id);
        cmd_1.Parameters.AddWithValue("@customer_id", uppercase);
        cmd_1.Parameters.AddWithValue("@amount", amount);
        cmd_1.Parameters.AddWithValue("@urgency", urgency);
        cmd_1.ExecuteNonQuery();
    }
```

- This method takes four arguments and uses them to insert the payment details from customers to the requested_services table in the database.
- Whenever a customer confirms a payment, this method will create a new record in the database that will hold all of the service details.
- Moreover, this method will assign a FALSE value to the reviews attribute of the table which will later enable the receptionist to filter the reviewed the unreviewed requests.
- The new record will contain the following values to store in the table: service ID, customer ID, False values for both the reviewed and service_completion to indicate that these events still haven't happened, amount of money paid, the urgency of the request, and the date of request.

declinePayment Class

- This class has only one mthod.

```
        SqlConnection con;
```

- Above is a string that initializes an instance of SqlConnection class.

```
public void decline(int i)

    {
        con = new SqlConnection(ConfigurationManager.ConnectionStrings["ACHdb"].ToString());
        con.Open();
        string command_2 = "update requested_services set payment_accepted = 0, reviewed = 1 where request_no =
@ID";
        SqlCommand cmd_2 = new SqlCommand(command_2, con);
        cmd_2.Parameters.AddWithValue("@ID", i);

        cmd_2.ExecuteNonQuery();
```

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

- This method takes one argument which is the request number.
- The request number is used to locate the payment record in the requested services table.
- After locating the payment, the method will insert a value FALSE in the payment_accepted attribute of the table.
- This indicates that the payment has been rejected, and that the team will not work on the request.

rec_profile Class

- This class has only one method.

```
private string name;
    private string phoneNumber;
```

- Above are backing fields that store the data for the name and phone number of the receptionist.

```
public string Name { get { return name; } set { name = value; } }
        public string PhoneNumber { get { return phoneNumber; } set { phoneNumber = value;} }
```

- Above are the class properties with a GET and SET methods to either assign the property or return its value.

```
public void fetch(string id)
    {
       con = new SqlConnection(ConfigurationManager.ConnectionStrings["ACHdb"].ToString());
       con.Open();

       string command_1 = "select rec_name, rec_phone_number from receptionists where rec_id = @id";
       SqlCommand cmd_1 = new SqlCommand(command_1, con);
       cmd_1.Parameters.AddWithValue("@id", id);
       SqlDataReader info = cmd_1.ExecuteReader();

       while (info.Read())
       {
          name = info.GetString(0);
          phoneNumber = info.GetString(1);
       }
       info.Close();
       con.Close();
    }
```

- This method extracts the needed from the database so it can be shown in the receptionist profile form.
- It starts by taking the receptionist ID as an argument.
- It uses this ID in a query to locate the record that stores the receptionist's information.
- After locating, it assigns the values in the record to the backing fields so that they can be accessed in the form via the properties.

recChangeContact Class

```
        string id;
    string number;
```

- The above are backing fields that are used to store the data that the constructor is holding.

```
public recChangeContact(string rec_id, string newNumber)
    {
      id = rec_id;
      number = newNumber;
    }
```

- This is the class constructor, it takes two strings as arguments.
- These strings are the receptionist ID and the new entered phone number.
- The constructor assigns the argument values to the backing fields.

