CAR INSURANCE DATA INTERFACE

Technical Report

Contents

Desktop Application Development Overview

The Car Insurance Data Interface, also known as CIDI, is a user-friendly GUI application linked to the MySQL server of a car insurance company. CIDI acts as the interface between users (company employees) and the MySQL database with customer details. As the employees may need to perform routine tasks of adding, deleting, and updating customer details, this application provides a GUI to do that whilst taking care of the MySQL commands in the background to make the necessary changes to the customer database. This allows employees with no prior knowledge of SQL to perform their duties without any issues.

Programming Languages and Frameworks

CIDI is developed in Windows Forms using the programming language C#, as part of the .NET framework. There were considerations to use Windows Presentation Framework but ultimately development went forward with WinForms as the simplicity of using forms and user controls match the goal of creating a simple and familiar UI for users to use without prior training. The WinForms component of the application is a client to a Windows Communication Framework service. The data input by the user will be sent to the service, and all MySQL connections take place at the service level for a more structured and scalable approach. There is a login form for user authentication before access to the main application is granted.

Application Logic and Functionality

The user is presented with a welcome screen where it clearly explains the application functionalities. The 'All Customers' tab shows all customers in order of the customer's ID (primary key) in the database, and the user is able to navigate through the list using the left and right buttons. The user can update each customer's details or they can also delete a customer if they wish using the buttons provided at the bottom. Similar functions are available in the 'Search Customers' tab where users can search for any customers matching the data provided. The 'Add Customer' tab is where users can add customers to the database.

Testing and Quality Assurance

Throughout development, this application went through rigorous testing and quality assurance. Visual Studio IDE helped ensure that all C# code syntax was correct and 'try catch' blocks were used with SQL operations so that errors could be caught while debugging. The client application cannot throw exceptions if there is a problem in the WCF service, so the catch blocks helped identify exceptions thrown in SQL coding while using stepping debugging as seen in Figure 1. Each test case of view, add, update and delete customers were tested and documented and results of testing corner cases were used to make sure the application throws errors if the user performs an illegal action (e.g. trying to add a customer with blank or incomplete details).

