

# **Application Development**

## **CS6004ES**

### **Individual coursework(2024/25)**

Name	Janarthana Kalaikumar
LMU ID	23057745
Esoft ID	E217463

# Table of Contents

Table of Figures .....	3
1. Introduction.....	4
2. Existing system and proposed solution.....	5
3. System requirements .....	6
3.1 Functional Requirements .....	6
3.2 Non functional requirements.....	7
3.3 Software requirements .....	7
3.4 Hardware requirements .....	8
4. System Design .....	9
4.1 Architecture diagram .....	9
4.2 Usecase diagram .....	10
5. User Interfaces .....	11
5.1 Login and Registration.....	11
5.2 Admin dashboard .....	13
5.3 Customer dashboard.....	19
6. Code Snippets .....	22
6.1 Classes.....	23
6.2 Login and Registration.....	25
6.2.1 Login .....	25
6.2.2 Customer Registration .....	27
6.3 Admin dashboard .....	30
6.3.1 User Management functions .....	30
6.3.2 Car Management functions .....	33
6.3.3 Part Management functions .....	35
6.3.4 Car order management functions .....	38
6.3.5 Part order management functions .....	40
6.4 Customer dashboard.....	42
6.4.1 Placing orders.....	42

6.4.2	Tracking order status.....	44
7.	Instructions to run the application.....	45
7.1	Instructions for Admin .....	45
7.2	Instructions for Customer .....	46
8.	Self-reflection on using C# and Visual Studio.....	47
8.1	Experience with using C# .....	48
8.2	Experience with using Visual Studio .....	48
9.	Conclusion .....	49

## Table of Figures

Figure 1 – System Architecture Diagram.....	9
Figure 2 - Usecase Diagram.....	10
Figure 3 - Login Page .....	11
Figure 4 - Customer registration form .....	12
Figure 5 - Admin Dashboard Customer Management .....	13
Figure 6 - Car Management .....	14
Figure 7 - Part management.....	15
Figure 8 - Car Order Management.....	16
Figure 9 - Part Order Management .....	17
Figure 10 - Admin Report Generation .....	18
Figure 11- Customer Car Order Page .....	19
Figure 12 - Customer Part Order Page.....	20
Figure 13 - Track Order Status.....	21

# 1. Introduction

In today's competitive market, maintaining efficient operations and delivering good customer service are important for success of this business. ABC Car Traders is a supplier of high-quality vehicles and transport solutions. The business seeks to improve its operational activities and customer satisfaction through the development of a comprehensive software system. This system aims to automate various aspects of their business, including car and parts management, customer interactions, and order processing.

The main goal of this project is to design and implement a software application that supports the operational needs of ABC Car Traders. The software will provide a user-friendly interface for both administrators and customers, ensuring that the business processes are managed effectively and that customer needs are met properly.

The main objectives of this project are to implement Admin functionality and Customer functionality.

- The administrator will be able to manage customer information, manage car and parts details, edit and track order information and generate reports related to orders and sales of cars and parts.
- The customer will be able to search for cars and car parts and place orders using this application. The customers can register themselves, but they do not have permission to edit their details and the details of the order after they are placed.

The software is developed using an object-oriented approach. The development of this software includes using Windows forms for windows user interface, C# as the coding language and Visual studio as the Integrated Development Environment (IDE). This application will have a well-structured database to store and manage application data.

This report contains the development process, detailed instructions to run the application, the architecture of the application, code snippets of functions and descriptions of relevant classes implemented. This document also contains self-reflection on using object-oriented approach, C#, Visual Studio, Windows Forms Application and SQL for the development of this comprehensive software application for ABC Car Traders.

## 2. Existing system and proposed solution

ABC car traders have a manual system for managing the operational activities of the business. The service center employees of ABC Car Traders provide information and consultation to customers to help them choose the right vehicle or parts for their needs. The employees also manually track the inventory and order details. The existing system requires a large quantity of labor, time and money and the possibility of errors can be high. The proposed solution aims to automate the major business operations of the business to improve the overall efficiency of the organization.

The major operations of ABC Car Traders are:

- Administrator
  - Manage car and parts details
  - Manage customer details
  - Manage order details
  - Generate reports
- Customer
  - Search Car and Parts
  - Place orders
  - Track the status of the order

The above-mentioned operational activities of the business will be implemented through the proposed solution. The proposed solution will improve the overall efficiency of the operational activities of the business. The software application will also improve customer satisfaction, improving the goodwill of the business.

## 3. System requirements

### 3.1 Functional Requirements

The functional requirements of this software project are:

- Login functionality
- Admin Functionality
  - Manage customer details
    - View customer profile
    - Update customer records
    - Delete customer records
  - Manage car details
    - View car information
    - Add car details
    - Update car details
    - Delete car details
  - Manage parts details
    - View part information
    - Add part details
    - Update part details
    - Delete part details
  - Manage car order details
    - View car orders
    - Update car orders
    - Delete car orders
  - Manage part order details
    - View part orders
    - Update part orders
    - Delete part orders
  - Generate reports

- Customer functionality
  - Customer registration
  - Browse car details
  - Browse part details
  - Place car/part order
  - Track order details and status
- System functionality
  - Authenticate users
  - Data Management
  - Responsive and user friendly interface
  - Error handling

### **3.2 Non functional requirements**

- Usability
- Reliability
- Scalability
- Security
- Maintainability
- Data Integrity

### **3.3 Software requirements**

- Operating System: Windows 10 or higher
- Development Environment: Visual Studio 2015 or later
- Database: SQL Server 2016 or higher
- Programming Language: C# .NET Framework 4.5 or higher
- Version Control: Git or any version control system
- Libraries/Frameworks: Entity Framework, Windows Forms or WPF

### **3.4 Hardware requirements**

- Processor: Intel Core i5 or higher
- RAM: 8 GB or higher
- Storage: 256 GB SSD or higher



