SAMSUNG

SAMSUNG SDS

INTERNSHIP REPORT

WINDOWS FORMS
AND
WEB APPLICATIONS
USING
.NET

Vedaanti Baliga

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to Mr. Sunil Kumar Singh, Deputy Manager, for their guidance and encouragement in carrying out this project work.

I would also like to express my gratitude to the officials and other staff members of Samsung India Electronics who rendered their help during the period of my project work.

PROJECT ABSTRACT

.NET Framework is a software framework developed by Microsoft that runs primarily on Microsoft Windows. It is a free cross platform which is open source. It provides a developer's platform for building desktop, mobile, web, gaming and IoT applications.

It includes a large class library named Framework Class Library (FCL) and provides language interoperability (each language can use code written in other languages) across several programming languages.

Programs written for .NET Framework execute in a software environment (in contrast to a hardware environment) named Common Language Runtime (CLR), an application virtual machine that provides services such as security, memory management, and exception handling.

The programming language used in this project is C#. C# (pronounced "C sharp") is a simple, modern, object-oriented, and type-safe programming language. Its roots in the C family of languages makes C# immediately familiar to C, C++, Java, and JavaScript programmers.

The project has an employee management database displayed in both, windows forms and a web application. It maintains the database of employees of an organization.

The project also includes a calculator made with windows forms having various functionalities.

CONTENTS

CHAPTER 1: INTRODUCTION

1.1 **Introduction to Work Done**

The .Net framework is used in many companies as it provides a flexible and easy to use interface. It is a graphical API to display data and manage user interactions with easier deployment and better security in client applications.

Windows Forms offers an extensive client library providing interface to access native Windows graphical interface elements and graphics from managed code. It is built with event-driven architecture similar to Windows clients and hence, its applications wait for user input for its execution. The first project done using windows forms is the employee management database, created at a basic level which has the ability to add, delete and update accounts ,search for accounts and filter the records according to the user's choice. The form also displays a grid view to list all the employee records.

The second windows form is a calculator application which offers the basic operations like addition, subtraction, multiplication and division. These operations can be done together and are displayed on the top of the calculator. The calculator also has basic functionalities like clearing the screen and clearing one digit.

ASP.NET is an open source web framework for building modern web apps and services. It creates websites based on HTML5, CSS, and JavaScript that are simple, fast, and can scale to millions of users. The project made with ASP.NET is a user friendly Employee Management, which is different from the windows forms as this is a real time scenario in which a user can register for an account, check his/her details, sign out and have a secure account protected by the SQL server.

1.2 **Project Statement and Objectives**

With this project I aim to provide an explanation of the working and features of the various scripts, queries and languages used to make the websites secure and easy to use.

The projects show an interactive interface with the help of javascript and html.

The following topics are covered in this project:

- Basic Calculator using windows form

- Employee management database using windows form and ASP.NET

CHAPTER 2: BACKGROUND OVERVIEW

2.1 **Conceptual Overview**

I learnt the basics of C# through some sample codes, like palindrome, generating prime numbers, etc. To begin with the windows form, a code in C# is auto generated which has to be completed according to specifications. There is a designer provided by Microsoft in which the labels, textboxes and buttons or the required tool can be dragged and edited in the designer respectively. Clicking on any tool generates an event in the .cs file which can then be filled with the appropriate code needed.

The employee database management provides the functionality to add an account by providing the employee name, phone, email and gender. The employee can also choose a project title provided by the drop down box. The employee can update his account by entering his employee ID and then make changes to the new windows form that opens on the click if the record is found.

The employee can delete his/her record by simply entering the employee ID and then confirming whether they wish to delete their account or not.

The employee can also filter the records displayed in the grid view on the basis of their gender. The form also has a search button where the employee can enter their name and email id and get their details displayed on the grid view.

The calculator application has a basic interface and a textbox where the users button click output is shown. There is a label which displays the operations and the result generated.

It has all validations like, entering only one "." for a decimal number and showing "Not Defined" when a number is divided by zero.

The ASP.NET application has a default page where the user can login to their existing account or register their account. There is also an admin's page which is only accessible by him/her in which there is a full record of all the employees that have an account. The admin can delete and update records.

On registering, an Employee ID is generated automatically which is to be remembered by the user at the time of logging in. There are validations provided by the web applications which don’t let the user enter invalid input and prompts the user to fill required fields.

After the registering of the account, the user can update his/her account according to their specifications. Once logged out of an account, the user cannot go back to the account page; he/she will have to log in again to get access.

The data is saved and accessed by the Microsoft SQL server. The tables and procedures are maintained inside the database. The queries for updating, deleting, adding, searching and filtering for the windows form and the web application are stored in a specific database which is safe from SQL injection.

2.2 **Technology Used**

a) C#

The programming language used for creating the windows forms and the web applications is C#. The inputs and parameters for updating, deleting, adding, searching and filtering are sent through functions written in C#.

For the calculators, in the same way, the button events are received by the functions in C# to generate an output at the label. The C# has functions with switch cases wherein it describes the functionalities for all operators.

b) HTML5

It is the standard markup language for creating web pages and web applications. With Cascading Style Sheets (CSS) and JavaScript, it forms a triad of cornerstone technologies for the World Wide Web. HTML5 is the latest specification of the HTML language. The objectives primarily include:

- HTML tables
- Links
- Forms
- Fonts
- Images
- Encouraging semantic (meaningful) markup
- Separating design from content
- Promoting accessibility and design responsiveness
- Reducing the overlap between HTML, CSS, and JavaScript
- Supporting rich media experiences while eliminating the need for plugins such as Flash or Java

In this project, HTML5 was used for providing an interface for the user to enter and update the required details, for submitting them and creating error messages for validations with fore colors.

c) JAVASCRIPT

It is used to program the behavior of web pages. Web pages are not the only place where JavaScript is used. Many desktop and server programs use JavaScript. Node.js is the best known. Some databases, like MongoDB and CouchDB, also use JavaScript as their programming language.

In this project, it is used to control the input allowed to enter by the user, for example, if there is a names field, it will disable the user from entering any numbers or special characters.

It also helps in creating functions which allow the user to give confirmations on leaving pages or making important changes to their data. These functions create a message box enabled with buttons for prompting the user to do something.

d) CSS

It is a simple mechanism for adding style (e.g., fonts, colors, spacing) to Web documents.

In this project, I have added a cascading sheet which has been linked to all pages to give a uniform look to the whole project.

e) SQL server

MS SQL Server is a relational database management system (RDBMS) developed by Microsoft. This product is built for the basic function of storing retrieving data as required by other applications. It can be run either on the same computer or on another across a network.

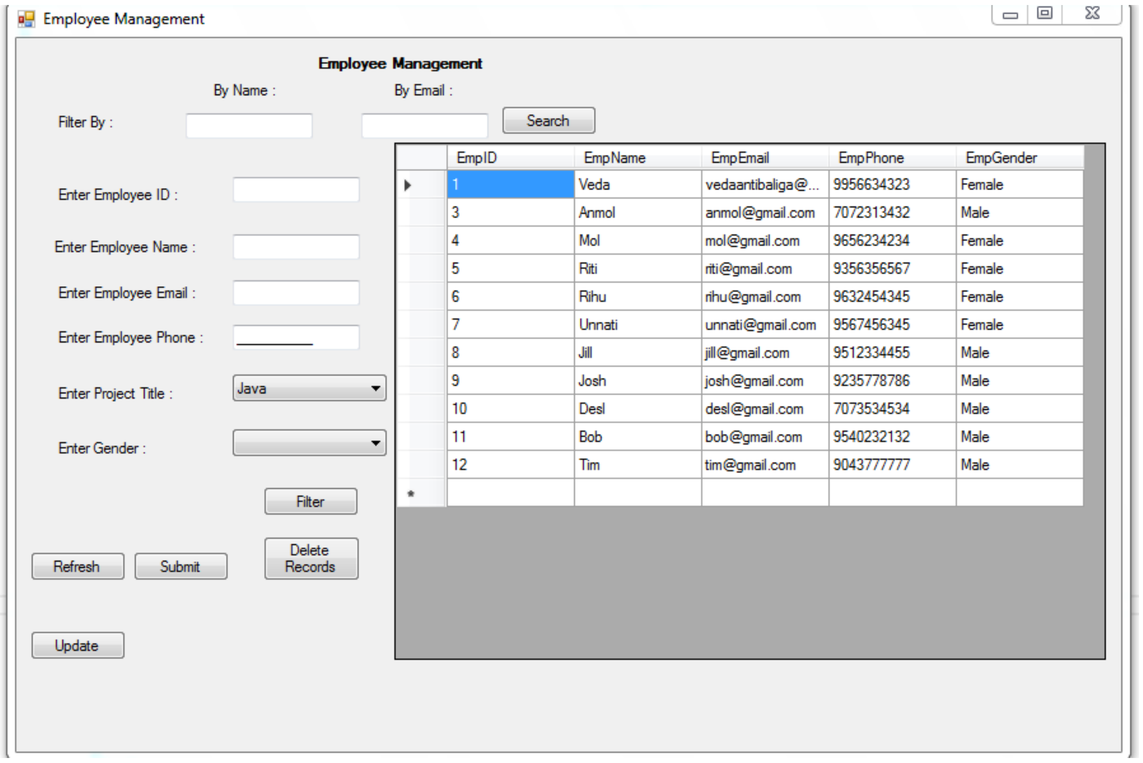
In this project, the SQL server has been used to store the data for the employees. All queries have been written inside stored procedures for inserting, deleting, updating, searching and filtering.

CHAPTER 3: METHODOLOGY

3.1 **Detailed Methodology**

1. **Employee Management (Windows Form):**

i. **Add An Account:**



Enter Project Title : Java

EmpID	EmpName	EmpEmail	EmpPhone	EmpGender
1	Veda	vedaantibaliga@...	9956634323	Female
3	Anmol	anmol@gmail.com	7072313432	Male
4	Mol	mol@gmail.com	9656234234	Female
5	Riti	riti@gmail.com	9356356567	Female
6	Rihu	rihu@gmail.com	9632454345	Female
7	Unnati	unnati@gmail.com	9567456345	Female
8	Jill	jill@gmail.com	9512334455	Male
9	Josh	josh@gmail.com	9235778786	Male
10	Desl	desl@gmail.com	7073534534	Male
11	Bob	bob@gmail.com	9540232132	Male
12	Tim	tim@gmail.com	9043777777	Male

Figure 1: Add an account

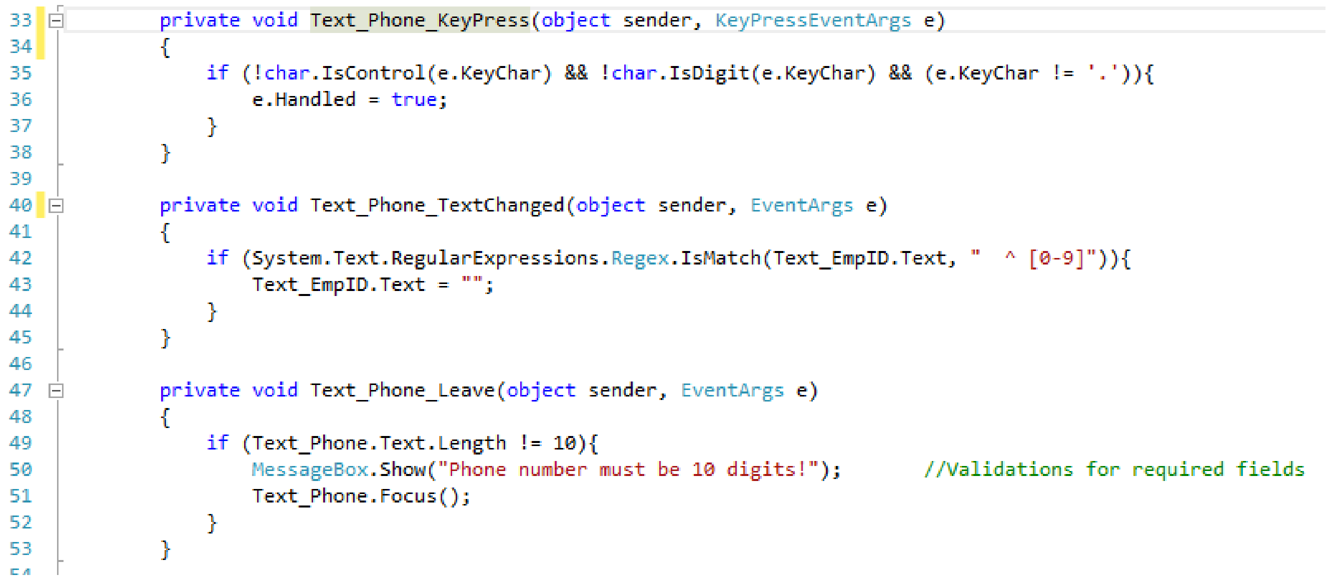
Figure 1 shows the interface for the windows form. In this form, the user has to enter his respective employee ID (except for the ones shown in the grid view).

The form doesn't allow to you to fill null values. There is an error box if that happens. The textbox for name doesn't allow the user to fill anything else other than spaces and letters.

The textbox for phone is a masked textbox and does not allow you to fill the user an entry, less or more digits than 10, as explained in the code in Figure 2.

After clicking on submit, an information box is displayed telling the user that the data was saved successfully, if the entries are not null and follow the rules set according to the functions of the code.

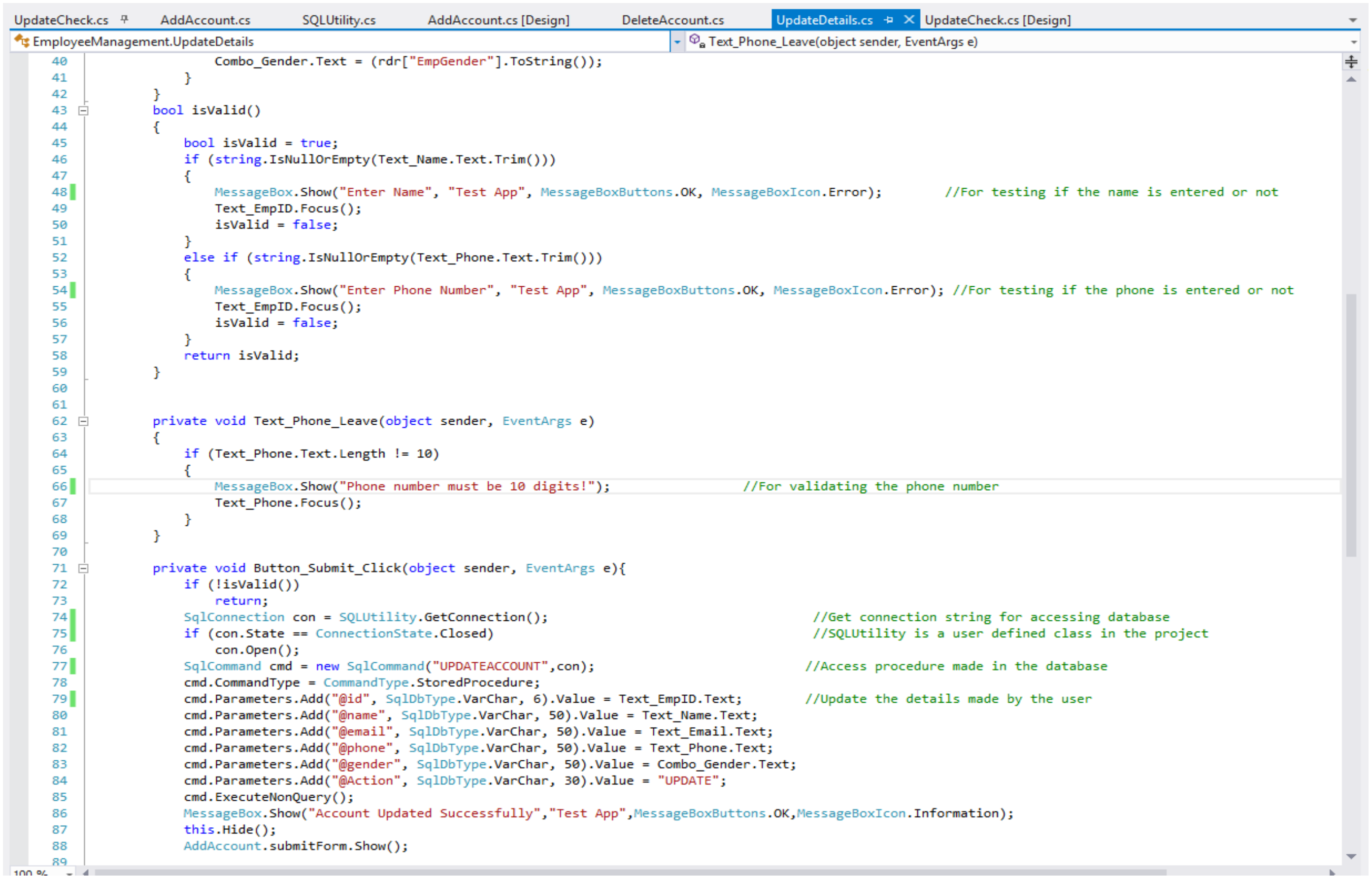
The textboxes also trim the extra spaces, in case they are entered by the user, by using the Trim() function.



```
        private void Text_Phone_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (!char.IsControl(e.KeyChar) && !char.IsDigit(e.KeyChar) && (e.KeyChar != '.')){
                e.Handled = true;
            }
        }

        private void Text_Phone_TextChanged(object sender, EventArgs e)
        {
            if (System.Text.RegularExpressions.Regex.IsMatch(Text_EmpID.Text, "  ^ [0-9]")){
                Text_EmpID.Text = "";
            }
        }

        private void Text_Phone_Leave(object sender, EventArgs e)
        {
            if (Text_Phone.Text.Length != 10){
                MessageBox.Show("Phone number must be 10 digits!");       //Validations for required fields
                Text_Phone.Focus();
            }
        }
```

Figure 2: Code for masked textbox

ii. **Update an Account:**



```
                Combo_Gender.Text = (rdr["EmpGender"].ToString());
            }
        }
        bool isValid()
        {
            bool isValid = true;
            if (string.IsNullOrEmpty(Text_Name.Text.Trim()))
            {
                MessageBox.Show("Enter Name", "Test App", MessageBoxButtons.OK, MessageBoxIcon.Error);        //For testing if the name is entered or not
                Text_EmpID.Focus();
                isValid = false;
            }
            else if (string.IsNullOrEmpty(Text_Phone.Text.Trim()))
            {
                MessageBox.Show("Enter Phone Number", "Test App", MessageBoxButtons.OK, MessageBoxIcon.Error); //For testing if the phone is entered or not
                Text_EmpID.Focus();
                isValid = false;
            }
            return isValid;
        }


        private void Text_Phone_Leave(object sender, EventArgs e)
        {
            if (Text_Phone.Text.Length != 10)
            {
                MessageBox.Show("Phone number must be 10 digits!");                 //For validating the phone number
                Text_Phone.Focus();
            }
        }

        private void Button_Submit_Click(object sender, EventArgs e){
            if (!isValid())
                return;
            SqlConnection con = SQLUtility.GetConnection();                                  //Get connection string for accessing database
            if (con.State == ConnectionState.Closed)                                         //SQLUtility is a user defined class in the project
                con.Open();
            SqlCommand cmd = new SqlCommand("UPDATEACCOUNT",con);                           //Access procedure made in the database
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Parameters.Add("@id", SqlDbType.VarChar, 6).Value = Text_EmpID.Text;        //Update the details made by the user
            cmd.Parameters.Add("@name", SqlDbType.VarChar, 50).Value = Text_Name.Text;
            cmd.Parameters.Add("@email", SqlDbType.VarChar, 50).Value = Text_Email.Text;
            cmd.Parameters.Add("@phone", SqlDbType.VarChar, 50).Value = Text_Phone.Text;
            cmd.Parameters.Add("@gender", SqlDbType.VarChar, 50).Value = Combo_Gender.Text;
            cmd.Parameters.Add("@Action", SqlDbType.VarChar, 30).Value = "UPDATE";
            cmd.ExecuteNonQuery();
            MessageBox.Show("Account Updated Successfully","Test App",MessageBoxButtons.OK,MessageBoxIcon.Information);
            this.Hide();
            AddAccount.submitForm.Show();
```

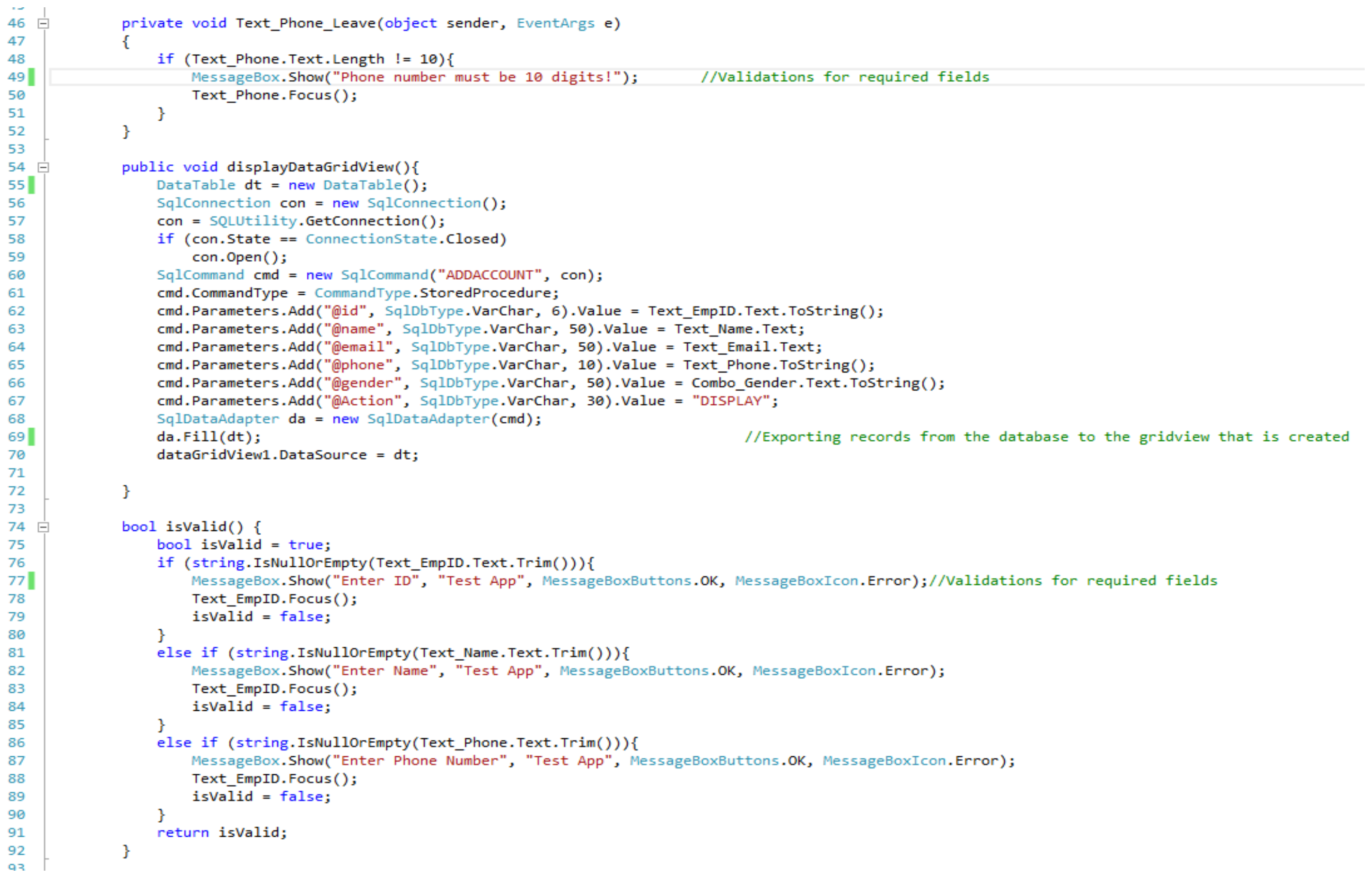
Figure 3: Code for updating

An account is updated when the user clicks on the update button on the form. The update form opens and the user enters the employee ID corresponding to the record that is to be updated.

If the user enters the wrong employee ID, then there is an error box displayed and the user is returned to the check record for update form otherwise the updating form appears where the user can update his/her data except for his/her employee ID as it is permanent and has been disabled by code.

After clicking on submit, a confirmation box appears asking the user if they want to submit the changes that they have made and on confirmation there is an information box displaying that the account was updated successfully. The grid view on refreshing would show the updated record.

iii. **Grid View:**



```
        private void Text_Phone_Leave(object sender, EventArgs e)
        {
            if (Text_Phone.Text.Length != 10){
                MessageBox.Show("Phone number must be 10 digits!");       //Validations for required fields
                Text_Phone.Focus();
            }
        }

        public void displayDataGridView(){
            DataTable dt = new DataTable();
            SqlConnection con = new SqlConnection();
            con = SQLUtility.GetConnection();
            if (con.State == ConnectionState.Closed)
                con.Open();
            SqlCommand cmd = new SqlCommand("ADDACCOUNT", con);
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Parameters.Add("@id", SqlDbType.VarChar, 6).Value = Text_EmpID.Text.ToString();
            cmd.Parameters.Add("@name", SqlDbType.VarChar, 50).Value = Text_Name.Text;
            cmd.Parameters.Add("@email", SqlDbType.VarChar, 50).Value = Text_Email.Text;
            cmd.Parameters.Add("@phone", SqlDbType.VarChar, 10).Value = Text_Phone.ToString();
            cmd.Parameters.Add("@gender", SqlDbType.VarChar, 50).Value = Combo_Gender.Text.ToString();
            cmd.Parameters.Add("@Action", SqlDbType.VarChar, 30).Value = "DISPLAY";
            SqlDataAdapter da = new SqlDataAdapter(cmd);
            da.Fill(dt);                                            //Exporting records from the database to the gridview that is created
            dataGridView1.DataSource = dt;

        }

        bool isValid() {
            bool isValid = true;
            if (string.IsNullOrEmpty(Text_EmpID.Text.Trim())){
                MessageBox.Show("Enter ID", "Test App", MessageBoxButtons.OK, MessageBoxIcon.Error);//Validations for required fields
                Text_EmpID.Focus();
                isValid = false;
            }
            else if (string.IsNullOrEmpty(Text_Name.Text.Trim())){
                MessageBox.Show("Enter Name", "Test App", MessageBoxButtons.OK, MessageBoxIcon.Error);
                Text_EmpID.Focus();
                isValid = false;
            }
            else if (string.IsNullOrEmpty(Text_Phone.Text.Trim())){
                MessageBox.Show("Enter Phone Number", "Test App", MessageBoxButtons.OK, MessageBoxIcon.Error);
                Text_EmpID.Focus();
                isValid = false;
            }
            return isValid;
        }
```

Figure 4: Code for grid view

The Grid View provides a powerful and flexible way to display data in a tabular format. I have used the Grid View to show read-only views of a small amount of data, and to scale it to show editable views of very large sets of data.

