Suppressor Application Technical Report

Aiden Bond - Whistl Fulfilment South West
Post-Development Version 3 - 12/02/2024

Contents

1.0.0 - Introduction

The Suppressor Application is intended to enhance the efficiency of recording and logging mailing suppressions and no-mails received by the Contact Centre. This application enables users to input data, creating a comprehensive record for subsequent reporting and distribution to relevant parties through the SSRS automated reporting service.

The application is designed to work in conjunction with Elucid, the ecommerce platform software utilised by Whistl Fulfilment South West. Elucid facilitates the taking, management, and fulfilment of customer orders.

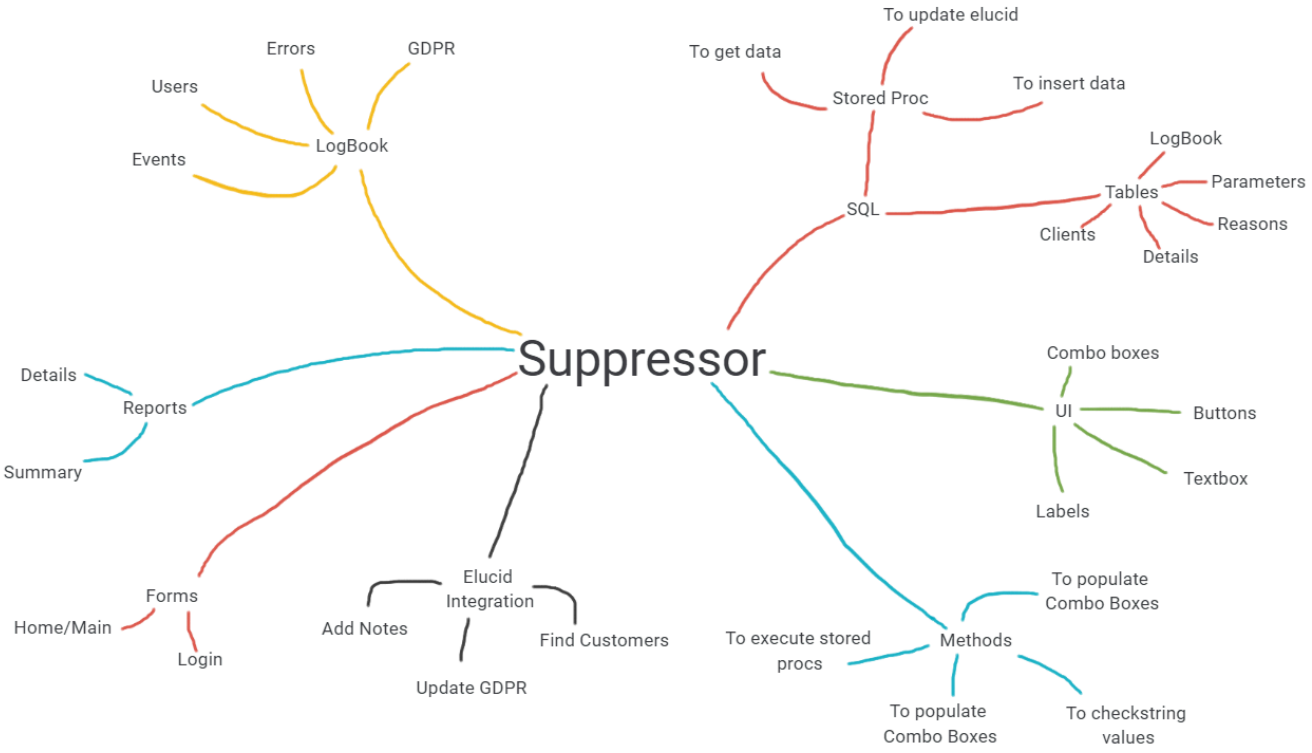
To ensure comprehensive tracking, the application incorporates an embedded logging process. This process captures instances such as application usage, errors, each saved entry, and every connection made to the Elucid system.

Throughout the project, an agile design methodology will be employed to optimise workflow and deliver optimal results for the client. The development of the application will be carried out using the C# programming language within the Visual Studio 2022 IDE. SQL data, tables, and stored procedures will be hosted on the SQL server SQL-SSRS

Project Lead	Aiden Bond
Project Start Date	11/01/2024
Project Deadline	29/01/2024
Quoted Time	15 hours
Actual Time	8 hours
Client Liaison	Nicki Neil

2.0.0 - Pre-Development Design & Planning

2.1.0 - Application & Usage



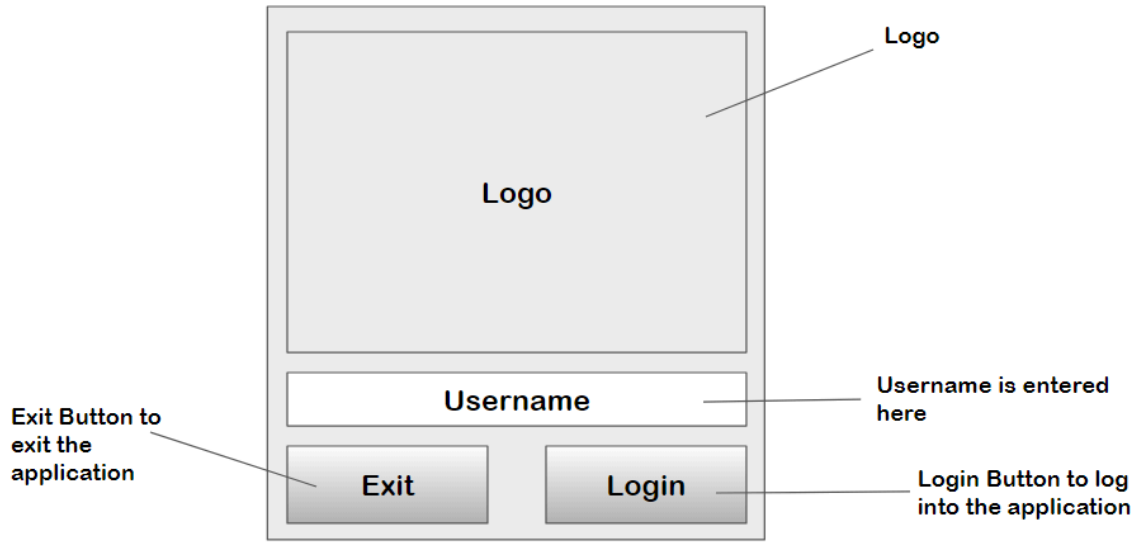
- A user needs to be able to input data based on a the information they receive as no-mails or mailing suppressions
- The user will have personal data such as a recipient's name, address, city and postcode which the recipient would like to be removed from mailing lists.
- A user needs to be able to choose which client they would like to record the mailing suppression against.
- The user needs to be able to specify the reason why the data is being recorded
- There will be 2 potential scenarios the user will face
  1. The recipient will already be a customer
  2. The recipient will not already be a customer

  If the recipient is already a customer, the user will be able to declare this on the form and this will cause a textbox to appear which will allow the user to input the customer number. The application will then link this to Elucid and fill in the relevant fields based on that customer number. Elucid is the customer management system used by the contact centre at Whistl Fulfilment South West

2.2.0 - Forms

2.2.1 - Login Form - Design

This is where the user will log into the application



- This form will allow users to log in using their Elucid username, no password is required.
- The username will be a free text field
- The “Exit” button will ask users if they are sure they want to exit the application. A message will appear saying “Are you sure you want to Exit?”
- The” Login” button will take the user to the Main Form
- The logo will be the whistl company logo or application logo

2.2.2 - Login Form - Error Handling & Dialog Boxes

In order to function as a robust application, error checking & handling will be in place. As well as dialog boxes to aid and enhance user experience.

Errors

- If the **Username** field is not completed when the user clicks the “Login” button, an error will occur
- If the **Username** field’s character limit is met, an error will occur
- If a special character is entered in the **Username** text box, an error will occur

Dialog Boxes

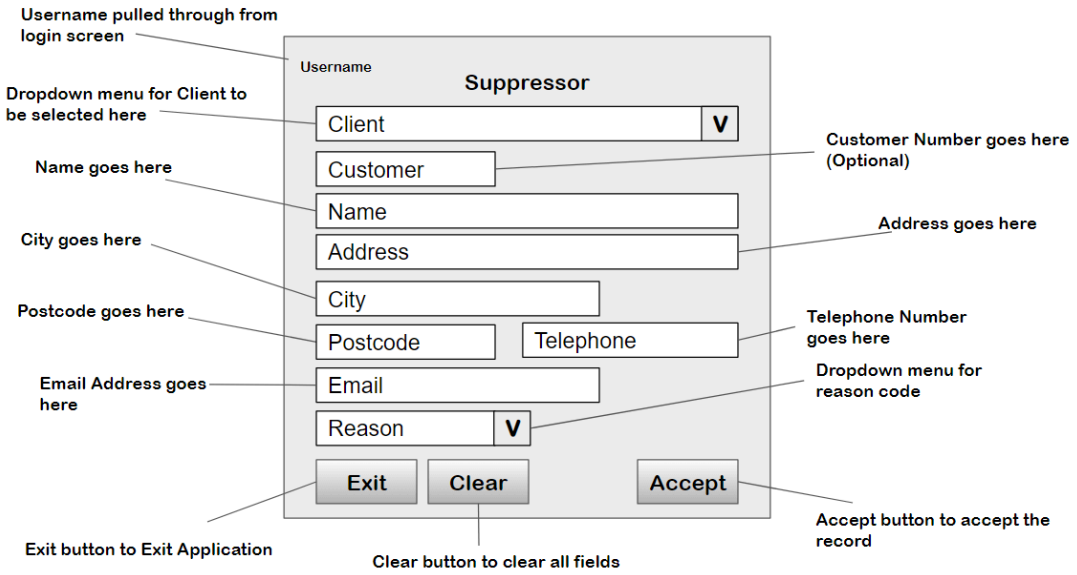
- If the user clicks the “Exit” button, a dialog box will appear asking the user if they are sure they want to leave the application.
- When the user clicks login, a message box will appear advising and asking the user that they understand the application is only to be used to record suppressions and cannot be used to record RTBF.

2.2.3 - Login Form - UI Elements

- **Username Field** - This will be a textbox name ***txbUsername***, it will have a character limit of 100 and will allow numbers and letters but will not allow any special characters
- **Login Button** - This will be a button named ***btnLogin*** and will have a “Click” method assigned to it.
- **Exit Button** - This will be a button named ***btnExit*** and will have a “Click” method assigned to it.
- **Logo** - This will be a picture box called ***pbLogo*** with the application logo in it.
- **Username Label** - this will be a prompting label called ***lblUsernametxb*** and will populated with the text “Username”

2.2.4 - Main Form - Design

The Main Form is where the user will be able to input the recipients data where it can then be recorded into the database



- This form is where the users will be able to record mailing suppressions based on recipients data.
- The user will be able to select which client this is for based on a dropdown menu.
- The username from the login form will feed through so as the user records data, the username will be recorded against each record as well as the Windows profile the user has logged into the server with.
- The “Customer” text box will allow users to input a customer number.
- The user will be able to clear all fields on this form by clicking the “Clear” button, this will also reset the Customer checkbox.
- When a customer number has been entered in the Customer Textbox the Name, Address, city and postcode textboxes will automatically be populated by data in Elucid. If no customer is found a message will appear saying the customer number does not match.
- This form has a dropdown menu where the user will be able to select the reason for the no-mail from a predefined list.
- The “Accept” button will allow users to save the recorded data they have inputted into the form. If any of the fields are blank, the application will give the user an error message.
- When a record is accepted, all fields will clear except for the “Client” field, this will only clear when the clear button is pressed
- The “Exit” button will ask the user if they want to exit, a message will appear saying “Are you sure you want to Exit?”

2.2.5 - Main Form - Error Handling & Dialog Boxes

Errors

- If the user clicks the **Accept** button and does not select an option from the **client** drop down box, an error will occur.
- If the user clicks the **Accept** button and the **Name** field is blank or empty an error will occur.
- If the user clicks the **Accept** button and either the **Email** field or the **Address** field is blank or empty an error will occur.
- If the user clicks the **Accept** button and does not select an option from the **Reason** drop down box, an error will occur.
- If the user enters a customer number into the **Customer** field that does not exist on the Elucid database, when the user attempts to leave the field, an error will occur.
- If the user enters an invalid email address into the **Email** field that does not exist on the Elucid database, when the user attempts to leave the field, an error will occur.
- If the character limit is met on the **Customer** field, an error will occur.
- If the character limit is met on the **Name** field, an error will occur.
- If the character limit is met on the **Address** field, an error will occur.
- If the character limit is met on the **City** field, an error will occur.
- If the character limit is met on the **Postcode** field, an error will occur.
- If the character limit is met on the **Email** field, an error will occur.
- If the character limit is met on the **Telephone** field, an error will occur.

Dialog Boxes

- If the user clicks the “Exit” button, a dialog box will appear asking the user if they are sure they want to leave the application.