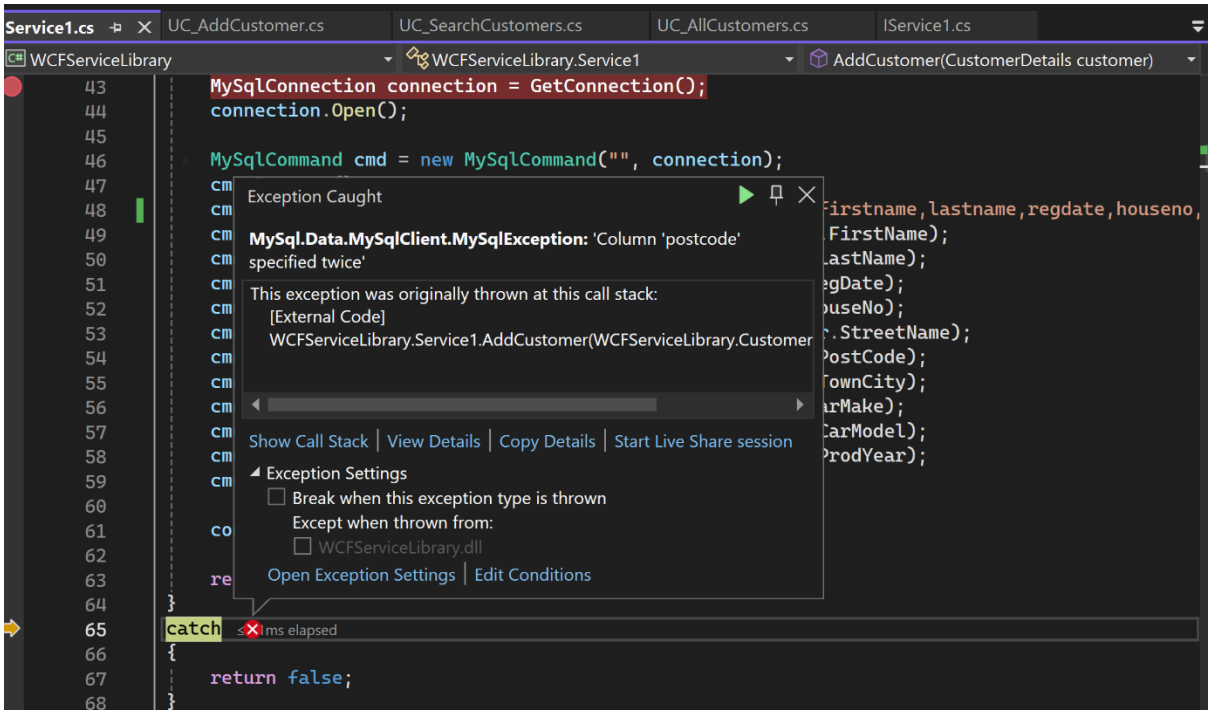

Figure 1

Deployment and Distribution

CIDI can be deployed by a car insurance company for use by staff with virtually no alterations to the code. Outside of testing, the application maintains a server connection to the database with login details for user authentication and the MySQL connection string will have the necessary credentials to connect to the remote server hosting the customer database. Due to the WCF service approach in this application, a company can alter the source code to allow for changes before distribution. This may include changes in database locations or to disable any features which can be done with relative ease by making changes to the service layer coding, preventing the client application from using those features.

Security and Data Protection

CIDI features user authentication as the user will be able to access and modify sensitive information. The login form is the first screen the user will see and will not be able to proceed unless they enter valid credentials that match those in the database. The MySQL server hosting the customer database is hosted by Azure. Azure enables security features such as encryption, firewall rules and requiring an SSL connection with an SSL certificate. Attempting to connect without SSL will be rejected as the server is configured to adhere to these strict security requirements in order to protect sensitive information. CIDI leverages these features in full so that customer details are secure from malicious attempts to view and modify as seen in Figure 2.