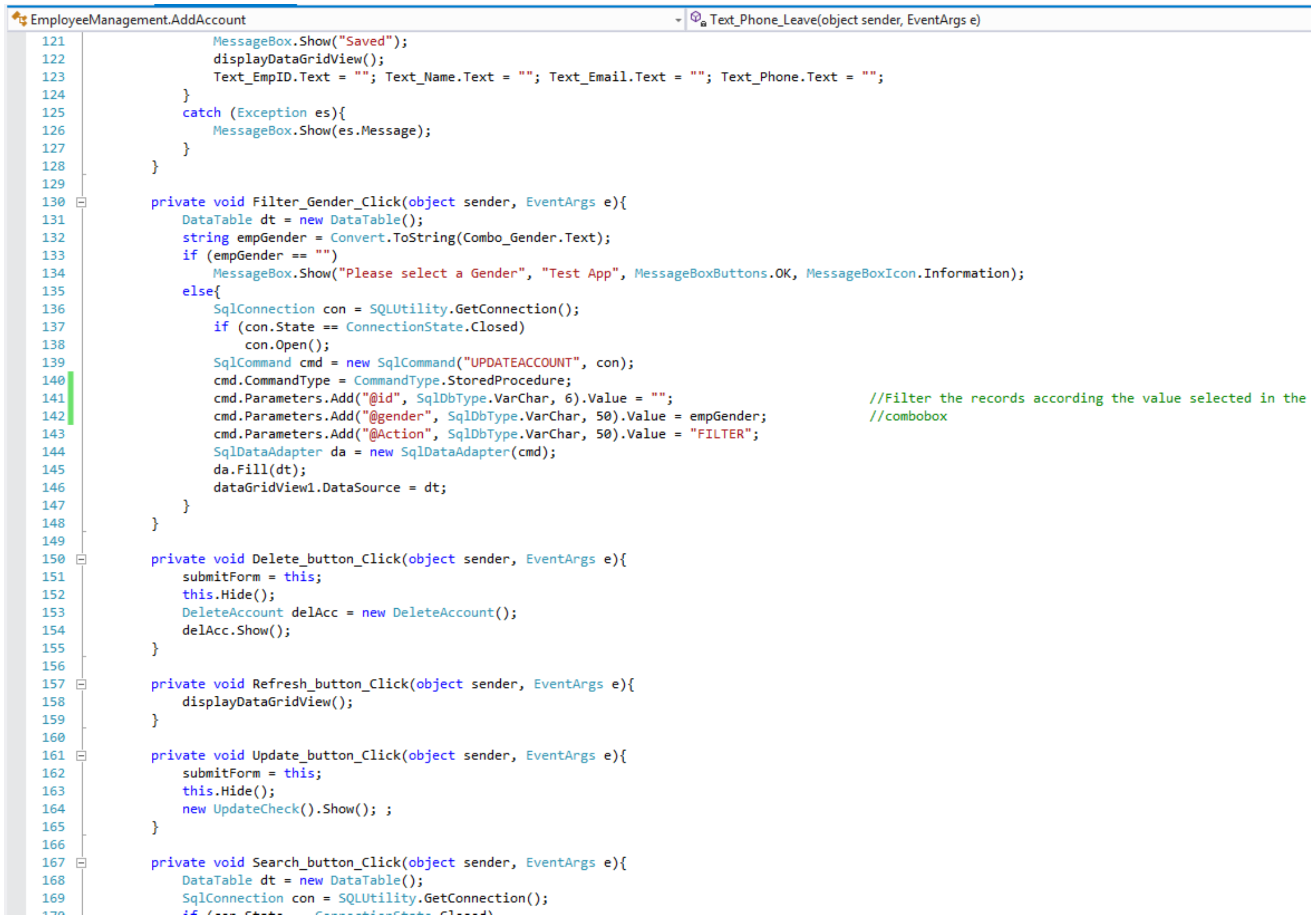
The grid view uses the SQL Adapter to access the database values. The object of the class SQL Adapter is then used with the function .fill ( ) to populate the grid view with the entries of the selected database. It then uses the .DataSource on the grid view to pass on the values.

iv. **Delete an Account:**

An account is deleted when the user clicks on the delete button on the form. The delete form opens and the user enters the employee ID corresponding to the record that is to be deleted.

If the user enters the wrong employee ID, then there is an error box displayed and the user is returned to the delete form otherwise there is an information box displaying that the account was deleted successfully. The grid view on refreshing would not show the deleted record.

v. **Filter:**


EmployeeManagement.AddAccount
Text_Phone_Leave(object sender, EventArgs e)

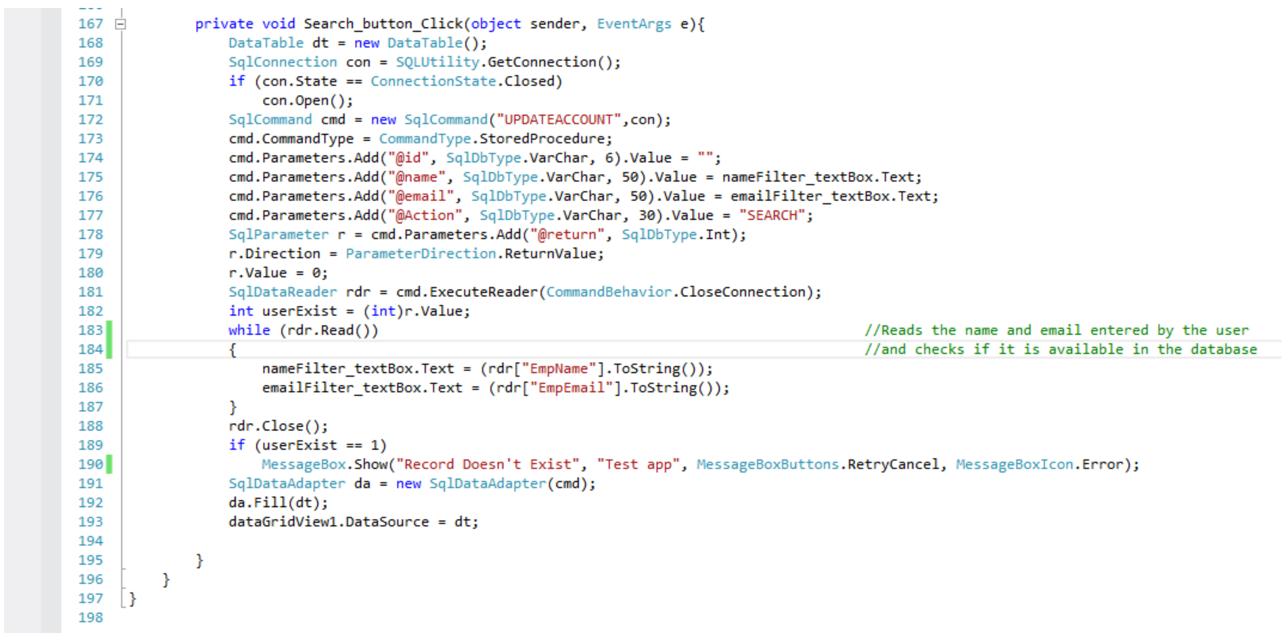
```
                MessageBox.Show("Saved");
                displayDataGridView();
                Text_EmpID.Text = ""; Text_Name.Text = ""; Text_Email.Text = ""; Text_Phone.Text = "";
            }
            catch (Exception es){
                MessageBox.Show(es.Message);
            }
        }

        private void Filter_Gender_Click(object sender, EventArgs e){
            DataTable dt = new DataTable();
            string empGender = Convert.ToString(Combo_Gender.Text);
            if (empGender == "")
                MessageBox.Show("Please select a Gender", "Test App", MessageBoxButtons.OK, MessageBoxIcon.Information);
            else{
                SqlConnection con = SQLUtility.GetConnection();
                if (con.State == ConnectionState.Closed)
                    con.Open();
                SqlCommand cmd = new SqlCommand("UPDATEACCOUNT", con);
                cmd.CommandType = CommandType.StoredProcedure;
                cmd.Parameters.Add("@id", SqlDbType.VarChar, 6).Value = "";                 //Filter the records according the value selected in the
                cmd.Parameters.Add("@gender", SqlDbType.VarChar, 50).Value = empGender;     //combobox
                cmd.Parameters.Add("@Action", SqlDbType.VarChar, 50).Value = "FILTER";
                SqlDataAdapter da = new SqlDataAdapter(cmd);
                da.Fill(dt);
                dataGridView1.DataSource = dt;
            }
        }

        private void Delete_button_Click(object sender, EventArgs e){
            submitForm = this;
            this.Hide();
            DeleteAccount delAcc = new DeleteAccount();
            delAcc.Show();
        }

        private void Refresh_button_Click(object sender, EventArgs e){
            displayDataGridView();
        }

        private void Update_button_Click(object sender, EventArgs e){
            submitForm = this;
            this.Hide();
            new UpdateCheck().Show(); ;
        }

        private void Search_button_Click(object sender, EventArgs e){
            DataTable dt = new DataTable();
            SqlConnection con = SQLUtility.GetConnection();
```

Figure 5: Code for filtering

The filter is done on the records on the basis of the gender selected in the drop down box. If the user doesn’t select a gender, the code generates a message box, prompting the user.

On selection of a gender, the records on the grid view are filtered, and only the records with the respective gender's entries are shown. The stored procedure has the query for filtering the database.

iv. **Search:**



```
        private void Search_button_Click(object sender, EventArgs e){
            DataTable dt = new DataTable();
            SqlConnection con = SQLUtility.GetConnection();
            if (con.State == ConnectionState.Closed)
                con.Open();
            SqlCommand cmd = new SqlCommand("UPDATEACCOUNT",con);
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Parameters.Add("@id", SqlDbType.VarChar, 6).Value = "";
            cmd.Parameters.Add("@name", SqlDbType.VarChar, 50).Value = nameFilter_textBox.Text;
            cmd.Parameters.Add("@email", SqlDbType.VarChar, 50).Value = emailFilter_textBox.Text;
            cmd.Parameters.Add("@Action", SqlDbType.VarChar, 30).Value = "SEARCH";
            SqlParameter r = cmd.Parameters.Add("@return", SqlDbType.Int);
            r.Direction = ParameterDirection.ReturnValue;
            r.Value = 0;
            SqlDataReader rdr = cmd.ExecuteReader(CommandBehavior.CloseConnection);
            int userExist = (int)r.Value;
            while (rdr.Read())                                                          //Reads the name and email entered by the user
            {                                                                           //and checks if it is available in the database
                nameFilter_textBox.Text = (rdr["EmpName"].ToString());
                emailFilter_textBox.Text = (rdr["EmpEmail"].ToString());
            }
            rdr.Close();
            if (userExist == 1)
                MessageBox.Show("Record Doesn't Exist", "Test app", MessageBoxButtons.RetryCancel, MessageBoxIcon.Error);
            SqlDataAdapter da = new SqlDataAdapter(cmd);
            da.Fill(dt);
            dataGridView1.DataSource = dt;

        }
    }
}
```

Figure 6: Code for searching

The search is done on the records on the basis of the name and the email entered in the text boxes. If the user doesn't enter either of the two, the name or the email, the code generates a message box, prompting the user to fill both text boxes.

On entry of a name and an email, the records on the grid view are searched, and only the record with the respective name and email id are shown. The stored procedure has the query for the database.

If the user enters a wrong name or a wrong email id or both, there will be a message box showing an error, as there would be no results after firing the query in the database.

2. **Calculator (Windows Form):**

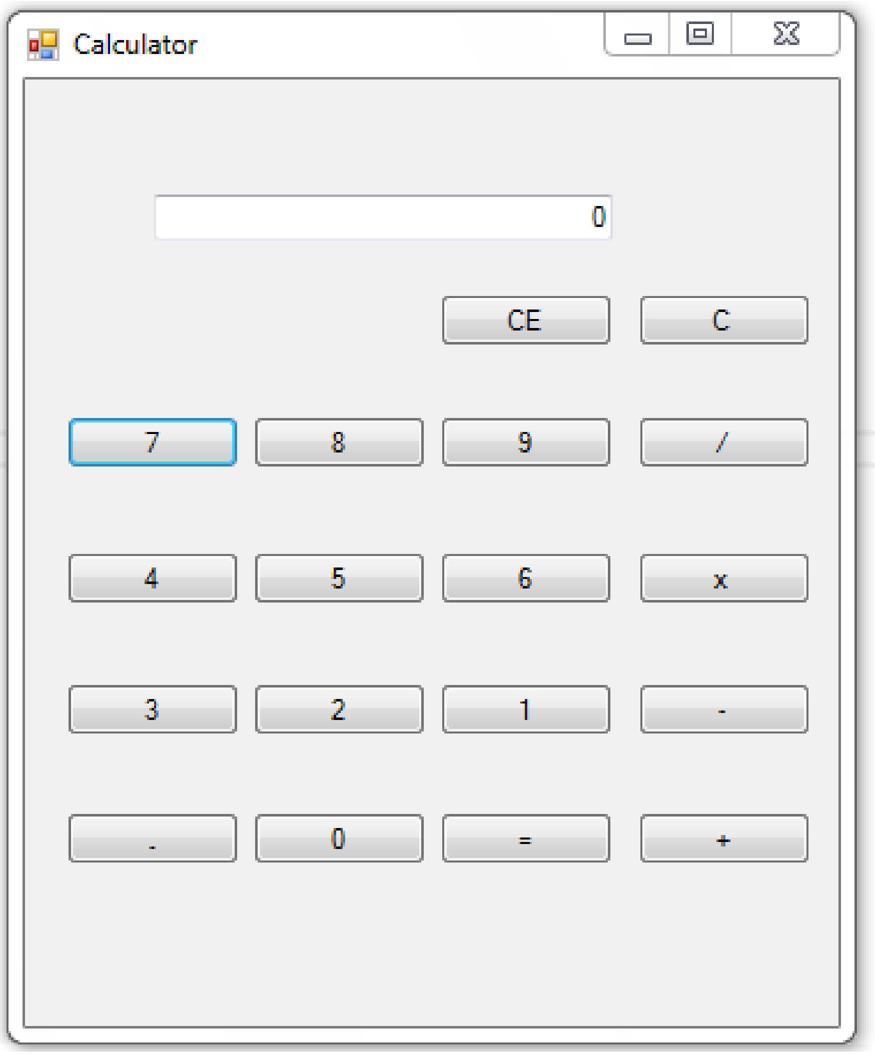


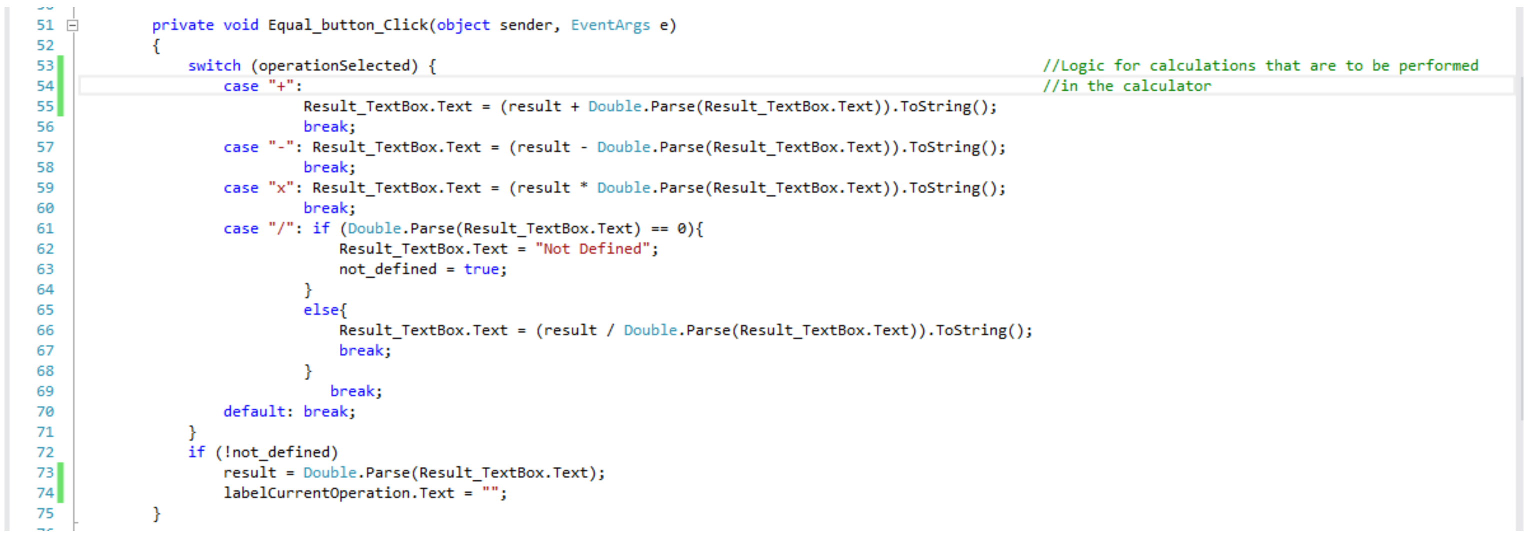
Figure 7: Calculator Interface

Figure 7 shows the interface made in the designer. The buttons are dragged into the form with the help of the toolbox and the properties can be changed in the code.

The buttons have the ability to display whatever their buttons display. The CE and C are used for clearing the whole calculation or just clearing a single digit.

For the operating buttons, there is a switch case. When the user clicks on any of the operators, it goes in the switch case ladder. Figure 8 shows the different functionalities of the operators entered.

For eg. If the user enters “/”, the switch case divides the previous input to the input entered after clicking the operator. If the input is zero, then the according to the ‘if’ case shown in the code, the result will be shown as “Not Defined”, after the equal to button is clicked.

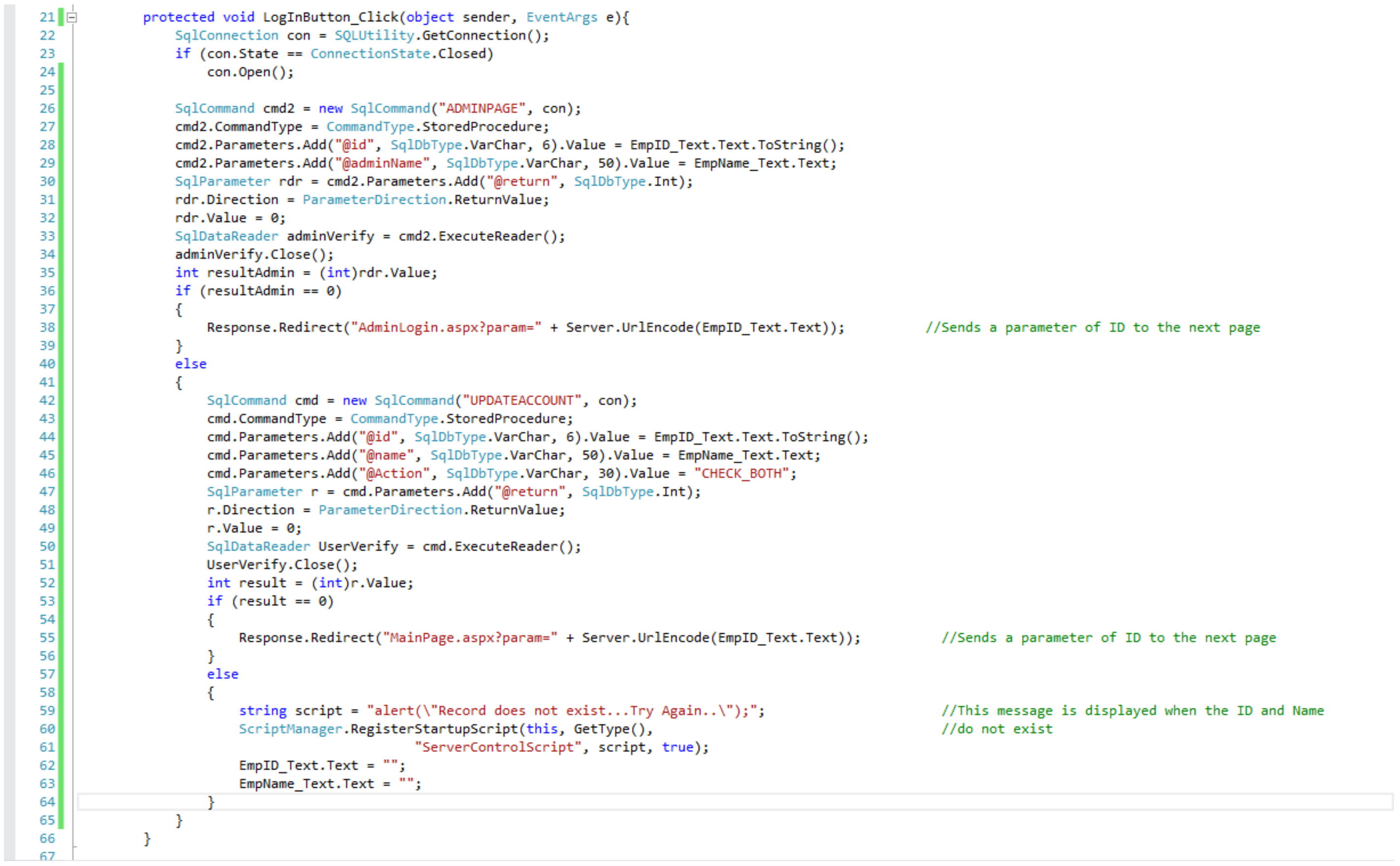


```
        private void Equal_button_Click(object sender, EventArgs e)
        {
            switch (operationSelected) {                                                                              //Logic for calculations that are to be performed
                case "+":                                                                                             //in the calculator
                        Result_TextBox.Text = (result + Double.Parse(Result_TextBox.Text)).ToString();
                        break;
                case "-": Result_TextBox.Text = (result - Double.Parse(Result_TextBox.Text)).ToString();
                        break;
                case "x": Result_TextBox.Text = (result * Double.Parse(Result_TextBox.Text)).ToString();
                        break;
                case "/": if (Double.Parse(Result_TextBox.Text) == 0){
                            Result_TextBox.Text = "Not Defined";
                            not_defined = true;
                        }
                        else{
                            Result_TextBox.Text = (result / Double.Parse(Result_TextBox.Text)).ToString();
                            break;
                        }
                           break;
                default: break;
            }
            if (!not_defined)
                result = Double.Parse(Result_TextBox.Text);
                labelCurrentOperation.Text = "";
        }
```

Figure 8: Calculator operator logic

3. **Website for Employee Database Management (ASP.NET):**

i. **Login Page Code:**



```
        protected void LogInButton_Click(object sender, EventArgs e){
            SqlConnection con = SQLUtility.GetConnection();
            if (con.State == ConnectionState.Closed)
                con.Open();

            SqlCommand cmd2 = new SqlCommand("ADMINPAGE", con);
            cmd2.CommandType = CommandType.StoredProcedure;
            cmd2.Parameters.Add("@id", SqlDbType.VarChar, 6).Value = EmpID_Text.Text.ToString();
            cmd2.Parameters.Add("@adminName", SqlDbType.VarChar, 50).Value = EmpName_Text.Text;
            SqlParameter rdr = cmd2.Parameters.Add("@return", SqlDbType.Int);
            rdr.Direction = ParameterDirection.ReturnValue;
            rdr.Value = 0;
            SqlDataReader adminVerify = cmd2.ExecuteReader();
            adminVerify.Close();
            int resultAdmin = (int)rdr.Value;
            if (resultAdmin == 0)
            {
                Response.Redirect("AdminLogin.aspx?param=" + Server.UrlEncode(EmpID_Text.Text));          //Sends a parameter of ID to the next page
            }
            else
            {
                SqlCommand cmd = new SqlCommand("UPDATEACCOUNT", con);
                cmd.CommandType = CommandType.StoredProcedure;
                cmd.Parameters.Add("@id", SqlDbType.VarChar, 6).Value = EmpID_Text.Text.ToString();
                cmd.Parameters.Add("@name", SqlDbType.VarChar, 50).Value = EmpName_Text.Text;
                cmd.Parameters.Add("@Action", SqlDbType.VarChar, 30).Value = "CHECK_BOTH";
                SqlParameter r = cmd.Parameters.Add("@return", SqlDbType.Int);
                r.Direction = ParameterDirection.ReturnValue;
                r.Value = 0;
                SqlDataReader UserVerify = cmd.ExecuteReader();
                UserVerify.Close();
                int result = (int)r.Value;
                if (result == 0)
                {
                    Response.Redirect("MainPage.aspx?param=" + Server.UrlEncode(EmpID_Text.Text));          //Sends a parameter of ID to the next page
                }
                else
                {
                    string script = "alert(\"Record does not exist...Try Again..\");";                        //This message is displayed when the ID and Name
                    ScriptManager.RegisterStartupScript(this, GetType(),                                       //do not exist
                                  "ServerControlScript", script, true);
                    EmpID_Text.Text = "";
                    EmpName_Text.Text = "";
                }
            }
        }
```

Figure 9: Login Page Code

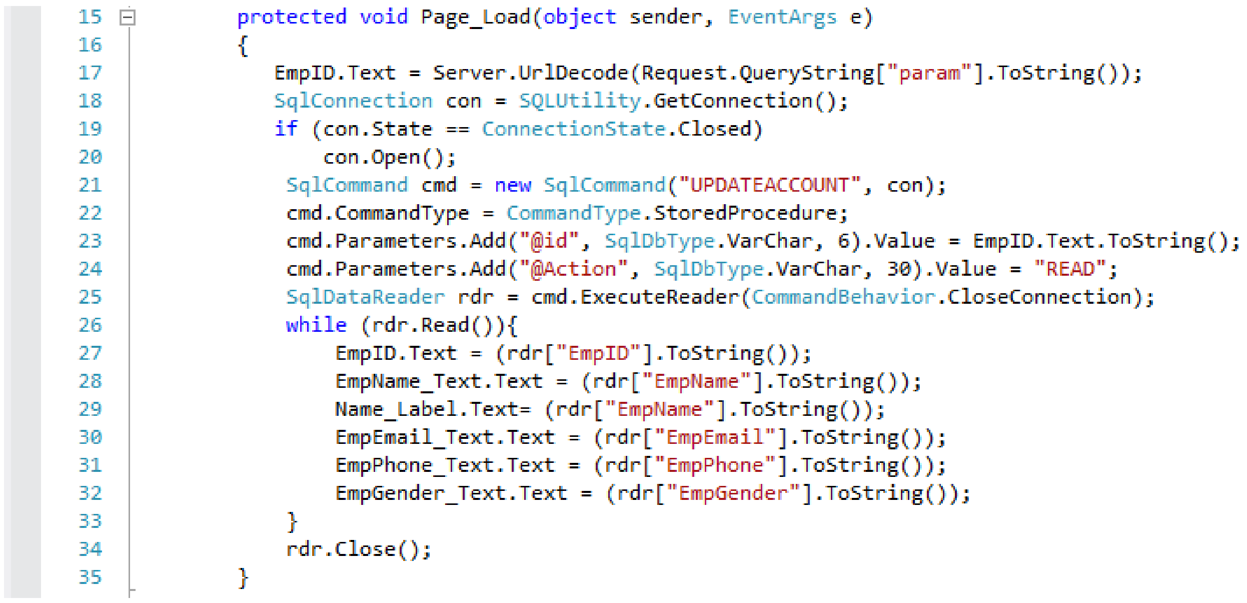
The login page provides the entry for the employee ID and Username. The login is for both, the user and the admin of the site. If the username and ID is that of the admin, the person will be directed to the admin page, if the username and ID is that of an employee, it will direct the employee to their record details.