```
public string changeContact()
    {
      con = new SqlConnection(ConfigurationManager.ConnectionStrings["ACHdb"].ToString());
      con.Open();

      string command = "update receptionists set rec_phone_number = @new where rec_id = @id";
      cmd_1 = new SqlCommand(command, con);
      cmd_1.Parameters.AddWithValue("@new", number);
      cmd_1.Parameters.AddWithValue("@id", id);
      int check = cmd_1.ExecuteNonQuery();
```

```
    con.Close();

    if (check != 0)
    {
       return "Your phone number is changed successfully";
    }
    else
    {
       return "Couldn't change your phone number !";
    }
}
```

- The above method uses the receptionist's ID to locate the data record in the database.
- In place of the rec_phone_number attribute, the method then updates the record with the new phone number in place of the old one.
- After updating the database, the method checks the returned results.
- If the method finds that there are more than one row effected in the database, it will return a statement 'Your phone number is changed successfully' to the 'changeContactNumberForm' to indicate a successful change of receptionist number.
- If the method finds that there are zero rows effected in the database, it will return a statement 'Couldn't change your phone number !' to the 'changeContactNumberForm' to indicate a failed change of receptionist number.

recChangePass Class

```
string id;
    string oldPassword;
```

- these packing fields are used to store data that come from the constructor and from the database.

```
public recChangePass(string rec_id)
    {
      id = rec_id;
    }
```

- The class constructor has one argument which is the receptionist ID.
- It assign the value of the argument to the local variable (backing field) called ‘id’ to be able to access and manipulate this data inside other methods in the class.

```
public string changePassword(string oldPass, string newPass)
    {
      con = new SqlConnection(ConfigurationManager.ConnectionStrings["ACHdb"].ToString());
      con.Open();
      string command_1 = "select rec_password from receptionists where rec_id = @re_id ";
      cmd_1 = new SqlCommand(command_1, con);
      cmd_1.Parameters.AddWithValue("@re_id", id);
      SqlDataReader retrieve = cmd_1.ExecuteReader();
      while (retrieve.Read())
      {
        oldPassword = retrieve.GetString(0);
      }
      retrieve.Close();
      con.Close();

      if (oldPassword == oldPass)
      {
```

```
        con = new SqlConnection(ConfigurationManager.ConnectionStrings["ACHdb"].ToString());
        con.Open();
        string command_2 = "update receptionists set rec_password = @new where rec_id = @id";
        cmd_2 = new SqlCommand(command_2, con);
        cmd_2.Parameters.AddWithValue("@new", newPass);
        cmd_2.Parameters.AddWithValue("@id", id);
        int check = cmd_2.ExecuteNonQuery();


        string command_3 = "update users_login set user_password = @new where user_id = @id ";
        SqlCommand cmd_3 = new SqlCommand(command_3, con);
        cmd_3.Parameters.AddWithValue("@new", newPass);
        cmd_3.Parameters.AddWithValue("@id", id);
        int check_2 = cmd_3.ExecuteNonQuery();
        con.Close();



        if (check != 0 & check_2 != 0)
        {
            return "Password is successfully updated !";
        }
        else
        {
            return "Failed to updae the password !";
        }
    }
    else
    {
        return "The old password you entered is incorrect !";
    }
}
```

- The above method takes the old and new passwords as arguments.
- It uses the receptionist ID stored in the id variable to access the receptionist record in the database.
- The method then extracts the old password to the backing field ‘oldPassword’
- After that, it compares the oldPassword value with the oldPass value of its argument.
- IF found that the two passwords match, the method will update the password field in users_login and receptionists tables with the new password.
- If found that there are more than zero rows effected in the database, the method returns a statement ‘Password is successfully updated !’ to the ‘recChangePass’ form to indicate a successful change of password.
- If found that there are zero rows effected in the database, the method returns a statement ‘Failed to updae the password !’ to the ‘recChangePass’ form to indicate a failed change of password.
- However, if the method finds that the ‘oldPass’ and ‘oldPassword’ don’t match, it will return a statement ‘The old password you entered is incorrect !’ to the ‘recChangePass’ form to indicate a mismatch of passwords.

register_cus Class

```
private string cus_name;
    private string cus_number;
    private string cus_pass;
    private string cus_id;
    int num;
```

- above are backing fields used to store data from the methods.