2.2.6 - Main Form - UI Elements

- **Client Field** - This will be a combo box called ***cbClient*** and will not allow the user to input any text, but it will allows the user to select an option from a dropdown menu
- **Customer Field** - This will be a textbox called ***txbCustomer*** and will have a character limit of 12 and allow numbers but not special characters
- **Name Field** - This will be a textbox called ***txbName*** and will have a character limit of 255 and allow numbers but not special characters
- **Address Field** - This will be a textbox called ***txbAddress*** and will have a character limit of 255 and allow numbers but not special characters
- **City Field** - This will be a textbox called ***txbCity*** and will have a character limit of 48 and allow numbers but not special characters
- **Postcode Field** - This will be a textbox called ***txbPostcode*** and will have a character limit of 10 and allow numbers but not special characters
- **Email Field** - This will be a textbox called ***txbEmail*** and will have a character limit of 48 and allow numbers and special characters
- **Telephone field** - This will be a textbox called ***txbTelephone*** and will have a character limit of 14 and only allow numbers or a ‘+’ symbol
- **Reason Field** - This will be a combo box called ***cbReason*** and will not allow the user to input any text, but it will allows the user to select an option from a dropdown menu
- **Accept Button** - This will be a button named ***btnExit*** and will have a “Click” method assigned to it.
- **Exit Button** - This will be a button named ***btnExit*** and will have a “Click” method assigned to it.
- **Clear Button** - This will be a button named ***btnClear*** and will have a “Click” method assigned to it.
- **Title Label -** This will be a label named ***lblTitle*** and will be populated with the text “Suppressor”

3.0.0 - SQL Database Construction

3.1.0 - Tables

3.1.1 - Suppressor_Details

This is the main table where the user data will be stored. This table will store all the data that the user inputs into the application

Columns

- **tKey -** This will be a unique key relating to the record, This will be a primary key for the table.
- **DT_Created-** This will be the date the record was inserted into the table
- **Client -** This will be the client the record was recorded against
- **Customer -** This will be the customer number (If there is one)
- **Customer_Flag -** This will be a boolean True/False field that will relate to if there is a customer number
- **Name -** This will be the name of the recipient
- **Address -** This will be the Address of the recipient
- **City -** This will be the City of the recipient
- **Postcode -** This will be the Postcode of the recipient
- **Email** - This will be the email of the recipient
- **Telephone** - This will be the Telephone number of the recipient
- **Reason -** This will be the reason for the suppression
- **Previous_Flags -** If the recipient is a customer, this will be the previous mailing flags
- **User_Created -** This will be the user who recorded the data

```
CREATE TABLE [dbo].[Suppressor_Details](
    [tKey] [varchar](255) NOT NULL,
    [DT_Created] [datetime] NULL,
    [Client] [varchar](255) NULL,
    [Customer] [varchar](48) NULL,
    [Customer_Flag] [int] NULL,
    [Name] [varchar](255) NULL,
    [Address] [varchar](500) NULL,
    [City] [varchar](255) NULL,
    [Postcode] [varchar](24) NULL,
    [Email] [varchar](48) NULL,
    [Telephone] [varchar](48) NULL,
    [Reason] [varchar](12) NULL,
    [Previous_Flags] [varchar](255) NULL,
    [User_Created] [varchar](48) NULL,
PRIMARY KEY CLUSTERED
(
    [tKey] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

3.1.2 - Suppressor_Reasons

This table will store the reasons for the data being recorded

Columns

- **ID -** This will be the sequence ID, This will be a primary key for the table.
- **Reason_Code -** This will be a 3 letter reason code
- **Description -** This will be the full description for the reason
- **Last_Updated -** This will be the last updated date
- **Last_Upd_User -** This will be the last updated user
- **Active** - This will be the status of the reason code

```
CREATE TABLE [dbo].[Suppressor_Reasons](
    [ID] [varchar](3) NOT NULL,
    [Reason_Code] [varchar](12) NOT NULL,
    [Description] [varchar](255) NULL,
    [Last_Updated] [datetime] NULL,
    [Last_Upd_User] [varchar](48) NULL,
    [Active] [int] NULL,
PRIMARY KEY CLUSTERED
(
    [ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY],
UNIQUE NONCLUSTERED
(
    [Reason_Code] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

3.1.3 - Suppressor_Clients

This table will store each client and relative SQL server data for them

Columns

- **ID -** This will be Row ID, This will be a primary key for the table.
- **Server -** This will be the SQL Server
- **Database -** This will be the SQL Database on the server
- **Site -** This will be the Elucid Site on the database
- **Description -** This will be a description of the Client
- **Last_Updated -** This will be the last updated date
- **Last_Upd_User -** This will be the last updated user
- **Active** - This will be the status of the Client

```
CREATE TABLE [dbo].[Suppressor_Clients](
    [ID] [varchar](6) NOT NULL,
    [Server] [varchar](48) NULL,
    [Database] [varchar](48) NULL,
    [Site] [varchar](48) NULL,
    [Description] [varchar](255) NULL,
    [Last_Updated] [datetime] NULL,
    [Last_Upd_User] [varchar](48) NULL,
    [Active] [int] NULL,
PRIMARY KEY CLUSTERED
(
    [ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

3.1.4 - Suppressor_LogBook

This table will work as a logbook logging the activity of the application

Columns

- **tKey** - This will be a unique key relating to the record, This will be a primary key for the table.
- **Domain_User** - This will be the user logged into the machine
- **User_Created** - This will be the user logged into the application
- **Form** - This will be the form the event took place on
- **Event** - This will be the event that took place
- **User** - This will be the user that did the event
- **Notes** - These will be any additional notes relating to the event.
- **DT_Created** - This will be the datetime the event took place

```
CREATE TABLE [dbo].[Suppressor_LogBook](
    [tKey] [varchar](255) NOT NULL,
    [Domain_User] [varchar](48) NULL,
    [User_Created] [varchar](48) NULL,
    [Form] [varchar](255) NULL,
    [Event] [varchar](255) NULL,
    [Notes] [varchar](4000) NULL,
    [DT_Created] [datetime] NULL,
PRIMARY KEY CLUSTERED
(
    [tKey] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

3.1.5 - Suppressor_Parameters

This table will be used to store fixed default parameters that the application can use in operations and functions.

Columns

- **ID** - This will be a unique ID for the parameter, This will be a primary key for the table.
- **Description** - This will be a description of the parameter
- **String_1** - This will be where a string can be stored
- **String_2** - This will be where a string can be stored
- **String_3** - This will be where a string can be stored
- **Int_1** - This will be where an int can be stored
- **Int_2** - This will be where an int can be stored
- **Int_3** - This will be where an int can be stored

```
CREATE TABLE [dbo].[Suppressor_Parameters](
    [ID] [varchar](3) NOT NULL,
    [Description] [varchar](100) NULL,
    [String_1] [varchar](4000) NULL,
    [String_2] [varchar](4000) NULL,
    [String_3] [varchar](4000) NULL,
    [Int_1] [int] NULL,
    [Int_2] [int] NULL,
    [Int_3] [int] NULL,
PRIMARY KEY CLUSTERED
(
    [ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

3.2.0 - Stored Procedures

3.2.1 - Suppressor_Get_Details

This stored procedure is designed to insert the data inputted on the form into the Suppressor_Details table. It employs an SQL insert command to add rows of data into the table, with fields determined by the provided parameters.
Additionally, this stored procedure updates the Elucid database table "cust_hist" by inserting a record into the table, which serves as an interaction note. This note is then displayed on the customer's account within the Elucid system.
Furthermore, the stored procedure gathers GDPR flags before the customer is suppressed and inserts them along with the suppression record. This step ensures that if the process needs to be reverted, the necessary data is available. The GDPR flags consist of true or false values, representing the customer's preferences regarding marketing and mailing activities.
The stored procedure will require 2 parameter (**@Customer, @Client**)

```
USE [CrossDatabase]
GO
/****** Object:  StoredProcedure [dbo].[Suppressor_Get_Details]    Script Date: 16/01/2024 15:28:34 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER OFF
GO


ALTER PROC [dbo].[Suppressor_Get_Details]

@Customer VARCHAR(24)
,@Client VARCHAR(255)

AS

DECLARE @FROM_CLAUSE VARCHAR(MAX)
DECLARE @Site VARCHAR(12)
DECLARE @SQL VARCHAR(MAX)

SET @Site = (SELECT [Site] FROM Suppressor_Clients WHERE [Description] = @Client)

SET @FROM_CLAUSE =
(
SELECT
'FROM [' + RTRIM([Server]) + '].[' + RTRIM([Database]) + '].dbo.cust'
FROM
Suppressor_Clients
WHERE
[Description] = @Client
)

SET @SQL =
'
SELECT
ISNULL(RTRIM(customer),''No Customer'') as Customer
,ISNULL(RTRIM(initials) + '' '', '''') + ISNULL(RTRIM(full_name), '''') as [Name]
,ISNULL(RTRIM([address]), '''') as [Address]
,ISNULL(RTRIM([city]), '''') as [City]
,ISNULL(RTRIM([postcode]), '''') as [Postcode]
,ISNULL(RTRIM([email_address]), '''') as [Email]
,ISNULL(RTRIM([phone_day]), '''') as [Telephone]
'
+ @FROM_CLAUSE +
'
WHERE
level_1 = ''' + @Site + '''
AND customer = ''' + @Customer + '''
'

PRINT (@SQL)
EXECUTE (@SQL)
```

3.2.2 - Suppressor_Insert_Details

This stored procedure will insert the data inputted on the form into the Suppressor_Details table

This stored procedure will use an SQL insert command to insert rows of data into the table with fields based on the provided parameters

This stored procedure will also update the Elucid database table cust_hist by inserting a record into the table serving as an interaction note, this will be displayed on the customer's account within the Elucid system.

This stored procedure gathers the GDPR flags before the customer is suppressed and inserts them along with the suppression record, this is so that if the process needs to be reverted it, we have the data to do so. The GDPR flags are true or false values that represent what type of marketing and mailing the customer agrees to.

The stored procedure will require 11 parameters (**@tKey**, **@Client**, **@Customer**, **@Name**, **@Address**, **@City**, **@Postcode**, **@Email**, **@Telephone**, **@Reason**, **@User**)

```
USE [CrossDatabase]
GO
/****** Object:  StoredProcedure [dbo].[Suppressor_Insert_Details]    Script Date: 07/02/2024 13:35:14 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER OFF
GO


ALTER PROC [dbo].[Suppressor_Insert_Details]

@tKey VARCHAR(255)
,@Client VARCHAR(255)
,@Customer VARCHAR(48)
,@Name VARCHAR(255)
,@Address VARCHAR(500)
,@City VARCHAR(255)
,@Postcode VARCHAR(24)
,@Email VARCHAR(48)
,@Telephone VARCHAR(48)
,@Reason VARCHAR(255)
,@User VARCHAR(48)

AS

--=============================================================================================================
-- DECLARE VARIABLES
--=============================================================================================================

DECLARE @ReasonCode VARCHAR(12) = (SELECT [Reason_Code] FROM [Suppressor_Reasons] WHERE [Description] = @Reason)
DECLARE @FROM_CLAUSE VARCHAR(MAX)
DECLARE @Site VARCHAR(12)
DECLARE @SQL_GET NVARCHAR(MAX)
DECLARE @SQL_UPDATE NVARCHAR(MAX)
DECLARE @Client_ID VARCHAR(3)
DECLARE @OG_No_Promote VARCHAR(1)
DECLARE @OG_Mail VARCHAR(1)
DECLARE @OG_Email VARCHAR(1)
DECLARE @OG_Phone VARCHAR(1)
DECLARE @OG_SMS VARCHAR(1)
DECLARE @OG_Survey VARCHAR(1)
DECLARE @OG_Rent VARCHAR(1)
DECLARE @OG_GDPR_Compliant VARCHAR(1)

SET @Customer = CASE WHEN @Customer = '' THEN NULL ELSE @Customer END
SET @Address = CASE WHEN @Address = '' THEN NULL ELSE @Address END
SET @City = CASE WHEN @City = '' THEN NULL ELSE @City END
SET @Postcode = CASE WHEN @Postcode = '' THEN NULL ELSE @Postcode END
SET @Email = CASE WHEN @Email = '' THEN NULL ELSE @Email END
SET @Telephone = CASE WHEN @Telephone = '' THEN NULL ELSE @Telephone END

```

```
52  --=============================================================================================
53  -- SET FROM CLAUSE
54  --=============================================================================================
55
56  SET @Site = (SELECT [Site] FROM Suppressor_Clients WHERE [Description] = @Client)
57  SET @Client_ID = (SELECT [ID] FROM Suppressor_Clients WHERE [Description] = @Client)
58
59  SET @FROM_CLAUSE =
60  (
61  SELECT
62              '[' + RTRIM([Server]) + '].[' + RTRIM([Database]) + '].dbo.'
63  FROM
64              Suppressor_Clients
65  WHERE
66              [Description] = @Client
67  )
68
69  --=============================================================================================
70  -- CREATE TEMP TABLE
71  --=============================================================================================
72
73  TRUNCATE TABLE OUTPUT_FLAGS
74
75  --BEGIN TRY
76
77  --DROP TABLE OUTPUT_FLAGS
78
79  --END TRY
80  --BEGIN CATCH
81
82  --END CATCH
83
84  --CREATE TABLE OUTPUT_FLAGS
85  --(
86  --  [OG_No_Promote] [varchar](1)  null,
87  --  [OG_Mail] [varchar](1) null,
88  --  [OG_Email] [varchar](1) null,
89  --  [OG_Phone] [varchar](1) null,
90  --  [OG_SMS] [varchar](1) null,
91  --  [OG_Survey] [varchar](1) null,
92  --  [OG_Rent] [varchar](1) null,
93  --  [OG_gdpr_Compliant] [varchar](1) null,
94  --)
95
96  --=============================================================================================
97  -- GET GDPR FLAG DATA
98  --=============================================================================================
```

```
--=============================================================================================
-- GET GDPR FLAG DATA
--=============================================================================================

SET @SQL_GET = '

INSERT INTO OUTPUT_FLAGS

SELECT
            RTRIM(no_promote)
            ,RTRIM(mail)
            ,RTRIM(email)
            ,RTRIM(phone)
            ,RTRIM(sms)
            ,RTRIM(survey)
            ,RTRIM(rent)
            ,RTRIM(gdpr_compliant)
FROM
            ' + @FROM_CLAUSE + 'cust_gdpr WITH(NOLOCK)
WHERE
            customer = ''' + @Customer + '''

INSERT INTO ' + @FROM_CLAUSE + 'cust_hist VALUES
(
    ''' + @tKey + '''
    ,''!''
    ,GETDATE()
    ,''' + @User + '''
    ,GETDATE()
    ,''' + @Site + '''
    ,''GROUP''
    ,CAST(GETDATE() as date)
    ,CAST(GETDATE() as time)
    ,''85''
    ,''Suppressor - No Mail Recorded''
    ,''' + @Customer + '''
    ,NULL
    ,NULL
    ,''SUP''
    ,NULL
    ,NULL
    ,NULL
    ,NULL
    ,NULL
    ,NULL
    ,NULL
    ,NULL
    ,''' + @User + '''
    ,''This customer has been recorded as a No Mail through the suppressor application.''
)
```

```
'


PRINT (@SQL_GET)
EXECUTE (@SQL_GET)

SET @OG_No_Promote = (SELECT OG_No_Promote FROM OUTPUT_FLAGS)
SET @OG_Mail = (SELECT OG_Mail FROM OUTPUT_FLAGS)
SET @OG_Email = (SELECT OG_Email FROM OUTPUT_FLAGS)
SET @OG_Phone = (SELECT OG_Phone FROM OUTPUT_FLAGS)
SET @OG_SMS = (SELECT OG_SMS FROM OUTPUT_FLAGS)
SET @OG_Survey = (SELECT OG_Survey FROM OUTPUT_FLAGS)
SET @OG_Rent = (SELECT OG_Rent FROM OUTPUT_FLAGS)
SET @OG_GDPR_Compliant = (SELECT OG_gdpr_Compliant FROM OUTPUT_FLAGS)

--DROP TABLE OUTPUT_FLAGS

--=============================================================================================================================
-- INSERT THE DATA
--=============================================================================================================================