If both of the cases are false, then there will be an error message displayed, as none of the entries would match. The Response.Redirect(). Directs the page to the page mentioned inside the function and passes the employee ID as a parameter to the redirected page.

If it is the admin who logs in, then the admin is directed to the ‘AdminLogin.aspx’ otherwise, if the record exists then the user is direct to ‘MainPage.aspx’.

The Jquery in this code is used to display the alert message when the record doesn’t exist.

ii. **Main User Page Code:**



```
        protected void Page_Load(object sender, EventArgs e)
        {
           EmpID.Text = Server.UrlDecode(Request.QueryString["param"].ToString());
           SqlConnection con = SQLUtility.GetConnection();
           if (con.State == ConnectionState.Closed)
               con.Open();
            SqlCommand cmd = new SqlCommand("UPDATEACCOUNT", con);
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Parameters.Add("@id", SqlDbType.VarChar, 6).Value = EmpID.Text.ToString();
            cmd.Parameters.Add("@Action", SqlDbType.VarChar, 30).Value = "READ";
            SqlDataReader rdr = cmd.ExecuteReader(CommandBehavior.CloseConnection);
            while (rdr.Read()){
                EmpID.Text = (rdr["EmpID"].ToString());
                EmpName_Text.Text = (rdr["EmpName"].ToString());
                Name_Label.Text= (rdr["EmpName"].ToString());
                EmpEmail_Text.Text = (rdr["EmpEmail"].ToString());
                EmpPhone_Text.Text = (rdr["EmpPhone"].ToString());
                EmpGender_Text.Text = (rdr["EmpGender"].ToString());
            }
            rdr.Close();
        }
```

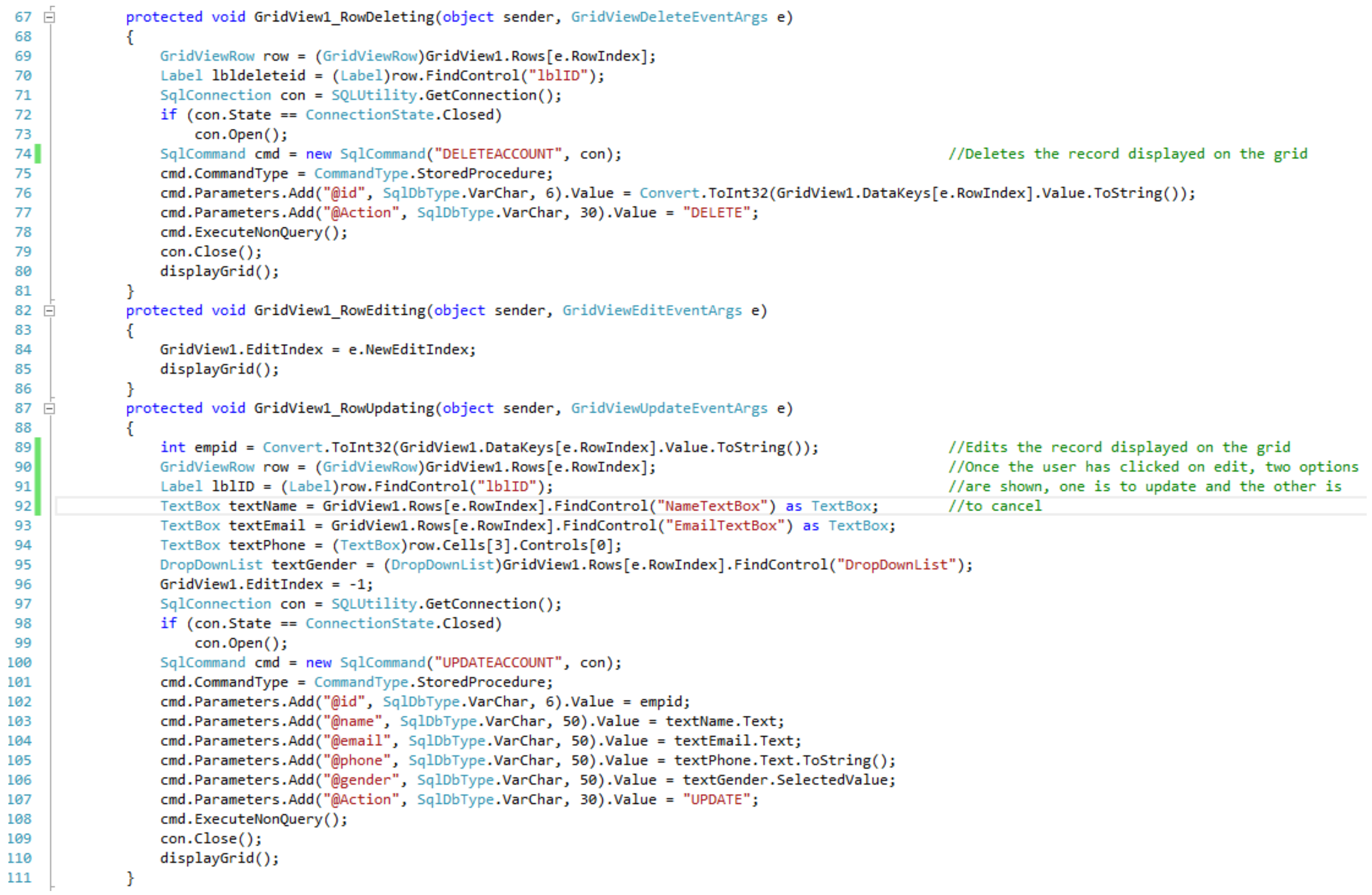
Figure 10: Main Page Code

This function is executed as soon as the employee ID and name matches with the one in the database. Page_Load() refers to the values displayed on the web form as soon as the page is accessed.

The rdr is an object of the SqlDataReader class which enables C# to read the data from the database corresponding to the employee ID and name. The .ToString() function converts all the data to string so that it can be displayed on the webpage.

The main user page offers the user to delete, update and sign out from their account securely. The validations used on the page are done with the help of html and javascript. After registration, the user is directed to the same page that is the main user page.

iii. **Admin Page Code:**



```
protected void GridView1_RowDeleting(object sender, GridViewDeleteEventArgs e)
{
    GridViewRow row = (GridViewRow)GridView1.Rows[e.RowIndex];
    Label lbldeleteid = (Label)row.FindControl("lblID");
    SqlConnection con = SQLUtility.GetConnection();
    if (con.State == ConnectionState.Closed)
        con.Open();
    SqlCommand cmd = new SqlCommand("DELETEACCOUNT", con);                         //Deletes the record displayed on the grid
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.Parameters.Add("@id", SqlDbType.VarChar, 6).Value = Convert.ToInt32(GridView1.DataKeys[e.RowIndex].Value.ToString());
    cmd.Parameters.Add("@Action", SqlDbType.VarChar, 30).Value = "DELETE";
    cmd.ExecuteNonQuery();
    con.Close();
    displayGrid();
}
protected void GridView1_RowEditing(object sender, GridViewEditEventArgs e)
{
    GridView1.EditIndex = e.NewEditIndex;
    displayGrid();
}
protected void GridView1_RowUpdating(object sender, GridViewUpdateEventArgs e)
{
    int empid = Convert.ToInt32(GridView1.DataKeys[e.RowIndex].Value.ToString());          //Edits the record displayed on the grid
    GridViewRow row = (GridViewRow)GridView1.Rows[e.RowIndex];                             //Once the user has clicked on edit, two options
    Label lblID = (Label)row.FindControl("lblID");                                        //are shown, one is to update and the other is
    TextBox textName = GridView1.Rows[e.RowIndex].FindControl("NameTextBox") as TextBox;   //to cancel
    TextBox textEmail = GridView1.Rows[e.RowIndex].FindControl("EmailTextBox") as TextBox;
    TextBox textPhone = (TextBox)row.Cells[3].Controls[0];
    DropDownList textGender = (DropDownList)GridView1.Rows[e.RowIndex].FindControl("DropDownList");
    GridView1.EditIndex = -1;
    SqlConnection con = SQLUtility.GetConnection();
    if (con.State == ConnectionState.Closed)
        con.Open();
    SqlCommand cmd = new SqlCommand("UPDATEACCOUNT", con);
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.Parameters.Add("@id", SqlDbType.VarChar, 6).Value = empid;
    cmd.Parameters.Add("@name", SqlDbType.VarChar, 50).Value = textName.Text;
    cmd.Parameters.Add("@email", SqlDbType.VarChar, 50).Value = textEmail.Text;
    cmd.Parameters.Add("@phone", SqlDbType.VarChar, 50).Value = textPhone.Text.ToString();
    cmd.Parameters.Add("@gender", SqlDbType.VarChar, 50).Value = textGender.SelectedValue;
    cmd.Parameters.Add("@Action", SqlDbType.VarChar, 30).Value = "UPDATE";
    cmd.ExecuteNonQuery();
    con.Close();
    displayGrid();
}
```

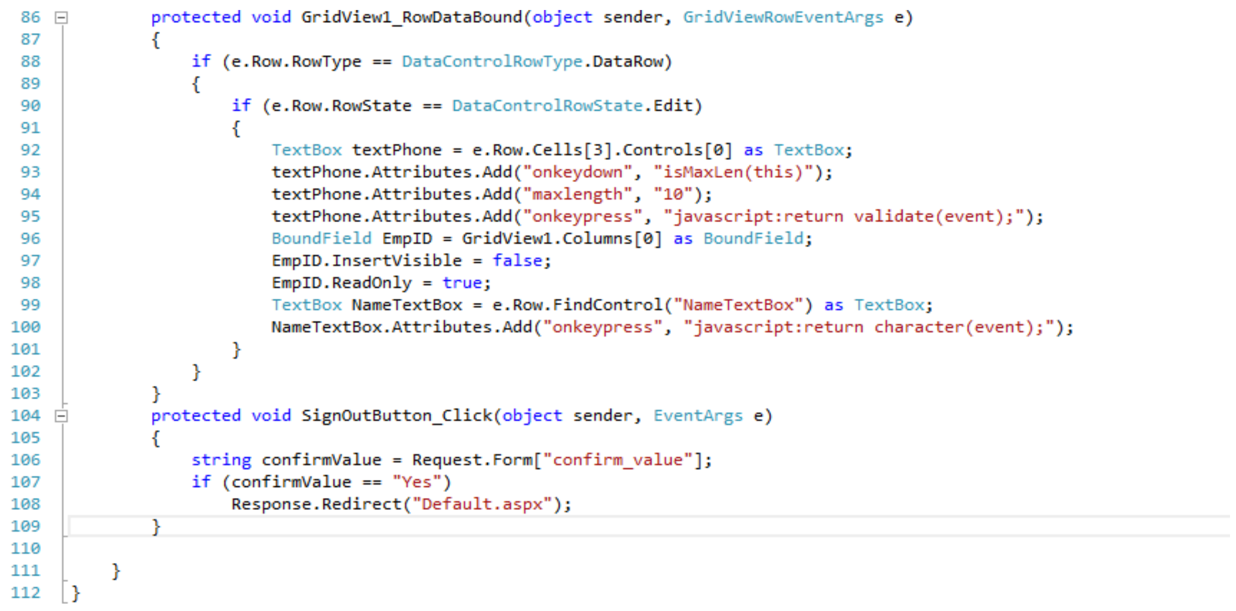
Figure 11: Admin Grid Page Code

The admin page can be accessed only through the admin employee ID and name; any other entry would either be an existing employee's account details or a false entry which would then lead to an error box.

The admin page offers the admin to delete and update all the employee's records. There is a grid displayed on the admin page which is populated with all the employee database information.

On clicking on the edit hyperlink, the admin would be showed two more hyperlinks, allowing him/her to cancel or update the account. On clicking the delete hyperlink, the admin would be able to delete an account.

The admin can leave the page, by clicking on the sign out button which will direct him/ her to the default page or the home page, after asking for a confirmation to log out.

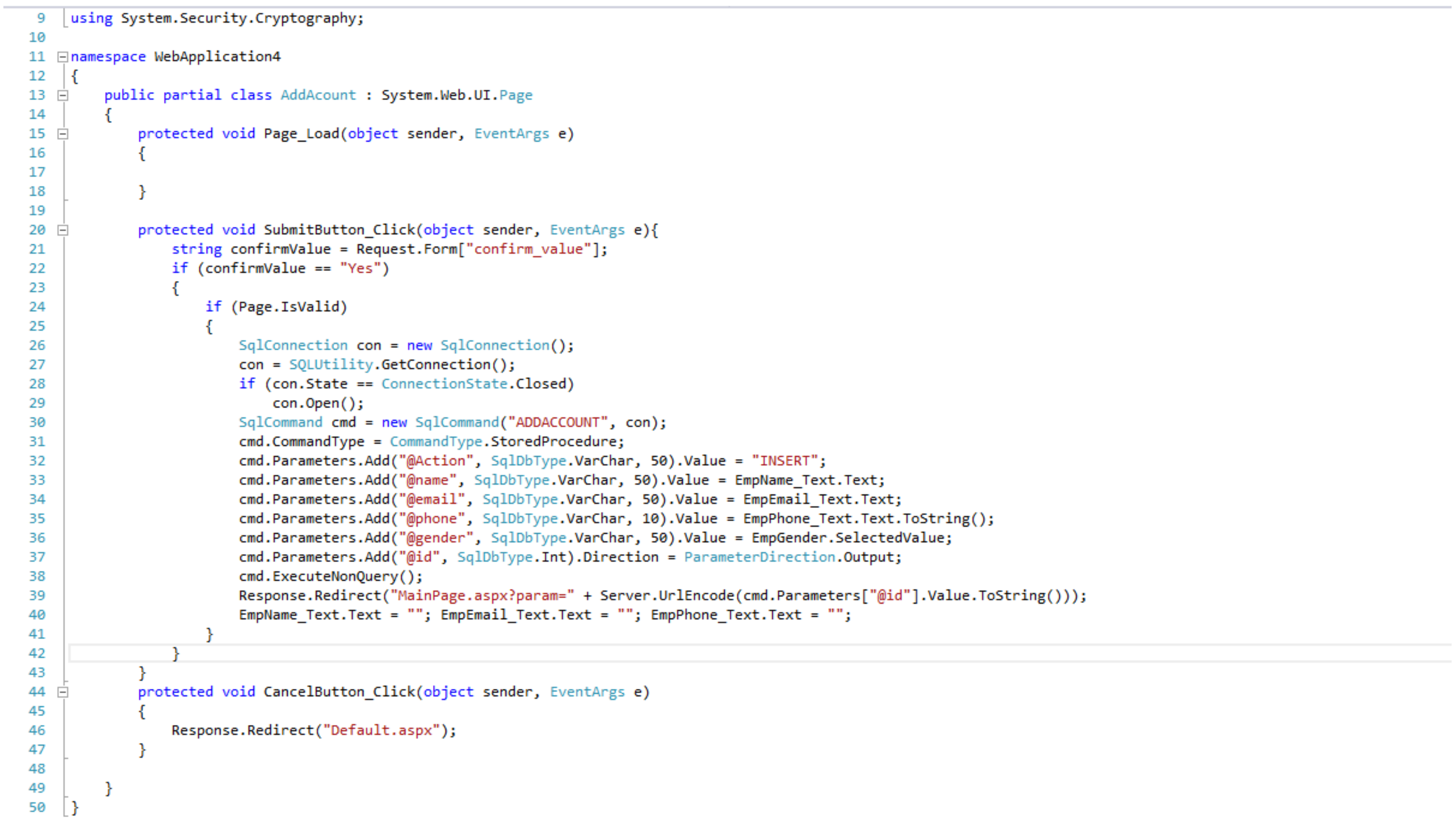


```
        protected void GridView1_RowDataBound(object sender, GridViewRowEventArgs e)
        {
            if (e.Row.RowType == DataControlRowType.DataRow)
            {
                if (e.Row.RowState == DataControlRowState.Edit)
                {
                    TextBox textPhone = e.Row.Cells[3].Controls[0] as TextBox;
                    textPhone.Attributes.Add("onkeydown", "isMaxLen(this)");
                    textPhone.Attributes.Add("maxlength", "10");
                    textPhone.Attributes.Add("onkeypress", "javascript:return validate(event);");
                    BoundField EmpID = GridView1.Columns[0] as BoundField;
                    EmpID.InsertVisible = false;
                    EmpID.ReadOnly = true;
                    TextBox NameTextBox = e.Row.FindControl("NameTextBox") as TextBox;
                    NameTextBox.Attributes.Add("onkeypress", "javascript:return character(event);");
                }
            }
        }
        protected void SignOutButton_Click(object sender, EventArgs e)
        {
            string confirmValue = Request.Form["confirm_value"];
            if (confirmValue == "Yes")
                Response.Redirect("Default.aspx");
        }

    }
}
```

Figure 12: Admin sign out and validation page Code

As shown in Figure 12, when the ‘confirmValue’ would be ‘Yes’, that is , only if the admin clicks on Yes, he/she will be directed to ‘Default.aspx’ which is the home page. The admin can only make valid changes to the account, that is, he/she cannot leave the name empty, or enter an invalid phone number.

iv. **Registration Page Code:**



```
using System.Security.Cryptography;

namespace WebApplication4
{
    public partial class AddAcount : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        protected void SubmitButton_Click(object sender, EventArgs e){
            string confirmValue = Request.Form["confirm_value"];
            if (confirmValue == "Yes")
            {
                if (Page.IsValid)
                {
                    SqlConnection con = new SqlConnection();
                    con = SQLUtility.GetConnection();
                    if (con.State == ConnectionState.Closed)
                        con.Open();
                    SqlCommand cmd = new SqlCommand("ADDACCOUNT", con);
                    cmd.CommandType = CommandType.StoredProcedure;
                    cmd.Parameters.Add("@Action", SqlDbType.VarChar, 50).Value = "INSERT";
                    cmd.Parameters.Add("@name", SqlDbType.VarChar, 50).Value = EmpName_Text.Text;
                    cmd.Parameters.Add("@email", SqlDbType.VarChar, 50).Value = EmpEmail_Text.Text;
                    cmd.Parameters.Add("@phone", SqlDbType.VarChar, 10).Value = EmpPhone_Text.Text.ToString();
                    cmd.Parameters.Add("@gender", SqlDbType.VarChar, 50).Value = EmpGender.SelectedValue;
                    cmd.Parameters.Add("@id", SqlDbType.Int).Direction = ParameterDirection.Output;
                    cmd.ExecuteNonQuery();
                    Response.Redirect("MainPage.aspx?param=" + Server.UrlEncode(cmd.Parameters["@id"].Value.ToString()));
                    EmpName_Text.Text = ""; EmpEmail_Text.Text = ""; EmpPhone_Text.Text = "";
                }
            }
        }
        protected void CancelButton_Click(object sender, EventArgs e)
        {
            Response.Redirect("Default.aspx");
        }

    }
}
```

Figure 13: Registration Code

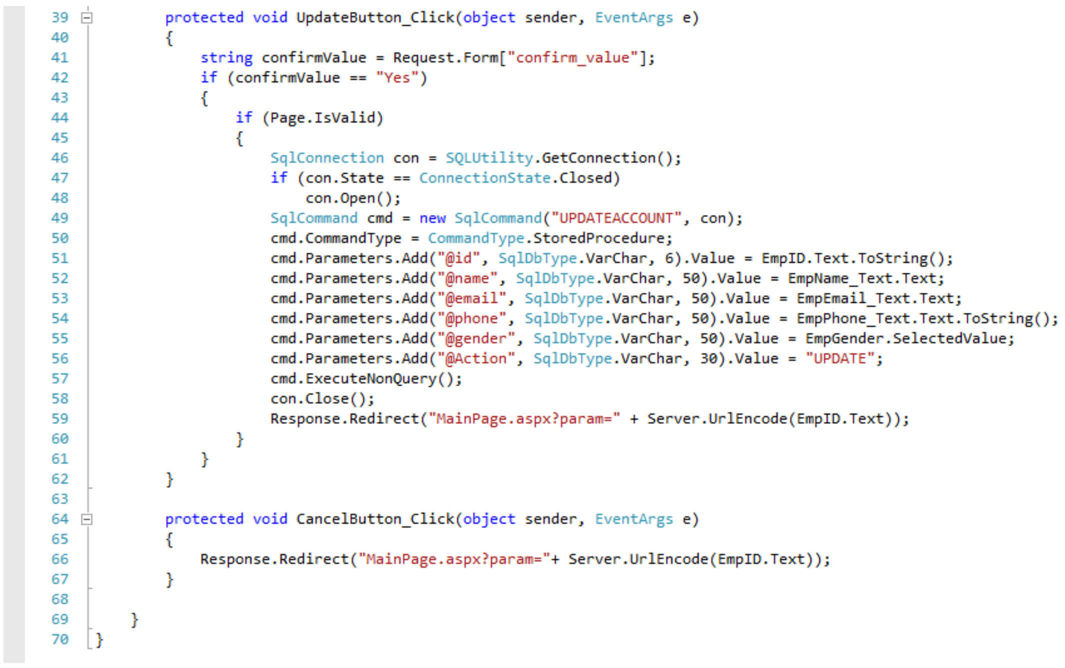
If a user doesn’t have an account in the database, they need to register themselves. For that, they can click on the ‘Register’ button, which will generate a buttonClick event and direct them to the ‘AddAccount.aspx’ page.

Here, the user will have to enter his/her name, email, phone number and gender. Some fields are mandatory; hence, if the user doesn’t fill them, there will be an asterisk with an error message displayed with that particular field.

The phone number has to be of 10 digits only, not less, not more. The user can’t fill more, because of a jquery written inside the html which will not allow the user to enter more numbers from the keyboard. If the number of digits is less, again an error message will be shown through the html tag of ‘RegularExpressionValidator’ for entering a valid number.

There will be no textbox for entering the employee ID, as it is automatically generated for each user that registers. The ID is shown as soon as the user has registered, there is also a message, telling the user to remember their ID so that they can login the next time. The user can click on the ‘submit’ button for completing their registration. They will be directed to the main page, with the details that they fill and their appointed employee ID.

v. **Update Account Page Code:**



```
        protected void UpdateButton_Click(object sender, EventArgs e)
        {
            string confirmValue = Request.Form["confirm_value"];
            if (confirmValue == "Yes")
            {
                if (Page.IsValid)
                {
                    SqlConnection con = SQLUtility.GetConnection();
                    if (con.State == ConnectionState.Closed)
                        con.Open();
                    SqlCommand cmd = new SqlCommand("UPDATEACCOUNT", con);
                    cmd.CommandType = CommandType.StoredProcedure;
                    cmd.Parameters.Add("@id", SqlDbType.VarChar, 6).Value = EmpID.Text.ToString();
                    cmd.Parameters.Add("@name", SqlDbType.VarChar, 50).Value = EmpName_Text.Text;
                    cmd.Parameters.Add("@email", SqlDbType.VarChar, 50).Value = EmpEmail_Text.Text;
                    cmd.Parameters.Add("@phone", SqlDbType.VarChar, 50).Value = EmpPhone_Text.Text.ToString();
                    cmd.Parameters.Add("@gender", SqlDbType.VarChar, 50).Value = EmpGender.SelectedValue;
                    cmd.Parameters.Add("@Action", SqlDbType.VarChar, 30).Value = "UPDATE";
                    cmd.ExecuteNonQuery();
                    con.Close();
                    Response.Redirect("MainPage.aspx?param=" + Server.UrlEncode(EmpID.Text));
                }
            }
        }

        protected void CancelButton_Click(object sender, EventArgs e)
        {
            Response.Redirect("MainPage.aspx?param="+ Server.UrlEncode(EmpID.Text));
        }

    }
}
```

Figure 14: Code for updating account

The update button on the main page directs the user to the ‘UpdateDetails.aspx’ page. Here he/she can make changes to anything that they wish to, except for the employee ID.

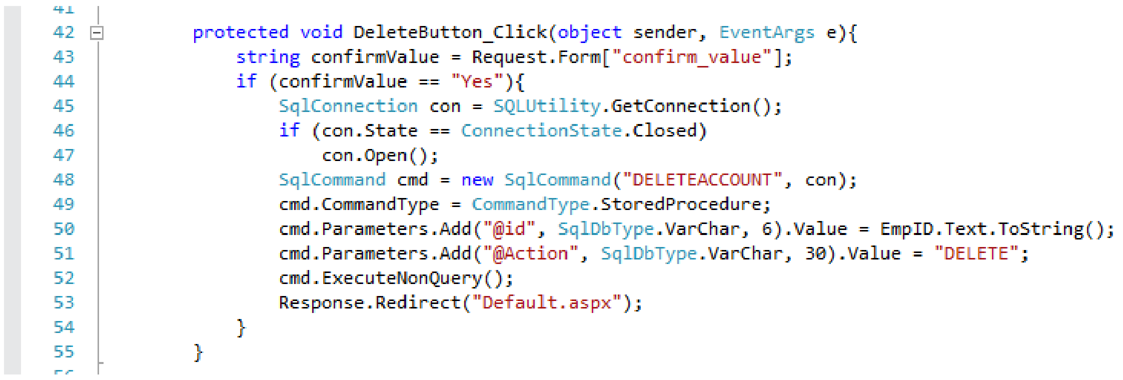
Since the employee ID is automatically generated, no changes can be made to it, so the editable property of the employee ID is disabled by code.