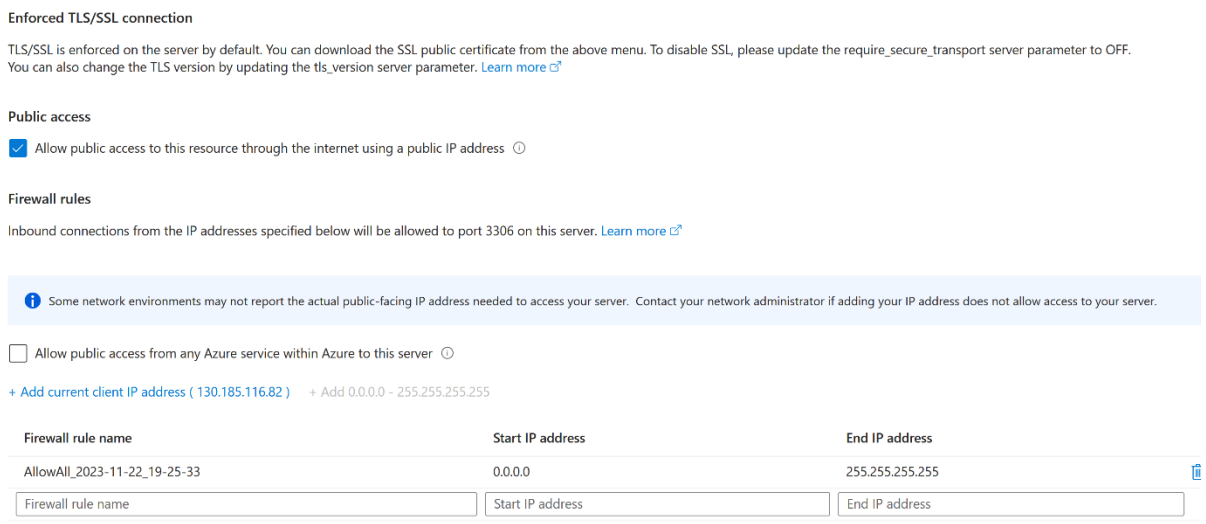

Figure 2

Companies can add their public IP addresses as a firewall rule to make sure that only authorised networks can access the data. At present, there is a firewall rule present allowing all IP addresses to connect to the server for testing purposes but this will be removed before deployment. Robust data validation is included to protect the data input by the user as shown in Figure 3 and reduces the chance of human error affecting data accuracy.

```
1 reference
private bool DataValidation()
{
    if (String.IsNullOrEmpty(FirstNameTextBox.Text) || (FirstNameTextBox.Text).Contains(" "))
        return false;
    else if (String.IsNullOrWhiteSpace(LastNameTextBox.Text) || (LastNameTextBox.Text).Contains(" "))
        return false;
    else if (this.RegDatePicker.Value > System.DateTime.Today)
        return false;
    else if (String.IsNullOrWhiteSpace(HouseNoTextBox.Text) || (HouseNoTextBox.Text).Contains(" "))
        return false;
    else if (StreetTextBox.Text.Any(char.IsDigit) || TownTextBox.Text.Any(char.IsDigit) || (TownTextBox.Text).Contains(" "))
        return false;
    else if (String.IsNullOrEmpty(PostcodeTextBox.Text))
        return false;
    else if (String.IsNullOrEmpty(CarMakeTextBox.Text))
        return false;
    else if (String.IsNullOrEmpty(CarModelTextBox.Text))
        return false;
    else if (YearTextBox.Text.All(char.IsDigit) == false || (YearTextBox.Text).Contains(" ") || (YearTextBox.Text).Length > 4
        return false;
    else
        return true;
}
```

Figure 3

Case Studies and Examples

The following case studies illustrate the application's core functionalities.

User adds blank customer:

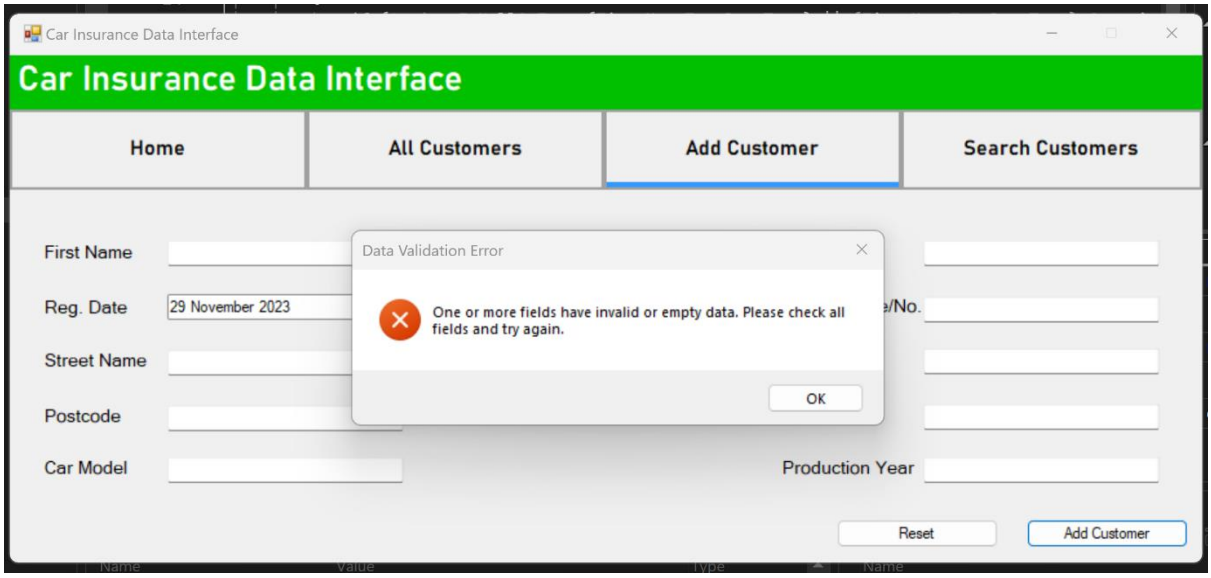

Figure 4

User adds customer and fails data validation:

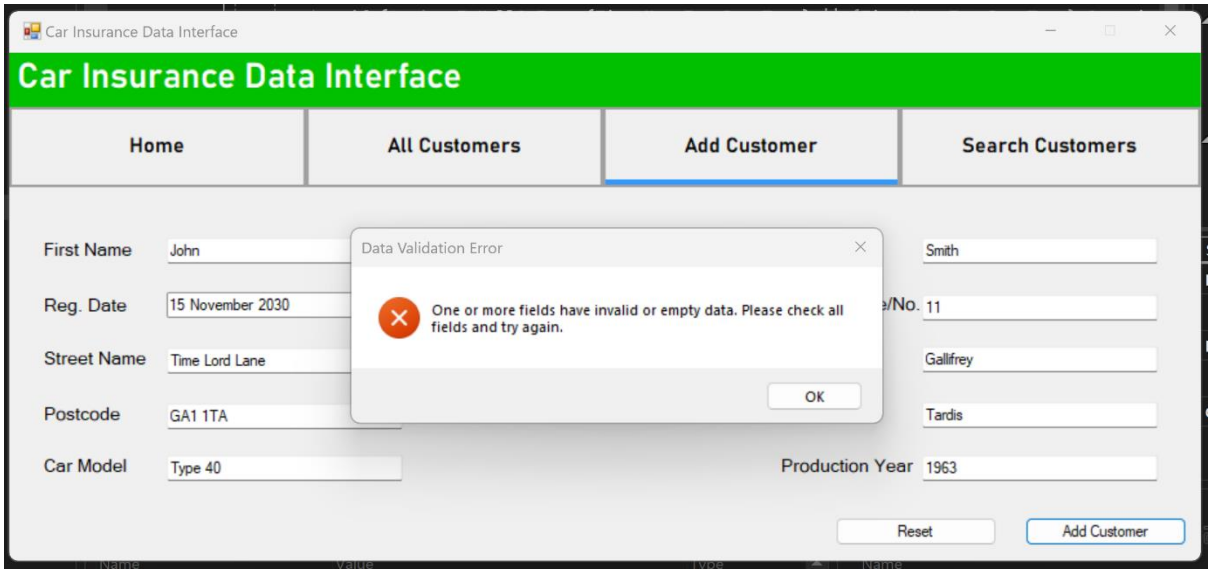

Figure 5

User adds customer successfully:

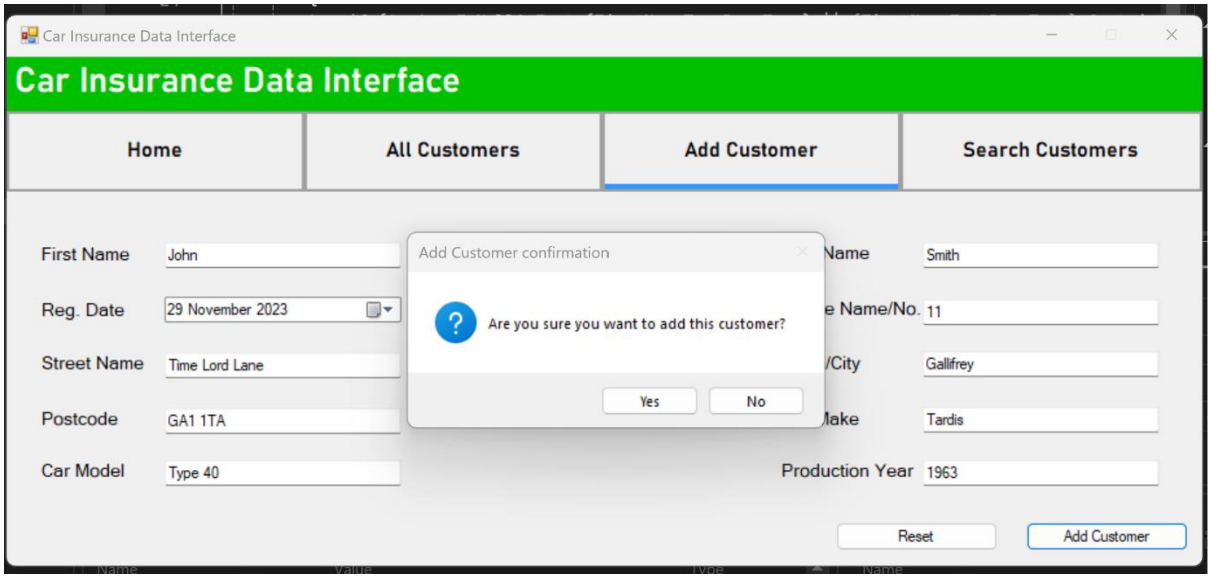

Figure 6

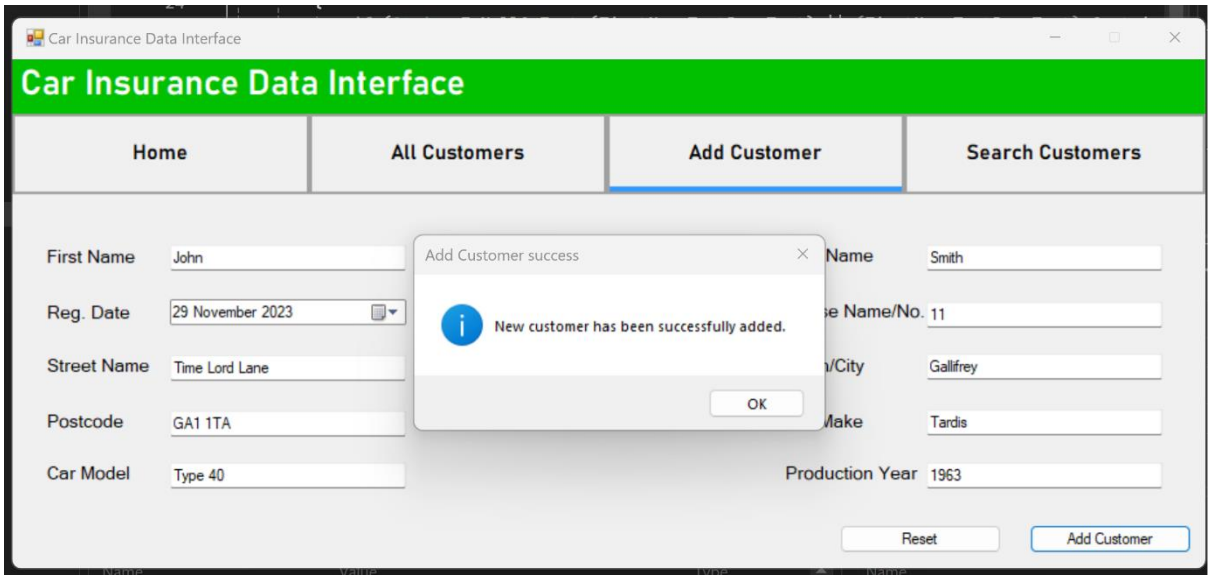

Figure 7

User tries to search for a blank customer:

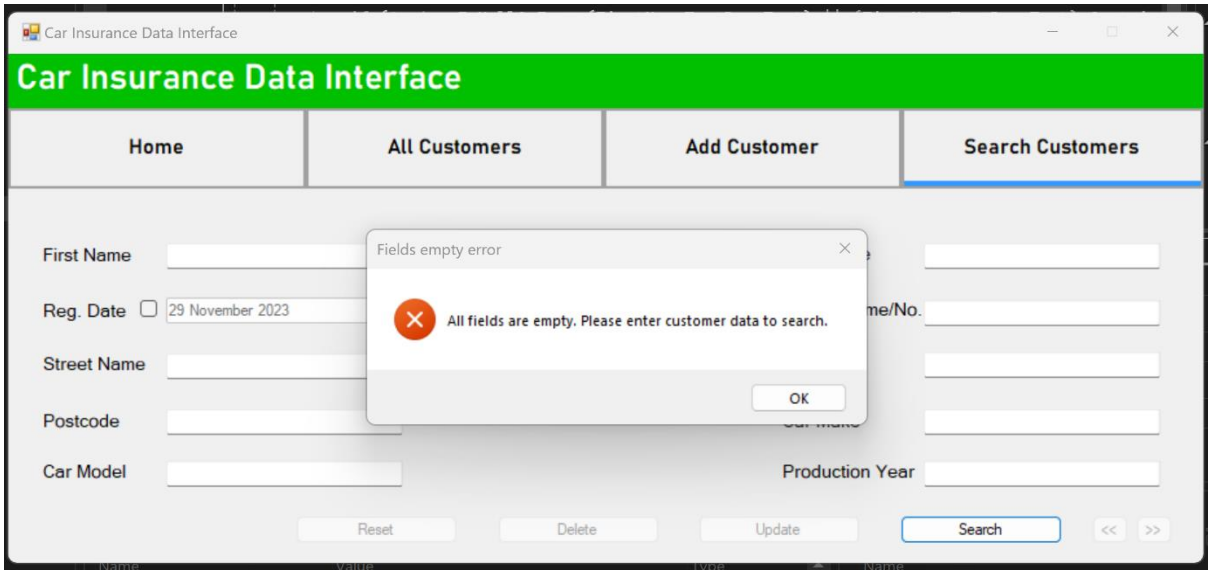

Figure 8

User updates an existing customer:

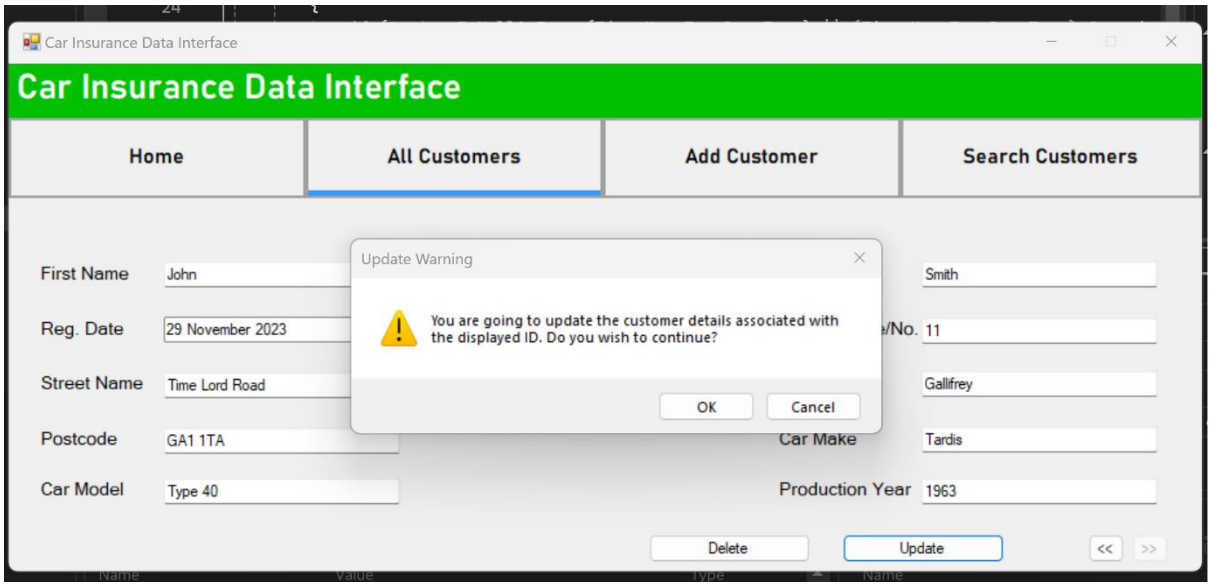

Figure 9

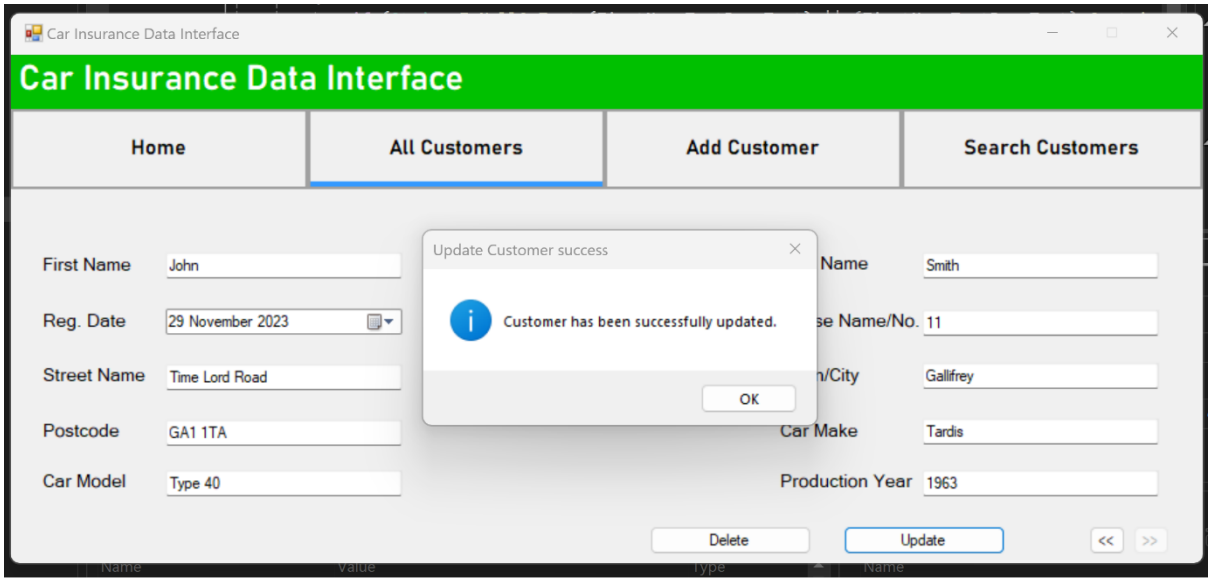

Figure 10

User searches for a customer with multiple data fields, then deletes customer:

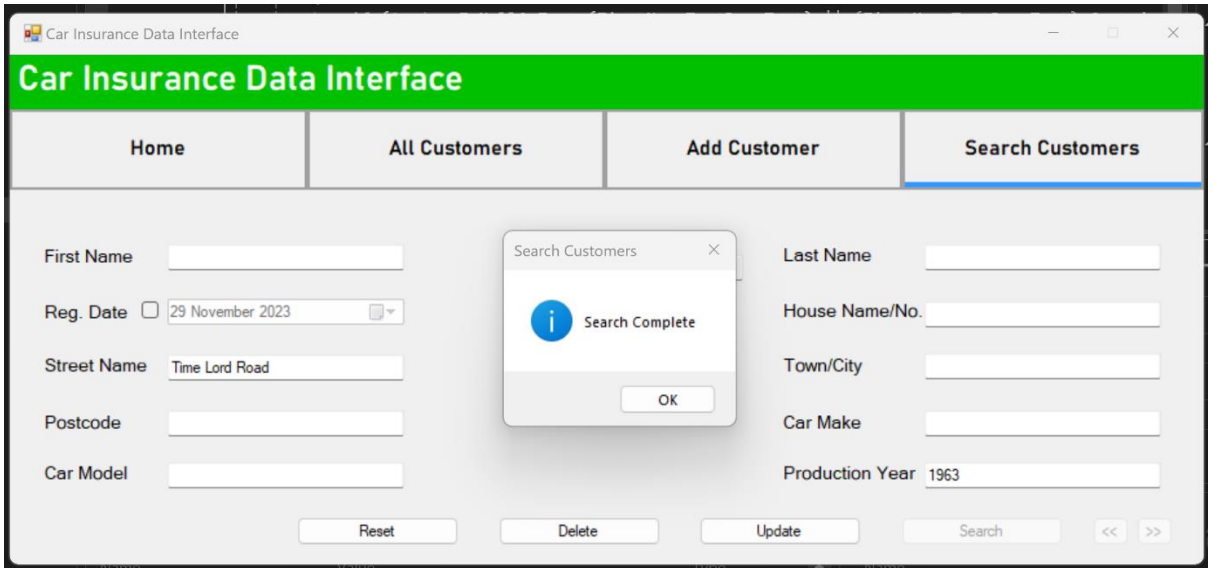

Figure 11

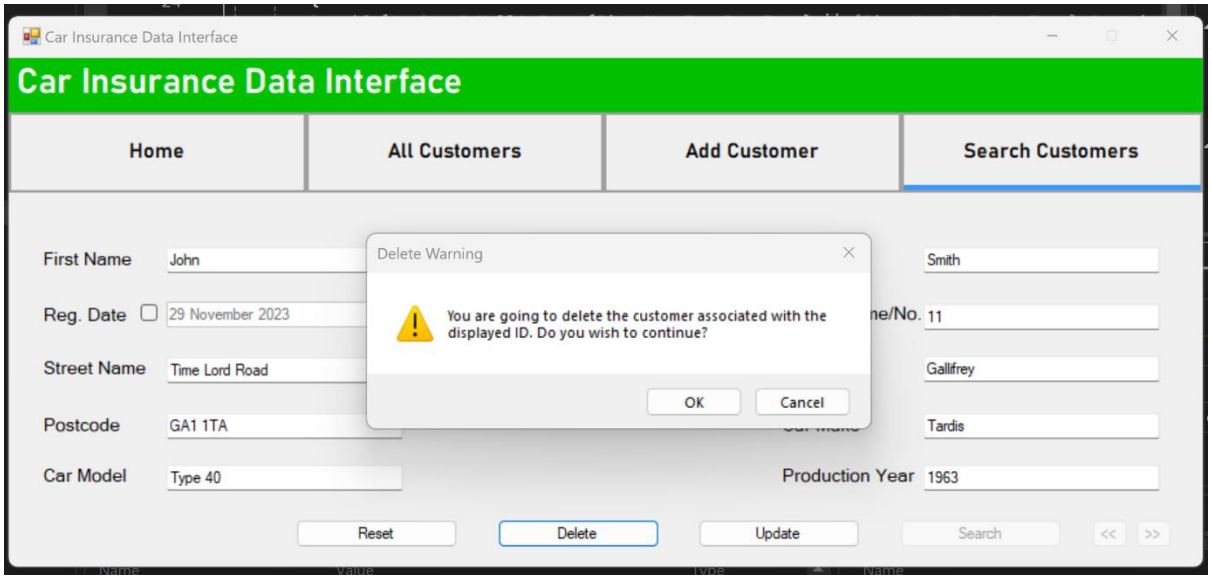

Figure 12

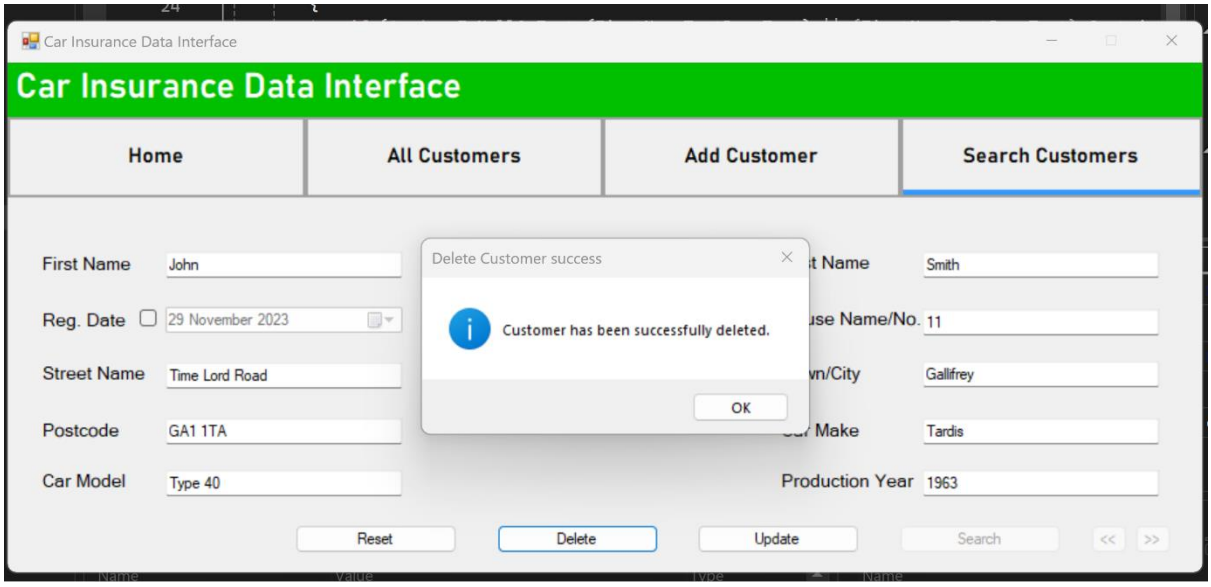

Figure 13

Future Trends and Challenges

Future challenges that may arise include having to update and redeploy the application if the database locations change and require new connection strings. There is also the possibility for the discontinuation of Azure's MySQL server hosting service. This may require the company using CIDI to look for alternative providers with similar levels of on-demand availability and security. Moreover, Microsoft may decide to fully deprecate the WCF framework and pull support for the .NET framework which may mean that CIDI will start to become outdated. In this case, the application will need to be reworked for the latest .NET versions.

Conclusion

To conclude, CIDI was designed to be an alternative interface to interact with MySQL databases. The aim was to create a user-friendly application that has no SQL coding or properties visible to the user, but still allows them to execute SQL queries seamlessly. This application achieves this goal and looking beyond the scope of this project, can even be redesigned and applied to other industries and commercial purposes as long as it involves SQL.