IF NOT EXISTS (SELECT Customer FROM Suppressor_Details WHERE Customer = @Customer AND Customer <> '' AND Client = @Client)
INSERT INTO Suppressor_Details VALUES
(
    @tKey  -- BATCH NUMBER
    ,GETDATE() -- DATE CREATED
    ,@Client_ID -- CLIENT ID
    ,@Client -- CLIENT
    ,@Customer -- CUSTOMER
    ,CASE WHEN ISNULL(@Customer, '') = '' THEN 0 ELSE 1 END -- CUSTOMER FLAG
    ,@Name -- NAME
    ,@Address -- ADDRESS
    ,@City -- CITY
    ,@Postcode -- POSTCODE
    ,@Email -- EMAIL ADDRESS
    ,@Telephone -- TELEPHONE
    ,@ReasonCode -- REASON CODE
    ,@OG_No_Promote -- NO PROMOTE
    ,@OG_Mail -- MAIL
    ,@OG_Email -- EMAIL
    ,@OG_Phone -- PHONE
    ,@OG_SMS -- SMS
    ,@OG_Survey -- SURVEY
    ,@OG_Rent -- RENT
    ,@OG_GDPR_Compliant -- GDPR COMPLIANT
    ,@User -- USER CREATED

)
```

3.2.3 - Suppressor_Insert_LogBook

This stored procedure is intended to create a log of events executed by the application. The logged events will be restricted to the opening and closing of the application, along with a record of data insertion and every time an Elucid database table is interacted with.
In order to aid in error tracking, any occurrences of errors will also be recorded.
6 parameters are required: **@tKey**, **@Domain_User**, **@User_Created**, **@Form**, **@Event**, **@Notes**

```
USE [CrossDatabase]
GO
/****** Object:  StoredProcedure [dbo].[Suppressor_Insert_LogBook]
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER OFF
GO


ALTER PROC [dbo].[Suppressor_Insert_LogBook]

@tKey VARCHAR(255)
,@Domain_User VARCHAR(48)
,@User_Created VARCHAR(48)
,@Form VARCHAR(255)
,@Event VARCHAR(255)
,@Notes VARCHAR(255)



AS

INSERT INTO Suppressor_LogBook VALUES
(
@tKey
,@Domain_User
,@User_Created
,@Form
,@Event
,@Notes
,GETDATE()

)

```

4.0.0 - Development

4.1.0 - Login Form

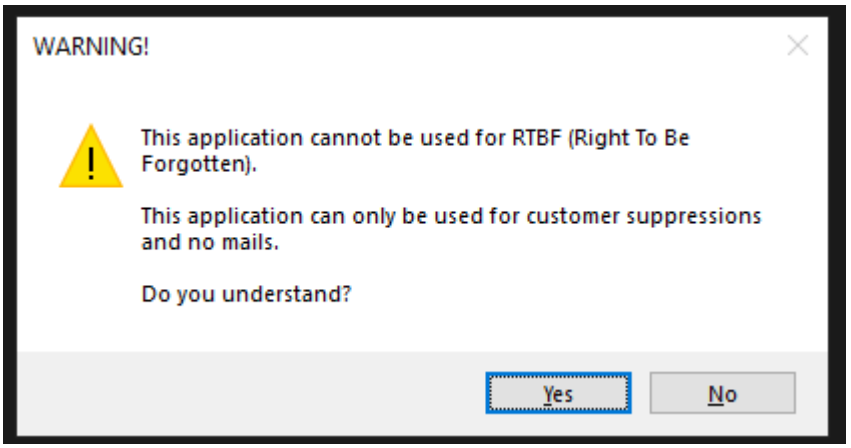
4.1.1 - Login Form - Design

Here we have the Login Form, as specified in the pre-development and planning stages. It features a text box for users to enter their username, buttons for exiting and logging into the application, and a label displaying the application name and the currently open form. The application logo is centred on the form.

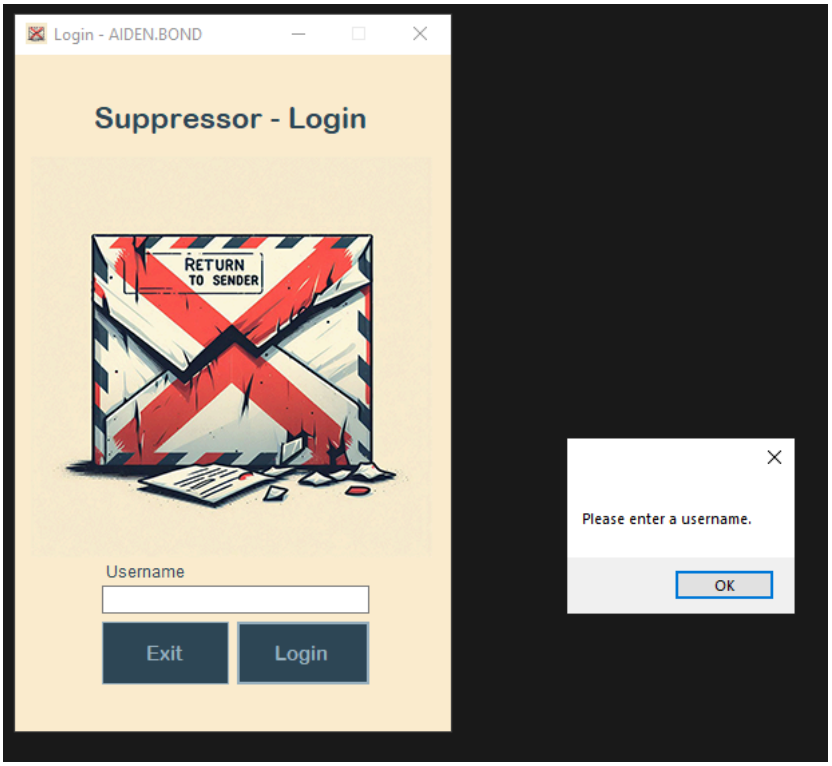
To enhance user experience and streamline design, the window of the application is non-resizable. This decision was made to prevent complications when scaling UI elements for various resolution displays, ultimately reducing the application's complexity and design time.

As no colour scheme was decided during the planning stage, a choice was made to adopt a light flesh colour with dark blue, light grey, and red accents. This colour scheme is consistently applied across all forms.

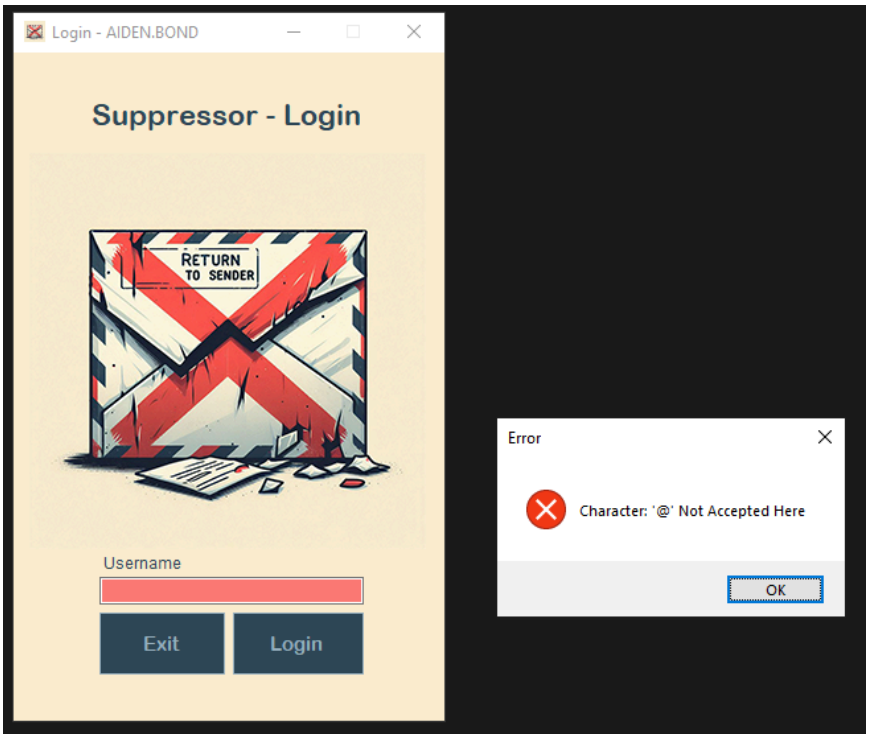
As specified in the pre-development design planning, here is the message which pops up when a user logs in, this informs the user that this application cannot be user for RTBF (Right to be forgotten) and is only to be used for no mails and customer suppressions



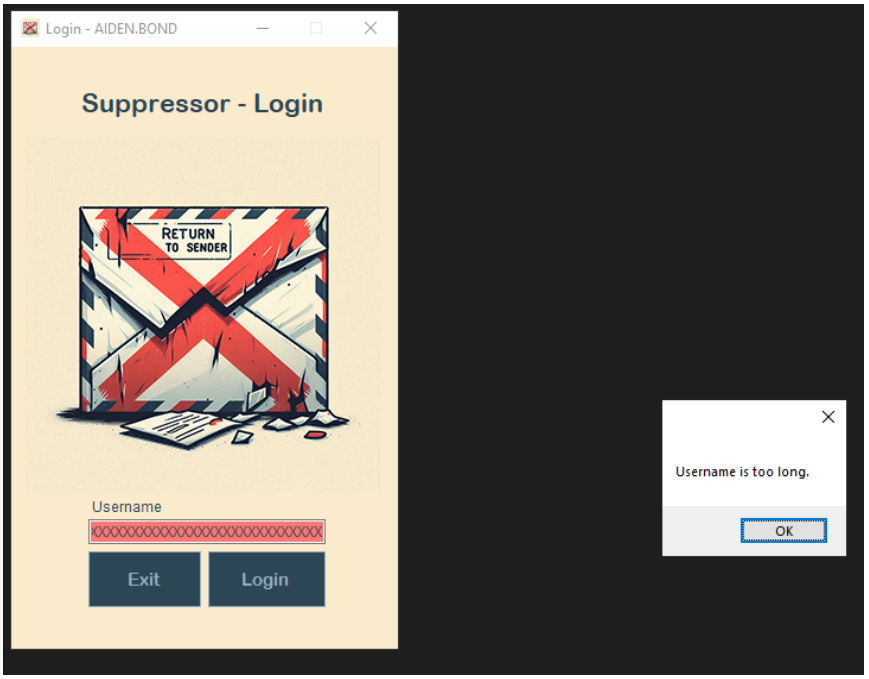
4.1.2 - Login Form - Errors



To ensure the robust operation of the application, error handling has been implemented in the login form. This specific error occurs when the user fails to enter a username into the designated field, triggering the display of a message box to notify the user.



This error shows the user has attempted to enter a special character into the username field. This has prompted the message box to appear notifying the user.



This error shows the user has entered a username that has met the character limit of the field, this has prompted the message box to appear notifying the user.

4.1.3 - Login Form - Methods

Login Form - Initialization

```
1 reference
public LoginForm()
{
    InitializeComponent();
    this.FormClosing += MainForm_FormClosing;
    this.MaximizeBox = false;
    this.KeyPreview = true;
    this.KeyDown += LoginForm_KeyDown;
    txbUsername.KeyPress += txbUsername_KeyPress;
    txbUsername.KeyDown += txbUsername_KeyDown;
    Text = $"Login - {Environment.UserName.ToUpper()}";
}
```

Here, we observe the code for initialising the form. In this process, the window's ability to maximise has been disabled, and various operational methods are being subscribed to. Additionally, the form text is set to display the form name along with the domain user currently using the application.

LoginForm - MainForm_FormClosing

```
// Exit Application Method
1 reference
private void MainForm_FormClosing(object sender, FormClosingEventArgs e)
{
    if (e.CloseReason == CloseReason.UserClosing)
    {
        DialogResult result = MessageBox.Show("Are you sure you want to Exit the application?", "Confirmation", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
        if (result == DialogResult.Yes)
        {
            Application.Exit();
        }
        else if (result == DialogResult.No)
        {
            e.Cancel = true;
        }
    }
}
```

Here, the method for closing the form is presented. In this method, a message box is triggered, prompting the user and asking if they are sure they want to exit the application. This particular method is invoked when the "X" button in the control box is clicked.

LoginForm - txbUsername_KeyPress

```
40          // Prevent Special Characters (txbUsername)
            1 reference
41          private void txbUsername_KeyPress(object sender, KeyPressEventArgs e)
42          {
43              if (!char.IsLetterOrDigit(e.KeyChar) && !char.IsControl(e.KeyChar))
44              {
45                  MessageBox.Show($"Character: '{e.KeyChar}' Not Accepted Here", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
46                  e.Handled = true;
47              }
48          }
```

Here, the method for validating the characters entered into the username text box (***txbUsername***) is presented. This method checks that the characters entered do not include special characters. If special characters are detected, a message box is triggered. Additionally, the method cancels the keypress, preventing the entry of special characters into the text box.

LoginForm - Environment Events

```
54          // Change Textbox Colours When Focused
            1 reference
55          private void txbUsername_Enter(object sender, EventArgs e)
56          {
57              txbUsername.BackColor = Color.FromArgb(253, 122, 115);
58          }
59
            1 reference
60          private void txbUsername_Leave(object sender, EventArgs e)
61          {
62              txbUsername.BackColor = Color.White;
63          }
64
65          // Change Button Colors
            1 reference
66          private void btnLogin_MouseEnter(object sender, EventArgs e)
67          {
68              btnLogin.BackColor = Color.FromArgb(152, 179, 194);
69              btnLogin.ForeColor = Color.FromArgb(46, 72, 89);
70          }
71
            1 reference
72          private void btnLogin_MouseLeave(object sender, EventArgs e)
73          {
74              btnLogin.BackColor = Color.FromArgb(46, 72, 89);
75              btnLogin.ForeColor = Color.FromArgb(152, 179, 194);
76          }
77
            1 reference
78          private void btnExit_MouseEnter(object sender, EventArgs e)
79          {
80              btnExit.BackColor = Color.FromArgb(152, 179, 194);
81              btnExit.ForeColor = Color.FromArgb(46, 72, 89);
82          }
83
            1 reference
84          private void btnExit_MouseLeave(object sender, EventArgs e)
85          {
86              btnExit.BackColor = Color.FromArgb(46, 72, 89);
87              btnExit.ForeColor = Color.FromArgb(152, 179, 194);
88          }
```

Here, several methods are shown, and these are invoked when UI elements are interacted with. When triggered, these methods alter the appearance of the UI elements, offering visual feedback to the user. This implementation is aimed at enhancing the user experience and making the application more engaging.

LoginForm - txbUsername_TextChanged