All other entries that are made should be valid, as they are validated through the ‘RequiredFieldValidator’, HTML tag. Also, the phone number field is also validated. All these validations are present in all those pages in which there is registration or updating of the account.

After making the changes and on clicking the update button, the page, again takes confirmation from the user if they wish to make changes to their main account or cancel the submission

vi. **Delete Account Code:**

If the user wishes to delete their account, they can simply do so, by clicking on the delete button displayed on the main page.



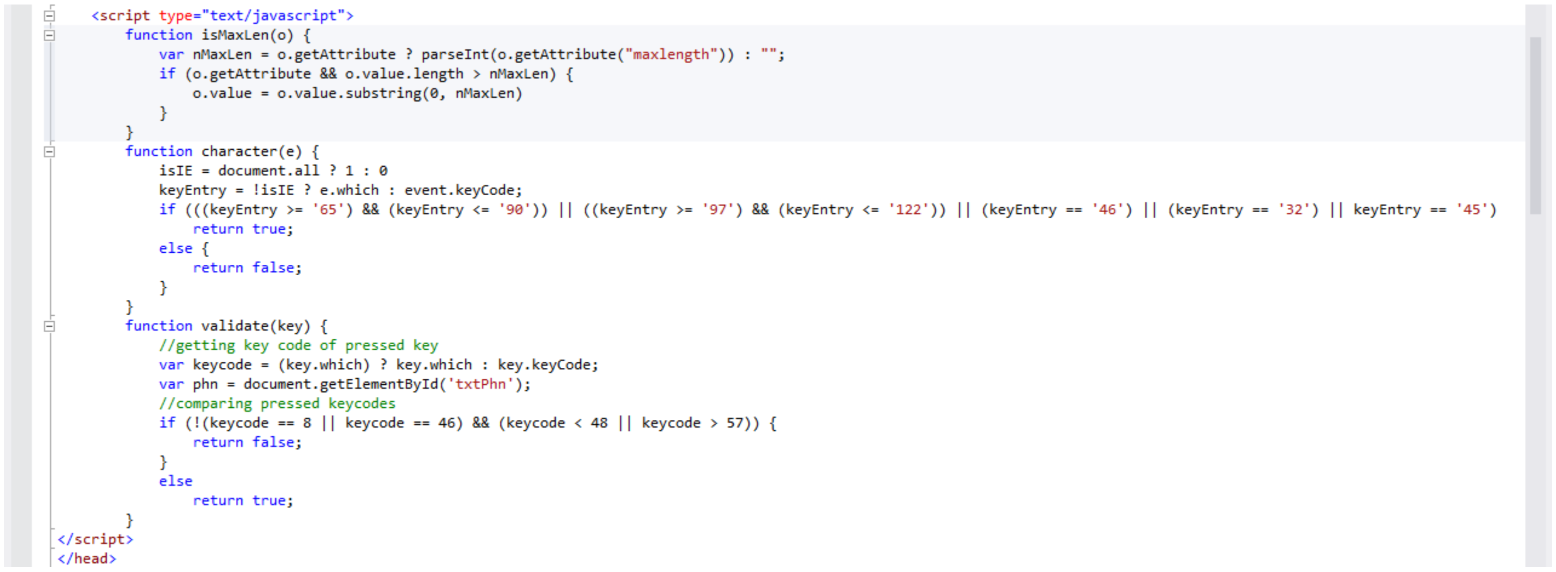
```
        protected void DeleteButton_Click(object sender, EventArgs e){
            string confirmValue = Request.Form["confirm_value"];
            if (confirmValue == "Yes"){
                SqlConnection con = SQLUtility.GetConnection();
                if (con.State == ConnectionState.Closed)
                    con.Open();
                SqlCommand cmd = new SqlCommand("DELETEACCOUNT", con);
                cmd.CommandType = CommandType.StoredProcedure;
                cmd.Parameters.Add("@id", SqlDbType.VarChar, 6).Value = EmpID.Text.ToString();
                cmd.Parameters.Add("@Action", SqlDbType.VarChar, 30).Value = "DELETE";
                cmd.ExecuteNonQuery();
                Response.Redirect("Default.aspx");
            }
        }
```

Figure 15: Code for deleting account

The page prompts the user to confirm the deletion of their account, and only then, does the account get deleted.

The user is then directed to the home page directly after deletion takes place. The change would also be reflected in the admin page, as it is connected to the database.

4. **HTML5 and Javascript Codes Used by the Employee Management Website:**



```
<script type="text/javascript">
    function isMaxLen(o) {
        var nMaxLen = o.getAttribute ? parseInt(o.getAttribute("maxlength")) : "";
        if (o.getAttribute && o.value.length > nMaxLen) {
            o.value = o.value.substring(0, nMaxLen)
        }
    }
    function character(e) {
        isIE = document.all ? 1 : 0
        keyEntry = !isIE ? e.which : event.keyCode;
        if (((keyEntry >= '65') && (keyEntry <= '90')) || ((keyEntry >= '97') && (keyEntry <= '122')) || (keyEntry == '46') || (keyEntry == '32') || keyEntry == '45')
            return true;
        else {
            return false;
        }
    }
    function validate(key) {
        //getting key code of pressed key
        var keycode = (key.which) ? key.which : key.keyCode;
        var phn = document.getElementById('txtPhn');
        //comparing pressed keycodes
        if (!(keycode == 8 || keycode == 46) && (keycode < 48 || keycode > 57)) {
            return false;
        }
        else
            return true;
    }
</script>
</head>
```

Figure 16: Javascript functions for validations

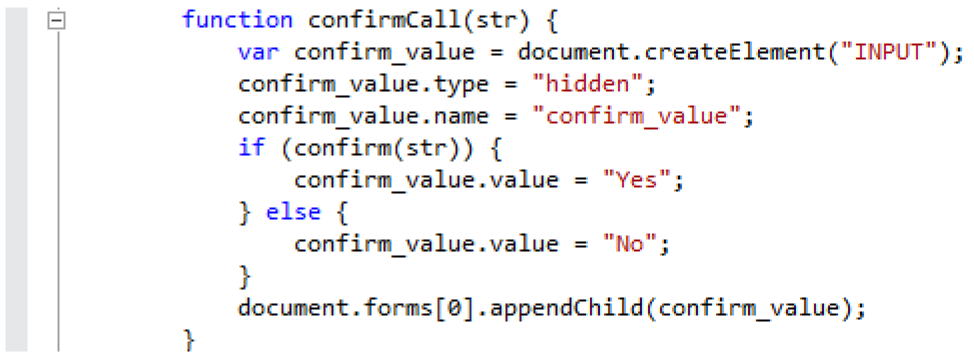
This HTML code shows three functions:

a) isMaxLen()
b) character()
c) validate()

The isMaxLen() function is for the phone number, this javascript function would be invoked for all the phone text boxes in the project. It checks if the number is less than or more than 10.

The character() function is for disabling the user to enter anything else, other than letters and blank spaces.

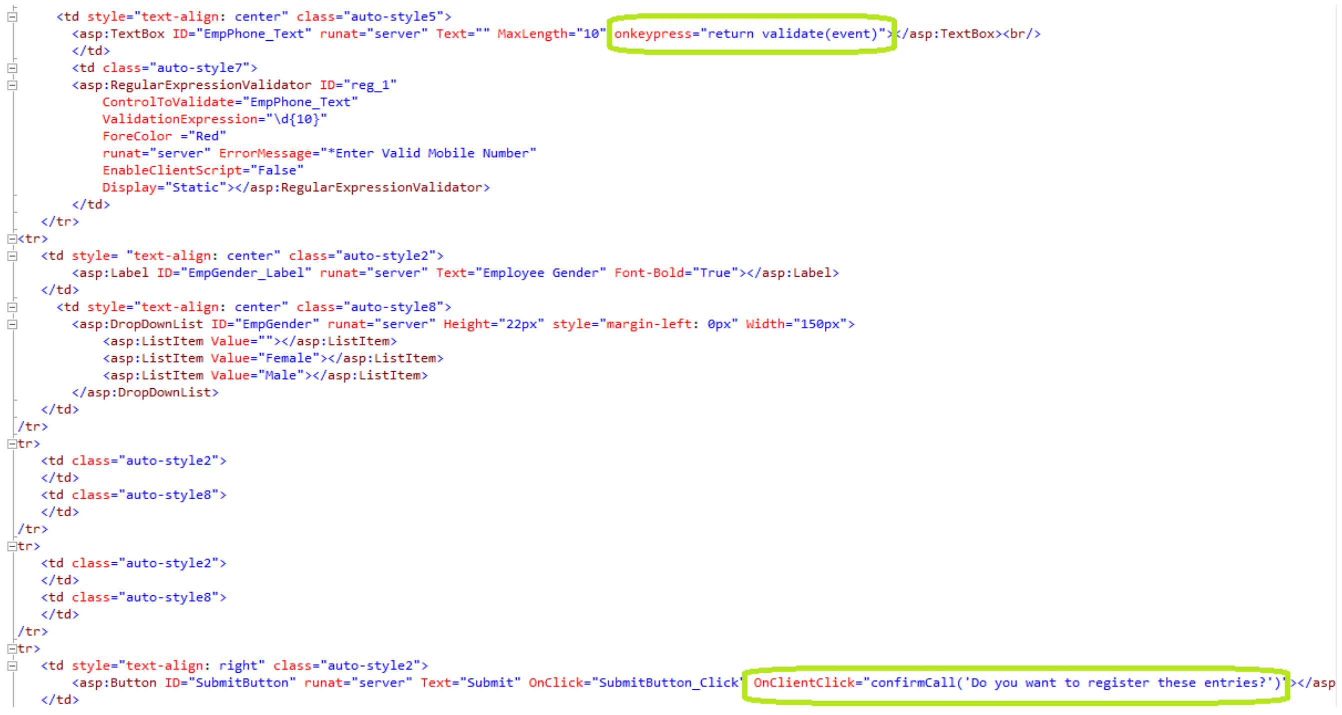
The validate() function is for disabling the user to enter anything else, other than digits.



```
function confirmCall(str) {
    var confirm_value = document.createElement("INPUT");
    confirm_value.type = "hidden";
    confirm_value.name = "confirm_value";
    if (confirm(str)) {
        confirm_value.value = "Yes";
    } else {
        confirm_value.value = "No";
    }
    document.forms[0].appendChild(confirm_value);
}
```

Figure 17: Javascript function for confirmation

The confirmation for all the message boxes is invoked from this javascript function; it shows two buttons, ‘Yes’ and ‘No’. If the user clicks on ‘Yes’ that value is sent to the C# function where it decides the next event that is to take place. Same case is there when the user clicks on ‘No’, the C# function decides the next event.

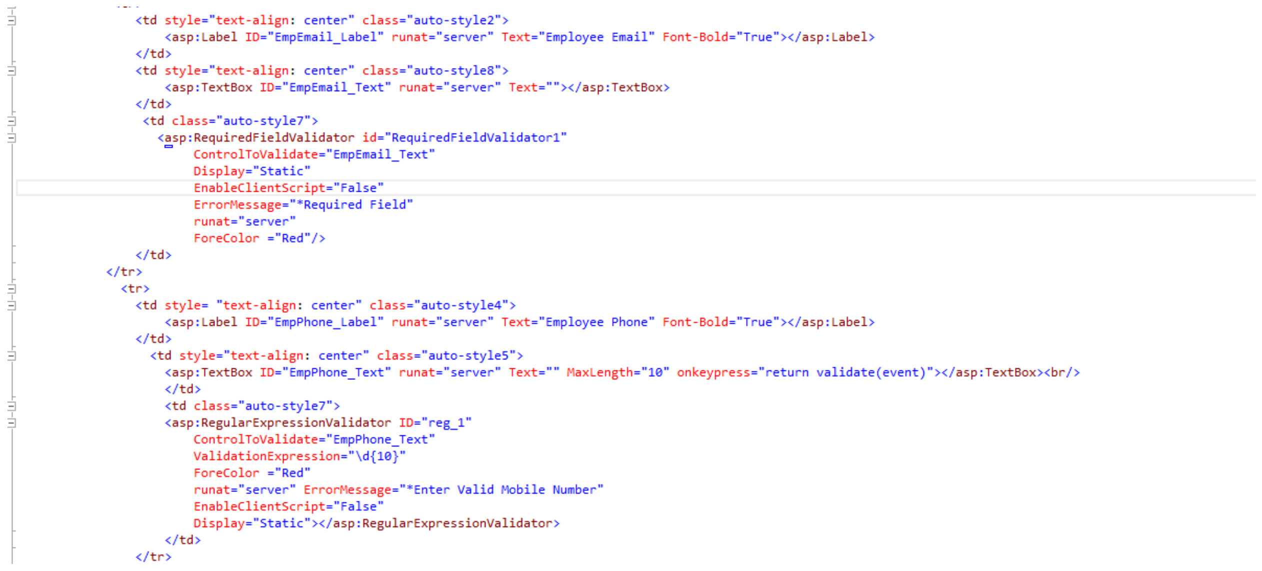


```
      <td style="text-align: center" class="auto-style5">
        <asp:TextBox ID="EmpPhone_Text" runat="server" Text="" MaxLength="10" onkeypress="return validate(event)"></asp:TextBox><br/>
        </td>
        <td class="auto-style7">
        <asp:RegularExpressionValidator ID="reg_1"
            ControlToValidate="EmpPhone_Text"
            ValidationExpression="\d{10}"
            ForeColor ="Red"
            runat="server" ErrorMessage="*Enter Valid Mobile Number"
            EnableClientScript="False"
            Display="Static"></asp:RegularExpressionValidator>
        </td>
    </tr>
<tr>
    <td style= "text-align: center" class="auto-style2">
        <asp:Label ID="EmpGender_Label" runat="server" Text="Employee Gender" Font-Bold="True"></asp:Label>
    </td>
      <td style="text-align: center" class="auto-style8">
        <asp:DropDownList ID="EmpGender" runat="server" Height="22px" style="margin-left: 0px" Width="150px">
            <asp:ListItem Value=""></asp:ListItem>
            <asp:ListItem Value="Female"></asp:ListItem>
            <asp:ListItem Value="Male"></asp:ListItem>
        </asp:DropDownList>
    </td>
/tr>
tr>
    <td class="auto-style2">
    </td>
    <td class="auto-style8">
    </td>
/tr>
tr>
    <td class="auto-style2">
    </td>
    <td class="auto-style8">
    </td>
/tr>
tr>
    <td style="text-align: right" class="auto-style2">
        <asp:Button ID="SubmitButton" runat="server" Text="Submit" OnClick="SubmitButton_Click" OnClientClick="confirmCall('Do you want to register these entries?')"></asp
    </td>
```

Figure 18: HTML5 tags

Figure 18 shows the javascript functions being called from the HTML5 tags. They are passing the respective arguments of the specific datatype required by the functions.

The onkeypress event is for the validate (), character () and isMaxLen () functions, on the key press events, they are invoked. Similarly, the onClientClick event is for passing the specific string or question to be displayed on the message box at the time of confirmation to the function.



```
        <td style="text-align: center" class="auto-style2">
            <asp:Label ID="EmpEmail_Label" runat="server" Text="Employee Email" Font-Bold="True"></asp:Label>
        </td>
        <td style="text-align: center" class="auto-style8">
            <asp:TextBox ID="EmpEmail_Text" runat="server" Text=""></asp:TextBox>
        </td>
         <td class="auto-style7">
           <asp:RequiredFieldValidator id="RequiredFieldValidator1"
                ControlToValidate="EmpEmail_Text"
                Display="Static"
                EnableClientScript="False"
                ErrorMessage="*Required Field"
                runat="server"
                ForeColor ="Red"/>
        </td>
    </tr>
      <tr>
        <td style= "text-align: center" class="auto-style4">
            <asp:Label ID="EmpPhone_Label" runat="server" Text="Employee Phone" Font-Bold="True"></asp:Label>
        </td>
          <td style="text-align: center" class="auto-style5">
            <asp:TextBox ID="EmpPhone_Text" runat="server" Text="" MaxLength="10" onkeypress="return validate(event)"></asp:TextBox><br/>
            </td>
            <td class="auto-style7">
            <asp:RegularExpressionValidator ID="reg_1"
                ControlToValidate="EmpPhone_Text"
                ValidationExpression="\d{10}"
                ForeColor ="Red"
                runat="server" ErrorMessage="*Enter Valid Mobile Number"
                EnableClientScript="False"
                Display="Static"></asp:RegularExpressionValidator>
            </td>
        </tr>
```

Figure 19: HTML5 Validations

The 'RequiredFieldValidator' is basically doing validations against the required fields in the front end. The user must fill in all the mandatory fields; to ensure this we make use of it. It has certain properties that we need to set for it to function the way we want it to, for example: it should show an error message whenever the details are not filled in.

For a login form we want the user name and word, both fields, to be filled in; only then can the user click the button and the server-side processing can occur. If either of the fields is empty then an error message should be invoked or displayed.

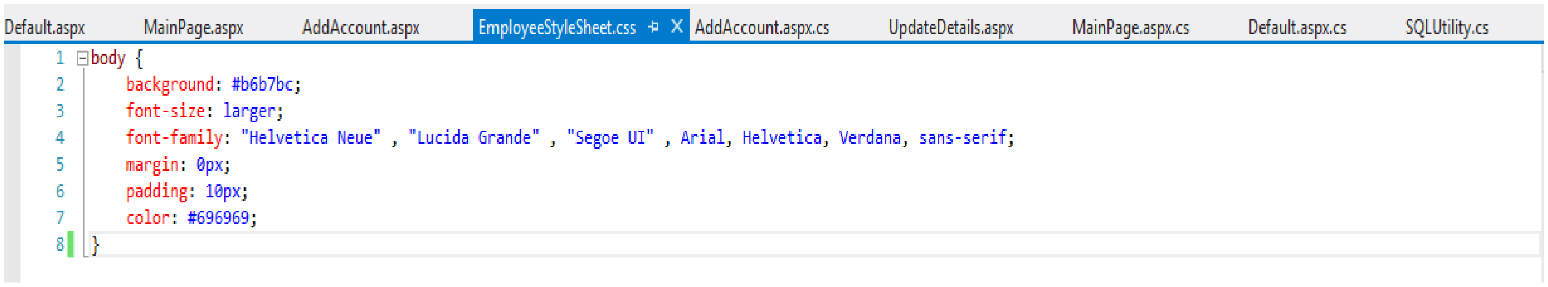
In case of a login form we just want to perform validation against the two required fields (Username and word). In the default case, whenever there is a RequiredFieldValidator inside a page and whenever the button is clicked then it will validate all the controls falling inside that page, but sometimes there exists a necessity for two or more panels and each one should be independently validated when the respective button within that panel is clicked.

To do that we have to register the ‘RequiredFieldValidator’ against the group of controls falling inside that particular panel.

The ‘RequiredExpressionValidator’ is useful when you are creating Regular Expression. This tool will helps in checking syntax of the regular expression typed in.

Also it can be checked whether any particular value matches with the regular expression exactly once or multiple matches are available depending on the option chosen. In this application we have used it for validating phone number. The validation is shown in the code in Figure 19 as ‘\d{10}’, which validates for 10 numbers.

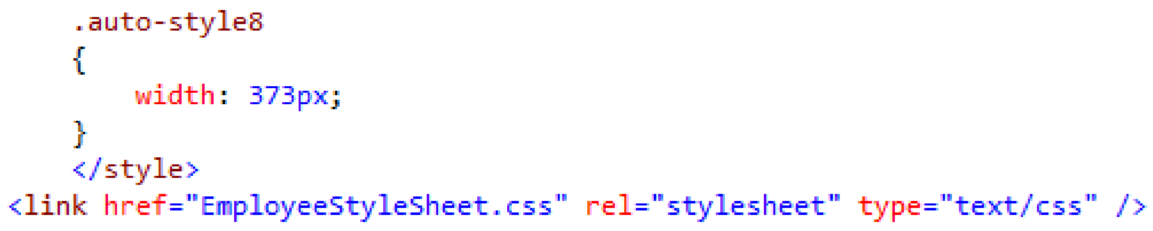
5. **Cascading Style Sheet Used by the Employee Management Website:**



```css
body {
    background: #b6b7bc;
    font-size: larger;
    font-family: "Helvetica Neue" , "Lucida Grande" , "Segoe UI" , Arial, Helvetica, Verdana, sans-serif;
    margin: 0px;
    padding: 10px;
    color: #696969;
}
```

Figure 20: Cascading Style Sheet

The project uses on style sheet common for all the web pages to maintain uniformity. The CSS is used for the making the site look presentable and give it style. The font family property specifies the font for an element and can hold several font names as a "fallback" system. If the browser does not support the first font, it tries the next font.



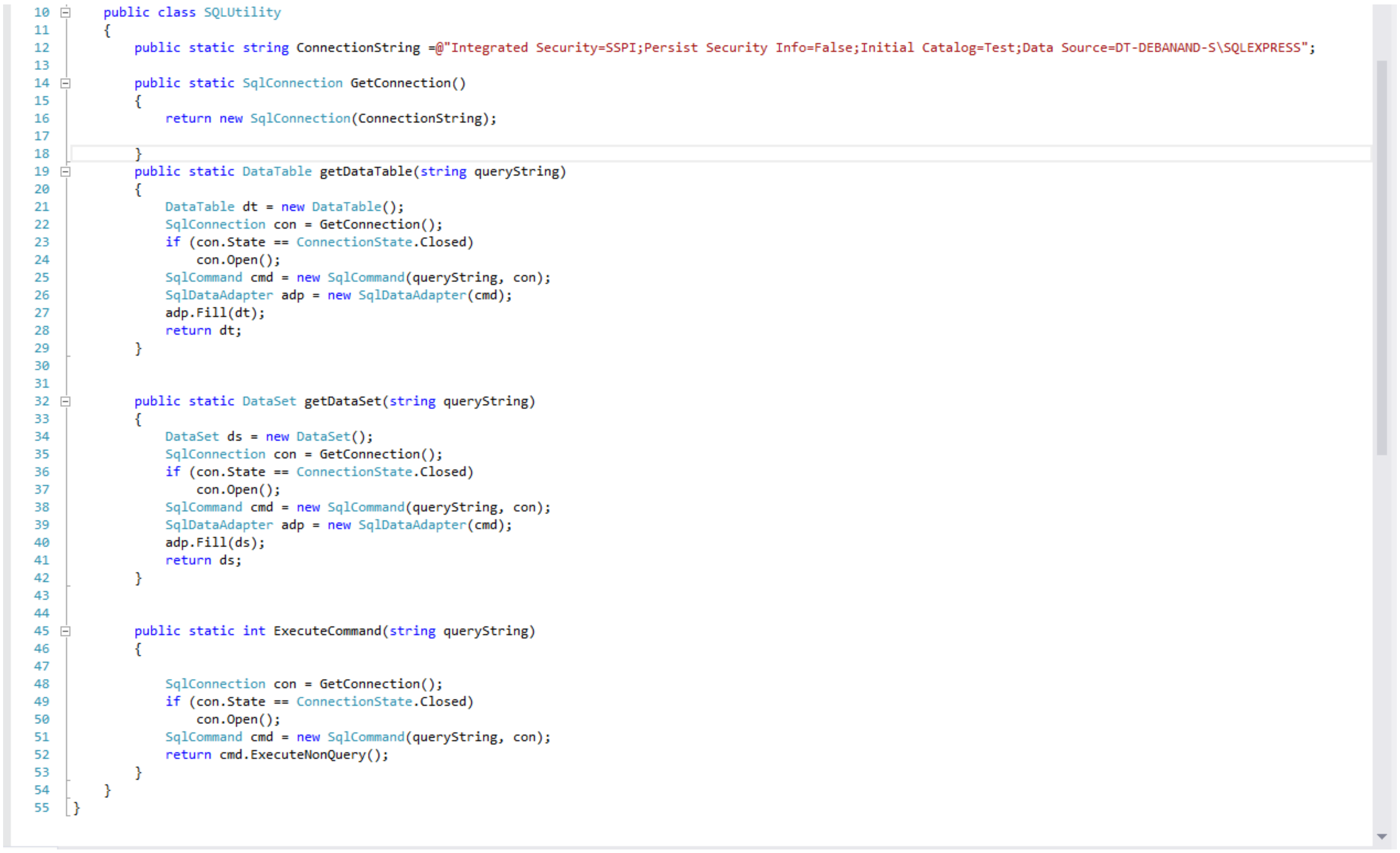
```html
    .auto-style8
    {
        width: 373px;
    }
    </style>
<link href="EmployeeStyleSheet.css" rel="stylesheet" type="text/css" />
```

Figure 21: Link of Cascading Style Sheet

Figure 21 shows the link reference of the CSS to be used in the project. This reference has been written everywhere. Hence, instead of copying the code in Figure 20, only the link has to be copied in the documents where the CSS is to be used.

This ensures reusability and saves time. In case there are any changes, they can be made in one place and they would be reflected in all the web pages.

6. **SQL Utility Class Used by the Employee Management Website and Windows Form:**



```
    public class SQLUtility
    {
        public static string ConnectionString =@"Integrated Security=SSPI;Persist Security Info=False;Initial Catalog=Test;Data Source=DT-DEBANAND-S\SQLEXPRESS";

        public static SqlConnection GetConnection()
        {
            return new SqlConnection(ConnectionString);

        }
        public static DataTable getDataTable(string queryString)
        {
            DataTable dt = new DataTable();
            SqlConnection con = GetConnection();
            if (con.State == ConnectionState.Closed)
                con.Open();
            SqlCommand cmd = new SqlCommand(queryString, con);
            SqlDataAdapter adp = new SqlDataAdapter(cmd);
            adp.Fill(dt);
            return dt;
        }


        public static DataSet getDataSet(string queryString)
        {
            DataSet ds = new DataSet();
            SqlConnection con = GetConnection();
            if (con.State == ConnectionState.Closed)
                con.Open();
            SqlCommand cmd = new SqlCommand(queryString, con);
            SqlDataAdapter adp = new SqlDataAdapter(cmd);
            adp.Fill(ds);
            return ds;
        }


        public static int ExecuteCommand(string queryString)
        {

            SqlConnection con = GetConnection();
            if (con.State == ConnectionState.Closed)
                con.Open();
            SqlCommand cmd = new SqlCommand(queryString, con);
            return cmd.ExecuteNonQuery();
        }
    }
}
```

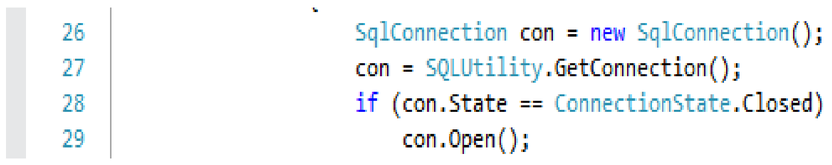
Figure 22: Code for SQL utility class

A SqlConnection object represents a unique session to a SQL Server data source. With a client/server database system, it is equivalent to a network connection to the server. The SqlConnection GetConnection () is used for getting the connection string of the database. The connection string that includes the source database name and other parameters needed to establish the initial connection. The default value is an empty string.