## 4. System Design

### 4.1 Architecture diagram

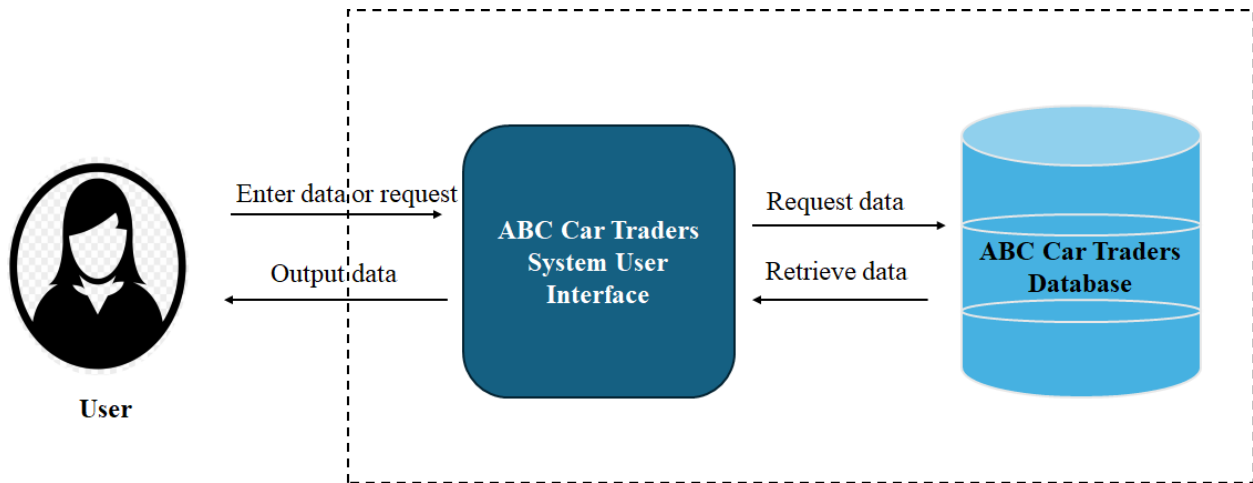


Figure 1 – System Architecture Diagram

The above architecture diagram depicts the user interaction with the system and the way the system interface interacts with the database of the software application. This application has a two-tier architecture where the client interacts directly with the server.

## 4.2 Usecase diagram

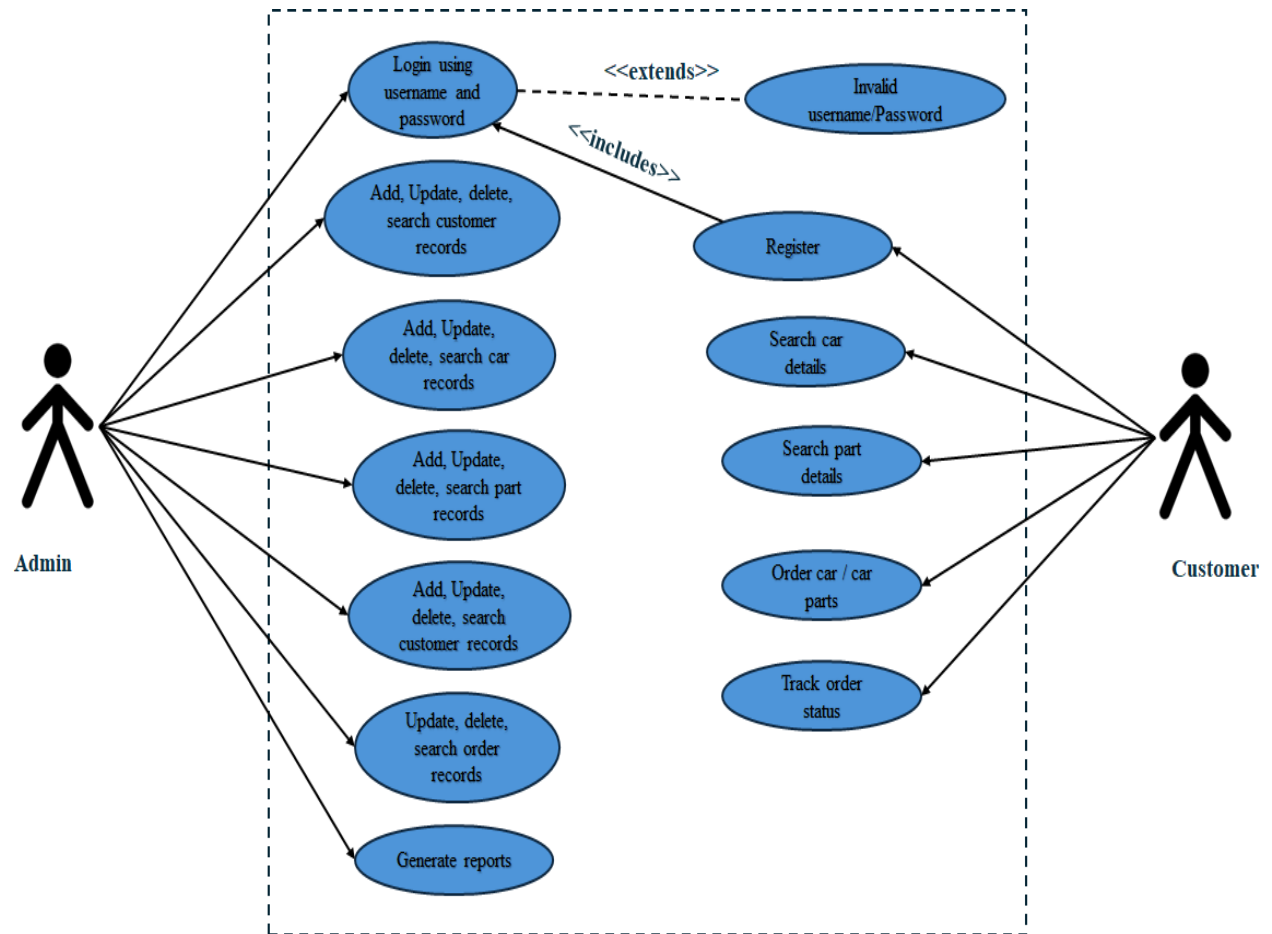


Figure 2 - Usecase Diagram

## 5. User Interfaces

### 5.1 Login and Registration

Login

ABC TRADERS

LOGIN


Username

Password

User Type

LOGIN

Not a user? [Register](#)






