INTERVIEW PROJECT REPORT

MEHODOLOGY

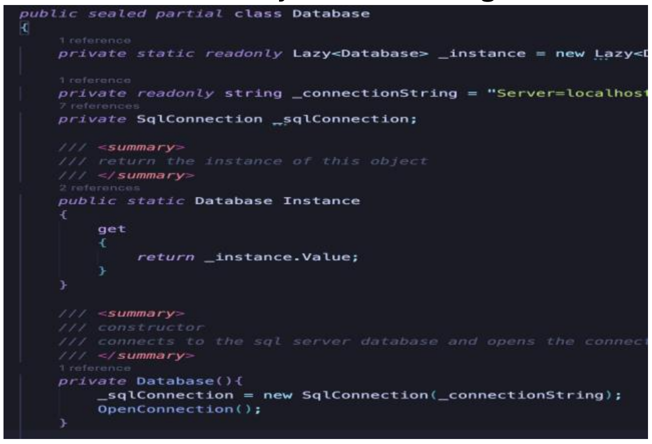
This project follows an AGILE methodology and was divided into three sprints. The first sprint involved writing the T-SQL scripts for the database, the second sprint focused on writing the C# code, and the final sprint was dedicated to designing the front end using Blazor. By following this approach, I was able to complete the project on time and had extra time to implement additional functionalities, such as adding a customer through the web app.

T-SQL SCRIPT

The T-SQL script for this project, called "interview.sql," can be found in the Scripts folder. It contains scripts to create a database, define the tables and their appropriate data types, and create stored procedures to insert data into the tables. It also includes stored procedures to select a customer and select products.

DESIGN PATTERNS

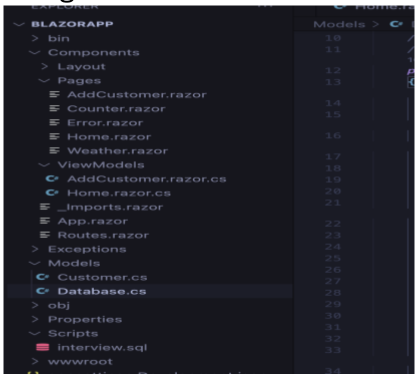
Singleton: Singleton design pattern was implemented in this project for the database class. Singleton was used to create a single instance of the database class which could then be accessed by the other classes, rather than creating a new instance of the database class every time it is being used. See Figure 1 below.



```
public sealed partial class Database
{
    1 reference
    private static readonly Lazy<Database> _instance = new Lazy<D

    1 reference
    private readonly string _connectionString = "Server=localhost
    7 references
    private SqlConnection _sqlConnection;

    /// <summary>
    /// return the instance of this object
    /// </summary>
    2 references
    public static Database Instance
    {
        get
        {
            return _instance.Value;
        }
    }

    /// <summary>
    /// constructor
    /// connects to the sql server database and opens the connect
    /// </summary>
    1 reference
    private Database(){
        _sqlConnection = new SqlConnection(_connectionString);
        OpenConnection();
    }
```

Figure 1.

Model-View-ViewModel (MVVM): The MVVM design pattern was used as the project structure to enhance readability and organization. The MVVM arrangement is illustrated in Figure 2 below.

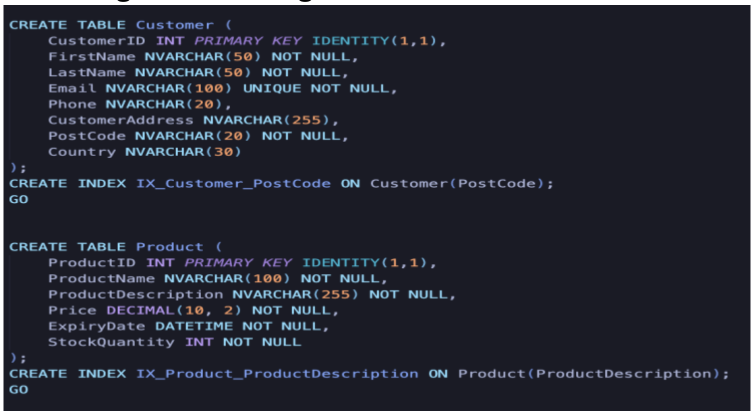


```
BLAZORAPP
  > bin
  ∨ Components
    > Layout
    ∨ Pages
        AddCustomer.razor
        Counter.razor
        Error.razor
        Home.razor
        Weather.razor
    ∨ ViewModels
        AddCustomer.razor.cs
        Home.razor.cs
      _Imports.razor
      App.razor
      Routes.razor
  > Exceptions
  ∨ Models
      Customer.cs
      Database.cs
  > obj
  > Properties
  ∨ Scripts
      interview.sql
  > wwwroot
```

Figure 2

INDEXING

Indexes were created on the postcode in the Customer table and on the product description in the Product table. This optimization improves speed and efficiency when retrieving data. See Figure 3 below.