```
90          // Set Character Limit for Username
            1 reference
91          private void txbUsername_TextChanged(object sender, EventArgs e)
92          {
93              string username = txbUsername.Text;
94              int usernameLength = username.Length;
95
96              if (usernameLength > 100)
97              {
98                  MessageBox.Show("Username is too long.");
99                  txbUsername.Text = username.Substring(0, Math.Min(username.Length, 100));
100                 txbUsername.SelectAll();
101                 return;
102             }
103         }
```

Here, the method validating the length of the string entered into ***txbUsername*** is presented. Each time the text value in the textbox changes, this method is invoked, counting the current length of the string. If the string reaches its limit, a message box is triggered, and the last character entry is undone.

LoginForm - btnLogin_Click

```
110     // Login Button Event
        1 reference
111     private void btnLogin_Click(object sender, EventArgs e)
112     {
113         string userName = txbUsername.Text.ToUpper();
114
115         if (string.IsNullOrEmpty(userName))
116         {
117             MessageBox.Show("Please enter a username.");
118             return;
119         }
120
121         // Display T&Cs
122         DialogResult result = MessageBox.Show("This application cannot be used for RTBF (Right To Be Forgotten). \n\nThis application can only be used for customer suppressions and no mails. \n\nDo you understand?", "WARNING!", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);
123         if (result == DialogResult.Yes)
124         {
125             MainForm MainForm = new MainForm();
126             MainForm.userName = userName;
127             MainForm.Show();
128             this.Hide();
129         }
130
131     }
```

Here, the method invoked when the user clicks the login button is presented. This method checks whether ***txbUsername*** is null or empty. If it is, a message box is triggered, displaying an error. If ***txbUsername*** is not null or empty, the method triggers a dialog result and a message box, asking the user to confirm their understanding that the application is only to be used for customer suppression and not RTBF. If the user clicks "No," nothing happens, and the main form does not load. If the user clicks "Yes," the main form loads, and the username entered into ***txbUsername*** is passed through to the main form.

LoginForm - btnExit_Click

```
133         //Exit Button Event
134
            2 references
135         private void btnExit_Click(object sender, EventArgs e)
136         {
137             DialogResult result = MessageBox.Show("Are you sure you want to Exit the application?", "Confirmation", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
138             if (result == DialogResult.Yes)
139             {
140                 Application.Exit();
141             }
142         }
```

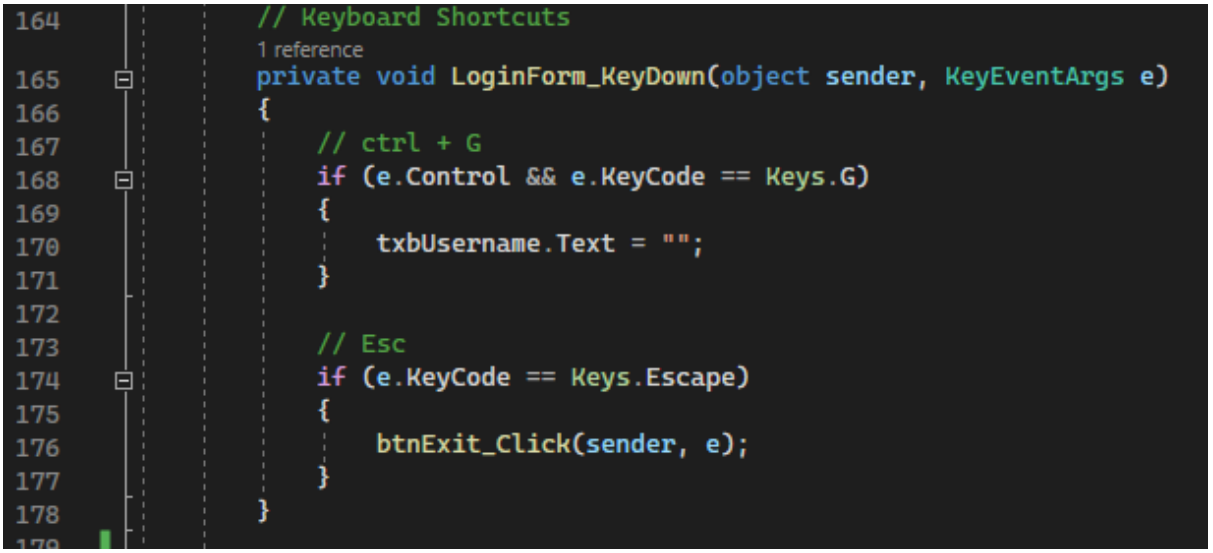
Here, the method invoked when the user clicks the exit button is presented. This method triggers a message box asking the user if they are sure they want to exit the application. If the user clicks "Yes," the application is closed, if the user clicks "No," the operation is cancelled.

LoginForm - txbUsername_KeyDown

```
148         // ctrl + V Event (txbUsername)
            1 reference
149         private void txbUsername_KeyDown(object sender, KeyEventArgs e)
150         {
151             if (e.Control && e.KeyCode == Keys.V)
152             {
153                 // Get the text from the clipboard
154                 string clipboardText = Clipboard.GetText();
155
156                 // Filter out invalid characters and set the filtered text to the TextBox
157                 txbUsername.Text = new string(clipboardText.Where(c => char.IsLetterOrDigit(c) || char.IsControl(c)).ToArray());
158
159                 // Cancel the paste operation
160                 e.SuppressKeyPress = true;
161             }
162         }
```

Here, the method invoked when the user presses any key while focused on the username field (***txbUsername***) is presented. This method checks if the keys pressed are "Ctrl" and "V". If they are, the text stored in the clipboard is retrieved, validated, and any special characters are removed before pasting the clipboard text into the field.

LoginForm - LoginForm_KeyDown

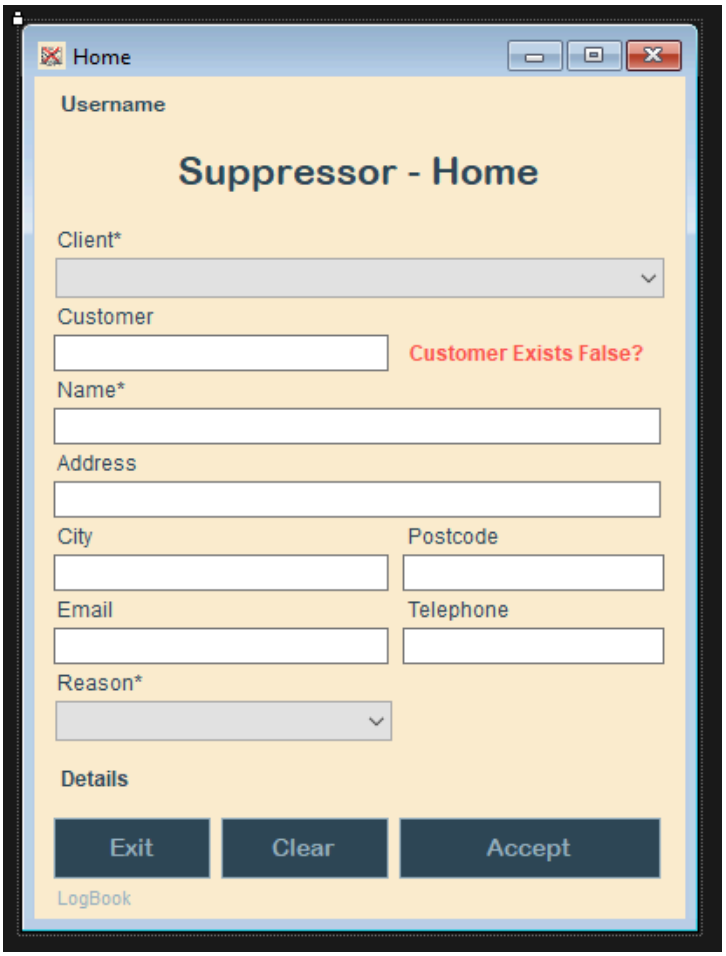