In this case, my connection string was for the database with the name ‘Test’, hence, as shown in Figure 22, the Initial catalog is stated as ‘Test’. The connection string is generated in a .dll file. Any .cs file within the project can refer to this connection string without copying it again and again.

SqlConnection is used together with SqlDataAdapter and SqlCommand to increase performance when connecting to a Microsoft SQL Server database.

The SqlDataAdapter serves as a bridge between a DataSet and SQL Server for retrieving and saving data. The SqlDataAdapter provides this bridge by mapping Fill, which changes the data in the DataSet to match the data in the data source, and Update, which changes the data in the data source to match the data in the DataSet, using the appropriate Transact-SQL statements against the data source.



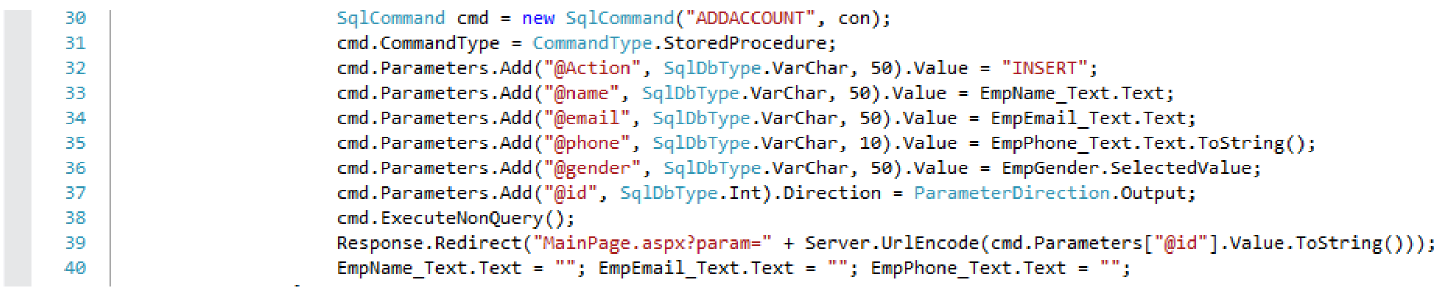
```
SqlConnection con = new SqlConnection();
con = SQLUtility.GetConnection();
if (con.State == ConnectionState.Closed)
    con.Open();
```

Figure 23: Code for using SQL utility class

The update is performed on a by-row basis. For every inserted, modified, and deleted row, the Update method determines the type of change that has been performed on it (Insert, Update, or Delete).

Depending on the type of change, the Insert, Update, or Delete command template executes to propagate the modified row to the data source. When the SqlDataAdapter fills a DataSet, it creates the necessary tables and columns for the returned data if they do not already exist.

The SqlDataAdapter also includes the SelectCommand, InsertCommand
, DeleteCommand, UpdateCommand, and TableMappings properties to facilitate the loading and updating of data. When an instance of SqlDataAdapter is created, the read/write properties are set to initial values

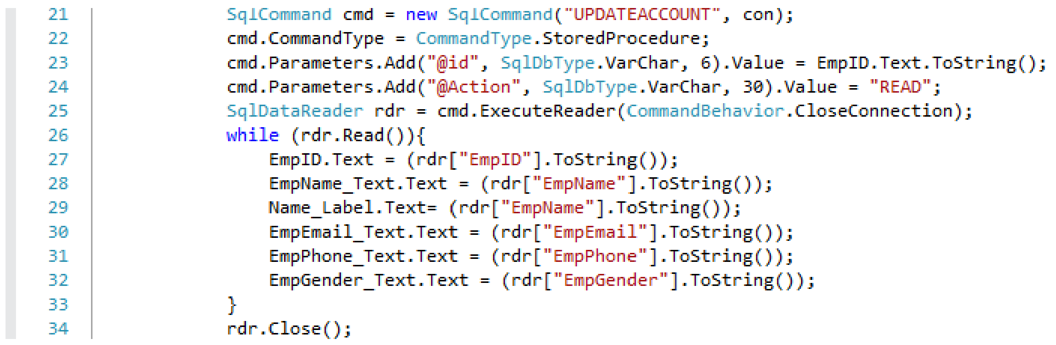


```
SqlCommand cmd = new SqlCommand("ADDACCOUNT", con);
cmd.CommandType = CommandType.StoredProcedure;
cmd.Parameters.Add("@Action", SqlDbType.VarChar, 50).Value = "INSERT";
cmd.Parameters.Add("@name", SqlDbType.VarChar, 50).Value = EmpName_Text.Text;
cmd.Parameters.Add("@email", SqlDbType.VarChar, 50).Value = EmpEmail_Text.Text;
cmd.Parameters.Add("@phone", SqlDbType.VarChar, 10).Value = EmpPhone_Text.Text.ToString();
cmd.Parameters.Add("@gender", SqlDbType.VarChar, 50).Value = EmpGender.SelectedValue;
cmd.Parameters.Add("@id", SqlDbType.Int).Direction = ParameterDirection.Output;
cmd.ExecuteNonQuery();
Response.Redirect("MainPage.aspx?param=" + Server.UrlEncode(cmd.Parameters["@id"].Value.ToString()));
EmpName_Text.Text = ""; EmpEmail_Text.Text = ""; EmpPhone_Text.Text = "";
```

Figure 24: Code for using SQL utility class for adding account

The InsertCommand, DeleteCommand, and UpdateCommand are generic templates that are automatically filled with individual values from every modified row through the parameters mechanism.

For every column that you propagate to the data source on Update, a parameter should be added to the InsertCommand, UpdateCommand, or DeleteCommand. The SourceColumn property of the DbParameter object should be set to the name of the column. This setting indicates that the value of the parameter is not set manually, but is taken from the particular column in the currently processed row.



```
SqlCommand cmd = new SqlCommand("UPDATEACCOUNT", con);
cmd.CommandType = CommandType.StoredProcedure;
cmd.Parameters.Add("@id", SqlDbType.VarChar, 6).Value = EmpID.Text.ToString();
cmd.Parameters.Add("@Action", SqlDbType.VarChar, 30).Value = "READ";
SqlDataReader rdr = cmd.ExecuteReader(CommandBehavior.CloseConnection);
while (rdr.Read()){
    EmpID.Text = (rdr["EmpID"].ToString());
    EmpName_Text.Text = (rdr["EmpName"].ToString());
    Name_Label.Text= (rdr["EmpName"].ToString());
    EmpEmail_Text.Text = (rdr["EmpEmail"].ToString());
    EmpPhone_Text.Text = (rdr["EmpPhone"].ToString());
    EmpGender_Text.Text = (rdr["EmpGender"].ToString());
}
rdr.Close();
```

Figure 25: Code for using SQL utility class for reading from an account

Figure 25 shows how the object of the class SqlCommand is able to access an account that is present in the specified database. To create a SqlDataReader, the ExecuteReader method of the SqlCommand object has to be called, instead of directly using a constructor.

While the SqlDataReader is being used, the associated SqlConnection is busy serving the SqlDataReader, and no other operations can be performed on the SqlConnection other than closing it.

This is the case until the Close method of the SqlDataReader is called. For example, you cannot retrieve output parameters until after you call Close. Changes made to a result set by another process or thread while data is being read may be visible to the user of the SqlDataReader. However, the precise behavior is timing dependent.

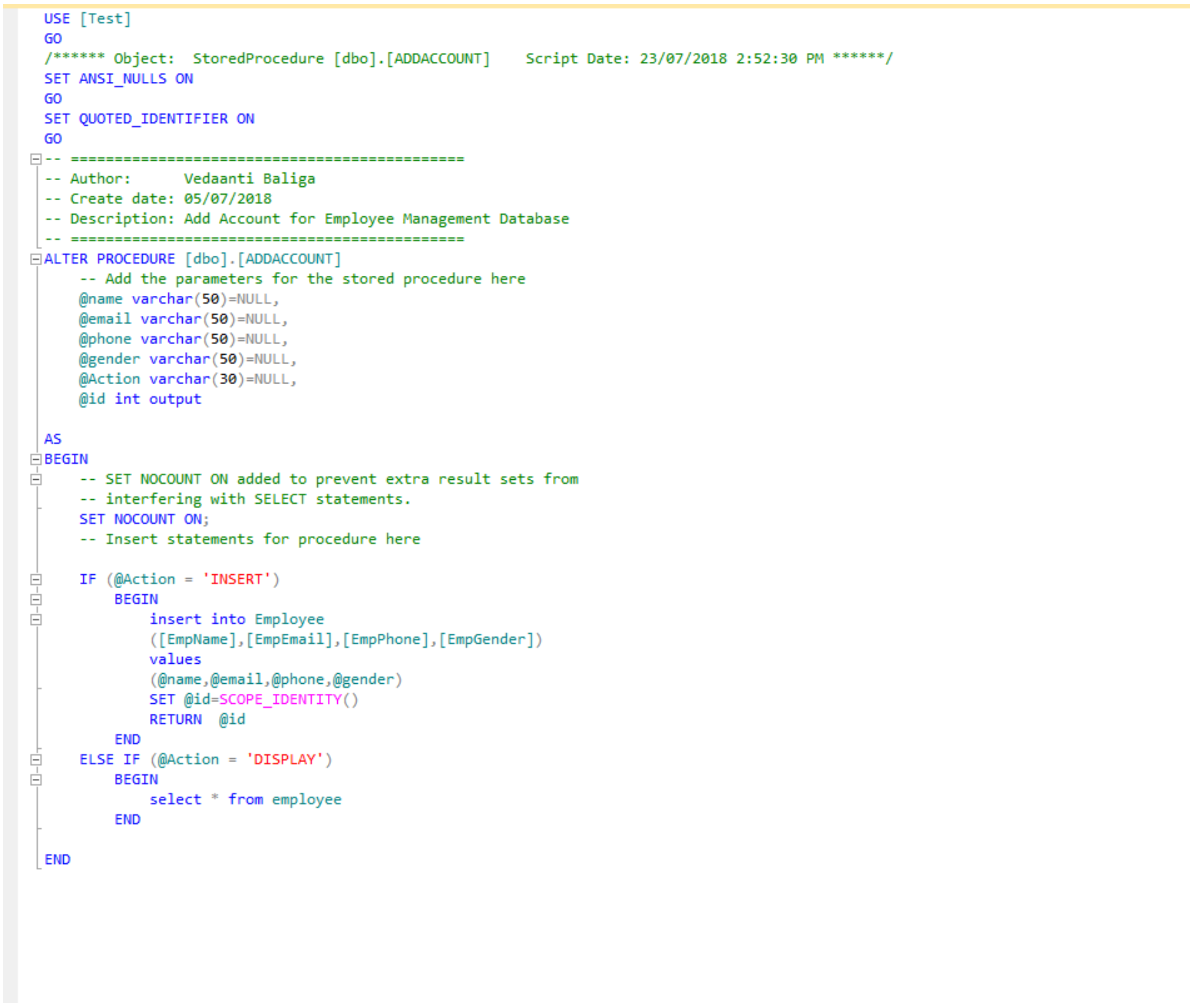
IsClosed and RecordsAffected are the only properties that you can call after the SqlDataReader is closed. Although the RecordsAffected property may be accessed while the SqlDataReader exists, Close has to be called, always, before returning the value of RecordsAffected to guarantee an accurate return value.

The getDataTable method fills a DataTable object with the result of a SQL query, and additional parameter is passed to specify the database where we want extract data.

Similarly, the getDataSet method returns a .NET DataSet object that contains a DataTable. The DataTable contains the results for each of the rule set that was checked. Using the DataSet and DataTable in this fashion is useful if you want to assign the results to any object which accepts a .NET DataSet as a source that can be bound to.

7. SQL Stored Procedures Used by the Employee Management Website and Windows Form:

i. Add an Account:

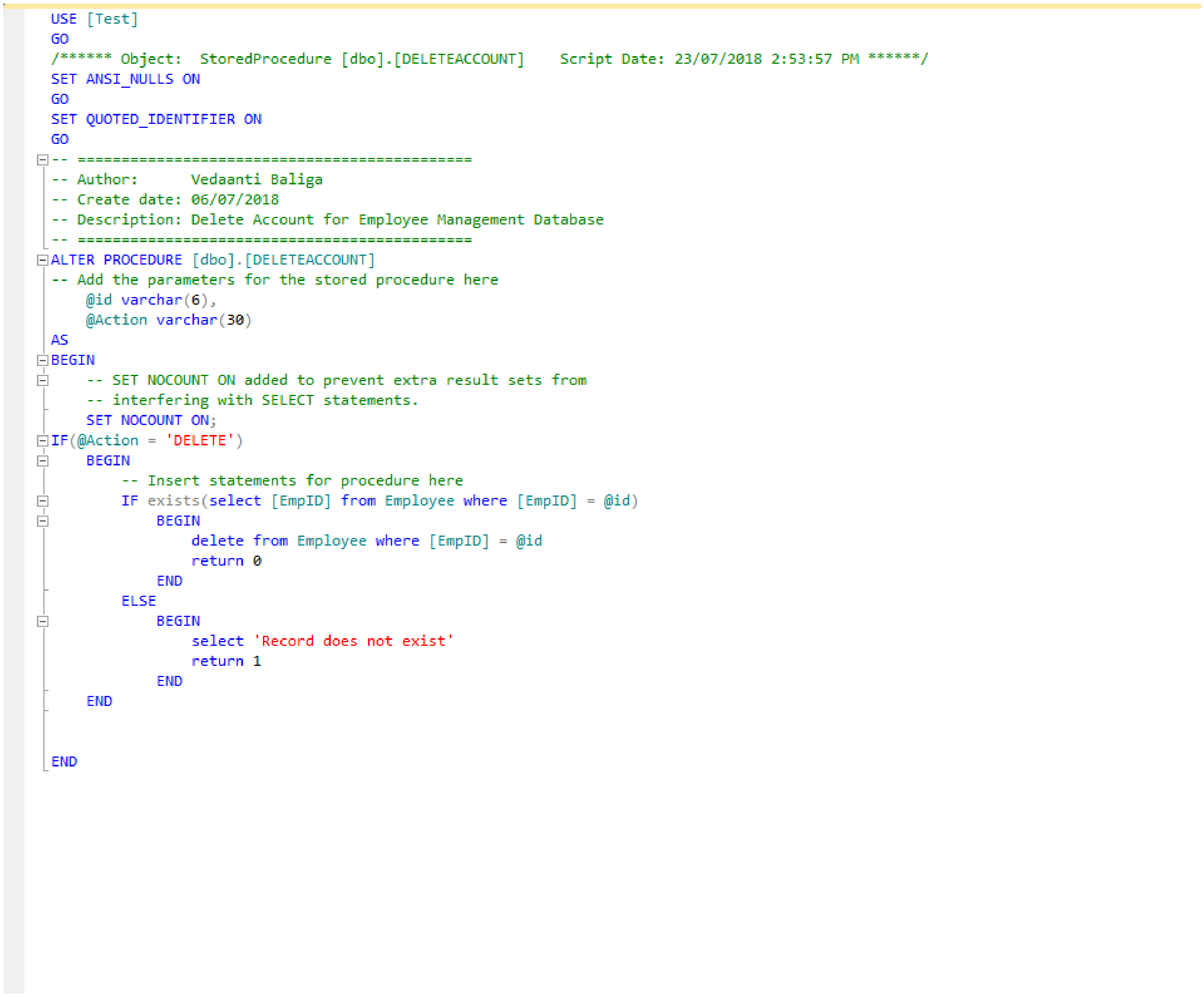


```
USE [Test]
GO
/****** Object:  StoredProcedure [dbo].[ADDACCOUNT]    Script Date: 23/07/2018 2:52:30 PM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      Vedaanti Baliga
-- Create date: 05/07/2018
-- Description: Add Account for Employee Management Database
-- =============================================
ALTER PROCEDURE [dbo].[ADDACCOUNT]
    -- Add the parameters for the stored procedure here
    @name varchar(50)=NULL,
    @email varchar(50)=NULL,
    @phone varchar(50)=NULL,
    @gender varchar(50)=NULL,
    @Action varchar(30)=NULL,
    @id int output

AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;
    -- Insert statements for procedure here

    IF (@Action = 'INSERT')
        BEGIN
            insert into Employee
            ([EmpName],[EmpEmail],[EmpPhone],[EmpGender])
            values
            (@name,@email,@phone,@gender)
            SET @id=SCOPE_IDENTITY()
            RETURN  @id
        END
    ELSE IF (@Action = 'DISPLAY')
        BEGIN
            select * from employee
        END

END
```

Figure 26: Stored Procedure for adding an account

This procedure is called in the C# code to perform the INSERT operation. The procedure returns the auto generated ID to the code. The inputs given by the user are inserted by the query shown in Figure 26.

ii. **Delete an Account:**

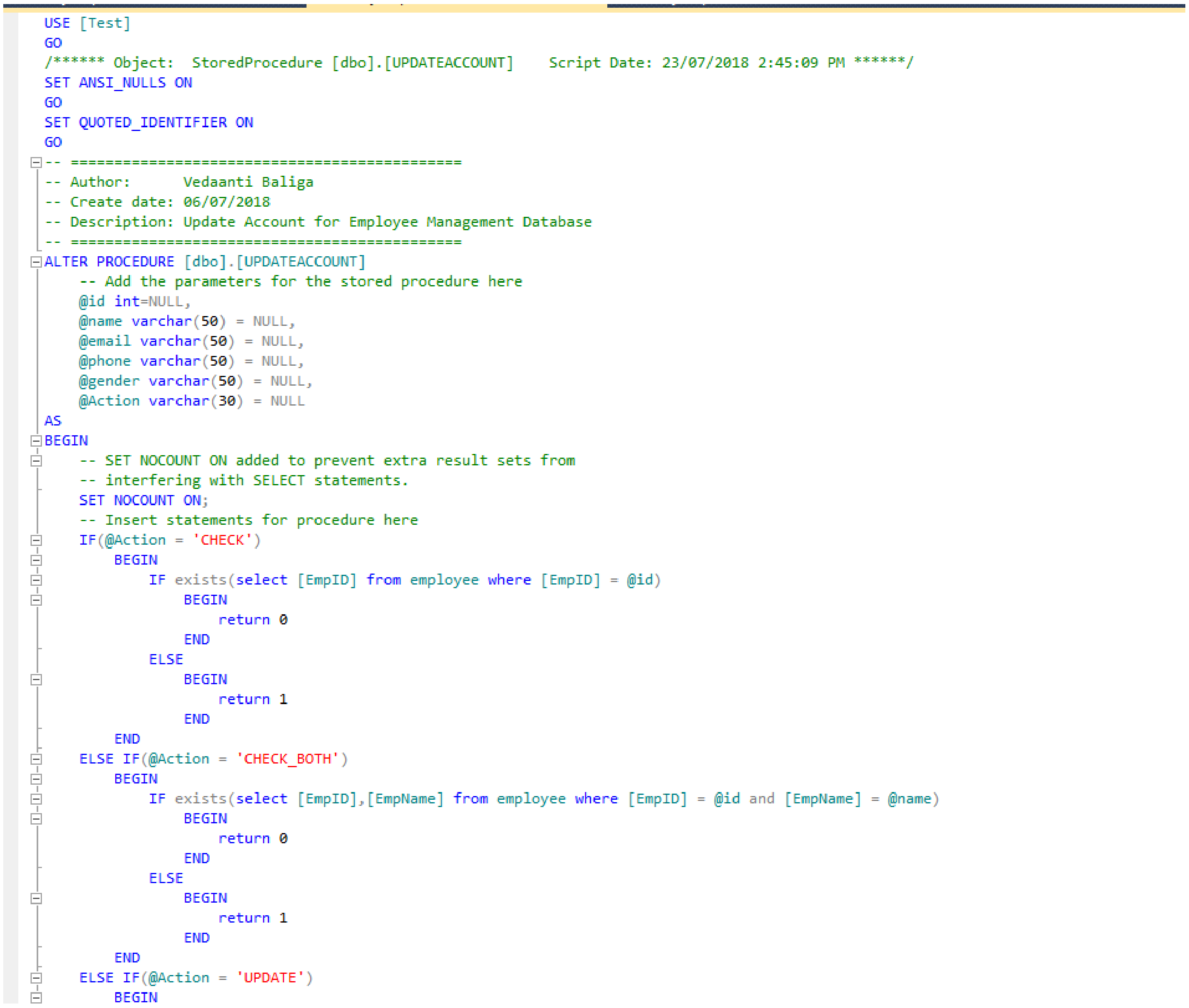


```
USE [Test]
GO
/****** Object:  StoredProcedure [dbo].[DELETEACCOUNT]    Script Date: 23/07/2018 2:53:57 PM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      Vedaanti Baliga
-- Create date: 06/07/2018
-- Description: Delete Account for Employee Management Database
-- =============================================
ALTER PROCEDURE [dbo].[DELETEACCOUNT]
-- Add the parameters for the stored procedure here
    @id varchar(6),
    @Action varchar(30)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;
IF(@Action = 'DELETE')
    BEGIN
        -- Insert statements for procedure here
        IF exists(select [EmpID] from Employee where [EmpID] = @id)
            BEGIN
                delete from Employee where [EmpID] = @id
                return 0
            END
        ELSE
            BEGIN
                select 'Record does not exist'
                return 1
            END
    END


END
```

Figure 27: Stored Procedure for deleting an account

This procedure is called in the C# code to perform the DELETE operation. The procedure returns the ‘0’ if the record corresponding to the ID exists, otherwise, it returns ‘1’. The account is deleted by the query shown in Figure 27.

iii. **Update an Account:**

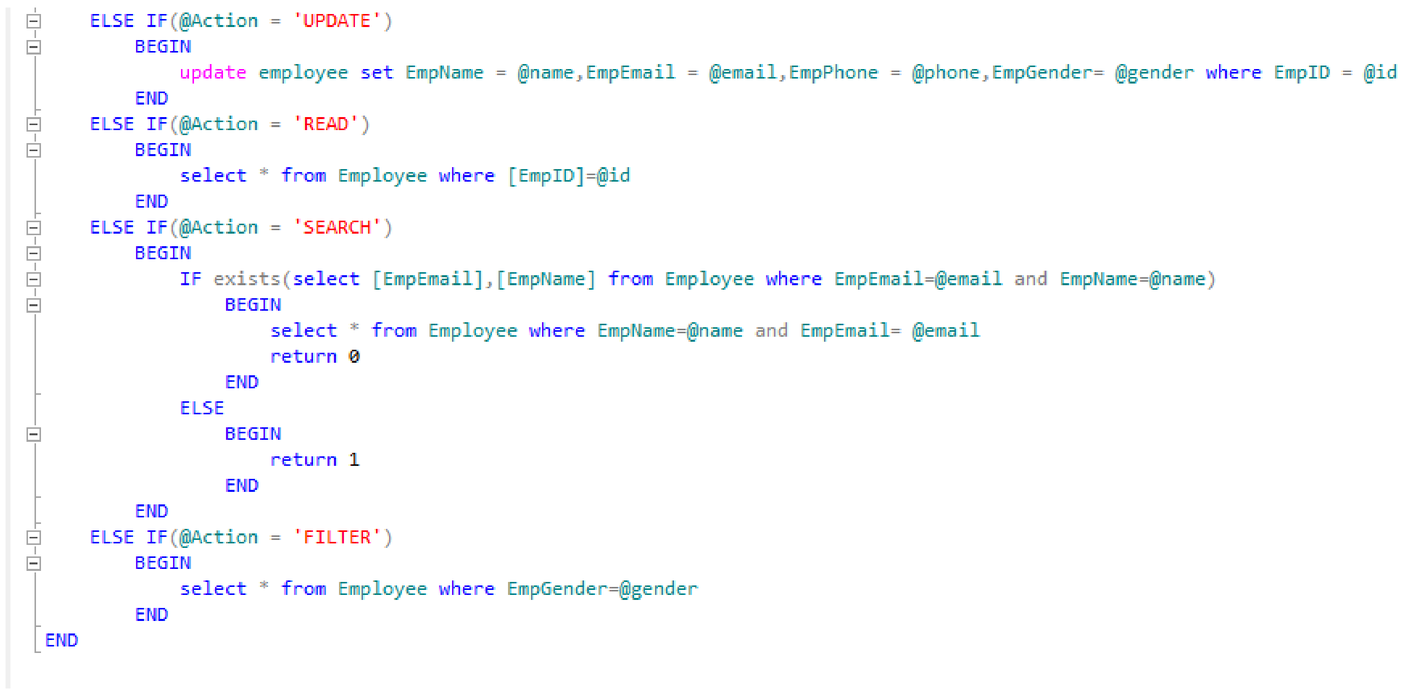


```
USE [Test]
GO
/****** Object:  StoredProcedure [dbo].[UPDATEACCOUNT]    Script Date: 23/07/2018 2:45:09 PM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      Vedaanti Baliga
-- Create date: 06/07/2018
-- Description: Update Account for Employee Management Database
-- =============================================
ALTER PROCEDURE [dbo].[UPDATEACCOUNT]
    -- Add the parameters for the stored procedure here
    @id int=NULL,
    @name varchar(50) = NULL,
    @email varchar(50) = NULL,
    @phone varchar(50) = NULL,
    @gender varchar(50) = NULL,
    @Action varchar(30) = NULL
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;
    -- Insert statements for procedure here
    IF(@Action = 'CHECK')
        BEGIN
            IF exists(select [EmpID] from employee where [EmpID] = @id)
                BEGIN
                    return 0
                END
            ELSE
                BEGIN
                    return 1
                END
        END
    ELSE IF(@Action = 'CHECK_BOTH')
        BEGIN
            IF exists(select [EmpID],[EmpName] from employee where [EmpID] = @id and [EmpName] = @name)
                BEGIN
                    return 0
                END
            ELSE
                BEGIN
                    return 1
                END
        END
    ELSE IF(@Action = 'UPDATE')
        BEGIN
```

Figure 28: Stored Procedure for updating an account

This procedure is called in the C# code to perform the UPDATE operation. The procedure returns has various functionalities.