Figure 3 - Login Page

Registration Form

## ABC TRADERS

### REGISTRATION

First Name  Last Name

NIC  Email

Address

Username  Password

Already a registered user? [Login](#)





Figure 4 - Customer registration form

## 5.2 Admin dashboard

The screenshot shows a web application window titled "Admin Home". The main header displays "ABC COMPANY" in a large, bold, serif font. To the right of the header is a "LOGOUT" button. Below the header is a navigation bar with tabs: "Customer", "Car Details", "Part Details", "Car Orders", "Parts Order", and "Reports". The "Customer" tab is currently selected. The main content area contains a form for managing customer details. It starts with a label "Enter User ID to edit customer details" followed by a text input field. Below this are several input fields for "First Name", "Last Name", "NIC", "Email", "Address", "Username", and "Password". At the bottom of the form are three buttons: "REGISTER", "UPDATE", and "DELETE". Below these buttons is a search section with a text input field and a "SEARCH" button. At the very bottom of the form area is a large, empty gray rectangular box.

Admin Home

**ABC COMPANY** **LOGOUT**

**Customer** Car Details Part Details Car Orders Parts Order Reports

Enter User ID to edit customer details

First Name  Last Name

NIC  Email

Address

Username  Password

**REGISTER** **UPDATE** **DELETE**

**SEARCH**

Figure 5 - Admin Dashboard Customer Management

Admin Home

ABC COMPANY

LOGOUT

Customer Car Details Part Details Car Orders Parts Order Reports

Brand  Model

Year  Colour

Price  Body Type

Car ID

**\*Delete or update car records using Car ID**

ADD UPDATE DELETE

SEARCH

Figure 6 - Car Management

Admin Home

ABC COMPANY

LOGOUT

Customer Car Details Part Details Car Orders Parts Order Reports

Brand  Category

Colour  Size

Warranty(Years)  Price

Part ID

**\*Delete or update car records using Car ID**

ADD UPDATE DELETE

SEARCH

Figure 7 - Part management

Admin Home

ABC COMPANY

LOGOUT

Customer Car Details Part Details Car Orders Parts Order Reports

Order ID  Car ID

CustomerID  Date

Status

UPDATE DELETE

SEARCH

Figure 8 - Car Order Management



Admin Home

ABC COMPANY

LOGOUT

Customer Car Details Part Details Car Orders **Parts Order** Reports

Order ID  Part ID

Customer ID  Date

Status

Figure 9 - Part Order Management

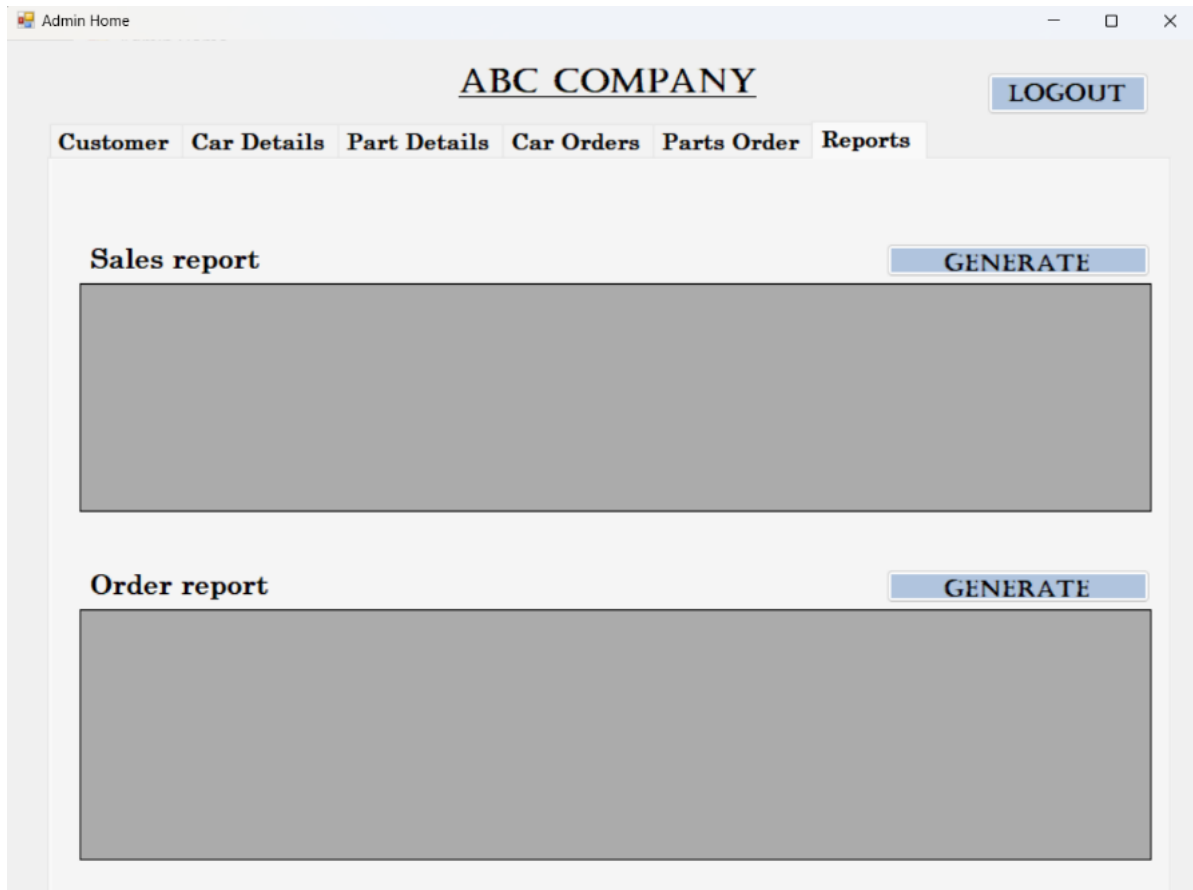


Figure 10 - Admin Report Generation

## 5.3 Customer dashboard

The screenshot shows a web application window titled "Customer Home". The main header displays "ABC COMPANY" in a large, bold, serif font. To the right of the header is a "LOGOUT" button. Below the header, there are three tabs: "Place Car Order" (which is active), "Place Part Order", and "Track Order Status". The main content area has a heading "\*Seach for cars using model, Brand, price etc." followed by a search input field and a "SEARCH" button. Below this is a large, empty rectangular box, likely for search results. At the bottom, there are input fields for "Order ID", "Customer ID", and "Status" (a dropdown menu). To the right of these are input fields for "Car ID" and "Date" (a dropdown menu showing "Friday"). A "PLACE ORDER" button is located at the bottom center.