```
164         // Keyboard Shortcuts
            1 reference
165         private void LoginForm_KeyDown(object sender, KeyEventArgs e)
166         {
167             // ctrl + G
168             if (e.Control && e.KeyCode == Keys.G)
169             {
170                 txbUsername.Text = "";
171             }
172
173             // Esc
174             if (e.KeyCode == Keys.Escape)
175             {
176                 btnExit_Click(sender, e);
177             }
178         }
```

Here, the method invoked when the user presses any key while the form is open is presented. This method checks if the keys pressed are either "Ctrl" and "G" or "Esc" and performs a specific action based on the key combination. If the keys pressed are "Ctrl" and "G," the field for username is set to empty strings, effectively clearing them. If the user presses "Esc," the **btnExit_Click** method is invoked.

4.2.0 - Main Form

4.2.1 - Main Form - Design



Here, we have the Main Form as specified in the original design planning. The form includes text boxes for customer details such as name, address, city, postcode, email, and telephone, along with combo boxes for client and reason fields. Buttons for various operations are also provided.

There is a title label displaying the application name and the currently open form, along with a label indicating the username currently logged into the application.

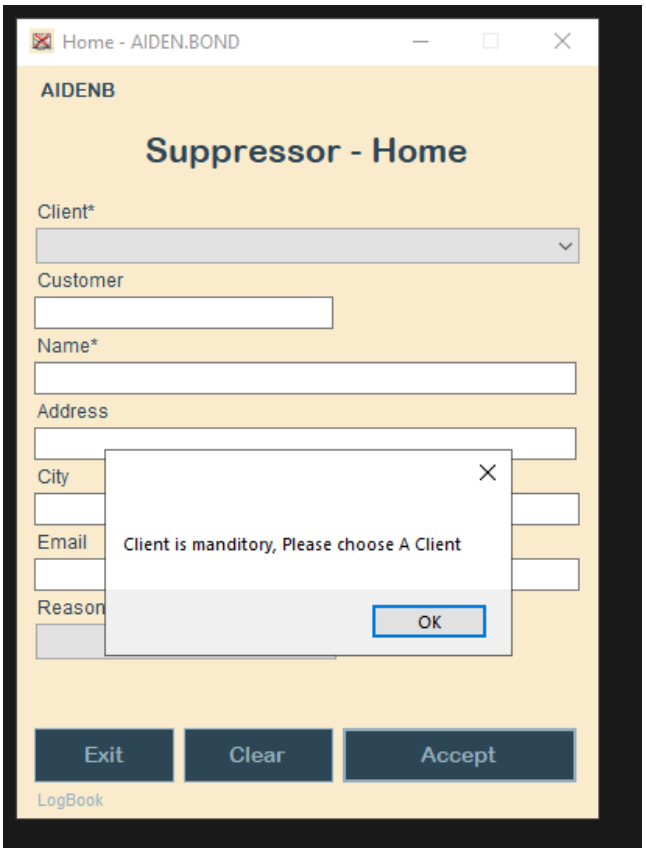
Additionally, a label with the placeholder text "Details" is present but remains hidden until the user successfully completes the form and clicks the "Accept" button. Upon successful completion, this label's value changes to inform the user of the successful operation and provides details of the entered record and timestamp.

Another label with the placeholder value "Customer Exists False?" is hidden until the user enters a customer that does not exist in the Elucid database.

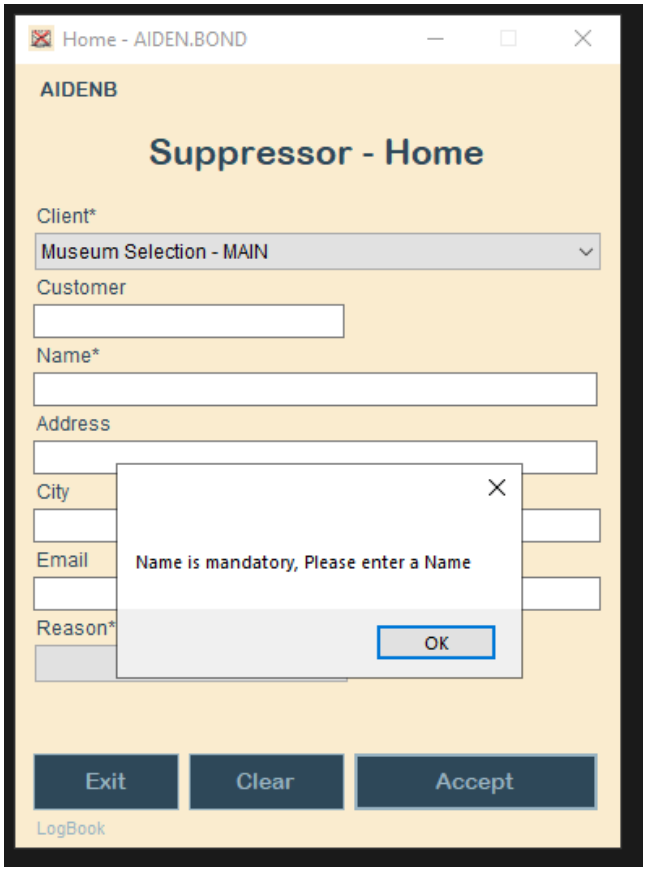
Similar to the Login Form, the window's ability to resize has been disabled to avoid complications when scaling UI elements for different resolution displays, aiming to simplify the application's design.

As no colour scheme was decided during the planning stage, a decision was made to adopt a light flesh colour with dark blue, light grey, and red accents, providing a consistent colour scheme across all forms.

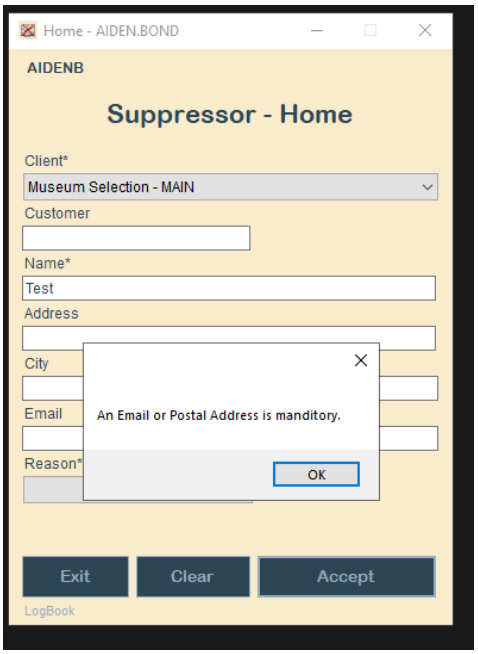
4.1.2 - Main Form - Errors



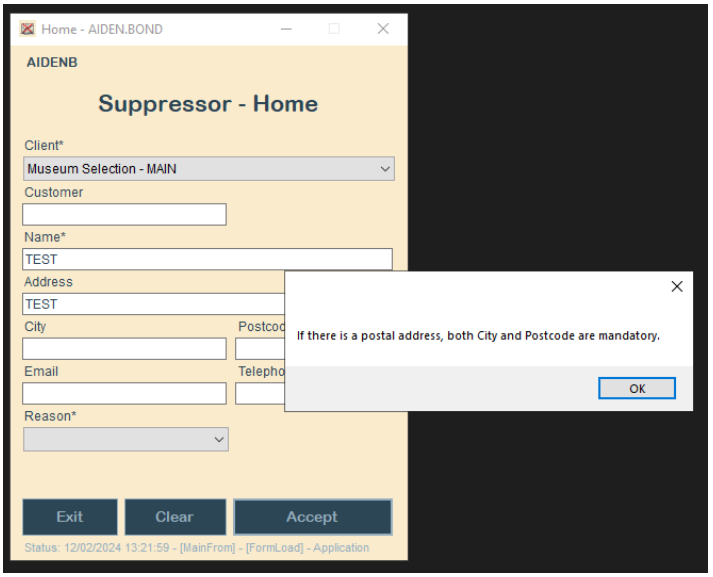
This error indicates that the user attempted to click the "Accept" button without selecting an option from the Client field. In response, a message box is triggered to notify the user of the omission.



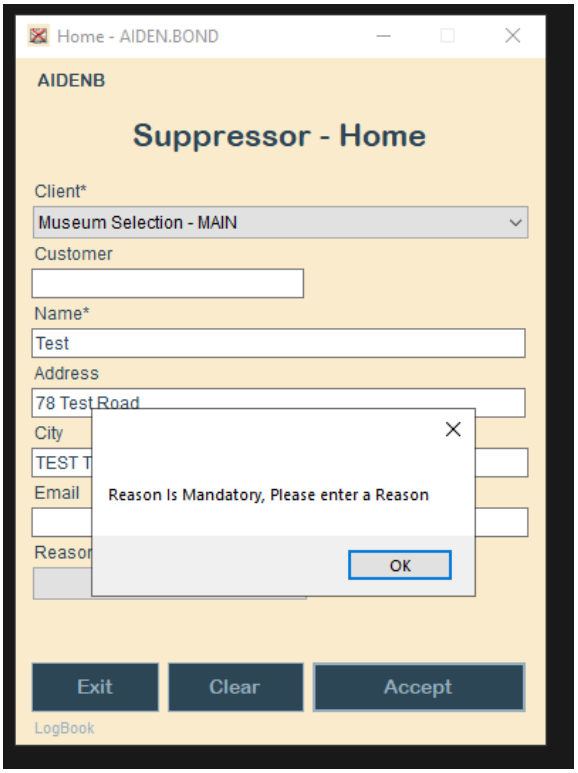
This error indicates that the user attempted to click the "Accept" button without entering a value in the Name field. As a result, a message box is triggered to notify the user of the missing information.



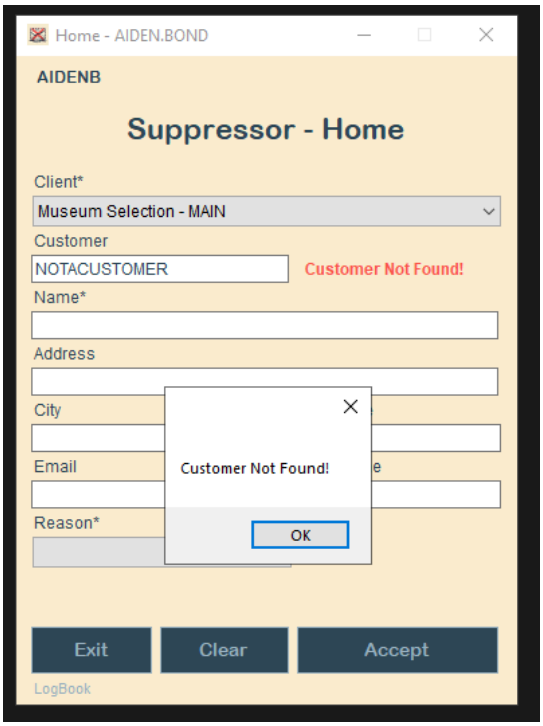
This error signifies that the user tried to click the "Accept" button without entering a value in either the email field or the Address field. Consequently, a message box is triggered to notify the user of the missing information.



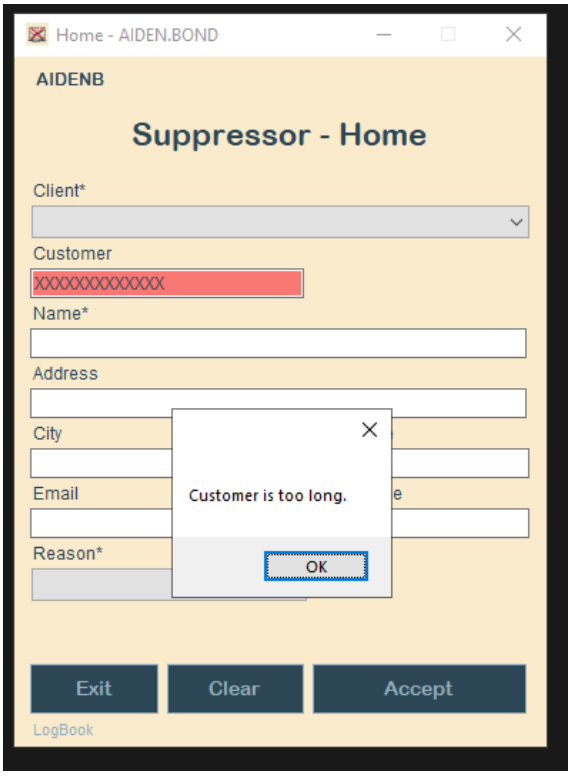
This error signifies that the user tried to click the "Accept" button without entering a value in the City field and/or the postcode field when there is a value in the address field. Consequently, a message box is triggered to notify the user of the missing information.



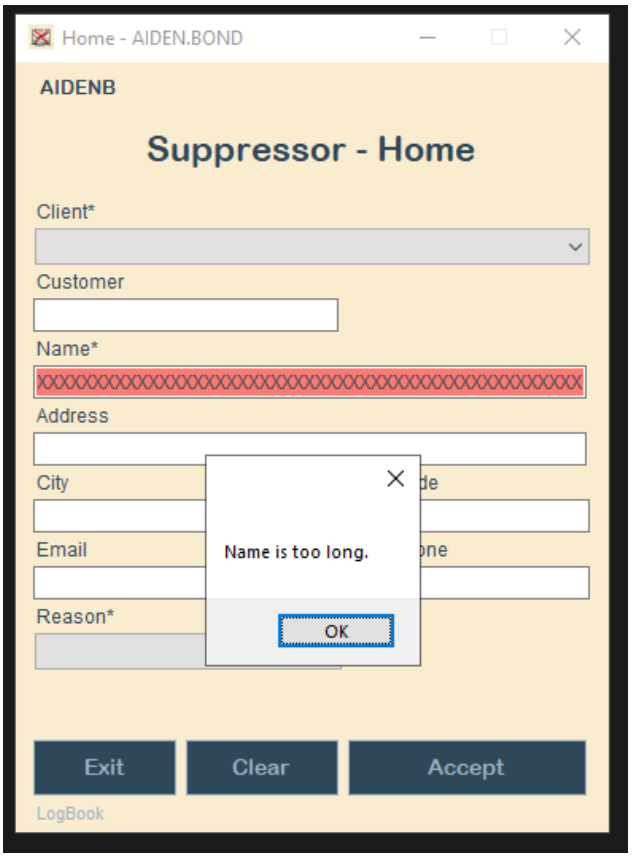
This error indicates that the user tried to click the "Accept" button without selecting an option from the reason field. As a result, a message box is triggered to notify the user of the omission.



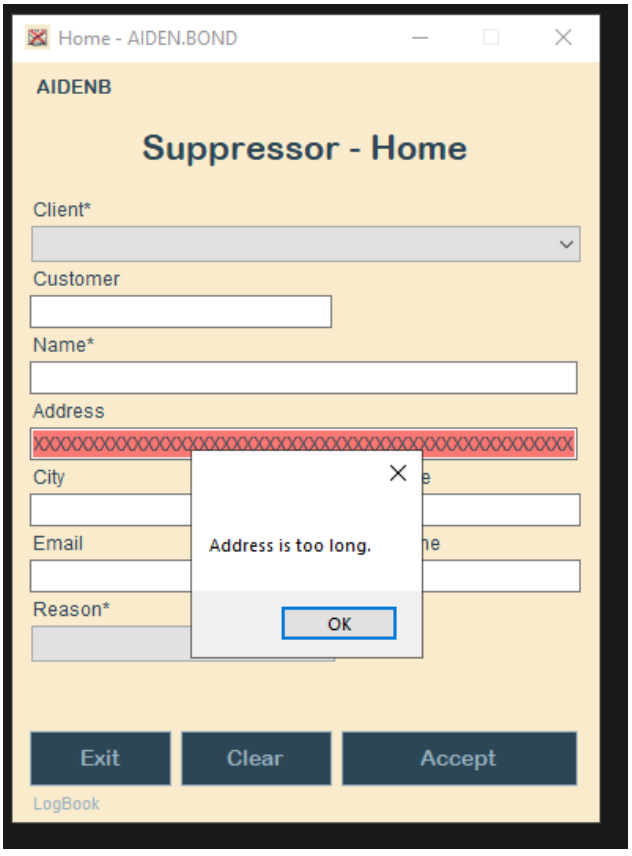
This error signifies that the user entered a customer number that does not exist in the Elucid database. In response, a message box and a label appear, notifying the user of the discrepancy.



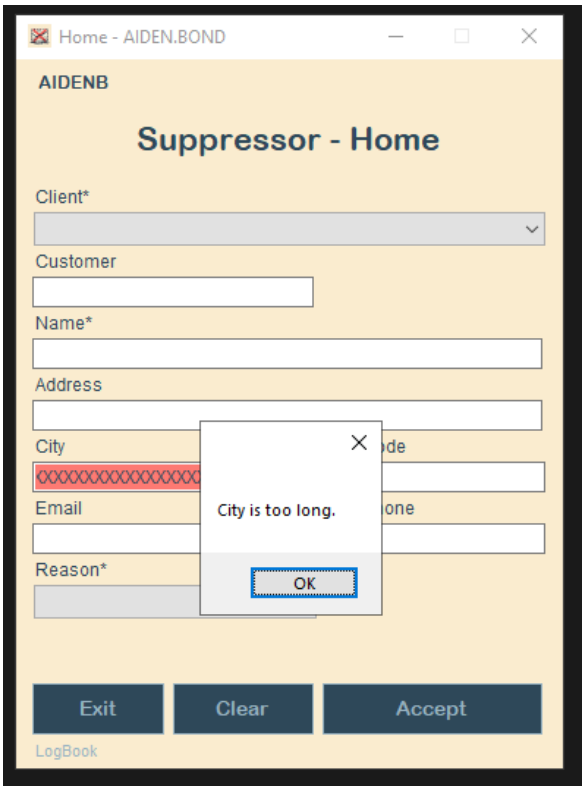
This error indicates that the user attempted to exceed the character limit for the Customer field. As a result, a message box appears, notifying the user of the character limit and preventing the input from exceeding it.



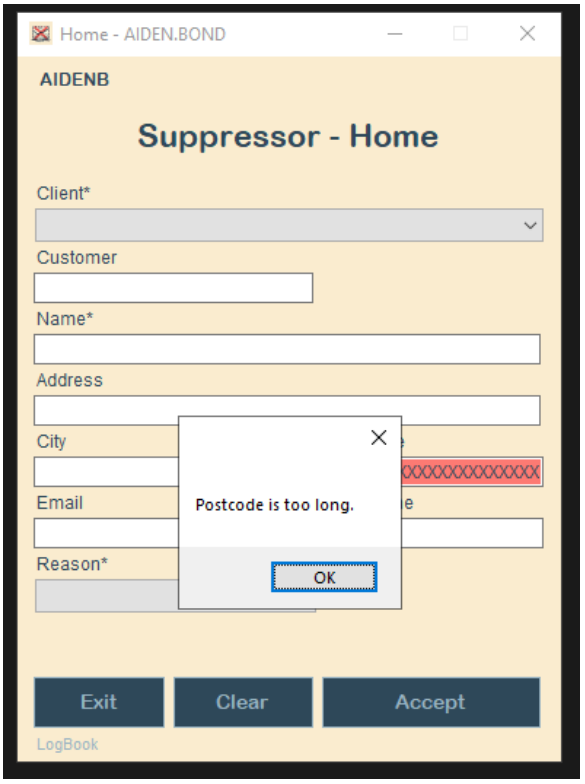
This error indicates that the user attempted to exceed the character limit for the Name field. Consequently, a message box appears, notifying the user of the character limit and preventing the input from surpassing it.



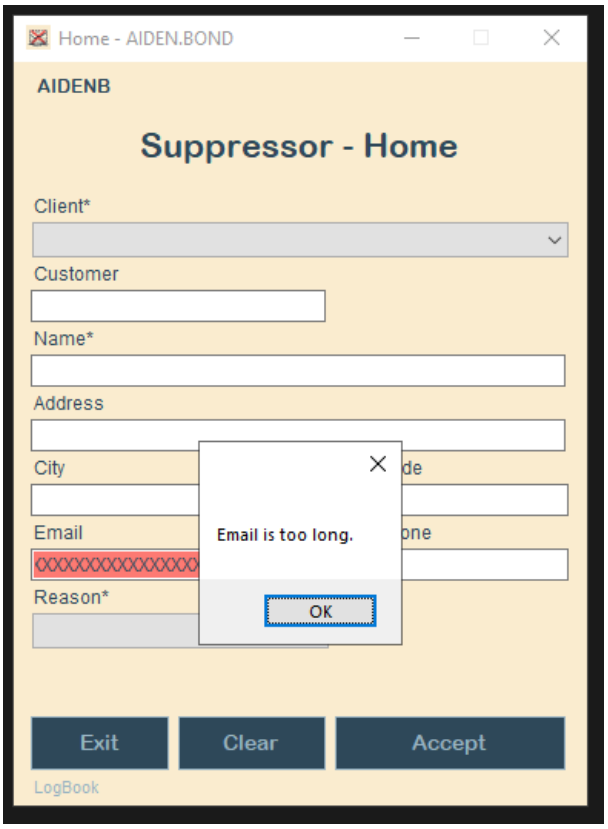
This error indicates that the user attempted to exceed the character limit for the Address field. In response, a message box appears, notifying the user of the character limit and preventing the input from surpassing it.



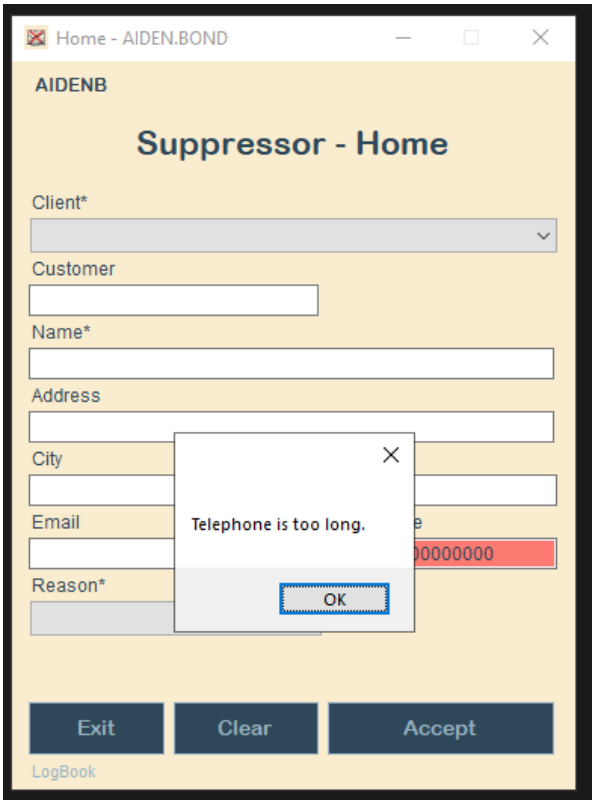
This error indicates that the user attempted to exceed the character limit for the City field. As a result, a message box appears, notifying the user of the character limit and preventing the input from surpassing it.



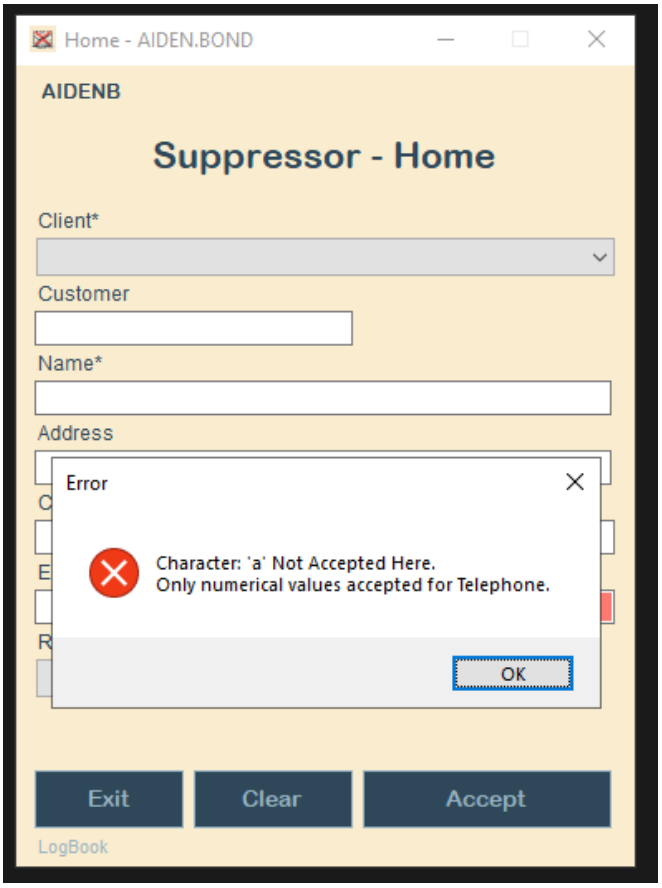
This error indicates that the user attempted to exceed the character limit for the Postcode field. In response, a message box appears, notifying the user of the character limit and preventing the input from surpassing it.



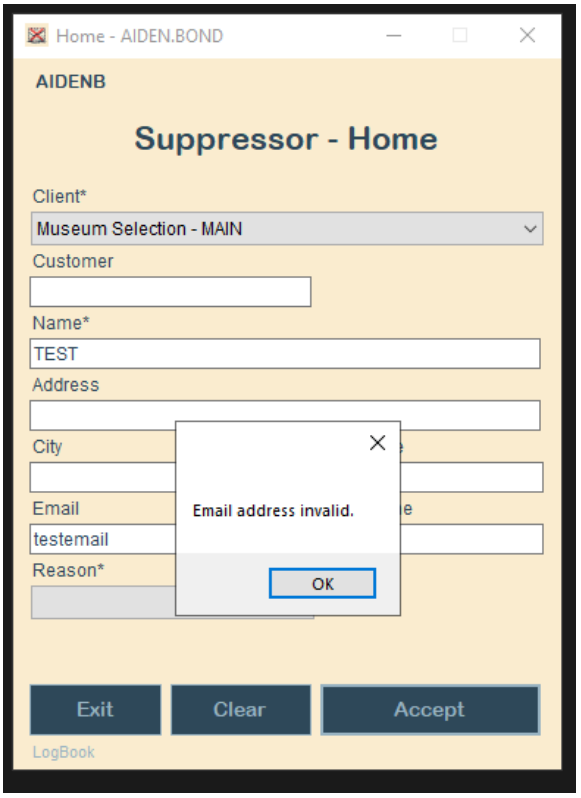
This error indicates that the user attempted to exceed the character limit for the Email field. As a result, a message box appears, notifying the user of the character limit and preventing the input from surpassing it.



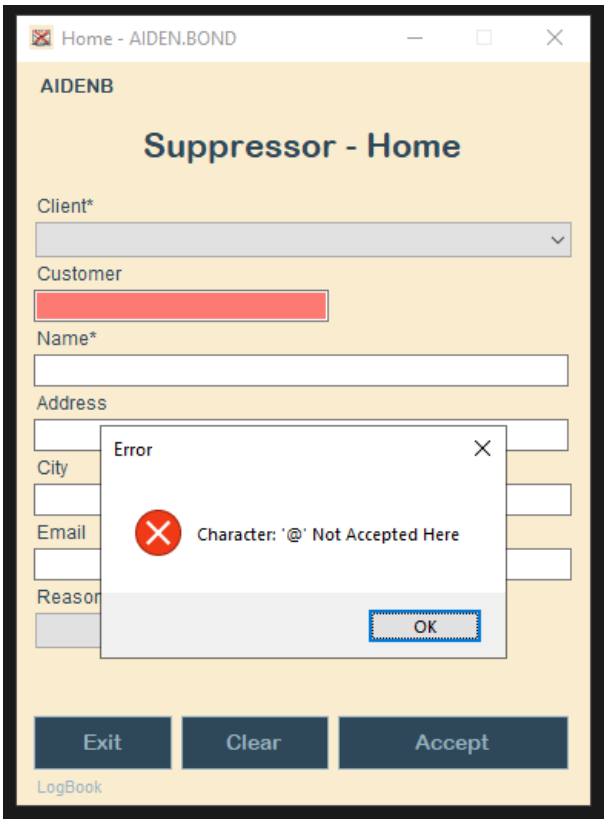
This error indicates that the user attempted to exceed the character limit for the Telephone field. As a result, a message box appears, notifying the user of the character limit and preventing the input from surpassing it.



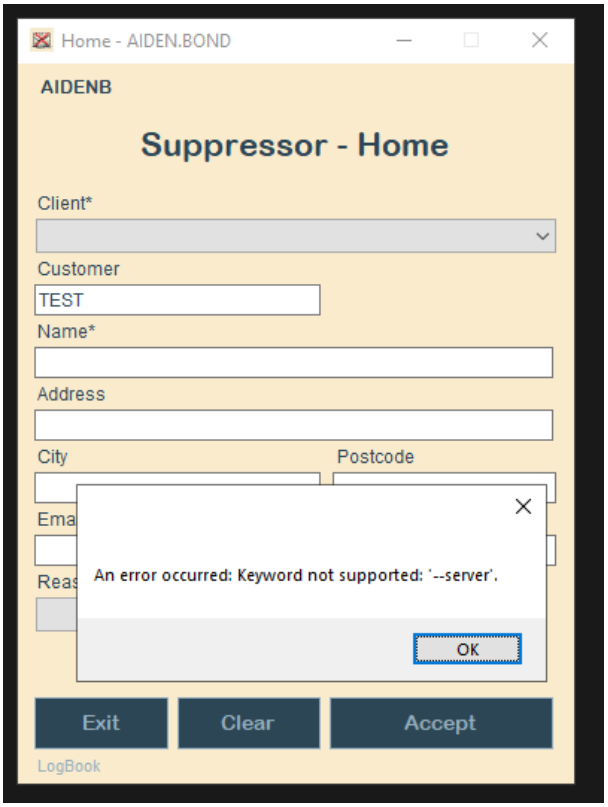
This error indicates that the user attempted to enter a character other than a number or the '+' symbol into the Telephone field. As a result, a message box appears, notifying the user of the invalid input and preventing the entry of non-numeric characters.



This error indicates that the user attempted to click the "Accept" button without entering a valid email address into the Email field. As a result, a message box appears, notifying the user of the invalid email address and prompting them to provide a valid one.



This error indicates that the user attempted to enter a special character into the Customer field. As a result, a message box appears, notifying the user of the invalid input and preventing the entry of special characters into the Customer field.



Throughout the methods, try/catch statements have been implemented to ensure no unhandled exceptions occur.

This error occurred after the SQL connection string was purposely altered to make it invalid. As a result, the message box displays the exception caught by the try/catch statement, demonstrating the effectiveness of the error handling mechanism.

4.2.3 - Main Form - Methods

Main Form - Initialization