```
public void id_generator()
    {

        con = new SqlConnection(ConfigurationManager.ConnectionStrings["ACHdb"].ToString());
        con.Open();

        string Q = "select COUNT(cus_id) from customers;";
        SqlCommand cmd_1 = new SqlCommand(Q, con);
        SqlDataReader numOfRows = cmd_1.ExecuteReader();
        while (numOfRows.Read())
        {
            num = numOfRows.GetInt32(0);
        }
        numOfRows.Close();
        con.Close();

        num = num + 1;
        cus_id = "CU00" + num.ToString();
    }
```

- This is the first method in this class.
- It is used to auto-generate the customer ID by incrementing one to the total number of records in the customers table.
- It assigns the final ID in the backing field ‘cus_id’.

```
public string registerer(string name, string number, string password)
    {
        cus_name = name;
        cus_number = number;
        cus_pass = password;

        con = new SqlConnection(ConfigurationManager.ConnectionStrings["ACHdb"].ToString());
        con.Open();
```

```
    string command_1 = "insert into customers values ( @id , @name , @number , @pass );";

    SqlCommand cmd_1 = new SqlCommand(command_1, con);
    cmd_1.Parameters.AddWithValue("@id", cus_id);
    cmd_1.Parameters.AddWithValue("@name", cus_name);
    cmd_1.Parameters.AddWithValue("@number", cus_number);
    cmd_1.Parameters.AddWithValue("@pass", cus_pass);
    int test = cmd_1.ExecuteNonQuery();

    string command_2 = "insert into users_login values ( @id , @pass , 'customer' , @name );";

    SqlCommand cmd_2 = new SqlCommand(command_2, con);
    cmd_2.Parameters.AddWithValue("@id", cus_id);
    cmd_2.Parameters.AddWithValue("@name", cus_name);
    cmd_2.Parameters.AddWithValue("@pass", cus_pass);
    int test_2 = cmd_2.ExecuteNonQuery();

    if (test != 0 & test_2 != 0)
    {
       con.Close();
       return "customer account was created successfully";
    }
    else
    {
       con.Close();
       return "Could not create customer account";
    }

}
```

- The above method takes the customer name, number, password as its arguments.
- It assigns these arguments to their according variables (‘cus_name’, ‘cus_number’, ‘cus_pass’).
- It create a new records in the customers table and users_login table.
- The new records data is brought from the data stored in the backing fields.
- After creating the new records, the method checks if there are more than 0 affected rows in the two table:
  1- If yes, the method will return a statement ‘customer account was created successfully’.
  2- If no, the method will return a statement ‘Could not create customer account’.

customerNumber Class

```
internal class customerNumber
    {
        private string cus_id;
        private string cus_name;
        private string cus_contact;
        private string cus_password;
```

These are the backing fields of customerNumber, This class will be used to change the customers details(contact number, password).

```
        public string Cus_id { get { return cus_id; } set { cus_id = value; } }
        public string Cus_name { get { return cus_name; } set { cus_name = value; } }
        public string Cus_contact { get { return cus_contact; } set { cus_contact =
value; } }
        public string Cus_password { get { return cus_password; } set { cus_password =
value; } }
```

Above are the class properties with a GET and SET methods to either assign the property or return its value.

```
        public customerNumber(string Cus_id, string Cus_name, string Cus_contact,
string  Cus_password)
        {
            this.cus_id = Cus_id;
            this.cus_name = Cus_name;
            this.cus_contact = Cus_contact;
            this.cus_password = Cus_password;

        }
```

In the above class, customer ID, name, contact Number and password are declared and there is one Object, which is the instance of the class, which provides the ability to access the customer data such as ID, name, , contact number and password.