For example. In the ‘CHECK’ action, it checks if an employee exists or not by checking for its ID, and returns the respective value. In the ‘CHECK_BOTH’ action, it checks if an employee exists or not by checking for its ID and his/her name, and returns the respective value.



```
    ELSE IF(@Action = 'UPDATE')
        BEGIN
            update employee set EmpName = @name,EmpEmail = @email,EmpPhone = @phone,EmpGender= @gender where EmpID = @id
        END
    ELSE IF(@Action = 'READ')
        BEGIN
            select * from Employee where [EmpID]=@id
        END
    ELSE IF(@Action = 'SEARCH')
        BEGIN
            IF exists(select [EmpEmail],[EmpName] from Employee where EmpEmail=@email and EmpName=@name)
                BEGIN
                    select * from Employee where EmpName=@name and EmpEmail= @email
                    return 0
                END
            ELSE
                BEGIN
                    return 1
                END
        END
    ELSE IF(@Action = 'FILTER')
        BEGIN
            select * from Employee where EmpGender=@gender
        END
END
```

Figure 29: Stored Procedure for updating an account

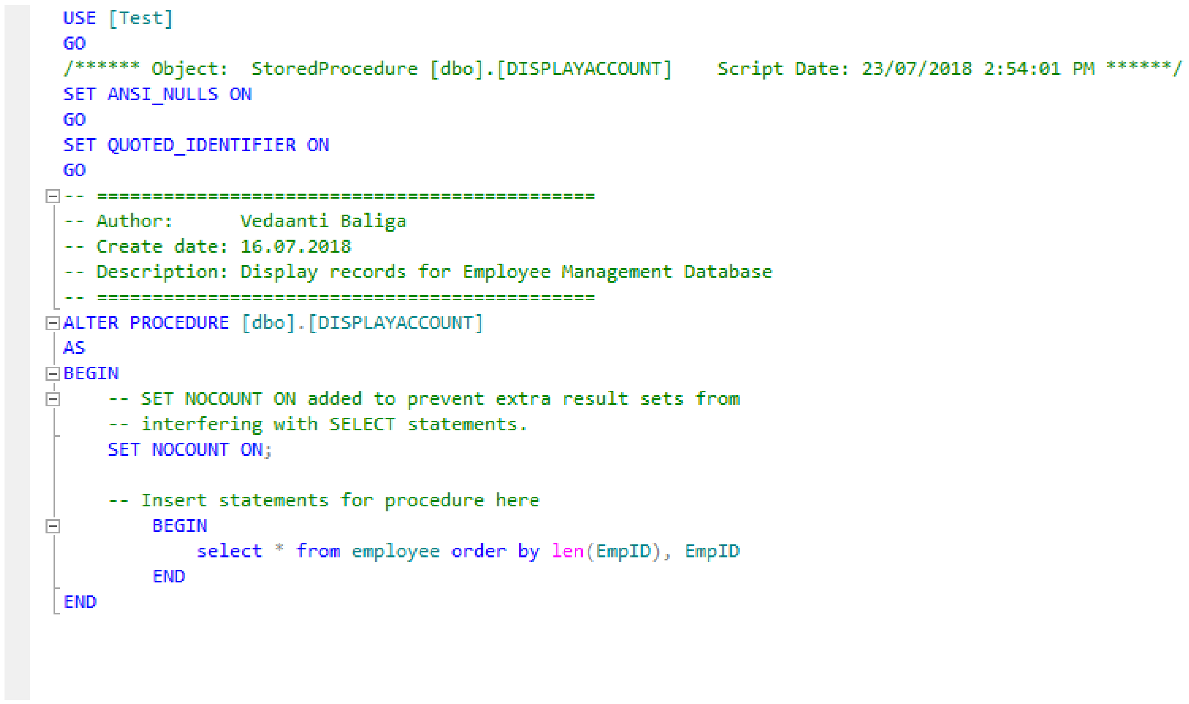
In the ‘UPDATE’ action, the procedure updates the changes made by the employee, if there are any. Figure 29 shows the query for updating the records in the database.

In the ‘READ’ action, the procedure reads the record corresponding the employee ID, if there are any. Figure 29 shows the query for reading from the records in the database.

In the ‘SEARCH’ action, the procedure first checks for existence of records corresponding to the employee name and email corresponding the records, if there are records present, then it returns a ‘0’ otherwise a ‘1’ is returned . Figure 29 shows the query for searching the records in the database.

In the ‘FILTER’ action, the procedure first checks for existence of records corresponding to the employee gender corresponding the records. Figure 29 shows the query for filtering the records in the database.

iv. **Display Accounts:**

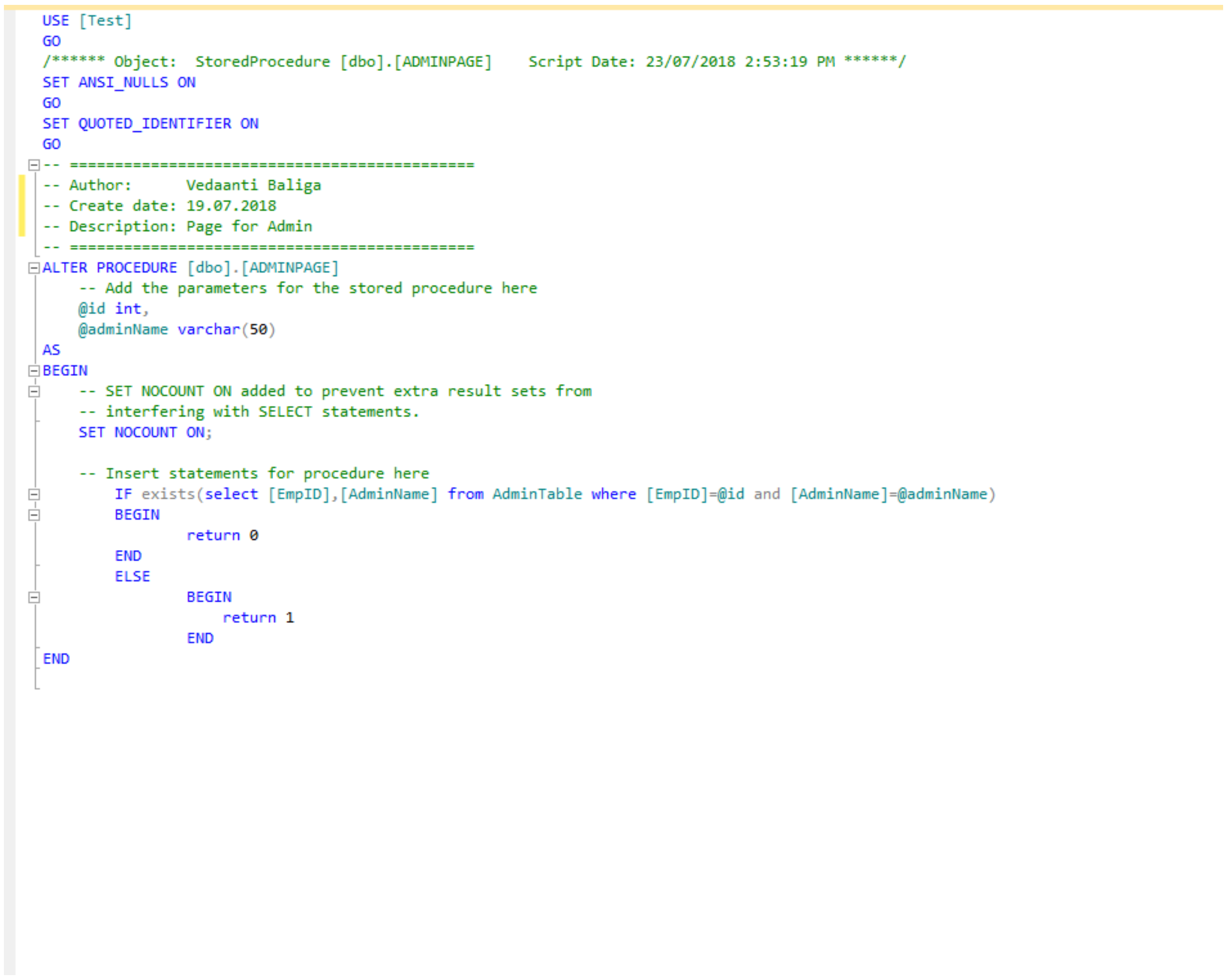


```
USE [Test]
GO
/****** Object:  StoredProcedure [dbo].[DISPLAYACCOUNT]    Script Date: 23/07/2018 2:54:01 PM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      Vedaanti Baliga
-- Create date: 16.07.2018
-- Description: Display records for Employee Management Database
-- =============================================
ALTER PROCEDURE [dbo].[DISPLAYACCOUNT]
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
        BEGIN
            select * from employee order by len(EmpID), EmpID
        END
END
```

Figure 30: Stored Procedure for displaying accounts

This procedure displays the records in ascending order, if there are any. Figure 29 shows the query for updating the records in the database.

v. **Admin Account:**



```
USE [Test]
GO
/****** Object:  StoredProcedure [dbo].[ADMINPAGE]    Script Date: 23/07/2018 2:53:19 PM ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      Vedaanti Baliga
-- Create date: 19.07.2018
-- Description: Page for Admin
-- =============================================
ALTER PROCEDURE [dbo].[ADMINPAGE]
    -- Add the parameters for the stored procedure here
    @id int,
    @adminName varchar(50)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
        IF exists(select [EmpID],[AdminName] from AdminTable where [EmpID]=@id and [AdminName]=@adminName)
        BEGIN
                return 0
        END
        ELSE
                BEGIN
                    return 1
                END
END
```

Figure 31: Stored Procedure for displaying admin account

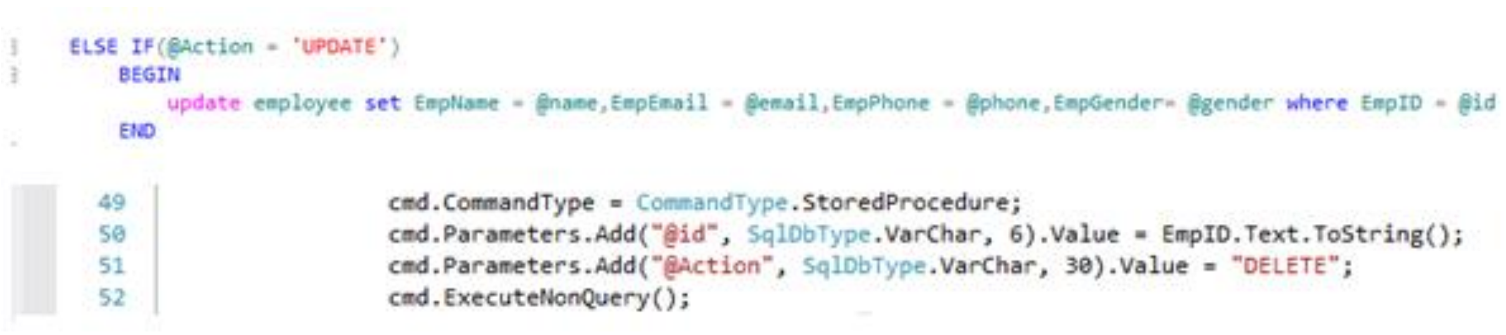
This procedure is for validating the admin account. The procedure checks if the user entries of ID and name match with the entries of the ‘AdminTable’. If they exist, the procedure returns ‘0’ otherwise, it returns ‘1’.

8. SQL Injection:

SQL injection is a code injection technique that might destroy your database. It is one of the most common web hacking techniques. It is the placement of malicious code in SQL statements, via web page input.

SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will unknowingly run on your database.

To protect a web site from SQL injection, I have used SQL parameters. SQL parameters are values that are added to an SQL query at execution time, in a controlled manner. Figure 32 shows the SQL server and the C# code.



```
ELSE IF(@Action = 'UPDATE')
    BEGIN
        update employee set EmpName = @name,EmpEmail = @email,EmpPhone = @phone,EmpGender= @gender where EmpID = @id
    END
```

```
                cmd.CommandType = CommandType.StoredProcedure;
                cmd.Parameters.Add("@id", SqlDbType.VarChar, 6).Value = EmpID.Text.ToString();
                cmd.Parameters.Add("@Action", SqlDbType.VarChar, 30).Value = "DELETE";
                cmd.ExecuteNonQuery();
```

Figure 32: Stored Procedure and C# Code

The parameters are represented in the SQL statement by a @ marker. The SQL engine checks each parameter to ensure that it is correct for its column and are treated literally, and not as part of the SQL to be executed.

CHAPTER 4: IMPLEMENTATION AND RESULTS

4.1 **Modules:**

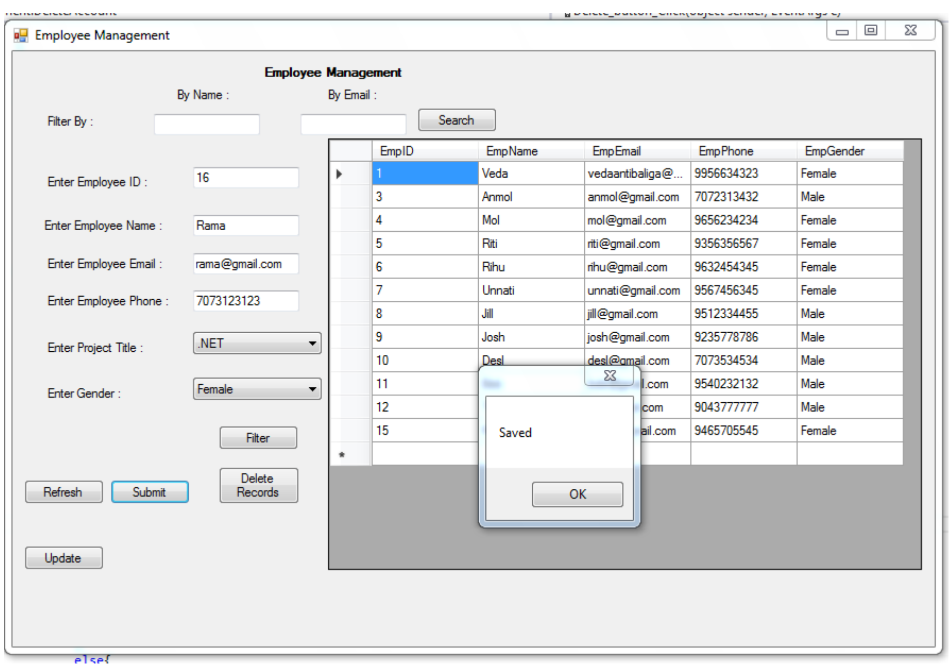
The following were the main models used in my code:

- System
- System.Collections.Generic
- System.Data
- System.Data.SqlClient
- System.Linq
- System.Web
- System.Web.UI
- System.Web.UI.WebControls
- System.Security.Cryptography
- System.ComponentModel
- System.Drawing
- System.Text
- System.Threading.Tasks
- System.Windows.Forms

4.2 **Prototype:**

1. **Employee Management (Windows Form):**

Attached hereon are snippets of the output produced:



Employee Management

Employee Management

Filter By : By Name : By Email : Search

Enter Employee ID : 16

Enter Employee Name : Rama

Enter Employee Email : rama@gmail.com

Enter Employee Phone : 7073123123

Enter Project Title : .NET

Enter Gender : Female

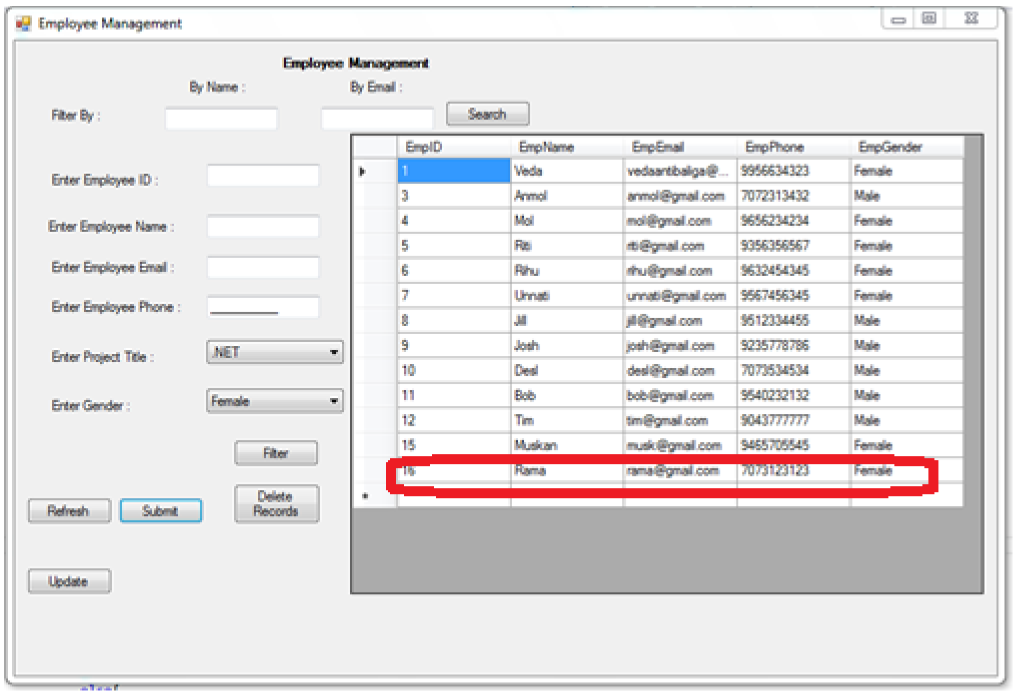
Filter | Refresh | Submit | Delete Records | Update

EmpID	EmpName	EmpEmail	EmpPhone	EmpGender
1	Veda	vedaantibaliga@...	9956634323	Female
3	Anmol	anmol@gmail.com	7072313432	Male
4	Mol	mol@gmail.com	9656234234	Female
5	Riti	riti@gmail.com	9356356567	Female
6	Rihu	rihu@gmail.com	9632454345	Female
7	Unnati	unnati@gmail.com	9567456345	Female
8	Jill	jill@gmail.com	9512334455	Male
9	Josh	josh@gmail.com	9235778786	Male
10	Desi	desi@gmail.com	7073534534	Male
11		l.com	9540232132	Male
12		com	9043777777	Male
15		ail.com	9465705545	Female

Saved

OK

Figure 33: Account saved



Employee Management

Employee Management

Filter By : By Name : By Email : Search

Enter Employee ID :

Enter Employee Name :

Enter Employee Email :

Enter Employee Phone :

Enter Project Title : .NET

Enter Gender : Female

Filter | Refresh | Submit | Delete Records | Update

EmpID	EmpName	EmpEmail	EmpPhone	EmpGender
1	Veda	vedaantibaliga@...	9956634323	Female
3	Anmol	anmol@gmail.com	7072313432	Male
4	Mol	mol@gmail.com	9656234234	Female
5	Riti	riti@gmail.com	9356356567	Female
6	Rihu	rihu@gmail.com	9632454345	Female
7	Unnati	unnati@gmail.com	9567456345	Female
8	Jill	jill@gmail.com	9512334455	Male
9	Josh	josh@gmail.com	9235778786	Male
10	Desi	desi@gmail.com	7073534534	Male
11	Bob	bob@gmail.com	9540232132	Male
12	Tim	tim@gmail.com	9043777777	Male
15	Muskan	musk@gmail.com	9465705545	Female
16	Rama	rama@gmail.com	7073123123	Female

Figure 34: Account added successfully

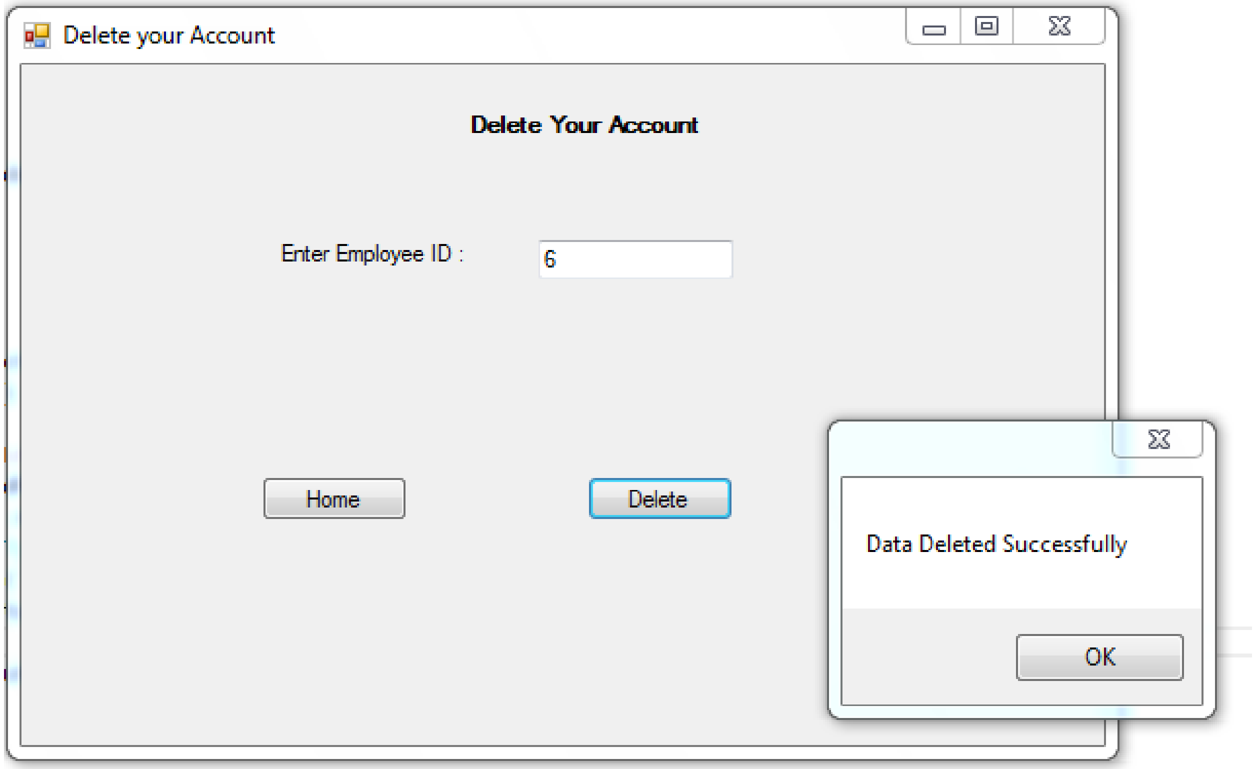


Figure 35: Account deleted successfully

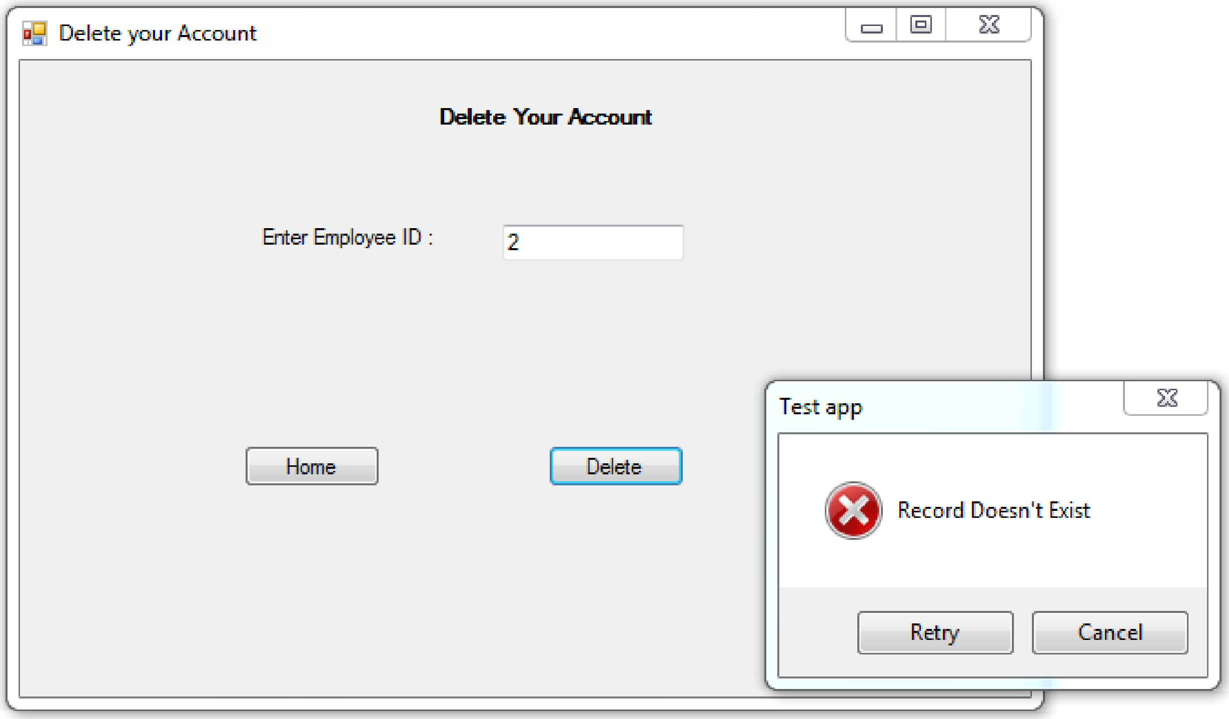
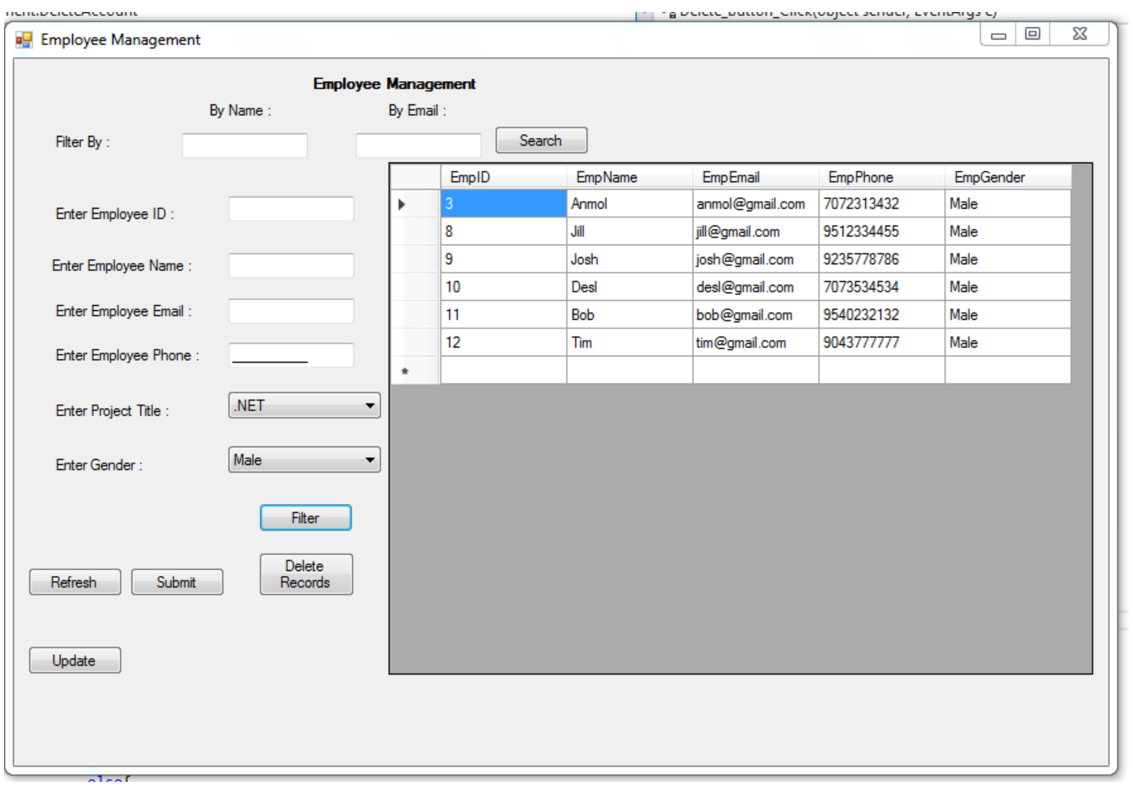
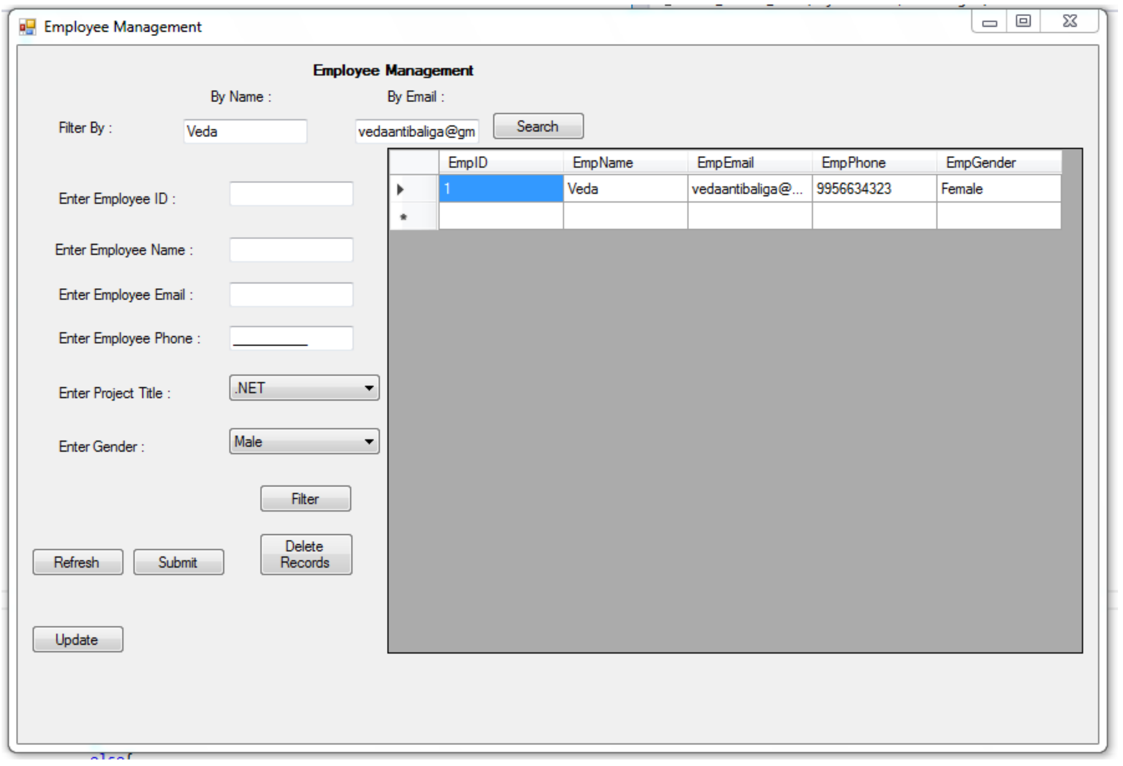