```
13      // Declare Global Variables
        8 references
14      public string userName { get; set; }
15
16      public int cust = 0;
17
18      // Set SQL Connection String
19      private const string connectionString = "Server=SQL-SSRS;Database=CrossDatabase;Integrated Security=True;Encrypt=False;";
        1 reference
20      public MainForm()
21      {
22          InitializeComponent();
23          this.FormClosing += MainForm_FormClosing;
24          this.MaximizeBox = false; // Diasble Maximize window option
25          this.KeyPreview = true;
26          this.KeyDown += MainFrom_KeyDown;
27          PopulateComboBoxes(); // Populate the dropdown menus
28          btnClear.CausesValidation = false;
29          btnExit.CausesValidation = false;
30          Text = $"Home - {Environment.UserName.ToUpper()}";
31      }
```

Here is the initialization code for the Main Form. This code retrieves the username passed through from the Login Form. Additionally, it declares and sets a public int variable and a constant string used as the connection string for the SQL server.

As part of the initialization, the maximise button in the control box is disabled, certain key down and key preview events are subscribed to, and the form text is set to display the form name followed by the domain user logged in..

Main Form - MainForm_Load

```
33      // Load Form Method
        1 reference
34      private void MainForm_Load(object sender, EventArgs e)
35      {
36          lblUsername.Text = userName; // Show Username
37          LogBook("", "[MainFrom]", "[FormLoad]", "Application Started");
38      }
```

Here is the method for when the Main Form loads. This method changes the text value of the username label to the username passed through from the Login Form. Additionally, the method invokes the LogBook method to create an entry indicating that the application has been started.

Main Form - MainForm_FormClosing

```
// Exit Application Method
1 reference
private void MainForm_FormClosing(object sender, FormClosingEventArgs e)
{
    if (e.CloseReason == CloseReason.UserClosing)
    {
        DialogResult result = MessageBox.Show("Are you sure you want to Exit the application?", "Confirmation", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
        if (result == DialogResult.Yes)
        {
            LogBook("", "[MainForm]", "[FormClosing]", "Application Closed");
            Application.Exit();
        }
        else if (result == DialogResult.No)
        {
            e.Cancel = true;
        }
    }
}
```

Here is the method for closing the form. In this method, a message box is presented to the user, asking if they are sure they want to exit the application. This method is invoked when the "X" button in the control box is clicked.

Main Form - PopulateComboBoxes

```
// Populate Client Options
1 reference
private void PopulateComboBoxes()
{
    // Declare Variables
    string ClientQuery = "SELECT [Description] FROM Suppressor_Clients WHERE [Active] = '1' ORDER BY [ID]";
    string ReasonQuery = "SELECT [Description] FROM Suppressor_Reasons WHERE [Active] = '1' ORDER BY [ID]";

    try
    {
        // Execute SQL Command
        using (SqlConnection conn = new SqlConnection(connectionString))
        {

            conn.Open(); // Open SQL Connection
            using (SqlCommand cmdClient = new SqlCommand(ClientQuery, conn))
            {
                using (SqlDataReader reader = cmdClient.ExecuteReader())
                {
                    cbClient.Items.Clear();

                    while (reader.Read())
                    {
                        cbClient.Items.Add(reader["Description"].ToString());
                    }
                }
            }
            using (SqlCommand cmdReason = new SqlCommand(ReasonQuery, conn))
            {
                using (SqlDataReader reader = cmdReason.ExecuteReader())
                {
                    cbReason.Items.Clear();