```
        public customerNumber(string Cus_id)
        {
            this.cus_id = Cus_id;
        }

        public void change_contact(string contact)
        {
            this.cus_contact = contact;
        }

        public void change_password(string password)
        {
            this.cus_password = password;
        }
    }
  //Created an Object of customerNumber
 customerNumber cusinfo = new customerNumber(cusID);
//Change the Value of Attribute of the Object
 cusinfo.change_contact(NewContactNumbertxt.Text);
string query = "UPDATE customers SET cus_phone_number = " + "'" + cusinfo.Cus_contact +
"'" + " WHERE cus_id = " + "'" + cusinfo.Cus_id + "'";
 SqlCommand cmd = new SqlCommand(query, con);

customerNumber cusfo = new customerNumber(cusID);
cusfo.change_password(NewPasswordtxt.Text);
string query = "UPDATE customers SET cus_password = " + "'" + cusfo.Cus_password + "'" +
" WHERE cus_id = " + "'" + cusfo.Cus_id + "';";
```

```
string query_2 = "update users_login set user_password = " + "'" + cusfo.Cus_password +
"'" + " Where user_id = " + "'" + cusfo.Cus_id + "';";

SqlCommand cmd = new SqlCommand(query + query_2, con);
```

These are the methods of customerNumber Class. There are two methods inside this class. For the first and second method, change_contact () and change_password (), its purpose is to bring the data into an object, assigning value to backing field to perform further processing such as changing that detail of customers by using the properties of an object. The methods above are there to interact with the database those are just selecting and updating the profile in the customers table.

Test Plan and Test Cases

Test Case	Function	Test Objective	Expected Results	Actual Results	Remarks
1	Requested Services (Receptionist)	View all the requested services and their payments	Accurate display of all service requests and the payments amounts	Details are displayed as expected	None
2	Decline a Payment	Check if the program mark the payment as declined in the database	payment_accepted column in the database to have a value of 0	Database has been updated with the correct value	None
3	Accept a Payment	1- Check if the program mark the payment as accepted in the database 2- Check if the program adds the payment details to the income_report table	1- payment_accepted column in the database to have a value of 1 2- income_report table to be updated with the new payment and service type	Database has been updated with the correct value	None
4	Register a Customer	To create a customer record in the customers and users_login tables	The users_login and customers tables to have new records which have the same data as the one input in the form	The tables in the database have been modified and new data was added successfully	None
5	View Profile (Receptionist)	To show all the receptionist's account details	All the data relating to the customer is to be displayed accurately in the form	All data was shown perfectly	None
6	Change Password	To update the password in both users_login and	All tables should have the newly updated password	All relevant database tables	None

	(Receptionist)	receptionists tables in the database		were modified correctly	
7	Change Contact Number (Receptionist)	To update the phone number in the receptionists table in the database	The receptionists table should be updated with the new number	The receptionist record in the database was modified successfully	None
8	Search for Technicians (Admin)	Display all the Technician account in the database	All Technician account is displayed	All Technician account is shown	None
9	Search for Receptionists (Admin)	Display all the Receptionist account in the database	All Receptionist account is displayed	All Receptionist account is shown	None
10	Register new Technician (Admin)	Register new Technician into database	New Technician account added to database	New Technician account successfully added to database	None
11	Register new Receptionist (Admin)	Register new Receptionist into database	New Receptionist account added to database	New Receptionist account successfully added to database	None
12	Delete Account (Admin)	Delete account in the database	Add “[DEL]” in front of the account ID	Deleted Account have “[DEL]” in front of its ID	None
13	View Service Report (Admin)	Display Service Report to Admin	Display Service Report	Service Report is displayed	None
14	View Income Report (Admin)	Display Income Report to Admin	Display Income Report	Income Report is displayed	None
15	View Service Detail (Technician)	View the detail of the Service selected	Display the details of the Service selected	The details of the service are displayed	None
16	Add Description (Technician)	Add description to the service	Description added to the service in database	Description successfully added into database	None

17	Add Collection Date (Technician)	Add collection date to the service	Collection data added to the service in database	Collection date successfully added into database	None
18	Complete Service (Technician)	Mark service as completed	Mark service as completed and add the service to the completed_service table in database	Service marked as completed and successfully added the service to completed_service table in database	None