Figure 36: Account not found for deleting



EmpID	EmpName	EmpEmail	EmpPhone	EmpGender
3	Anmol	anmol@gmail.com	7072313432	Male
8	Jill	jill@gmail.com	9512334455	Male
9	Josh	josh@gmail.com	9235778786	Male
10	Desl	desl@gmail.com	7073534534	Male
11	Bob	bob@gmail.com	9540232132	Male
12	Tim	tim@gmail.com	9043777777	Male

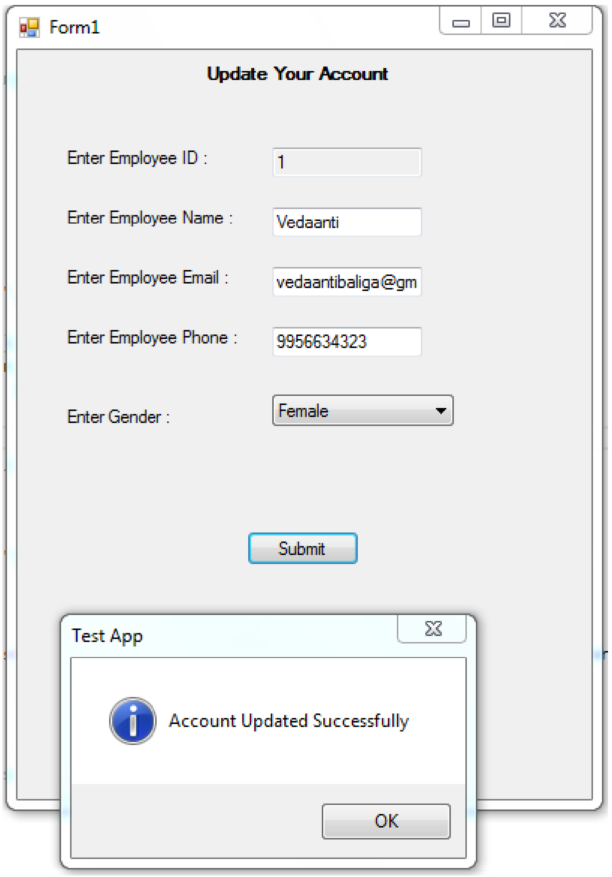
Figure 37: Records filtered according to gender



Filter By : Veda vedaantibaliga@gm

EmpID	EmpName	EmpEmail	EmpPhone	EmpGender
1	Veda	vedaantibaliga@...	9956634323	Female

Figure 38: Records searched according to name and email id


Form1

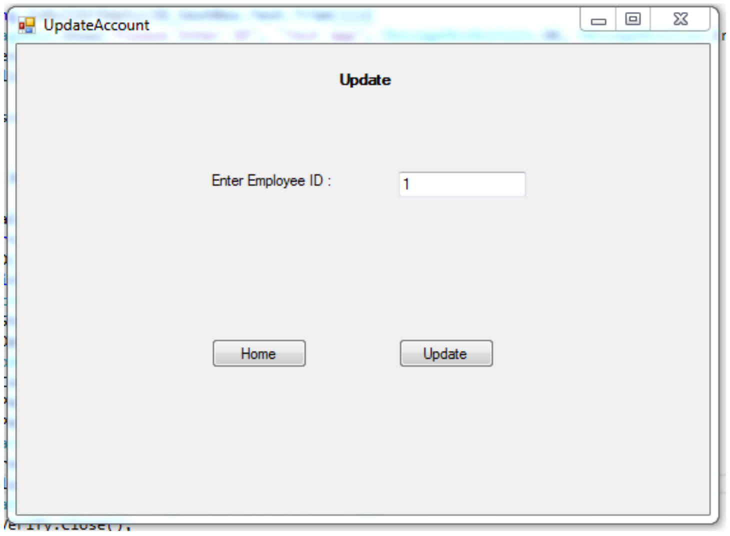
Update Your Account

Enter Employee ID :	1
Enter Employee Name :	Vedaanti
Enter Employee Email :	vedaantibaliga@gm
Enter Employee Phone :	9956634323
Enter Gender :	Female

Test App

Account Updated Successfully

Figure 39: Records updated successfully


UpdateAccount

Update

Enter Employee ID : 1

Figure 40: Update Form

2. Calculator (Windows Form):

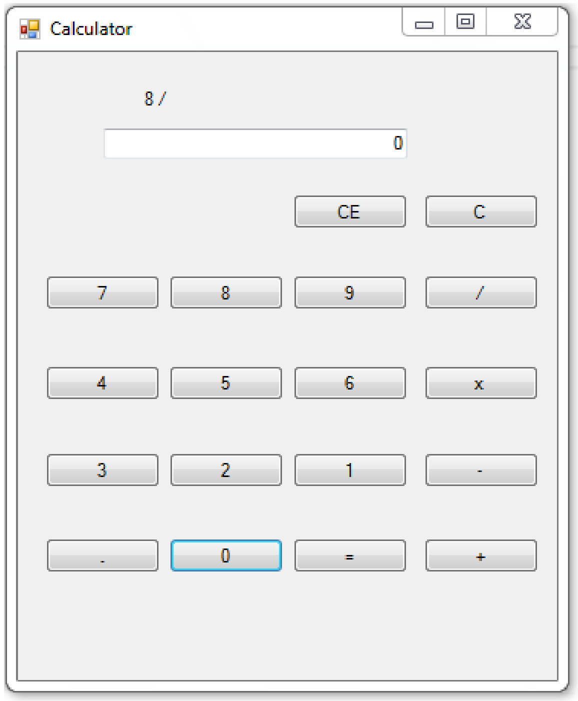


Figure 40: Divide operation

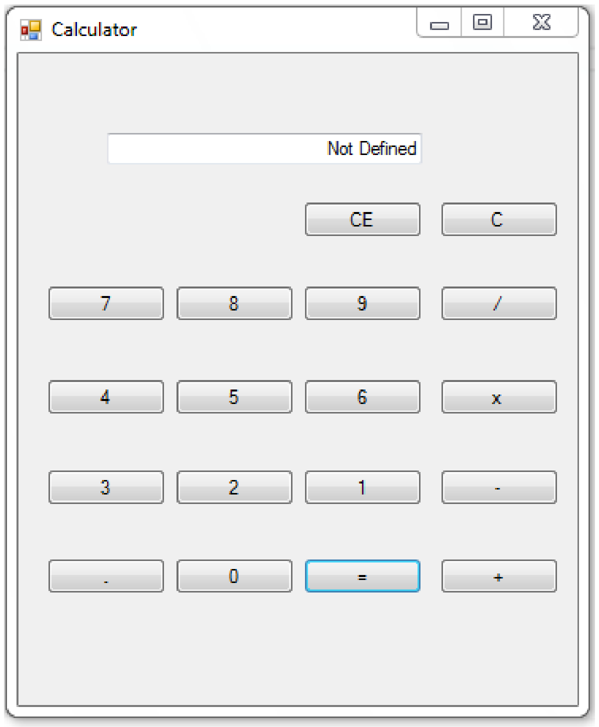


Figure 41: Validation for divide operation

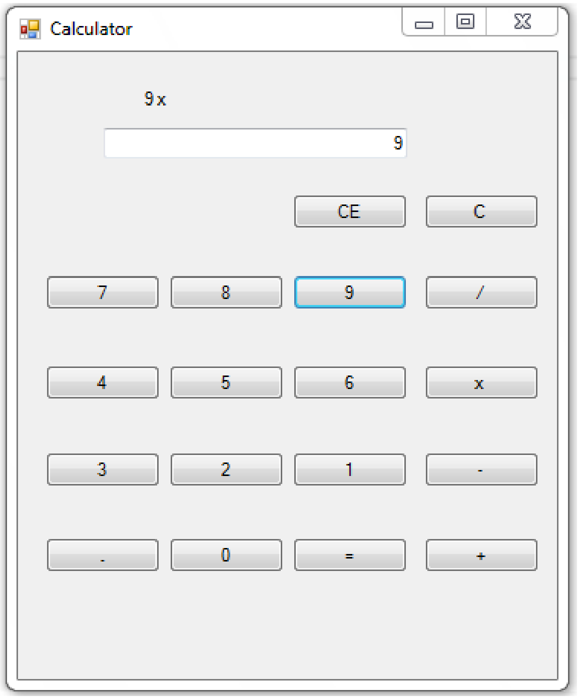


Figure 42: Multiply operation

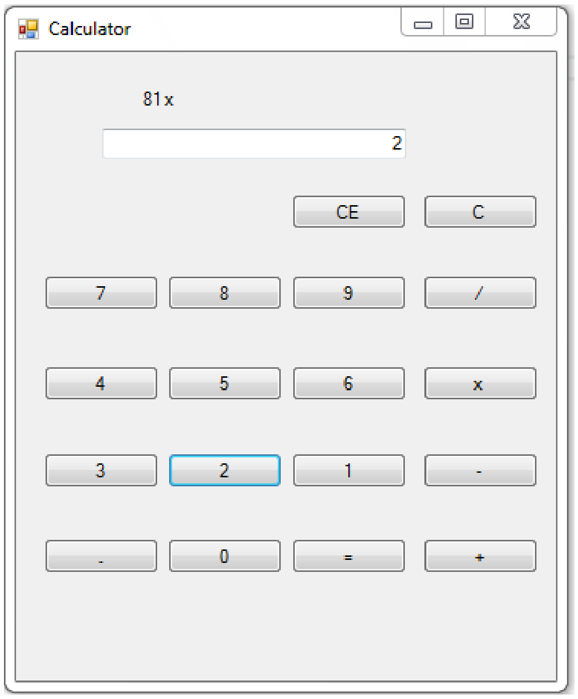


Figure 43: Multiply operation result

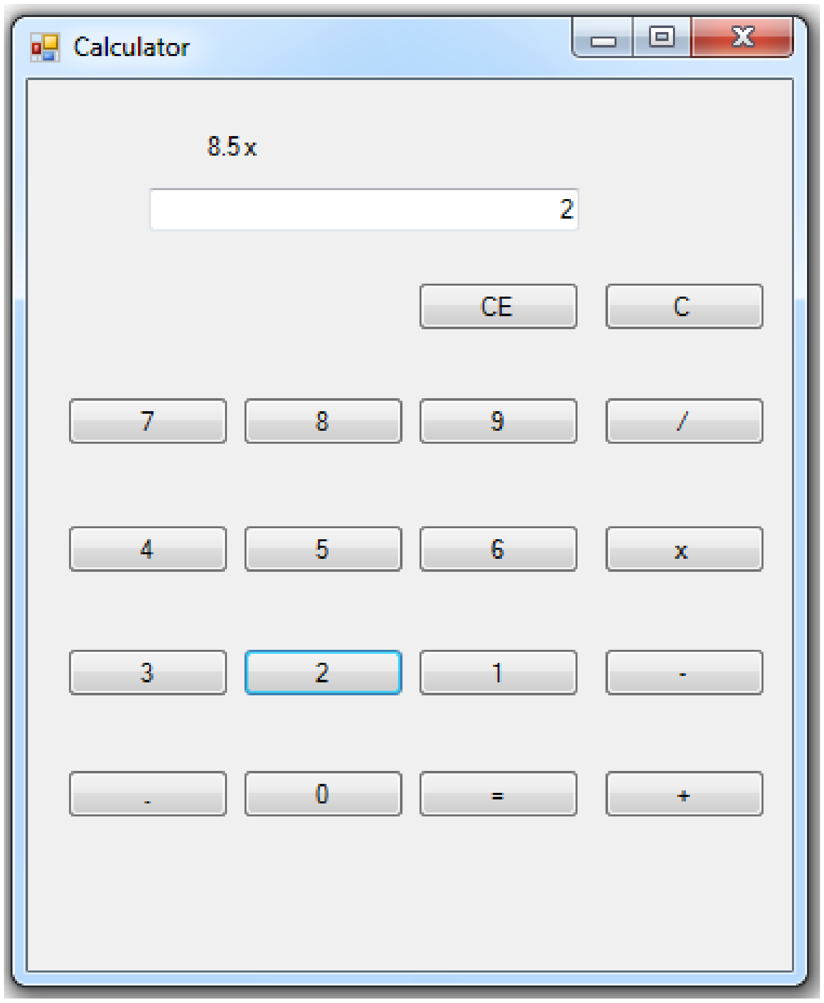


Figure 44: Multiply operation with decimals

3. Website for Employee Database Management (ASP.NET):

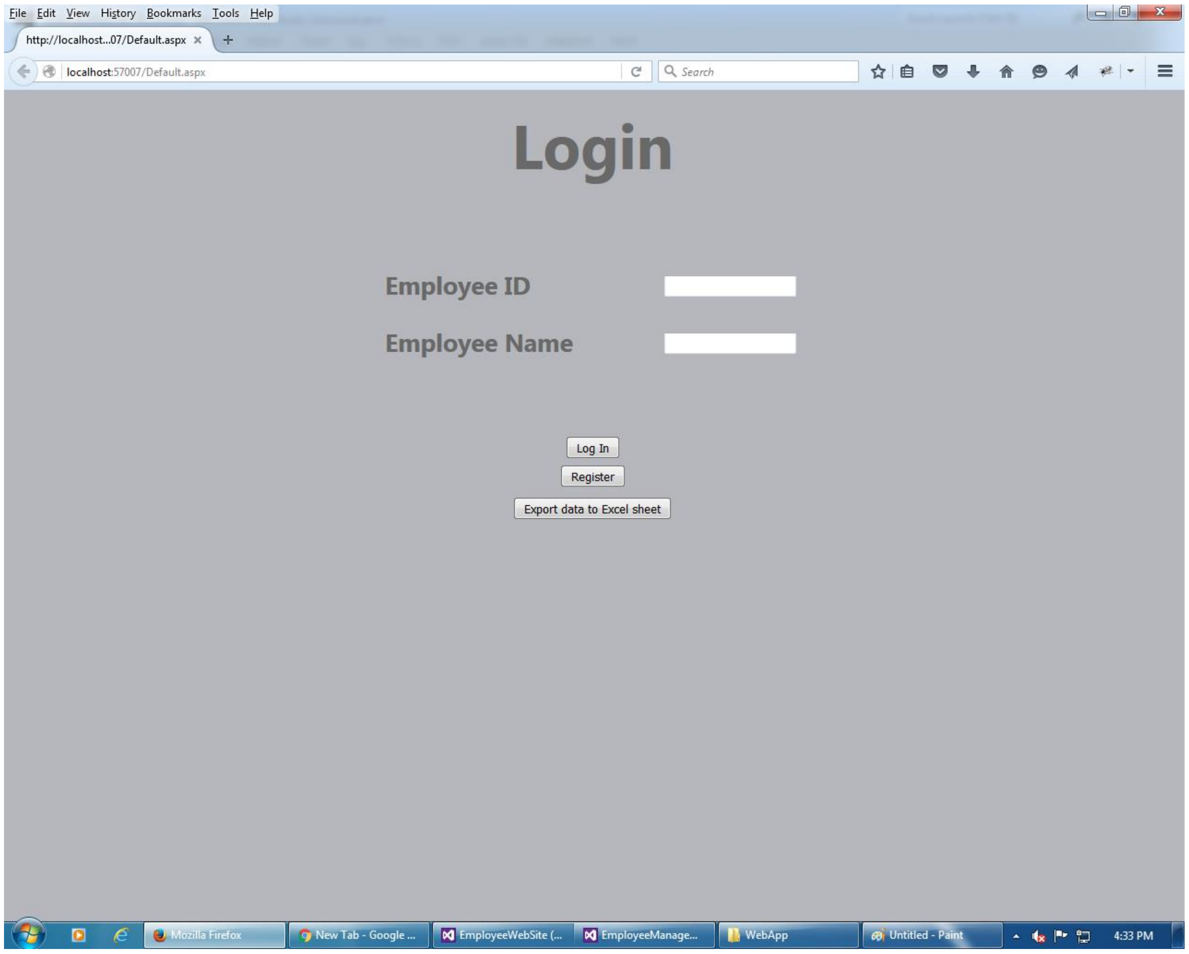
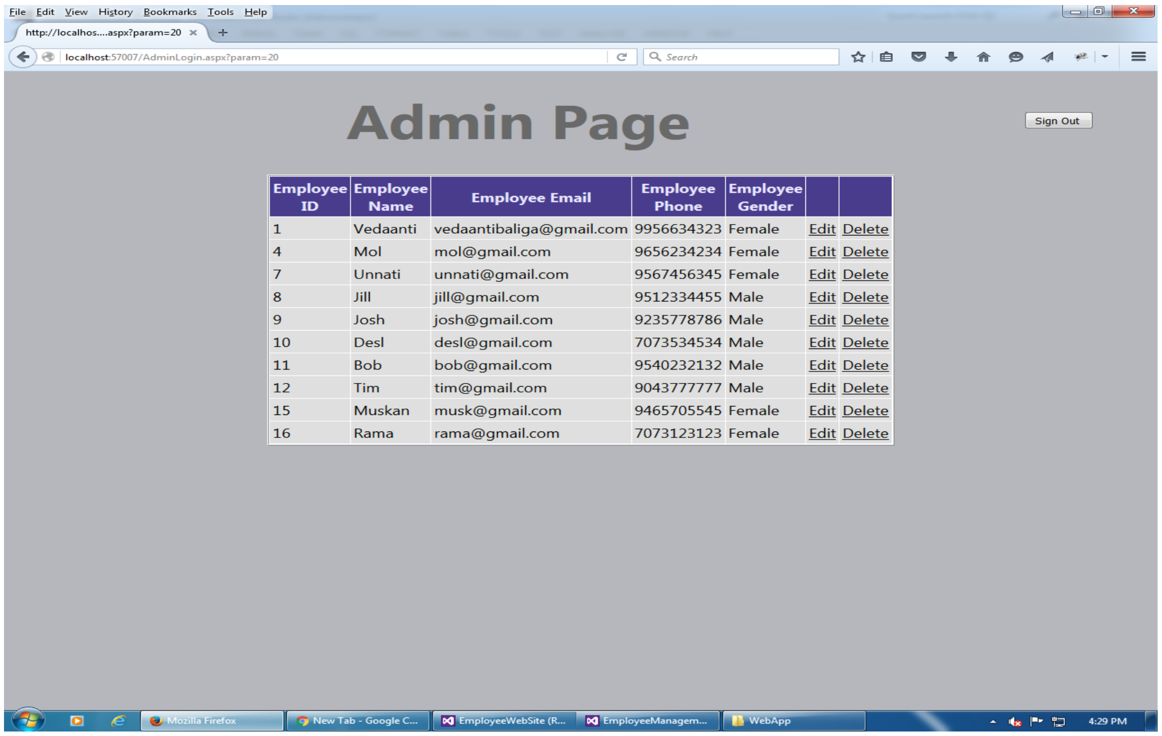


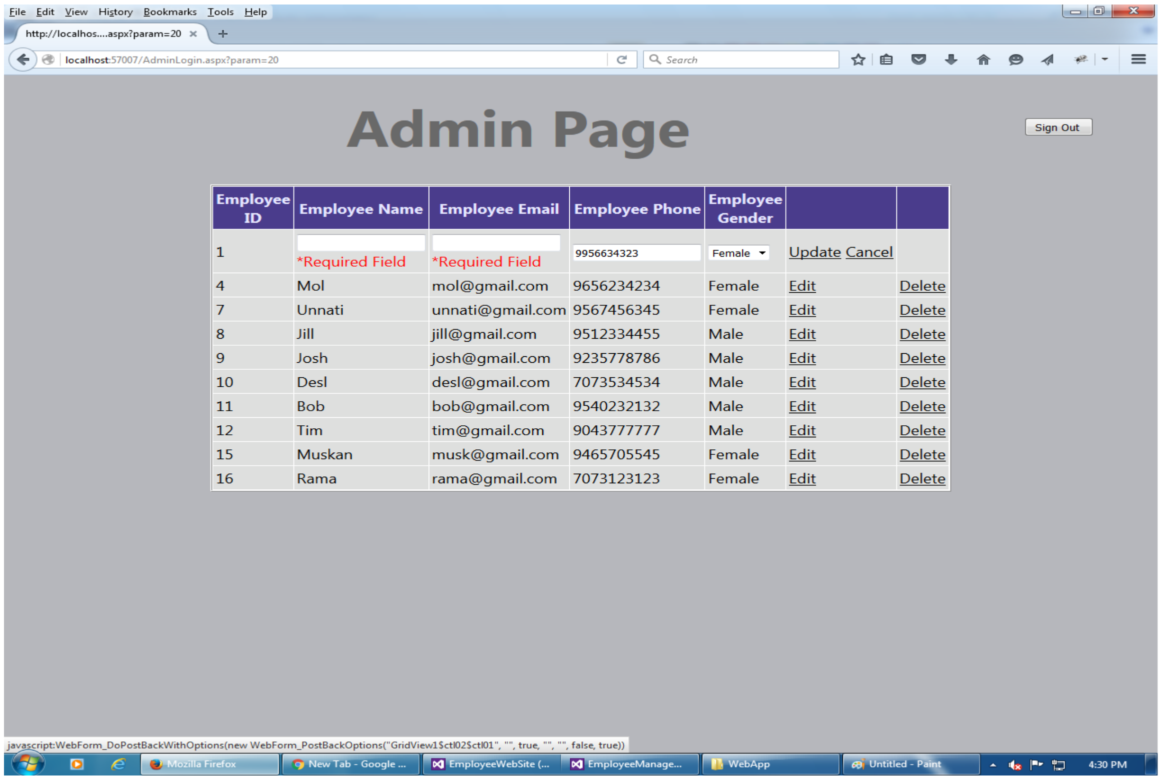
Figure 45: Login Page



Admin Page

Employee ID	Employee Name	Employee Email	Employee Phone	Employee Gender		
1	Vedaanti	vedaantibaliga@gmail.com	9956634323	Female	Edit	Delete
4	Mol	mol@gmail.com	9656234234	Female	Edit	Delete
7	Unnati	unnati@gmail.com	9567456345	Female	Edit	Delete
8	Jill	jill@gmail.com	9512334455	Male	Edit	Delete
9	Josh	josh@gmail.com	9235778786	Male	Edit	Delete
10	Desl	desl@gmail.com	7073534534	Male	Edit	Delete
11	Bob	bob@gmail.com	9540232132	Male	Edit	Delete
12	Tim	tim@gmail.com	9043777777	Male	Edit	Delete
15	Muskan	musk@gmail.com	9465705545	Female	Edit	Delete
16	Rama	rama@gmail.com	7073123123	Female	Edit	Delete