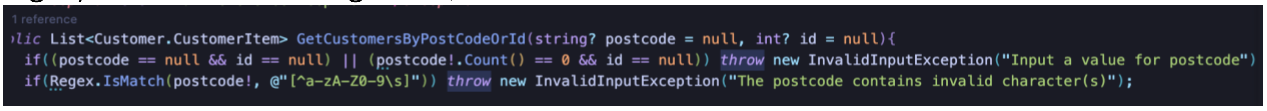
```sql
CREATE TABLE Customer (
    CustomerID INT PRIMARY KEY IDENTITY(1,1),
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    Email NVARCHAR(100) UNIQUE NOT NULL,
    Phone NVARCHAR(20),
    CustomerAddress NVARCHAR(255),
    PostCode NVARCHAR(20) NOT NULL,
    Country NVARCHAR(30)
);
CREATE INDEX IX_Customer_PostCode ON Customer(PostCode);
GO

CREATE TABLE Product (
    ProductID INT PRIMARY KEY IDENTITY(1,1),
    ProductName NVARCHAR(100) NOT NULL,
    ProductDescription NVARCHAR(255) NOT NULL,
    Price DECIMAL(10, 2) NOT NULL,
    ExpiryDate DATETIME NOT NULL,
    StockQuantity INT NOT NULL
);
CREATE INDEX IX_Product_ProductDescription ON Product(ProductDescription);
GO
```

Figure 3

EXCEPTION HANDLING

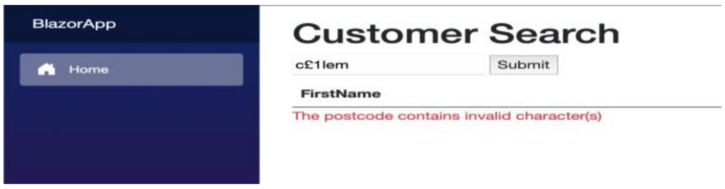
Custom exception classes were created in the Exceptions folder to handle exceptions in this project.

Invalid Input Exception: This exception is thrown when the user enters invalid input. For example, inputting special characters (like £@$) when searching for a user by postcode or entering an incorrectly formatted phone number (less or more than 11 digits). This is shown in Figures 4, 5 and 8 below.



```csharp
1 reference
lic List<Customer.CustomerItem> GetCustomersByPostCodeOrId(string? postcode = null, int? id = null){
  if((postcode == null && id == null) || (postcode!.Count() == 0 && id == null)) throw new InvalidInputException("Input a value for postcode")
  if(Regex.IsMatch(postcode!, @"[^a-zA-Z0-9\s]")) throw new InvalidInputException("The postcode contains invalid character(s)");
```

Figure 4



Customer Search

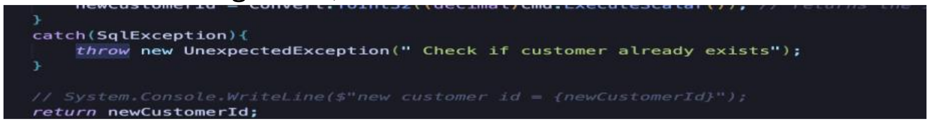
c£1lem Submit

FirstName

The postcode contains invalid character(s)

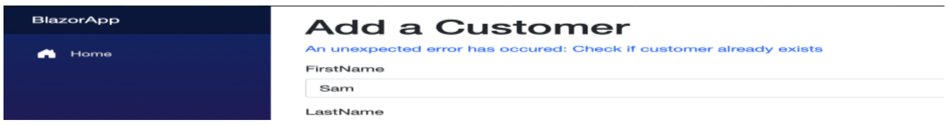
Figure 5

Unexpected Exception: This exception is thrown when there is an issue performing SQL operations. An example is attempting to add a duplicate user to the database. This can be seen in Figures 6, 7 and 8 below.



```csharp
}
catch(SqlException){
    throw new UnexpectedException(" Check if customer already exists");
}

// System.Console.WriteLine($"new customer id = {newCustomerId}");
return newCustomerId;
```