19	View Profile (Technician)	To display the profile window of technician	Display the profile window of technician	Technician profile window is displayed	None
20	Change Password (Technician)	Change his/her password in database	Update new password into the database	New password is updated in database	None
21	Change Contact Number (Technician)	Change his/her contact number in database	Update new contact number into the database	New contact number is updated in database	None
22	Display Available Services	To display all the available services in the combo box	All services names should be displayed in the combo box without repetition or messy ordering	Services were displayed in an ordered way without repetition	None
23	Display Service Price	To show the price of the selected service after checking one of the service urgency radio buttons	The accurate price for each service and its urgency should be shown in the service price label	Prices are shown accordingly and perfectly	None
24	Confirm Payment	To add the payment and service details into the requested_services table in the databsae	The creation of a new record in the requested_services table which has all the service and payment information	A new record is accurately created as expected	None
25	View Requested Services	To show the customer all of the services that has been requested by him/her	The services names, dates, statuses, collection dates, descriptions should be displayed in their assigned controls accurately	All the data is shown accurately	None

26	View Profile (Customer)	To display all of the customer account details from the database	All data displayed should be accurate and presented in the desired locations	All the data and its locations are accurately presented in the form	None
27	Change Password (customer)	To update the password in both users_login and customers tables in the database	All tables should have the newly updated password	All relevant database tables were modified correctly	None
28	Change Contact Number (customer)	To update the phone number in the customers table	The customer record in the customers table should have updated phone number	The record in the table was updated successfully with the new number	None
29	Login	To test the authentication process of the program	1- The program is expected to only give access to those who enter matched ID and password. 2- The program assigned the correct role for each user.	The program gave access and assigned roles accurately.	If the password is typed with the actual correct characters but with different capitalization at any point of the password, the program didn't show an incorrect password message nor did it give access to any other form. It basically performed no action. However, it didn't freeze and users can still reinput their details again.

Conclusion

In conclusion, the teamwork on this project was quite effective. Working as a team to create a specific project requires a great deal of planning and organization; as a result, the members of this group gained proficiency with Classes, Objects, and Methods. There were a lot of issues at first and significant bugs in the actual code, however, with research and collaboration we were able to get past these issues and figure out these bugs causes. In general, the program seems well put together, but when diving deep into the code there is a weakness which is that the login screen won't display or perform any action if the entered password is matched but with different capitalization. There are some other unhealthy coding practices that we couldn't go about such as, using several list arrays -instead of one- to store several values from the database, the existence of unutilized event handlers, or the use of primitive validation techniques. Nonetheless, there are aspects of this program that we believe are solid coding practices such as, the utilization of database wherever possible and avoiding hard-coded values, and the use of classes and methods to handle functions and processes. To sum up, this program is the fruit of what we have learnt so far about OOP, and we are keen to learn more extensive concepts of implementing programs more effectively and professionally using OOP. Concepts where all the aforementioned weaknesses are left behind and replaced with powerful, clean, and efficient code.

References

C# Classes and Objects. (n.d.-b). W3Schools.

https://www.w3schools.com/cs/cs_classes.php

Gillis, A. S., & Lewis, S. (2021, July 13). *object-oriented programming (OOP)*.

SearchAppArchitecture.

https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP

What is Class Diagram? (n.d.). Visual Paradigm.

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/

→ V. A. P. B. N. (2022, February 21). *Use Case Diagram Tutorial (Guide with Examples)*.

Creately Blog.

https://creately.com/blog/diagrams/use-case-diagram-tutorial/

Workload Matrix

Work Description	**Hasan Akram Abdullah Zaid** TP066635	**Chai Chean Han TP064801**	**Abdullah Zakaria Elkhatal TP064837**
Storyboard Designs	33.3%	33.3%	33.3%
Coding	40.0%	50.0%	10.0%
Creating Database	80.0%	20.0%	0.0%
Documentation	33.3%	33.3%	33.3%
Signatures