Figure 46: Admin Login



Admin Page

Employee ID	Employee Name	Employee Email	Employee Phone	Employee Gender		
1	*Required Field	*Required Field	9956634323	Female	Update Cancel	
4	Mol	mol@gmail.com	9656234234	Female	Edit	Delete
7	Unnati	unnati@gmail.com	9567456345	Female	Edit	Delete
8	Jill	jill@gmail.com	9512334455	Male	Edit	Delete
9	Josh	josh@gmail.com	9235778786	Male	Edit	Delete
10	Desl	desl@gmail.com	7073534534	Male	Edit	Delete
11	Bob	bob@gmail.com	9540232132	Male	Edit	Delete
12	Tim	tim@gmail.com	9043777777	Male	Edit	Delete
15	Muskan	musk@gmail.com	9465705545	Female	Edit	Delete
16	Rama	rama@gmail.com	7073123123	Female	Edit	Delete

```
javascript:WebForm_DoPostBackWithOptions(new WebForm_PostBackOptions("GridView1$ctl02$ctl01", "", true, "", "", false, true))
```

Figure 47: Admin updating validations

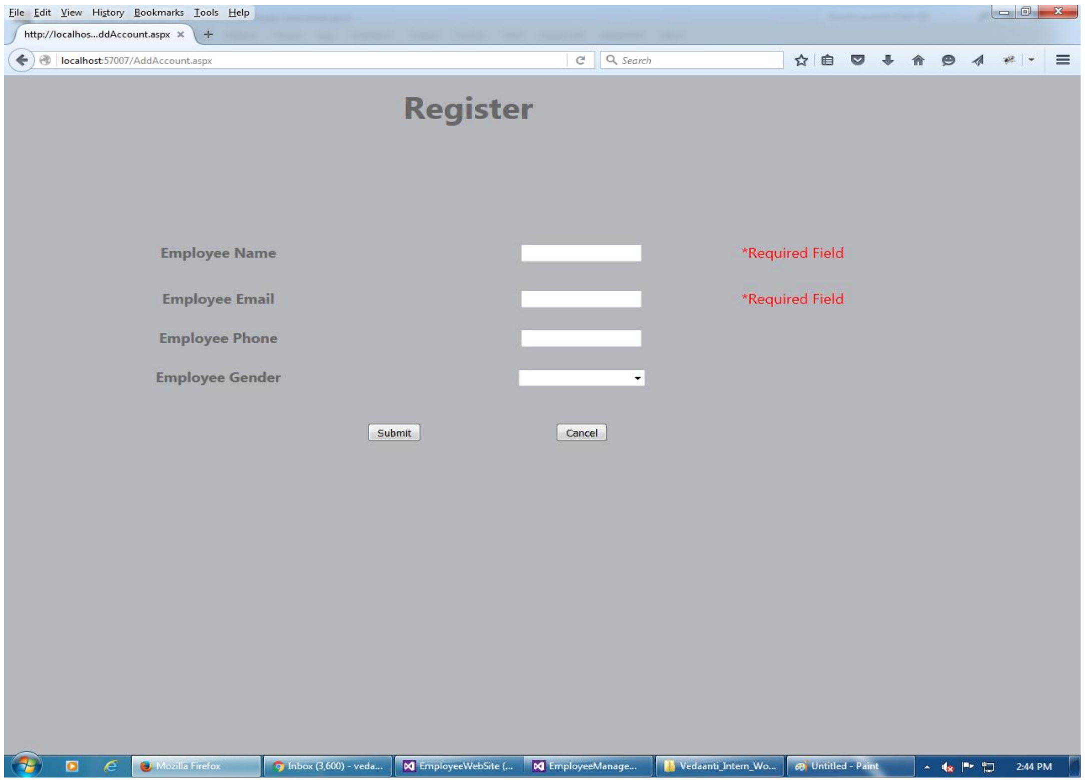


Figure 48: Register account

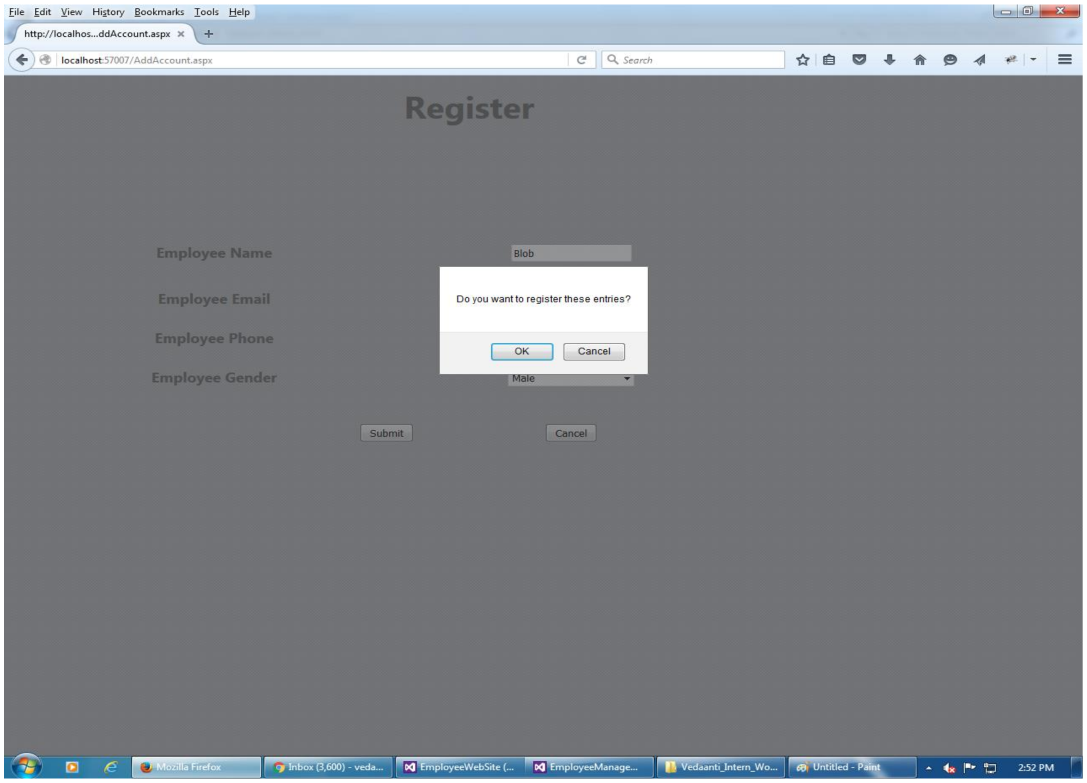
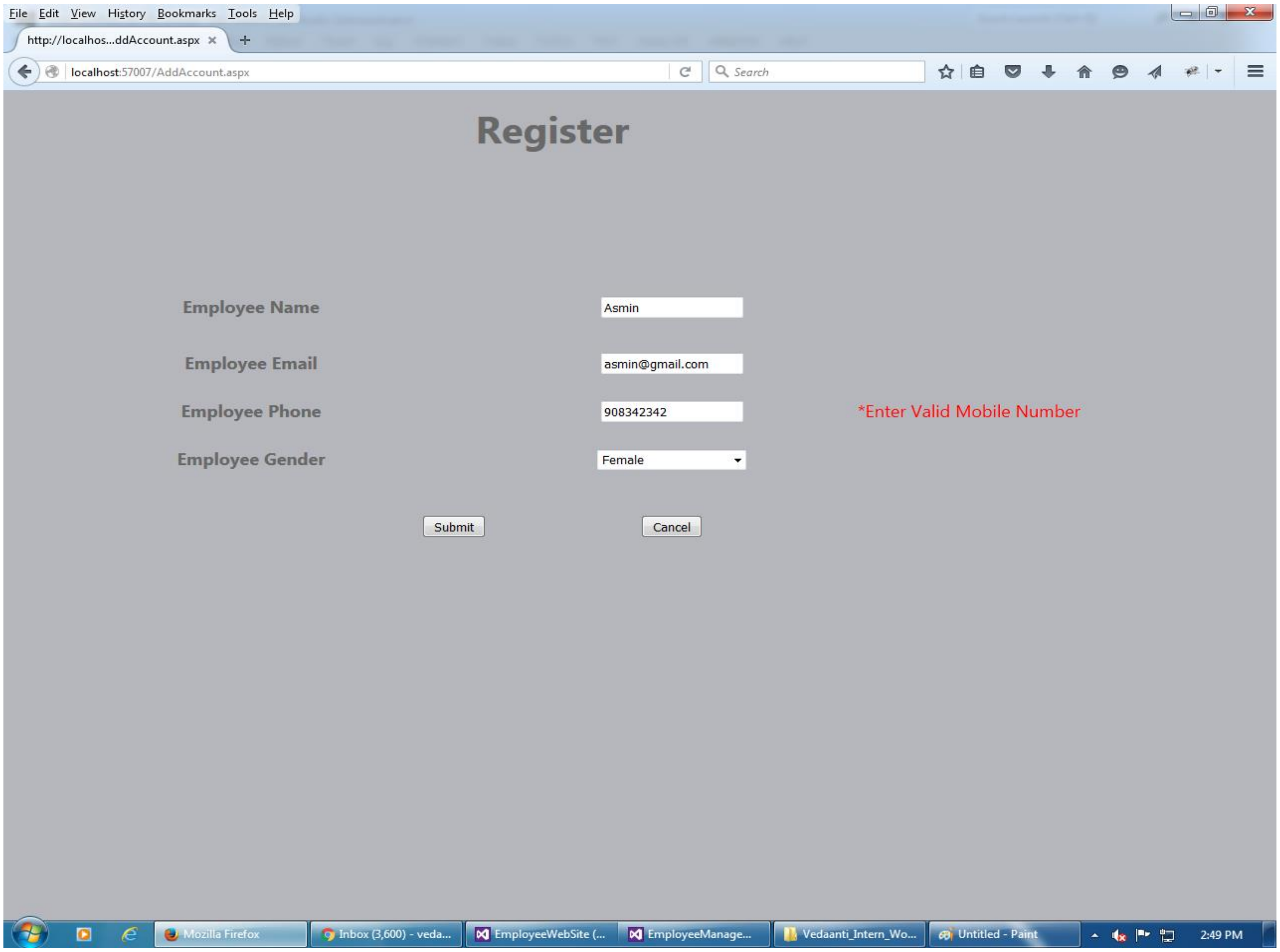


Figure 49: Register account confirmation

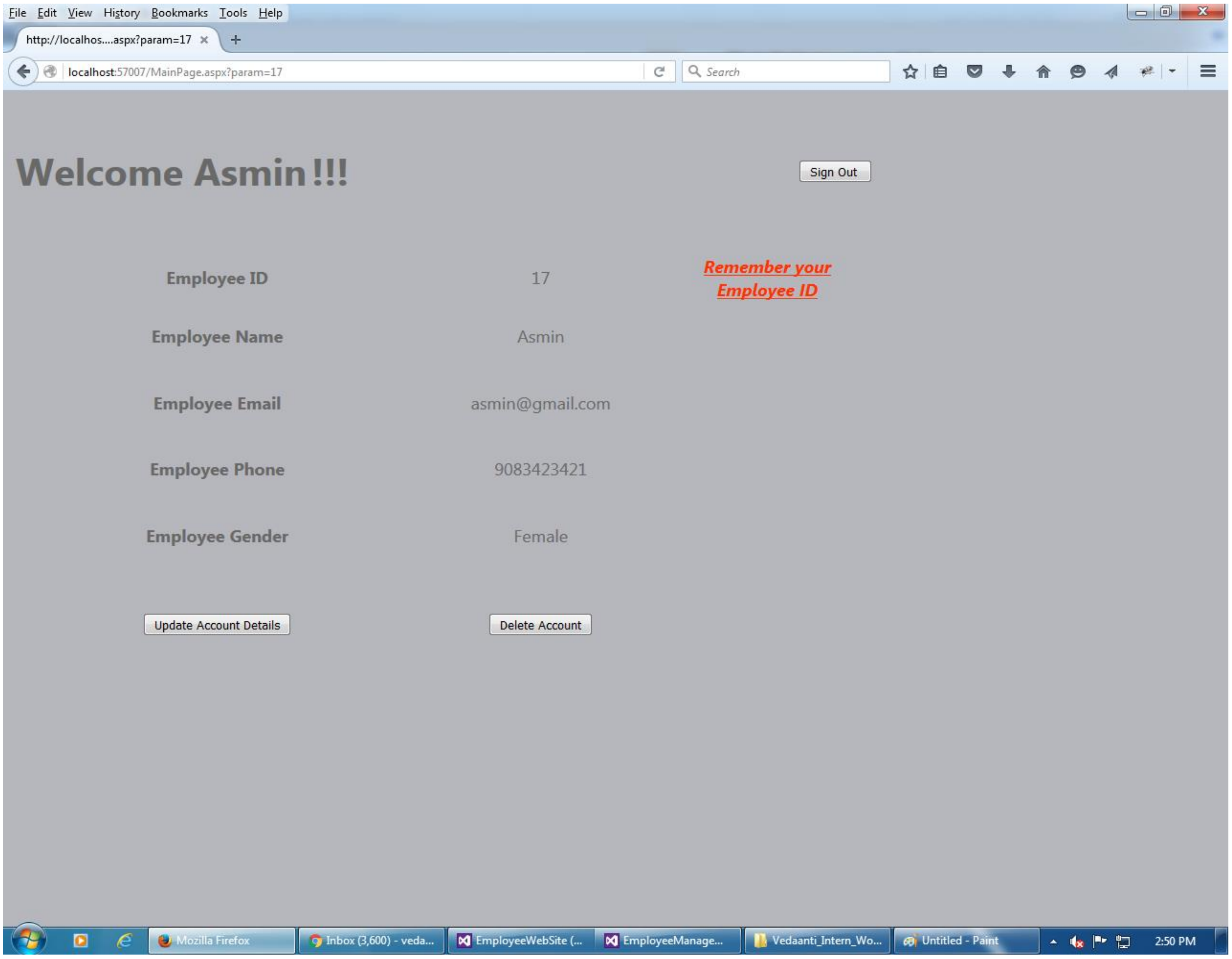


Register

Employee Name	Asmin
Employee Email	asmin@gmail.com
Employee Phone	908342342
Employee Gender	Female

*Enter Valid Mobile Number

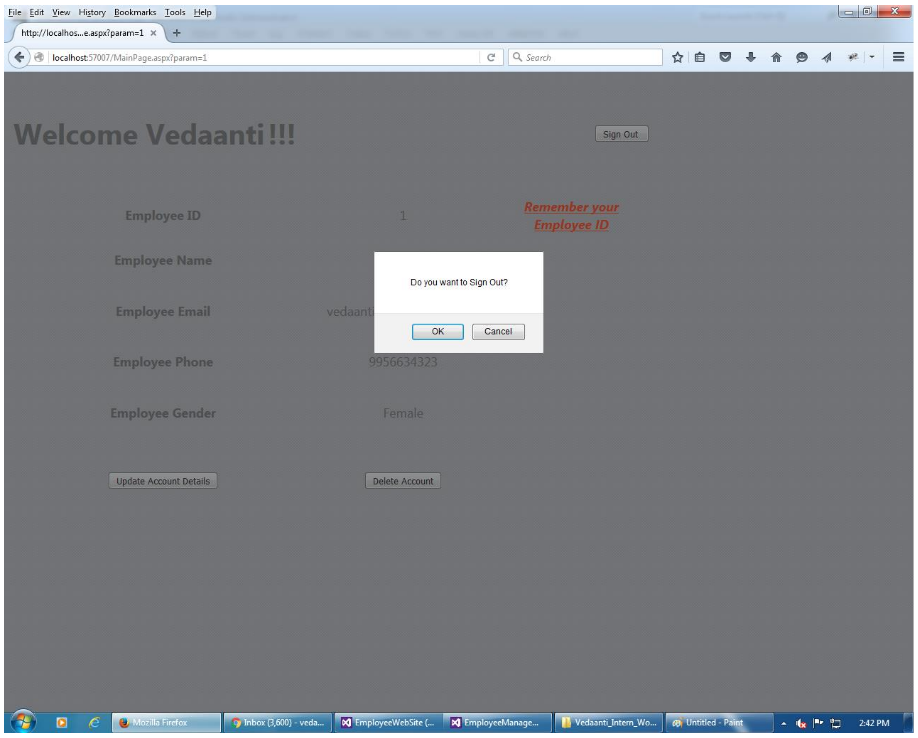
Figure 50: Phone number validation



Welcome Asmin !!!

Employee ID	17	Remember your Employee ID
Employee Name	Asmin	
Employee Email	asmin@gmail.com	
Employee Phone	9083423421	
Employee Gender	Female	

Figure 51: Registered Account

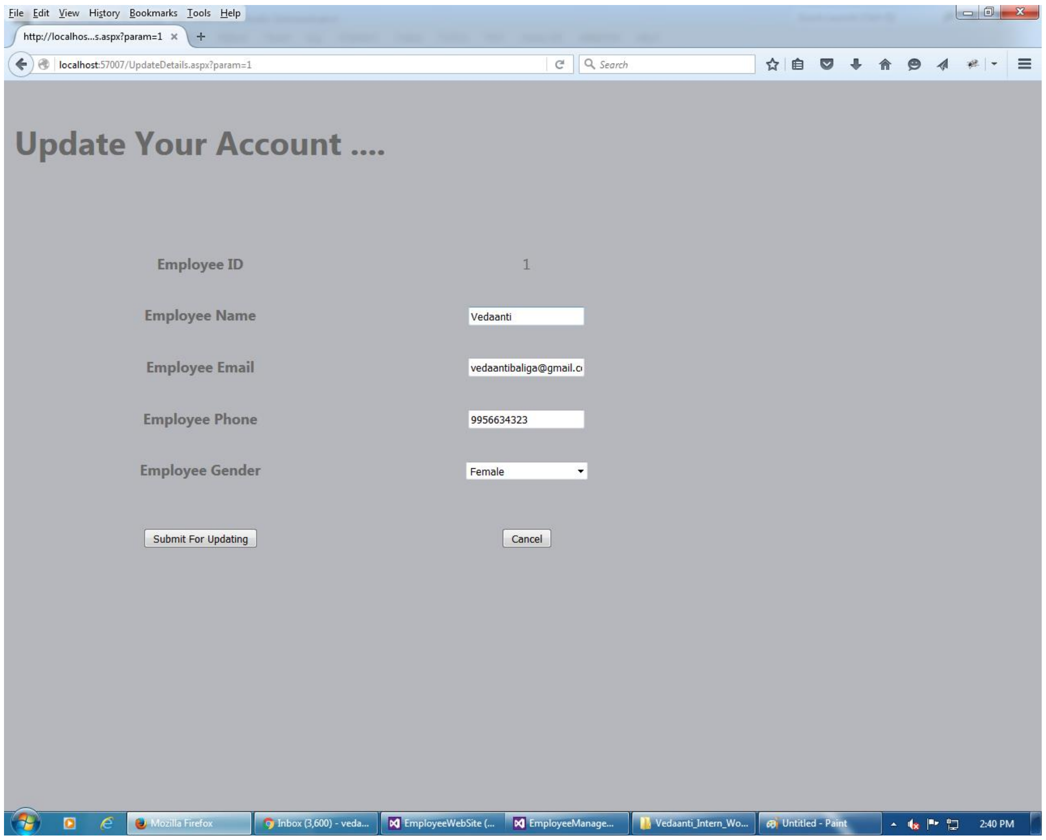


Welcome Vedaanti !!!

Employee ID	1	Remember your Employee ID
Employee Name		
Employee Email	vedaanti	
Employee Phone	9956634323	
Employee Gender	Female	

Do you want to Sign Out?

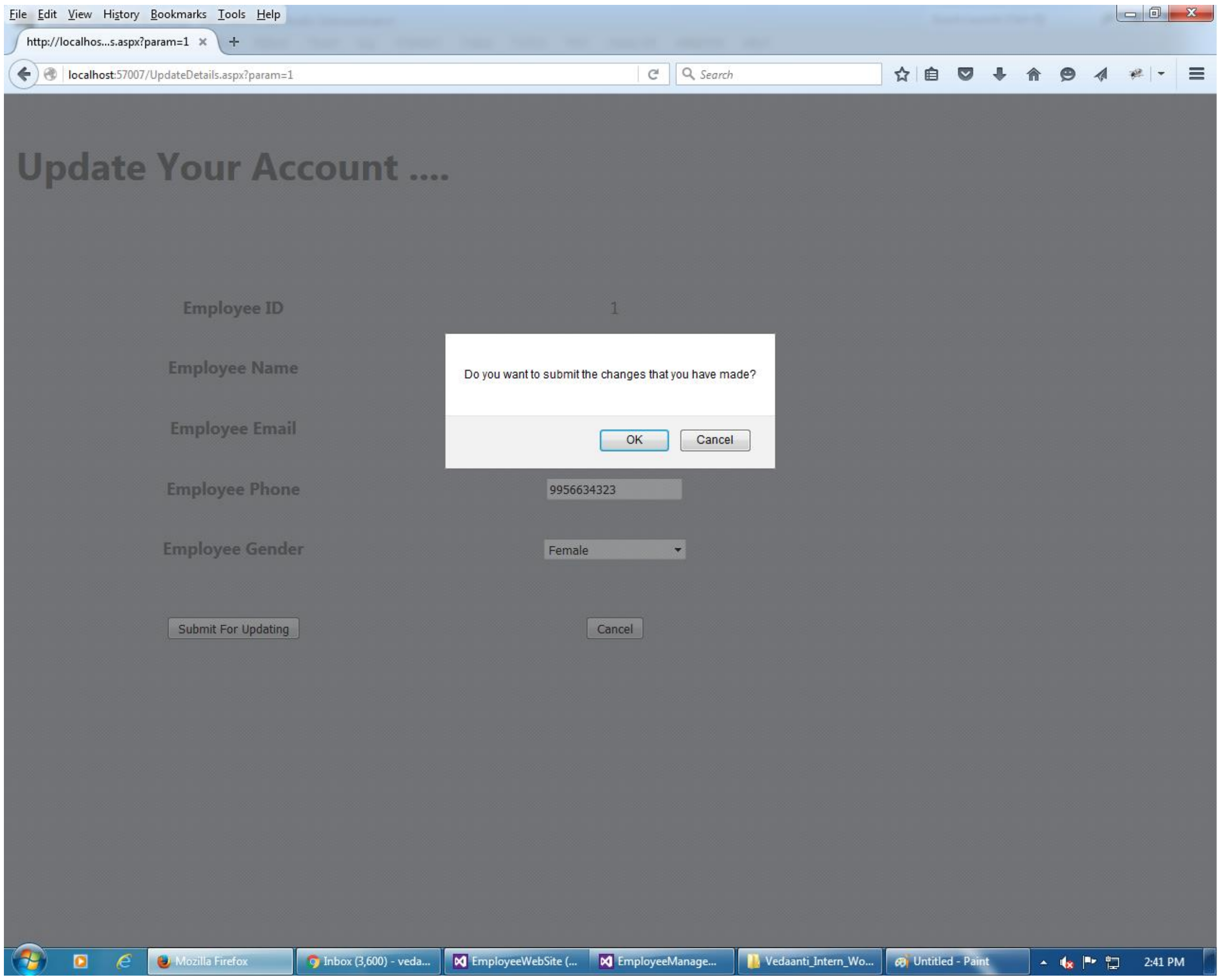
Figure 52: Sign out confirmation



Update Your Account ....

Employee ID	1
Employee Name	Vedaanti
Employee Email	vedaantibaliga@gmail.c
Employee Phone	9956634323
Employee Gender	Female

Figure 53: Updating Web Form



Update Your Account ....

Employee ID 1

Employee Name

Employee Email

Employee Phone 9956634323

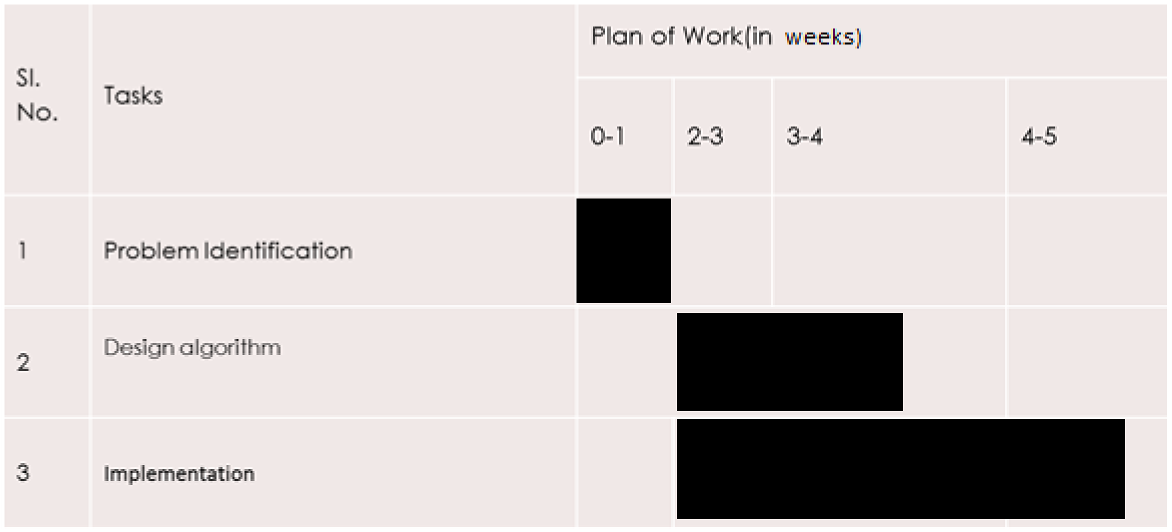
Employee Gender Female

Do you want to submit the changes that you have made?

Figure 54: Updating confirmation

CHAPTER 5: CONCLUSION

5.1 Progress Chart



Sl. No.	Tasks	Plan of Work(in weeks)			
		0-1	2-3	3-4	4-5
1	Problem Identification	■			
2	Design algorithm		■	■	
3	Implementation		■	■	■

5.2 Conclusion

To conclude, I have learnt the basics of .NET that is, creating windows forms, designing them, adding functionalities to them, like updating, deleting, adding, searching and filtering records. I have also learnt how to make web pages interactive and how to store data that they have, in the SQL in a safe and secure manner.

Along with learning C#, some basics of CSS, HTML and javascript, I have learnt their use in the real world that is, in the industry. I have also observed the project ‘Queue Management System’, which is developed on .NET.

I understood the proceedings and problems that took place while the project was going on. It was a good experience to see what kinds of software are required in the real world scenarios and how they are implemented.