Figure 6



Add a Customer

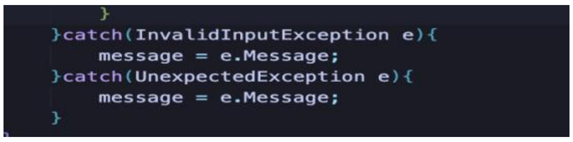
An unexpected error has occured: Check if customer already exists

FirstName

Sam

LastName

Figure 7

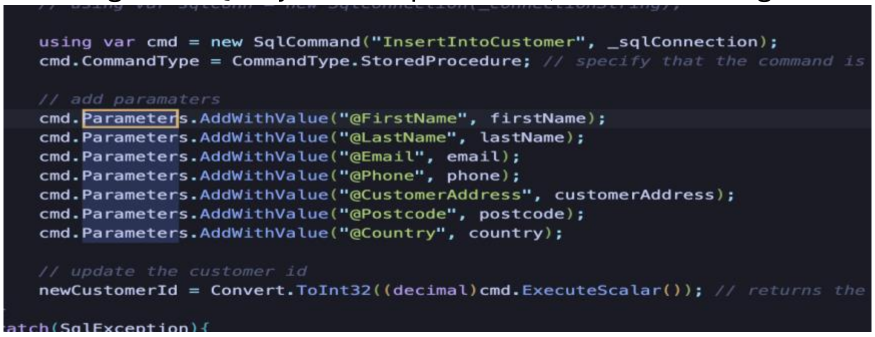


```
    }
}catch(InvalidInputException e){
    message = e.Message;
}catch(UnexpectedException e){
    message = e.Message;
}
```

Figure 8

SQL INJECTION PREVENTION

Assigning parameters when executing the procedure binds the parameters, thereby ensuring that SQL injection is prevented, as shown in Figure 9 below.



```
    using var cmd = new SqlCommand("InsertIntoCustomer", _sqlConnection);
    cmd.CommandType = CommandType.StoredProcedure; // specify that the command is

    // add paramaters
    cmd.Parameters.AddWithValue("@FirstName", firstName);
    cmd.Parameters.AddWithValue("@LastName", lastName);
    cmd.Parameters.AddWithValue("@Email", email);
    cmd.Parameters.AddWithValue("@Phone", phone);
    cmd.Parameters.AddWithValue("@CustomerAddress", customerAddress);
    cmd.Parameters.AddWithValue("@Postcode", postcode);
    cmd.Parameters.AddWithValue("@Country", country);

    // update the customer id
    newCustomerId = Convert.ToInt32((decimal)cmd.ExecuteScalar()); // returns the

atch(SqlException){
```

Figure 9

ADDITIONAL FUNCTIONALITY IMPLEMENTED

After completing the core functionalities of this project, I used the extra time available to implement an additional feature that allows a customer to be added to the database through the web app, as illustrated in Figures 10 and 11 below.

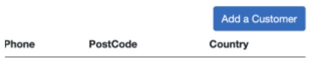


Figure 10

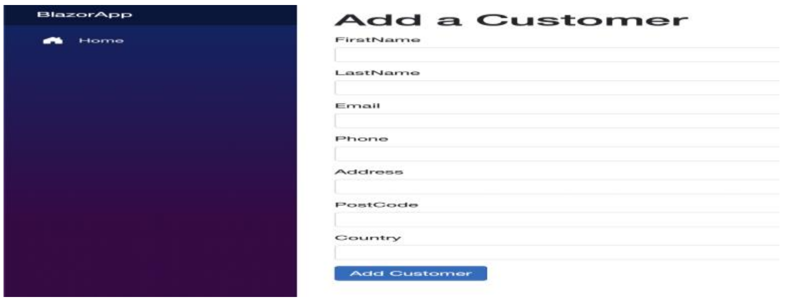


Figure 11

CHALLENGES FACED

A major challenge faced during this project was setting up an SQL server to test the T-SQL script. T-SQL is used by Microsoft SQL Server and is primarily designed for Windows computers. I use a MacBook and do not have access to a Windows computer, so I had to research on alternative methods to run the SQL server. I learned about Docker and Azure Data Studio, and by using these tools, I was able to get the SQL server running and connect my project to the database.
Another challenge was trying to use Visual Studio as the IDE for the project because Visual Studio was recently discontinued on MacBooks. To solve this problem, I switched to using Visual Studio Code.