**Telephone directory project**

**Introduction:**

This is a telephone directory project designed for managing and organizing contact information. It provides users with the ability to enter, store, update, and retrieve phone numbers and associated contact details. The system comprises four forms that enable users to perform various tasks, including adding new contacts, listing existing contacts, updating contact information, and deleting contacts. The application also utilizes a database to ensure data persistence and facilitate seamless data management. This report provides a detailed description and documentation of the project, including its functionality, forms, and data management. Additionally, an analysis of the project's benefits and features will be presented.

**Project Overview:**

The telephone directory project aims to provide users with a user-friendly interface for managing their contacts. The application consists of four forms that enable users to perform various tasks, such as adding new contacts, listing existing contacts, updating contact information, and deleting contacts. The project utilizes a database to ensure data persistence and facilitate seamless data management.

**Primary Form:**

The primary form serves as the central interface of the telephone directory project. It allows users to enter a phone number and provides options for further actions one of these options is deleting a number if the user entered wrong ,Also, Users can either add the entered phone number to the database or list all the contacts they have saved before. Additionally, the form provides the option to exit the application.

**Adding a Contact:**

If the user chooses to add a phone number, they are directed to a second form. This form allows the user to enter the contact's name, address, email, and select a category from a list (such as friend, family, favorite, or another person). The user has three options: save the entered information to the database, clear the form to enter new data, or exit the form without saving to be returned to the first form.

**Confirmation Form:**

After clicking the save button in the second form, a confirmation form appears to ensure the user's intention to save the contact. If the user confirms, the system displays a success message and proceeds to save the contact information in the database. The database consists of two tables: the first table stores the contact's ID (primary key), name, phone number, email, address, and category, while the second table maintains the category names along with their respective primary keys.

**Listing Contacts:**

The primary form features a "Contacts" button, which opens a fourth form responsible for displaying all the contacts saved in the database. This form provides several functionalities to manage contacts effectively:

Update: Users can modify the name, address, phone number, or category of a contact, allowing them to keep the contact information up to date.

Delete: Users can remove a contact's name, address, phone number, or category from the database, providing flexibility in managing their contacts.

Search: Users can search for specific contacts based on their name or any other relevant information. This feature streamlines the process of finding desired contacts.

Exit: Users can easily exit the contact management form when they have completed their desired tasks.

**Implementation Details:**

**Primary Form (frm\_contact):**

The primary form is the main interface of the application.

It includes buttons and text boxes for entering phone numbers and performing actions.

The form dynamically shows or hides buttons based on the presence of a phone number.

**frm\_contact :**

Inside this form, the methods unvis() and vis() are defined as private methods. These methods are used to show or hide buttons on the form.

The class has a public static string variable varnumber2, which is declared to store a phone number.

The frm\_contact\_Load method is an event handler for the form's load event. It gets executed when the form is loaded. In this case, it calls the unvis() method to hide buttons.

The class contains event handlers for various button click events such as button\_EXIT\_WOC\_Click, button\_ZERO\_WOC\_Click, button\_7\_WOC\_Click, etc. These event handlers are executed when the corresponding buttons on the form are clicked. They perform string concatenation on the txt\_NUMBER TextBox control based on the button clicked.

The event handler for the txt\_NUMBER\_TextChanged event is executed whenever the text in the txt\_NUMBER TextBox control changes. It checks if the text is empty and calls the unvis() method to hide buttons, or if it's not empty, it calls the vis() method to show buttons.

The event handler for the button\_DELETE\_NUMBER\_WOC\_Click event removes the last character from the txt\_NUMBER TextBox by creating a new string s1 and copying all characters except the last one from the existing text.

The event handler for the button\_ADD\_WOC\_Click event is executed when the button is clicked. It first validates the phone number entered in the txt\_NUMBER TextBox. If the number is invalid, it displays an error message using MessageBox.Show(). Otherwise, it assigns the phone number to the varnumber2 variable, creates an instance of the information\_form form, and displays it as a dialog box using form2.ShowDialog().

The event handler for the button\_CONTACT\_WOC\_Click event creates an instance of the Contacts\_form form and displays it as a dialog box using contacts\_Form.ShowDialog()

**Second Form (information\_form):**

This form is displayed when the user chooses to add a new contact.

It includes text boxes for entering contact details such as name, address, email, and a list box for selecting the category.

The form provides options to save the contact, clear the form, or exit without saving.