Customer Home

ABC COMPANY

LOGOUT

Place Car Order Place Part Order Track Order Status

\*Seach for cars using model, Brand, price etc.

SEARCH

Order ID

Customer ID

Status

Car ID

Date Friday

PLACE ORDER

Figure 11- Customer Car Order Page

Customer Home

ABC COMPANY

LOGOUT

Place Car Order Place Part Order Track Order Status

\*Seach for parts using model, Brand, price etc.

SEARCH

Order ID  Part ID

Customer ID  Date Tuesday , ▾

Status  ▾

PLACE ORDER

Figure 12 - Customer Part Order Page

Customer Home

**ABC COMPANY**

LOGOUT

Place Car Order Place Part Order **Track Order Status**

**\*Track order status using Customer Id or Order ID**

Order ID  Customer ID

Parts/Car??

SEARCH

Figure 13 - Track Order Status

## 6. Code Snippets and Algorithms

### 6.1 Searching Algorithms

In the ABC Car Traders software, search functionality is critical for both the admin and customer interfaces. This functionality allows users to quickly find cars and car parts based on specific criteria. The project utilizes linear search algorithm, tailored to the data structures and requirements of the application.

#### 1. Linear Search

Linear search is used when searching for specific items within a list or collection, such as searching for a car by its model name or finding a car part by its part number. This algorithm is simple and effective for small to medium-sized datasets where a quick, straightforward search is needed.

#### How Linear Search Works:

- The algorithm iterates through each item in the list, comparing it with the search query.
- If a match is found, the search stops, and the item is returned.
- If no match is found after checking all items, the search concludes without a result.

#### Example Use Case:

- Searching for a car by its model name in the list of available cars.
- Searching for a car part by its ID in the parts inventory.

#### Advantages:

- Easy to implement.
- Works well for small datasets.

#### Disadvantages:

- Inefficient for large datasets as the time complexity is  $O(n)$ , where  $n$  is the number of items.

## 6.2 Classes

### *User*

```
public class User
{
    public int UserId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string NIC { get; set; }
    public string Email { get; set; }
    public string Address { get; set; }
    public string Username { get; set; }
    public string Password { get; set; }
    public int UserType { get; set; }
}
```

### *Car*

```
public class Cars
{
    public int CarID { get; set; }
    public string Model { get; set; }
    public string Brand { get; set; }
    public string Colour { get; set; }
    public string Year { get; set; }
}
```

```
    public string BodyType { get; set; }

    public decimal Price { get; set; }

}
```

### **Parts**

```
public class Parts
{
    public int PartID { get; set; }

    public string Brand { get; set; }

    public string Category { get; set; }

    public string Colour { get; set; }

    public string Size { get; set; }

    public int Warranty { get; set; }

    public decimal Price { get; set; }

}
```

### **Car order**

```
public class CarOrder
{
    public int OrderID { get; set; }

    public int CarID { get; set; }

    public DateTime OrderDate { get; set; }

}
```



```

    public int UserID { get; set; }

    public string Status { get; set; }
}

```

### **Part order**

```

public class PartOrder
{
    public int OrderID { get; set; }

    public int PartID { get; set; }

    public DateTime OrderDate { get; set; }

    public int UserID { get; set; }

    public string Status { get; set; }
}

```

## **6.3 Login and Registration**

### **6.3.1 Login**

```

namespace ABC_Traders
{
    public partial class LoginForm : Form
    {

        SqlConnection conn = new SqlConnection(@"Data
Source=MALIFICENT;Initial Catalog=ABCCarTraders;Integrated
Security=True;TrustServerCertificate=True");

        public LoginForm()
        {
            InitializeComponent();
        }

        private void LoginForm_Load(object sender, EventArgs e)

```

```

    {

    }

    private void btnLogin_Click(object sender, EventArgs e)
    {

        try
        {
            conn.Open();

            SqlCommand cmd = new SqlCommand("SELECT username, password,
user_type FROM [user] WHERE username = '" + txtBoxUsername.Text + "' AND
password = '" + txtBoxPassword.Text + "' AND user_type =
'" + comboBoxUserType.SelectedIndex + "'", conn);

            SqlDataAdapter sda = new SqlDataAdapter(cmd);
            DataTable dtable = new DataTable();
            sda.Fill(dtable);

            if (dtable.Rows.Count > 0)
            {
                //open relevant user home pages
                if (comboBoxUserType.SelectedIndex == 0) //Customer
                {
                    CustomerHomeForm customerHomeForm = new
CustomerHomeForm();

                    customerHomeForm.Show();
                    this.Hide();
                }
                else if (comboBoxUserType.SelectedIndex ==
1) //Administrator
                {
                    AdminHomeForm adminHomeForm = new AdminHomeForm();
                    adminHomeForm.Show();
                    this.Hide();
                }
            }
            else
            {

```

```

        MessageBox.Show("Invalid login
credentials","Error",MessageBoxButtons.OK, MessageBoxIcon.Error);
        txtBoxUsername.Clear();
        txtBoxPassword.Clear();
    }
}

catch(Exception ex)
{
    MessageBox.Show(ex.Message,"Error",MessageBoxButtons.OK,
MessageBoxIcon.Error);
    txtBoxUsername.Clear();
    txtBoxPassword.Clear();
}
finally
{
    conn.Close();
}

}

private void linkLblRegister_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
{
    RegistrationForm registrationform = new RegistrationForm();
    registrationform.Show();
    this.Hide();
}
}
}