                    while (reader.Read())
                    {
                        cbReason.Items.Add(reader["Description"].ToString());
                    }
                }
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred getting combo box values: \n{ex.Message}");
        Application.Exit();
    }
}
```

Here is the method used to populate the combo boxes with options for the user to select. This method retrieves options from two SQL database tables, and using the SqlDataReader function, it adds the results to the combo box options.

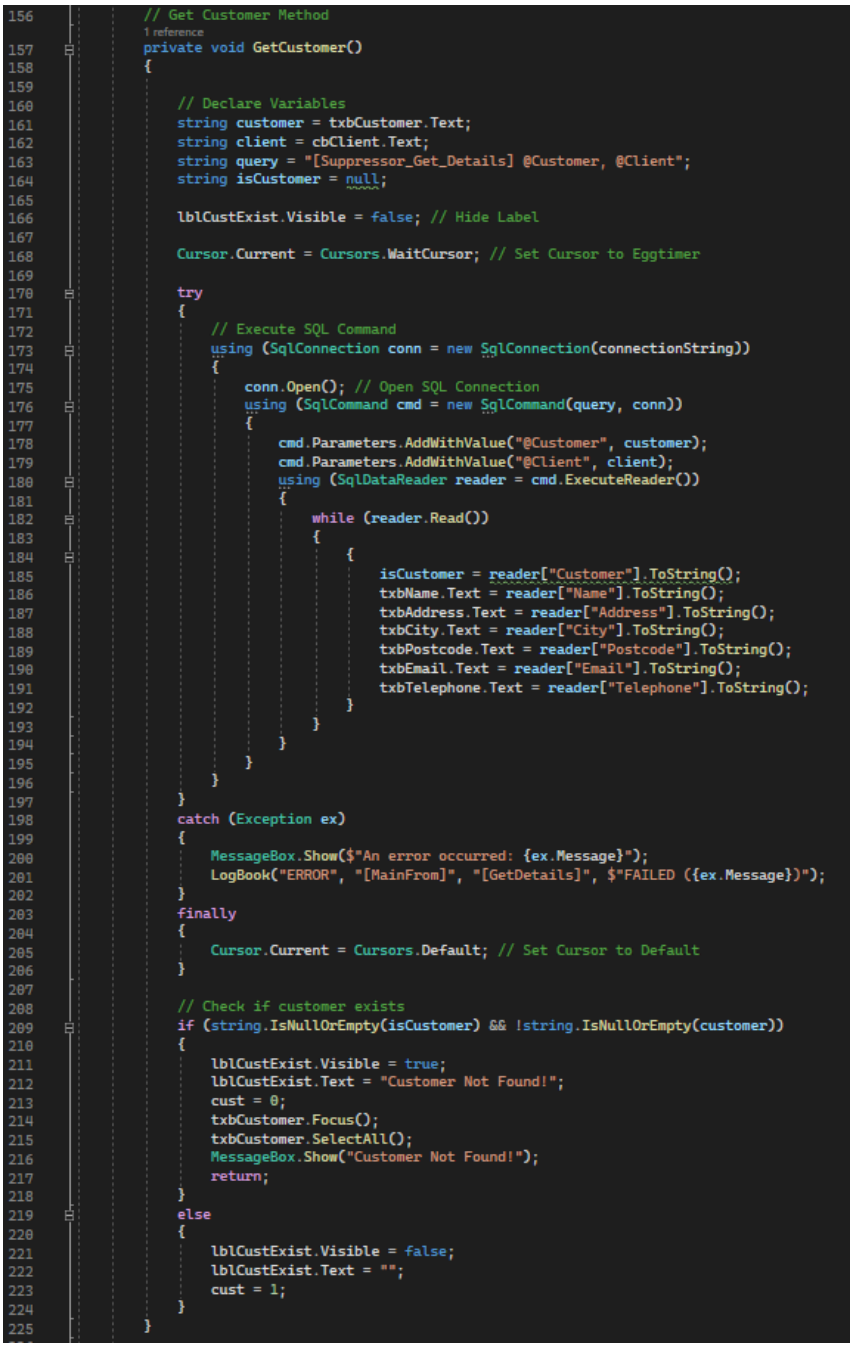
Main Form - LogBook

```
// LogBook Method
14 references
private void LogBook(string tKey, string Form, string Event, string notes)
{
    string query = "[Suppressor_Insert_LogBook] @tKey, @Domain_User, @User_Created, @Form, @Event, @Notes";
    string user = $"{Environment.MachineName}.{Environment.UserName}";
    string tKeySufix = Environment.UserName.Length >= 2 ? Environment.UserName.Substring(0, 2).ToUpper() : Environment.UserName.ToUpper();


    if (string.IsNullOrEmpty(tKey))
    {
        tKey = DateTime.Now.ToString("yyyyMMddHHmmssfff") + tKeySufix;
    }
    else if (tKey == "ERROR")
    {
        tKey = DateTime.Now.ToString("yyyyMMddHHmmssfff") + tKeySufix + "-ER";
    }

    try
    {
        // Execute SQL Query
        using (SqlConnection conn = new SqlConnection(connectionString))
        {
            conn.Open(); // Open SQL Connection
            using (SqlCommand cmd = new SqlCommand(query, conn))
            {
                cmd.Parameters.AddWithValue("@tKey", tKey);
                cmd.Parameters.AddWithValue("@Domain_User", user.ToUpper());
                cmd.Parameters.AddWithValue("@User_Created", userName);
                cmd.Parameters.AddWithValue("@Form", Form);
                cmd.Parameters.AddWithValue("@Event", Event);
                cmd.Parameters.AddWithValue("@Notes", notes);
                cmd.ExecuteNonQuery();
            }
            conn.Close(); // Close SQL Connection
        }
        if (Event != "[UpdateElucidGDPR]")
        {
            lblLogBook.Text = $"Status: {DateTime.Now} - {Form} - {Event} - {notes};";
        }
    }
    // Catch Errors
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}");
    }
}
```

Here is the method that is invoked whenever the application performs a process that needs to be logged or when an error occurs that needs to be logged. This method gathers data from parameters input when invoked, retrieves the domain user, generates a unique time key (referred to as a tKey), and inserts this information into the database table Suppressor_LogBook. The method creates a different tKey format depending on whether the invocation is due to an error or a regular application process.

This method also includes an SQL query that executes the Suppressor_Insert_LogBook stored procedure on the database.

Main Form - GetCustomer

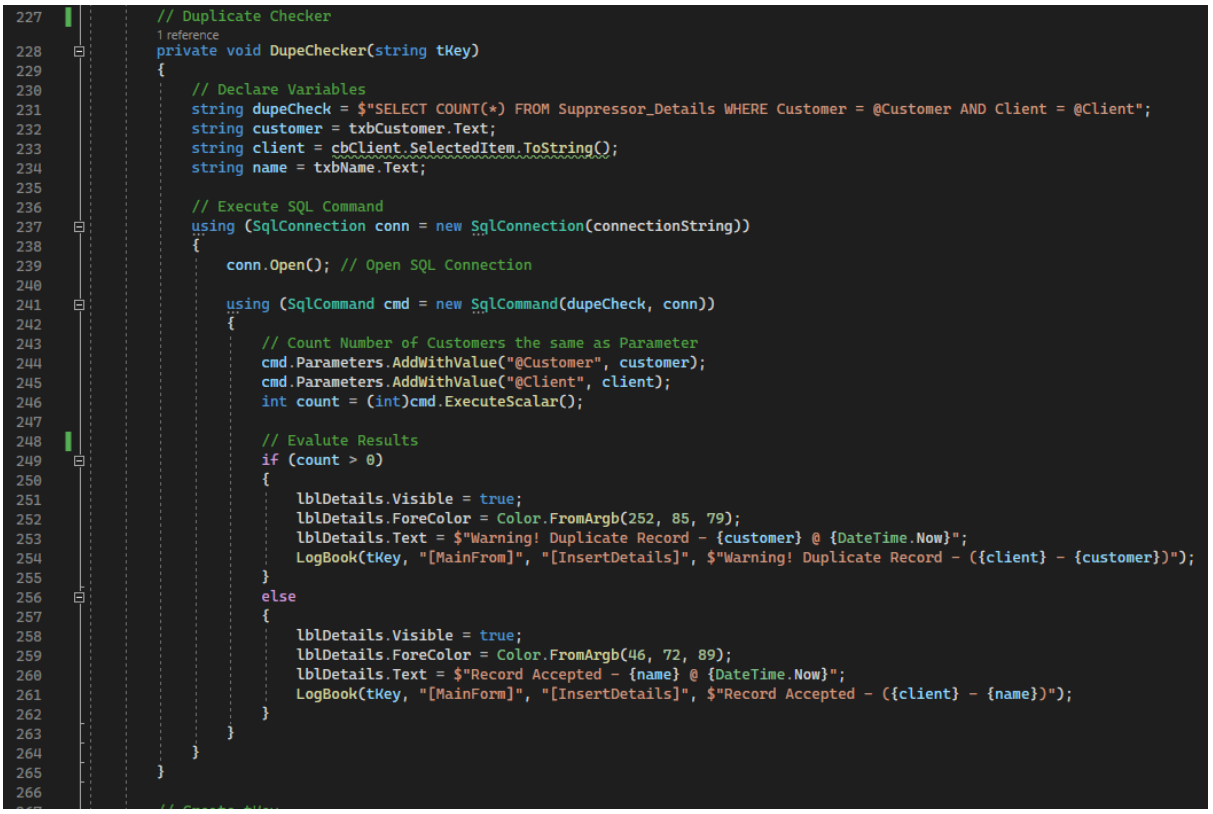


```
// Get Customer Method
1 reference
private void GetCustomer()
{

    // Declare Variables
    string customer = txbCustomer.Text;
    string client = cbClient.Text;
    string query = "[Suppressor_Get_Details] @Customer, @Client";
    string isCustomer = null;

    lblCustExist.Visible = false; // Hide Label

    Cursor.Current = Cursors.WaitCursor; // Set Cursor to Eggtimer

    try
    {
        // Execute SQL Command
        using (SqlConnection conn = new SqlConnection(connectionString))
        {
            conn.Open(); // Open SQL Connection
            using (SqlCommand cmd = new SqlCommand(query, conn))
            {
                cmd.Parameters.AddWithValue("@Customer", customer);
                cmd.Parameters.AddWithValue("@Client", client);
                using (SqlDataReader reader = cmd.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        {
                            isCustomer = reader["Customer"].ToString();
                            txbName.Text = reader["Name"].ToString();
                            txbAddress.Text = reader["Address"].ToString();
                            txbCity.Text = reader["City"].ToString();
                            txbPostcode.Text = reader["Postcode"].ToString();
                            txbEmail.Text = reader["Email"].ToString();
                            txbTelephone.Text = reader["Telephone"].ToString();
                        }
                    }
                }
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}");
        LogBook("ERROR", "[MainFrom]", "[GetDetails]", $"FAILED ({ex.Message})");
    }
    finally
    {
        Cursor.Current = Cursors.Default; // Set Cursor to Default
    }

    // Check if customer exists
    if (string.IsNullOrEmpty(isCustomer) && !string.IsNullOrEmpty(customer))
    {
        lblCustExist.Visible = true;
        lblCustExist.Text = "Customer Not Found!";
        cust = 0;
        txbCustomer.Focus();
        txbCustomer.SelectAll();
        MessageBox.Show("Customer Not Found!");
        return;
    }
    else
    {
        lblCustExist.Visible = false;
        lblCustExist.Text = "";
        cust = 1;
    }
}
```

Here is the method used to check the database and see if the customer value entered in the customer field matches a record in the database. This method initially hides the CustExists label, which warns the user if the customer number they entered is not valid. Then, it executes the SQL stored Procedure Suppressor_Get_Details. Using the results from the stored procedure, it determines if the value entered into the customer field is valid. If it is valid, the method populates the relevant fields based on the data gathered from the stored procedure.

If the value entered into the field is not a valid customer number, it triggers the error process where a message box appears, and the CustExist label is made visible to warn the user.

Main Form - DupeChecker



```
// Duplicate Checker
1 reference
private void DupeChecker(string tKey)
{
    // Declare Variables
    string dupeCheck = $"SELECT COUNT(*) FROM Suppressor_Details WHERE Customer = @Customer AND Client = @Client";
    string customer = txbCustomer.Text;
    string client = cbClient.SelectedItem.ToString();
    string name = txbName.Text;

    // Execute SQL Command
    using (SqlConnection conn = new SqlConnection(connectionString))
    {
        conn.Open(); // Open SQL Connection

        using (SqlCommand cmd = new SqlCommand(dupeCheck, conn))
        {
            // Count Number of Customers the same as Parameter
            cmd.Parameters.AddWithValue("@Customer", customer);
            cmd.Parameters.AddWithValue("@Client", client);
            int count = (int)cmd.ExecuteScalar();

            // Evalute Results
            if (count > 0)
            {
                lblDetails.Visible = true;
                lblDetails.ForeColor = Color.FromArgb(252, 85, 79);
                lblDetails.Text = $"Warning! Duplicate Record - {customer} @ {DateTime.Now}";
                LogBook(tKey, "[MainFrom]", "[InsertDetails]", $"Warning! Duplicate Record - ({client} - {customer})");
            }
            else
            {
                lblDetails.Visible = true;
                lblDetails.ForeColor = Color.FromArgb(46, 72, 89);
                lblDetails.Text = $"Record Accepted - {name} @ {DateTime.Now}";
                LogBook(tKey, "[MainForm]", "[InsertDetails]", $"Record Accepted - ({client} - {name})");
            }
        }
    }
}
```

Here is the method that checks for duplicate records. This method takes the customer number entered in the customer field and checks whether the customer has already been entered into the Suppressor_Details database table. If the customer already exists in the table, the LogBook method is invoked to record that a duplicate has been entered. Additionally, the details label, which mirrors the LogBook events, changes to provide visual feedback to the user that the values they have entered are duplicates.

Main Form - CreateTKey