Inside the information\_form class, several variables are declared as public and static. These variables are used to store contact information such as varnumber, varname, varemail, varaddress, and varcategory.

The information\_form\_Load method is an event handler that is triggered when the form is loaded. It sets the text of the txt\_NUMBER textbox to the value of the varnumber2 variable, which is likely a value passed from another form.

The button\_WOC2\_Click method is an event handler for the click event of the "Close" button. It simply closes the form.

The button\_CLEAR\_Click method is an event handler for the click event of the "Clear" button. It clears the text of several textboxes (txt\_NUMBER, txt\_NAME, txt\_ADDRESS, txt\_EMAIL).

The button\_SAVE\_Click method is an event handler for the click event of the "Save" button. It retrieves the values entered in the textboxes (txt\_NAME, txt\_NUMBER, txt\_EMAIL, txt\_ADDRESS) and assigns them to the corresponding variables (varname, varnumber, varemail, varaddress). It also assigns the selected item from the listbox\_CATEGORY listbox to the varcategory variable. Then, it hides the current form (this.Hide()) and creates an instance of the SAVE\_FORM form, and shows it as a dialog box (sAVE\_FORM.ShowDialog()).

The listbox\_CATEGORY\_SelectedIndexChanged method is an event handler that is triggered when the selected index of the listbox\_CATEGORY listbox changes.

**Confirmation Form (SAVE\_FORM):**

-This form is shown when the user clicks the save button in the second form.

-It prompts the user to confirm their intention to save the contact.

-The form includes buttons for "Yes" and "No" options.

-The code starts with some import statements that bring in necessary namespaces for working with SQL Server (using System.Data.SqlClient) The class contains a private field called "con" of type "SqlConnection". This field represents the connection to the SQL Server database and is initialized with the connection string "server=DESKTOP-P3FR6Q2\SQLEXPRESS;database=contacts;integrated security=true".

-The class has a constructor called "SAVE\_FORM" which is the form's initialization logic.

-The class also contains several event handler methods, such as "SAVE\_FORM\_Load", "button\_NO\_Click", "button\_YES\_Click", and "label2\_Click". These methods handle various events triggered by user interactions with the form's controls.

-The "button\_NO\_Click" method is executed when the user clicks the "NO" button on the form. It creates a new instance of the "information\_form" class, closes it, and then closes the current form.

-The "button\_YES\_Click" method is executed when the user clicks the "YES" button on the form. It prepares an SQL query string to insert data into the "contactphonebookk" table of the "contacts" database. The query includes placeholders (@fullname, @phonenumber, @email, @address, @category). A new instance of the "SqlCommand" class is created with the SQL query and the connection object ("con") as arguments.The parameters for the query are set using the "Parameters.AddWithValue" method.

-The parameter values are taken from the "varname", "varnumber", "varemail", "varaddress", and "varcategory" variables of the "information\_form" class. The connection to the database is opened using the "con.Open()" method.The query is executed against the database using the "sqlcmd.ExecuteNonQuery()" method. The connection is closed using the "con.Close()" method.

-A message box is displayed to the user indicating that the data has been added successfully. The "Close" method is called on the current form to close it.

**Fourth Form (Contacts\_form):**

-This form is opened when the user clicks the "Contacts" button in the primary form.

-It displays a data grid view that shows all the contacts saved in the database.

-Buttons are provided for updating, deleting, searching, and exiting the form.

-The "txt\_DATA\_TextChanged" event handler is called when the text in a text box named "txt\_DATA" changes. It filters the rows in the DataGridView based on the value in the text box. It sets the RowFilter property of the DefaultView of the DataGridView's DataSource to a filter expression that checks if the "fullname" column contains the entered text.

-The "updata\_BUTTON\_Click" event handler is called when the user clicks the update button. It iterates over each row in the DataGridView and constructs a SqlCommand object to update the corresponding contact in the "contactphonebookk" table. The values from the DataGridView are used as parameters for the update query. The connection is opened, the command is executed, and the connection is closed. After updating all rows, a message box is displayed indicating the successful update.

**Database Connectivity:**

-The project connects to a SQL Server database using a connection string.

-The SqlConnection class is used to establish a connection with the database.

-SqlCommand objects are used to execute SQL queries and commands for data retrieval, insertion, update, and deletion.

-SqlDataAdapter is used to fill a DataTable with data from the database, which is then bound to the data grid view.