```

### 6.3.2 Customer Registration

```

public partial class RegistrationForm : Form
{
    SqlConnection conn = new SqlConnection(@"Data Source=MALIFICENT;Initial
Catalog=ABCCarTraders;Integrated Security=True;TrustServerCertificate=True");

    public RegistrationForm()
    {

```

```

        InitializeComponent();
    }

    private void RegistrationForm_Load(object sender, EventArgs e)
    {

    }

    private void btnRegister_Click(object sender, EventArgs e)
    {
        try
        {
            if (txtBoxFirstName.Text != "" && txtBoxlastName.Text != "" &&
txtBoxNIC.Text != "" && txtBoxEmail.Text != "" && txtBoxAddress.Text != "" &&
txtBoxUsername.Text != "" && txtBoxPassword.Text != "")
            {
                int v = check(txtBoxEmail.Text);
                if (v == 0)
                {
                    conn.Open();
                    SqlCommand cmd = new SqlCommand("INSERT INTO [user]
VALUES(@First_name, @Last_name, @NIC, @email, @address, @username, @password,
@user_type)", conn);

                    cmd.Parameters.AddWithValue("@First_name",
txtBoxFirstName.Text);
                    cmd.Parameters.AddWithValue("@Last_name",
txtBoxlastName.Text);
                    cmd.Parameters.AddWithValue("@NIC", txtBoxNIC.Text);
                    cmd.Parameters.AddWithValue("@email", txtBoxEmail.Text);
                    cmd.Parameters.AddWithValue("@address",
txtBoxAddress.Text);
                    cmd.Parameters.AddWithValue("@username",
txtBoxUsername.Text);
                    cmd.Parameters.AddWithValue("@password",
txtBoxPassword.Text);
                    cmd.Parameters.AddWithValue("@user_type", 1);
                    cmd.ExecuteNonQuery();
                    conn.Close();
                    MessageBox.Show("Registered Successfully");
                }
            }
        }
    }

```

```

        txtBoxFirstName.Clear();
        txtBoxlastName.Clear();
        txtBoxNIC.Clear();
        txtBoxEmail.Clear();
        txtBoxAddress.Clear();
        txtBoxUsername.Clear();
        txtBoxPassword.Clear();

        conn.Close();

    }
    else
    {
        MessageBox.Show("You are already registered as a
user.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
}
else
{
    MessageBox.Show("Fill all the fields!!", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
}
}
catch(Exception ex)
{
    MessageBox.Show(ex.Message);
}
}

int check(string email)
{
    conn.Open();
    string query = "SELECT COUNT(*) FROM [user] WHERE email =' " +
txtBoxEmail + "'";
    SqlCommand cmd = new SqlCommand(query, conn);
    int v= (int)cmd.ExecuteScalar();
    conn.Close();
    return v;
}

```

```

        private void linkLblLogin_LinkClicked(object sender,
LinkLabelLinkClickedEventArgs e)
        {
            LoginForm loginForm = new LoginForm();
            loginForm.Show();
            this.Hide();
        }
    }
}

```

## 6.4 Admin dashboard

### 6.4.1 User Management functions

#### **User Register function**

```

public bool RegisterUser(User user)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = "INSERT INTO [user] (First_name, Last_name, NIC,
email, address, username, password, user_type) VALUES (@First_name, @last_name,
@NIC, @email, @address, @username, @password, @user_type)";
        SqlCommand cmd = new SqlCommand(query, conn);
        cmd.Parameters.AddWithValue("@First_name", user.FirstName);
        cmd.Parameters.AddWithValue("@last_name", user.LastName);
        cmd.Parameters.AddWithValue("@NIC", user.NIC);
        cmd.Parameters.AddWithValue("@email", user.Email);
        cmd.Parameters.AddWithValue("@address", user.Address);
        cmd.Parameters.AddWithValue("@username", user.Username);
        cmd.Parameters.AddWithValue("@password", user.Password);
        cmd.Parameters.AddWithValue("@user_type", user.UserType);

        conn.Open();
        int result = cmd.ExecuteNonQuery();
        return result > 0;
    }
}

```

```

//Customer Search function
public DataTable SearchUsers(string searchQuery)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = @"SELECT user_id, First_name, Last_name, NIC,
email, address, username, password FROM [user]
WHERE First_name LIKE @searchQuery
OR Last_name LIKE @searchQuery
OR NIC LIKE @searchQuery
OR email LIKE @searchQuery
OR address LIKE @searchQuery
OR username LIKE @searchQuery";
        SqlCommand cmd = new SqlCommand(query, conn);
        cmd.Parameters.AddWithValue("@searchQuery", "%" + searchQuery +
"%");

        SqlDataAdapter adapter = new SqlDataAdapter(cmd);
        DataTable dataTable = new DataTable();
        adapter.Fill(dataTable);
        return dataTable;
    }
}

```

### **Customer update function**

```

public bool UpdateUser(User user)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = @"UPDATE [user]
SET First_name = @FirstName,
Last_name = @LastName,
NIC = @NIC,
email = @Email,
address = @Address,
username = @Username,
password = @Password
WHERE user_id = @UserID";
    }
}

```

```

        using (SqlCommand cmd = new SqlCommand(query, conn))
        {

            cmd.Parameters.AddWithValue("@UserID", user.UserId);
            cmd.Parameters.AddWithValue("@FirstName", user.FirstName);
            cmd.Parameters.AddWithValue("@LastName", user.LastName);
            cmd.Parameters.AddWithValue("@NIC", user.NIC);
            cmd.Parameters.AddWithValue("@Email", user.Email);
            cmd.Parameters.AddWithValue("@Address", user.Address);
            cmd.Parameters.AddWithValue("@Username", user.Username);
            cmd.Parameters.AddWithValue("@Password", user.Password);

            conn.Open();
            int rowsAffected = cmd.ExecuteNonQuery();
            return rowsAffected > 0;
        }
    }
}

```

### **Customer delete function**

```

public bool DeleteCustomer(int userId)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = "DELETE FROM [user] WHERE user_id = @userId";
        using (SqlCommand cmd = new SqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("@userId", userId);

            conn.Open();
            int rowsAffected = cmd.ExecuteNonQuery();
            return rowsAffected > 0;
        }
    }
}

```



## 6.4.2 Car Management functions

### Add new car details

```
public bool AddCar(Cars car)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = "INSERT INTO [cars] (model, brand, colour, year,
body_type, price) VALUES (@model, @brand, @colour, @year, @body_type, @price)";
        using (SqlCommand cmd = new SqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("@model", car.Model);
            cmd.Parameters.AddWithValue("@brand", car.Brand);
            cmd.Parameters.AddWithValue("@colour", car.Colour);
            cmd.Parameters.AddWithValue("@year", car.Year);
            cmd.Parameters.AddWithValue("@body_type", car.BodyType);
            cmd.Parameters.AddWithValue("@price", car.Price);

            conn.Open();
            int result = cmd.ExecuteNonQuery();
            return result > 0;
        }
    }
}

public bool CarExists(string carId)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = "SELECT COUNT(*) FROM [cars] WHERE car_id = @carId";
        using (SqlCommand cmd = new SqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("@carId", carId);

            conn.Open();
            int count = (int)cmd.ExecuteScalar();
            return count > 0;
        }
    }
}
```

### **Update car details**

```
public bool UpdateCar(Cars car)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = @"UPDATE [cars]
            SET model = @Model,
            brand = @Brand,
            colour = @Colour,
            year = @Year,
            body_type = @BodyType
        WHERE car_id = @CarID";

        using (SqlCommand cmd = new SqlCommand(query, conn))
        {

            cmd.Parameters.AddWithValue("@CarID", car.CarID);
            cmd.Parameters.AddWithValue("@Model", car.Model);
            cmd.Parameters.AddWithValue("@Brand", car.Brand);
            cmd.Parameters.AddWithValue("@Colour", car.Colour);
            cmd.Parameters.AddWithValue("@Year", car.Year);
            cmd.Parameters.AddWithValue("@BodyType", car.BodyType);
            cmd.Parameters.AddWithValue("@Price", car.Price);

            conn.Open();
            int rowsAffected = cmd.ExecuteNonQuery();
            return rowsAffected > 0;
        }
    }
}
```

### **Delete car details**

```
public bool DeleteCar(int carId)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
```

```

        string query = "DELETE FROM [cars] WHERE car_id = @carId";
        using (SqlCommand cmd = new SqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("@carId", carId);

            conn.Open();
            int rowsAffected = cmd.ExecuteNonQuery();
            return rowsAffected > 0;
        }
    }
}

```

### **Car Search functionality**

```

public DataTable SearchCars(string searchQuery)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = @"SELECT car_id, model, brand, colour, year, body_type,
price FROM [cars]
        WHERE model LIKE @searchQuery
        OR brand LIKE @searchQuery
        OR colour LIKE @searchQuery
        OR year LIKE @searchQuery
        OR body_type LIKE @searchQuery
        OR price LIKE @searchQuery";

        SqlCommand cmd = new SqlCommand(query, conn);
        cmd.Parameters.AddWithValue("@searchQuery", "%" + searchQuery + "%");

        SqlDataAdapter adapter = new SqlDataAdapter(cmd);
        DataTable dataTable = new DataTable();
        adapter.Fill(dataTable);
        return dataTable;
    }
}

```

## **6.4.3 Part Management functions**

### **Add new parts to the database**

```

public bool RegisterParts(Parts part)

```

```

{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = "INSERT INTO [parts]
(part_id,brand,category,colour,size,warranty,price) VALUES (@part_id,@brand,
@category, @colour, @size, @warranty,@price)";
        SqlCommand cmd = new SqlCommand(query, conn);
        cmd.Parameters.AddWithValue("@part_id", part.PartID);
        cmd.Parameters.AddWithValue("@brand", part.Brand);
        cmd.Parameters.AddWithValue("@category", part.Category);
        cmd.Parameters.AddWithValue("@colour", part.Colour);
        cmd.Parameters.AddWithValue("@size", part.Size);
        cmd.Parameters.AddWithValue("@warranty", part.Warranty);
        cmd.Parameters.AddWithValue("@price", part.Price);

        conn.Open();
        int result = cmd.ExecuteNonQuery();
        return result > 0;
    }
}

```

### **Function to check if part exists**

```

public bool PartExists(string partID)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = "SELECT COUNT(*) FROM [parts] WHERE part_id = @partID";
        using (SqlCommand cmd = new SqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("@partID", partID);

            conn.Open();
            int count = (int)cmd.ExecuteScalar();
            return count > 0;
        }
    }
}

```

### **Update part details**

```
public bool UpdatePart(Parts part)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = @"UPDATE [parts]
            SET brand = @Brand,
                category = @Category,
                colour = @Colour,
                size = @Size,
                warranty = @Warranty,
                price = @Price
            WHERE part_id = @PartID";

        using (SqlCommand cmd = new SqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("@PartID", part.PartID);
            cmd.Parameters.AddWithValue("@Brand", part.Brand);
            cmd.Parameters.AddWithValue("@Category", part.Category);
            cmd.Parameters.AddWithValue("@Colour", part.Colour);
            cmd.Parameters.AddWithValue("@Size", part.Size);
            cmd.Parameters.AddWithValue("@Warranty", part.Warranty);
            cmd.Parameters.AddWithValue("@Price", part.Price);

            conn.Open();
            int rowsAffected = cmd.ExecuteNonQuery();
            return rowsAffected > 0;
        }
    }
}
```

### **Delete part details**

```
public bool DeletePart(int partId)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = "DELETE FROM [parts] WHERE part_id = @PartID";
        using (SqlCommand cmd = new SqlCommand(query, conn))
        {
```

```

        cmd.Parameters.AddWithValue("@PartID", partId);

        conn.Open();
        int rowsAffected = cmd.ExecuteNonQuery();
        return rowsAffected > 0;
    }
}

public DataTable SearchParts(string searchQuery)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = @"SELECT part_id, brand, category, colour, size,
warranty, price
                        FROM [parts]
                        WHERE brand LIKE @SearchQuery
                        OR category LIKE @SearchQuery
                        OR colour LIKE @SearchQuery
                        OR size LIKE @SearchQuery
                        OR warranty LIKE @SearchQuery";

        using (SqlCommand cmd = new SqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("@SearchQuery", "%" + searchQuery +
"%");

            SqlDataAdapter adapter = new SqlDataAdapter(cmd);
            DataTable dataTable = new DataTable();
            adapter.Fill(dataTable);
            return dataTable;
        }
    }
}

```

## 6.4.4 Car order management functions

### Update car orders

```
public bool UpdateCarOrder(CarOrder carOrder)
```

```

{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = @"UPDATE CarOrders
                        SET car_id = @CarID,
                          status = @Status,
                          user_id = @UserID,
                          date = @Date
                        WHERE order_id = @OrderID";

        using (SqlCommand cmd = new SqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("@OrderID", carOrder.OrderID);
            cmd.Parameters.AddWithValue("@CarID", carOrder.CarID);
            cmd.Parameters.AddWithValue("@Status", carOrder.Status);
            cmd.Parameters.AddWithValue("@UserID", carOrder.UserID);
            cmd.Parameters.AddWithValue("@Date", carOrder.OrderDate);

            conn.Open();
            int rowsAffected = cmd.ExecuteNonQuery();
            return rowsAffected > 0;
        }
    }
}

```

### **Delete car orders**

```

public bool DeleteCarOrder(int orderId)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = "DELETE FROM CarOrders WHERE order_id = @OrderID";

        using (SqlCommand cmd = new SqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("@OrderID", orderId);

            conn.Open();
            int rowsAffected = cmd.ExecuteNonQuery();
            return rowsAffected > 0;
        }
    }
}

```

```

    }
}
}

```

### **Search car orders**

```

public DataTable SearchCarOrders(string searchQuery)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = @"SELECT c.brand, c.colour, c.body_type AS body_model,
o.order_id, o.date, o.user_id
                        FROM CarOrders o
                        JOIN Cars c ON o.car_id = c.car_id
                        WHERE o.order_id LIKE @searchQuery
                        OR o.date LIKE @searchQuery
                        OR o.user_id LIKE @searchQuery";

        using (SqlCommand cmd = new SqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("@searchQuery", "%" + searchQuery +
"%");

            SqlDataAdapter adapter = new SqlDataAdapter(cmd);
            DataTable dataTable = new DataTable();
            adapter.Fill(dataTable);
            return dataTable;
        }
    }
}

```

## **6.4.5 Part order management functions**

### **Update Part orders**

```

public bool UpdatePartOrder(PartOrder partOrder)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {

```



```

        string query = @"UPDATE part_order
                        SET part_id = @PartID,
                          status = @Status,
                          user_id = @UserID,
                          date = @Date
                        WHERE order_id = @OrderID";

        using (SqlCommand cmd = new SqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("@OrderID", partOrder.OrderID);
            cmd.Parameters.AddWithValue("@PartID", partOrder.PartID);
            cmd.Parameters.AddWithValue("@Status", partOrder.Status);
            cmd.Parameters.AddWithValue("@UserID", partOrder.UserID);
            cmd.Parameters.AddWithValue("@Date", partOrder.OrderDate);

            conn.Open();
            int rowsAffected = cmd.ExecuteNonQuery();
            return rowsAffected > 0;
        }
    }
}

```

### **Delete parts order**

```

public bool DeletePartOrder(int orderId)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = "DELETE FROM part_order WHERE order_id = @OrderID ";

        using (SqlCommand cmd = new SqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("@OrderID", orderId);

            conn.Open();
            int rowsAffected = cmd.ExecuteNonQuery();
            return rowsAffected > 0;
        }
    }
}

```

### **Search parts order**

```
public DataTable SearchPartOrders(string searchQuery)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = @"SELECT p.brand, p.colour, p.category, p.warranty,
p.size, o.order_id, o.date, o.user_id
FROM part_order o
JOIN Parts p ON o.part_id = p.part_id
WHERE o.order_id LIKE @searchQuery
OR o.date LIKE @searchQuery
OR o.user_id LIKE @searchQuery";

        using (SqlCommand cmd = new SqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("@searchQuery", "%" + searchQuery +
"%");

            SqlDataAdapter adapter = new SqlDataAdapter(cmd);
            DataTable dataTable = new DataTable();
            adapter.Fill(dataTable);
            return dataTable;
        }
    }
}
```

## **6.5 Customer dashboard**

### **6.5.1 Placing orders**

#### **Register car orders**

```
public bool RegisterCarOrder(CarOrder carOrder)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = @"INSERT INTO car_order (order_id, car_id, status,
user_id, date)
VALUES (@OrderID, @CarID, @Status, @UserID, @Date)";
```

```

        using (SqlCommand cmd = new SqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("@OrderID", carOrder.OrderID);
            cmd.Parameters.AddWithValue("@CarID", carOrder.CarID);
            cmd.Parameters.AddWithValue("@Status", carOrder.Status);
            cmd.Parameters.AddWithValue("@UserID", carOrder.UserID);
            cmd.Parameters.AddWithValue("@Date", carOrder.OrderDate);

            conn.Open();
            int result = cmd.ExecuteNonQuery();
            return result > 0;
        }
    }
}

```

### **Register Part orders**

```

public bool RegisterPartOrder(PartOrder partOrder)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = @"INSERT INTO part_order
order_id,part_id,status,user_id,date)
        VALUES (@OrderID, @PartID, @Status, @UserID, @Date)";

        using (SqlCommand cmd = new SqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("@OrderID", partOrder.OrderID);
            cmd.Parameters.AddWithValue("@PartID", partOrder.PartID);
            cmd.Parameters.AddWithValue("@Status", partOrder.Status);
            cmd.Parameters.AddWithValue("@UserID", partOrder.UserID);
            cmd.Parameters.AddWithValue("@Date", partOrder.OrderDate);

            conn.Open();
            int result = cmd.ExecuteNonQuery();
            return result > 0;
        }
    }
}

```

## 6.5.2 Tracking order status

### **Track cars order**

```
public DataTable TrackCarOrders(string searchQuery)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = @"SELECT c.brand, c.colour, c.body_type AS body_model,
o.order_id, o.date, o.user_id, o.status
FROM CarOrders o
JOIN Cars c ON o.car_id = c.car_id
WHERE o.order_id LIKE @searchQuery
OR o.user_id LIKE @searchQuery";

        using (SqlCommand cmd = new SqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("@searchQuery", "%" + searchQuery + "%");

            SqlDataAdapter adapter = new SqlDataAdapter(cmd);
            DataTable dataTable = new DataTable();
            adapter.Fill(dataTable);
            return dataTable;
        }
    }
}
```

### **Track Parts Order**

```
public DataTable TrackPartOrders(string searchQuery)
{
    using (SqlConnection conn = new SqlConnection(_connection))
    {
        string query = @"SELECT p.brand, p.colour, p.category, o.order_id,
o.date, o.user_id, o.status
FROM part_order o
JOIN Parts p ON o.part_id = p.part_id";
```

```

        WHERE o.order_id LIKE @searchQuery
        OR o.user_id LIKE @searchQuery";

using (SqlCommand cmd = new SqlCommand(query, conn))
{
    cmd.Parameters.AddWithValue("@searchQuery", "%" + searchQuery +
"%");

    SqlDataAdapter adapter = new SqlDataAdapter(cmd);
    DataTable dataTable = new DataTable();
    adapter.Fill(dataTable);
    return dataTable;
}
}
}

```

## 7. Instructions to run the application

### 7.1 Instructions for Admin

The Admin portal is designed to manage car details, car parts, customer information, and orders. Admins can also generate and view sales and order reports.

#### Login

- Enter your admin username and password on the login screen.
- Click the "Login" button.

#### Manage car details

- Navigate to the "Car Management" section.
- To add a new car, fill all fields and click "Add".
- To edit a car, type the car id, make necessary changes, and click "Update."
- To delete a car, type the car id and click "Delete."

### **Manage parts details**

- Go to the "Parts Management" section.
- To add a new part, fill all fields and click "Add".
- To edit a part, type the part id, update the details, and click "Update."
- To delete a part, type the part id and click "Delete."

### **Manage customer details**

- Open the "Customer Management" section.
- To add a new customer, fill all fields and click "Add".
- To edit customer details, type the customer id, make changes, and click "Update."
- To delete a customer, type the customer id and click "Delete."

### **Manage orders**

- Go to Car or Parts management section.
- Type the Order Id or customer Id or date and click "Search" to view orders.
- To edit order details ,type the order Id , make changes and click "Update."
- To delete a order, type the order id and click "Delete."

### **Generate reports**

- Navigate to the "Reports" section.
- Choose either "Sales Report" or "Order Report."
- Click the "Generate Report" button to display the report in the DataGridView.

### **Logout**

- Click the "Logout" button in the top-right corner of the application.
- Confirm the logout action if prompted.

## **7.2 Instructions for Customer**

### **Registering account**

- Open the ABC Car Traders software.
- On the login screen, click "Register."

- Fill in the registration form with your details.
- Click "Submit" to create your account.

### **Login**

- Enter your customer username and password on the login screen.
- Click the "Login" button.

### **Search for cars**

- Use the search bar to enter car details or browse the list.

### **Search for parts**

- Use the search bar to enter car details or browse the list.

### **Place orders**

- Fill in the required order details.
- Click "Place Order" to place your order.

### **Track order status**

- Type your order id or customer id and select parts or cars.
- Click “view status” to view order status.

### **Logout**

- Click the "Logout" button in the top-right corner of the application.
- Confirm the logout action if prompted.

## **8. Self-reflection on using C# and Visual Studio**

Working with C# and Visual Studio for this project has been an enriching experience. As I undertook the task of developing a comprehensive software solution for ABC Car Traders, I

encountered various challenges and learning opportunities that contributed significantly to my growth as a developer.

## **8.1 Experience with using C#**

- C# is a robust language that strongly emphasizes OOP principles. Working with classes, inheritance, and polymorphism allows the application to be structured in a modular and maintainable way. The language enforces a clean separation of concerns, making the code easier to understand and extend.
- C#'s exception handling mechanisms are straightforward and effective. Implementing try-catch blocks helps manage unexpected situations gracefully, ensuring that the application remains stable even in the face of errors.
- Understanding and implementing events and delegates can be challenging. However, once mastered, they greatly improve the handling of user interactions and asynchronous operations within the application.

## **8.2 Experience with using Visual Studio**

- Visual Studio is a feature-rich IDE that provides an excellent environment for developing applications. Its extensive toolset, including IntelliSense, code refactoring tools, and the debugger, significantly boosts productivity. The real-time error checking and code suggestions are particularly useful for maintaining high-quality code.
- The debugging tools in Visual Studio are among the best available. The ability to step through code, inspect variables, and set breakpoints allows for quick identification and resolution of issues. Watching variable values change in real-time offers deep insights into how the code functions.
- Visual Studio's integration with Git makes version control seamless. Changes can be committed, branches created, and code pushed to remote repositories without leaving the IDE. This integration ensures that a well-documented history of changes is maintained and facilitates effective collaboration.



The overall experience of using C# and Visual Studio was challenging due to the learning curve and handling data storage and retrieval. By implementing robust error handling and validation routines, I minimized potential issues related to data handling.

## **9. Conclusion**

To sum up, building the software application for ABC Car Traders required learning C# and Visual Studio, which was a rewarding and educational experience. Strong features in Visual Studio, such IntelliSense and others, allowed for an orderly and user-friendly program.

The integrated debugger and the robust object-oriented programming support in C# helped this author build this application successfully.

Despite its early challenges, the learning curve provided useful information regarding effective User Interface Design, Data Management, and Coding Techniques. Apart from refining the author's technical proficiency, this endeavor let the author realize the importance of thorough testing, efficient error management and user-centered design, which are essential for generating trustworthy and expandable solutions.