```
// Create tKey
1 reference
private string CreateTKey()
{
    string name = txbName.Text;
    string tKeySufix = name.Length >= 2 ? name.Substring(0, 2).ToUpper() : name.ToUpper();
    string tKeyTimeStamp = DateTime.Now.ToString("yyyyMMddHHmmssfff");
    string tKey = tKeyTimeStamp + tKeySufix;
    return tKey;
}
```

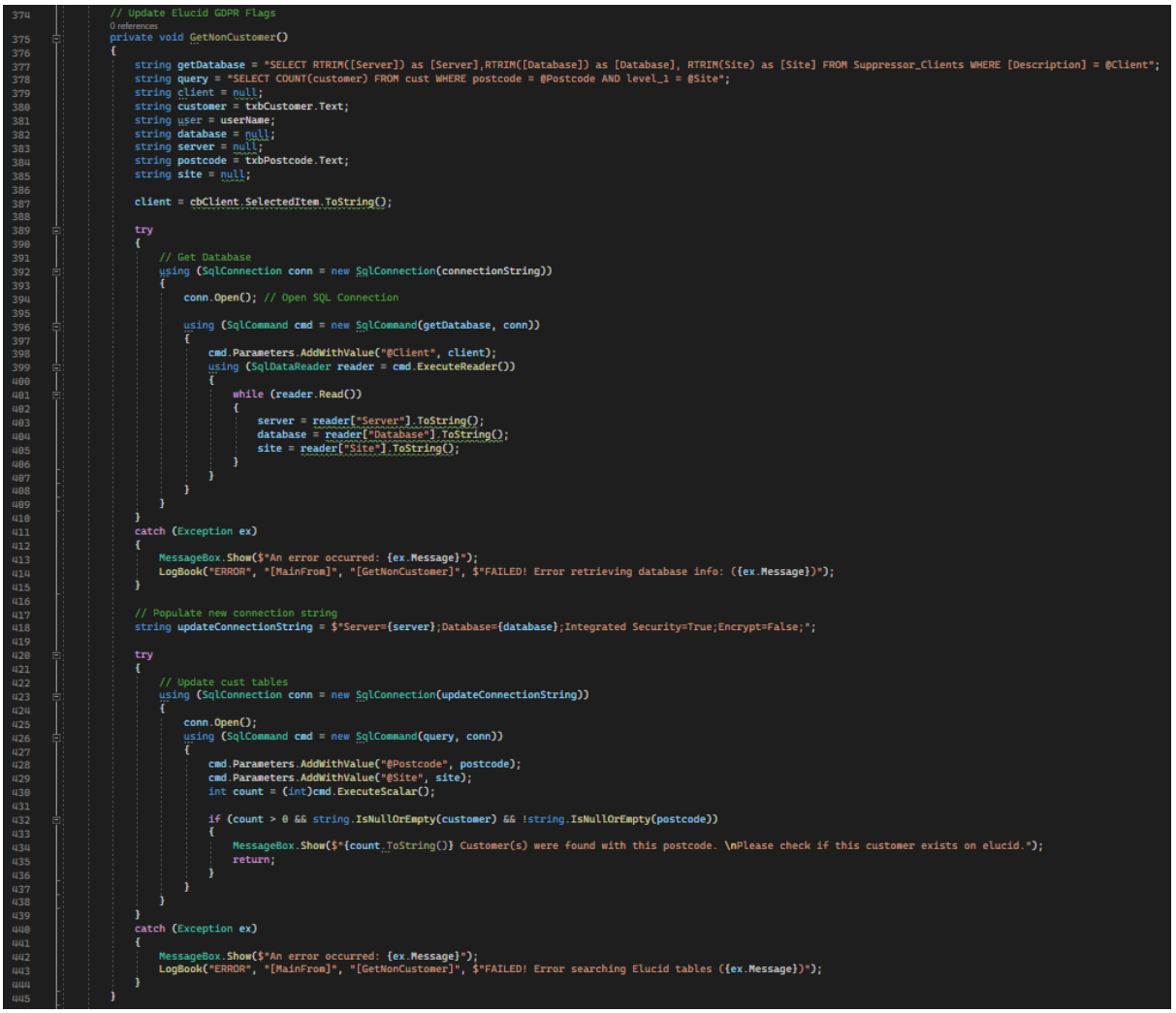
Here is the method used to create a unique tKey. This method gathers the customer's name, extracts the first 2 characters, and converts them to uppercase. The method then retrieves the current date and time up to the millisecond and formats it into a string. Subsequently, the datetime string and the first 2 characters of the name are concatenated together to form a unique tKey.

Main Form - UpdateElucidGDPR

```
// Update Elucid GDPR Flags
1 reference
private void UpdateElucidGDPR()
{
    LogBook("", "[MainForm]", "[UpdateElucidGDPR]", "Started");

    string getDatabase = "SELECT RTRIM([Server]) as [Server],RTRIM([Database]) as [Database] FROM Suppressor_Clients WHERE [Description] = @Client";
    string getQuery = "SELECT String_1, String_2 FROM Suppressor_Parameters WHERE ID = '001'";
    string client = null;
    string customer = txbCustomer.Text;
    string user = userName;
    string database = null;
    string server = null;
    string queryUpdate = null;
    string queryGDPRUpdate = null;

    client = cbClient.SelectedItem.ToString();

    try
    {
        // Get Database & Query Info
        using (SqlConnection conn = new SqlConnection(connectionString))
        {
            conn.Open(); // Open SQL Connection

            using (SqlCommand cmd = new SqlCommand(getDatabase, conn))
            {
                cmd.Parameters.AddWithValue("@Client", client);
                using (SqlDataReader reader = cmd.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        server = reader["Server"].ToString();
                        database = reader["Database"].ToString();
                    }
                }
            }

            using (SqlCommand cmd = new SqlCommand(getQuery, conn))
            {
                cmd.Parameters.AddWithValue("@Client", client);
                using (SqlDataReader reader = cmd.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        queryUpdate = reader["String_1"].ToString();
                        queryGDPRUpdate = reader["String_2"].ToString();

                    }
                }
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}");
        LogBook("ERROR", "[MainFrom]", "[UpdateElucidGDPR]", $"FAILED! Error retrieving database info: ({ex.Message})");
    }
    // Populate new connection string
    string updateConnectionString = $"Server={server};Database={database};Integrated Security=True;Encrypt=False;";

    try
    {
        // Update cust tables
        using (SqlConnection conn = new SqlConnection(updateConnectionString))
        {
            conn.Open();
            using (SqlCommand cmd = new SqlCommand(queryUpdate, conn))
            {
                cmd.Parameters.AddWithValue("@Customer", customer);
                cmd.Parameters.AddWithValue("@User", userName);
                cmd.ExecuteNonQuery();
            }

            LogBook("", "[MainForm]", "[UpdateElucidGDPR]", $"Update Cust Executed ({customer})");

            using (SqlCommand cmd = new SqlCommand(queryGDPRUpdate, conn))
            {
                cmd.Parameters.AddWithValue("@Customer", customer);
                cmd.Parameters.AddWithValue("@User", userName);
                cmd.ExecuteNonQuery();
            }
            conn.Close();

            LogBook("", "[MainForm]", "[UpdateElucidGDPR]", $"Update GDPR Executed ({customer})");
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}");
        LogBook("ERROR", "[MainFrom]", "[UpdateElucidGDPR]", $"FAILED! Error Updating Elucid tables ({ex.Message})");
    }

    LogBook("", "[MainForm]", "[UpdateElucidGDPR]", "Finished");

}
```

Here is the method used to update the GDPR database tables used by Elucid. This method gathers SQL Server connection values for the client selected by the user. Additionally, it collects two strings containing SQL Update queries. Using these values, a new connection string is formed to connect to the respective client's database. The SQL queries are then executed on that database. These queries are retrieved from the SQL database table Suppressor_Parameters and are pre-defined to update the Elucid GDPR tables to reflect the suppression of the customer.

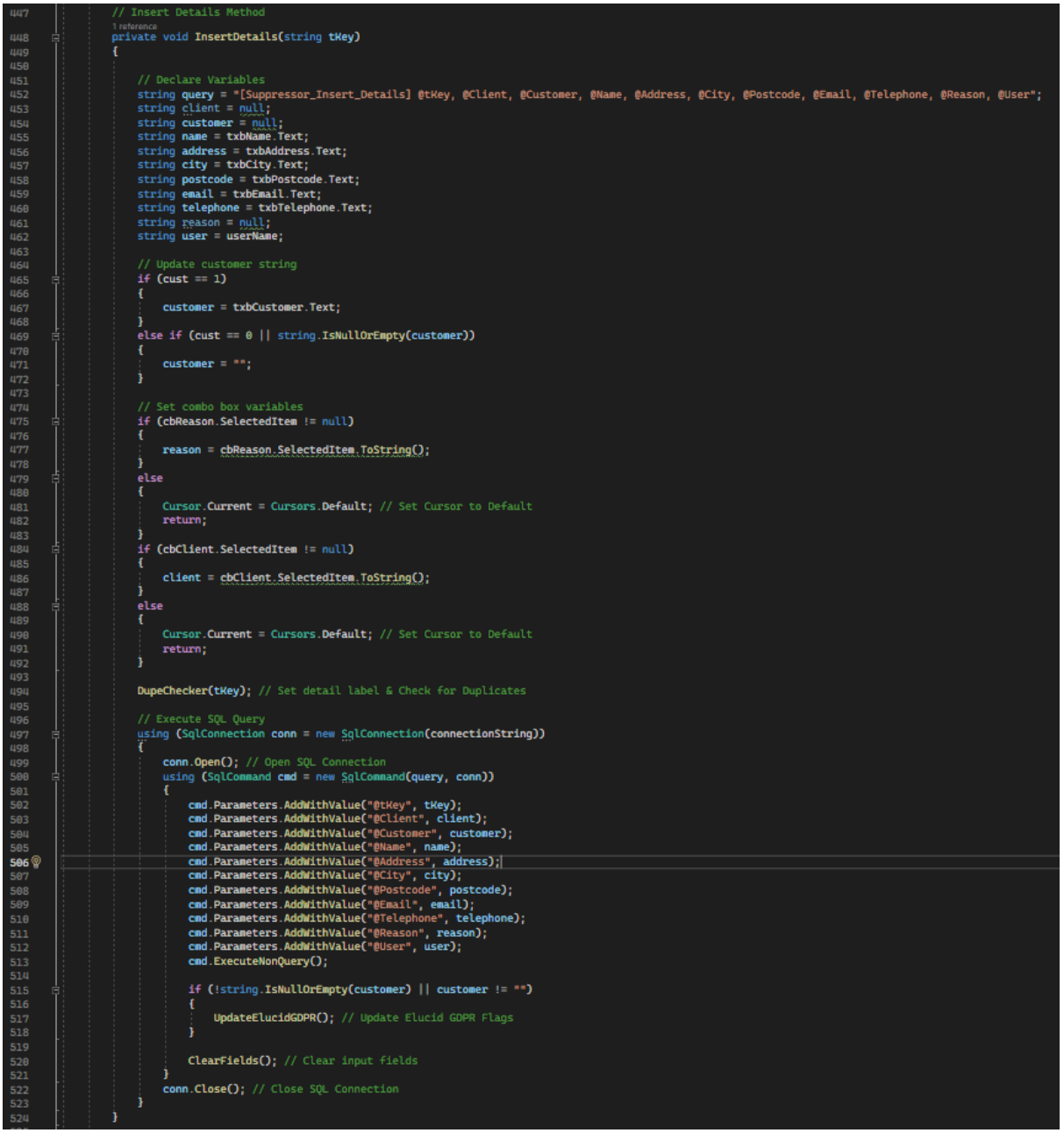
Main Form - GetNonCustomer



```
// Update Elucid GDPR Flags
0 references
private void GetNonCustomer()
{
    string getDatabase = "SELECT RTRIM([Server]) as [Server],RTRIM([Database]) as [Database], RTRIM(Site) as [Site] FROM Suppressor_Clients WHERE [Description] = @Client";
    string query = "SELECT COUNT(customer) FROM cust WHERE postcode = @Postcode AND level_1 = @Site";
    string client = null;
    string customer = txbCustomer.Text;
    string user = userName;
    string database = null;
    string server = null;
    string postcode = txbPostcode.Text;
    string site = null;

    client = cbClient.SelectedItem.ToString();

    try
    {
        // Get Database
        using (SqlConnection conn = new SqlConnection(connectionString))
        {
            conn.Open(); // Open SQL Connection

            using (SqlCommand cmd = new SqlCommand(getDatabase, conn))
            {
                cmd.Parameters.AddWithValue("@Client", client);
                using (SqlDataReader reader = cmd.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        server = reader["Server"].ToString();
                        database = reader["Database"].ToString();
                        site = reader["Site"].ToString();
                    }
                }
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}");
        LogBook("ERROR", "[MainFrom]", "[GetNonCustomer]", $"FAILED! Error retrieving database info: ({ex.Message})");
    }

    // Populate new connection string
    string updateConnectionString = $"Server={server};Database={database};Integrated Security=True;Encrypt=False;";

    try
    {
        // Update cust tables
        using (SqlConnection conn = new SqlConnection(updateConnectionString))
        {
            conn.Open();
            using (SqlCommand cmd = new SqlCommand(query, conn))
            {
                cmd.Parameters.AddWithValue("@Postcode", postcode);
                cmd.Parameters.AddWithValue("@Site", site);
                int count = (int)cmd.ExecuteScalar();

                if (count > 0 && string.IsNullOrEmpty(customer) && !string.IsNullOrEmpty(postcode))
                {
                    MessageBox.Show($"{count.ToString()} Customer(s) were found with this postcode. \nPlease check if this customer exists on elucid.");
                    return;
                }
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An error occurred: {ex.Message}");
        LogBook("ERROR", "[MainFrom]", "[GetNonCustomer]", $"FAILED! Error searching Elucid tables ({ex.Message})");
    }
}
```

Here is the method used to check if there are customers whose postcode matches the value inputted into the Postcode field. This method gathers SQL Server connection values for the client selected by the user. Using these values, a new connection string is formed, which is then utilised to check the database table "cust" for any customers who have the same postcode as the value entered into the Postcode field. If customers are found and the customer field is null or empty, a message box will appear, informing the user that customers have been found for this postcode. This prompts the user and asks them to check the Elucid system to see if these customers' details match the ones entered by the user.
(This method is not yet implemented and is pending client review to decide if it is required or not.)

Main Form - InsertDetails



```
// Insert Details Method
1 reference
private void InsertDetails(string tKey)
{

    // Declare Variables
    string query = "[Suppressor_Insert_Details] @tKey, @Client, @Customer, @Name, @Address, @City, @Postcode, @Email, @Telephone, @Reason, @User";
    string client = null;
    string customer = null;
    string name = txbName.Text;
    string address = txbAddress.Text;
    string city = txbCity.Text;
    string postcode = txbPostcode.Text;
    string email = txbEmail.Text;
    string telephone = txbTelephone.Text;
    string reason = null;
    string user = userName;

    // Update customer string
    if (cust == 1)
    {
        customer = txbCustomer.Text;
    }
    else if (cust == 0 || string.IsNullOrEmpty(customer))
    {
        customer = "";
    }

    // Set combo box variables
    if (cbReason.SelectedItem != null)
    {
        reason = cbReason.SelectedItem.ToString();
    }
    else
    {
        Cursor.Current = Cursors.Default; // Set Cursor to Default
        return;
    }
    if (cbClient.SelectedItem != null)
    {
        client = cbClient.SelectedItem.ToString();
    }
    else
    {
        Cursor.Current = Cursors.Default; // Set Cursor to Default
        return;
    }

    DupeChecker(tKey); // Set detail label & Check for Duplicates

    // Execute SQL Query
    using (SqlConnection conn = new SqlConnection(connectionString))
    {
        conn.Open(); // Open SQL Connection
        using (SqlCommand cmd = new SqlCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("@tKey", tKey);
            cmd.Parameters.AddWithValue("@Client", client);
            cmd.Parameters.AddWithValue("@Customer", customer);
            cmd.Parameters.AddWithValue("@Name", name);
            cmd.Parameters.AddWithValue("@Address", address);
            cmd.Parameters.AddWithValue("@City", city);
            cmd.Parameters.AddWithValue("@Postcode", postcode);
            cmd.Parameters.AddWithValue("@Email", email);
            cmd.Parameters.AddWithValue("@Telephone", telephone);
            cmd.Parameters.AddWithValue("@Reason", reason);
            cmd.Parameters.AddWithValue("@User", user);
            cmd.ExecuteNonQuery();

            if (!string.IsNullOrEmpty(customer) || customer != "")
            {
                UpdateElucidGDPR(); // Update Elucid GDPR Flags
            }

            ClearFields(); // Clear input fields
        }
        conn.Close(); // Close SQL Connection
    }
}
```

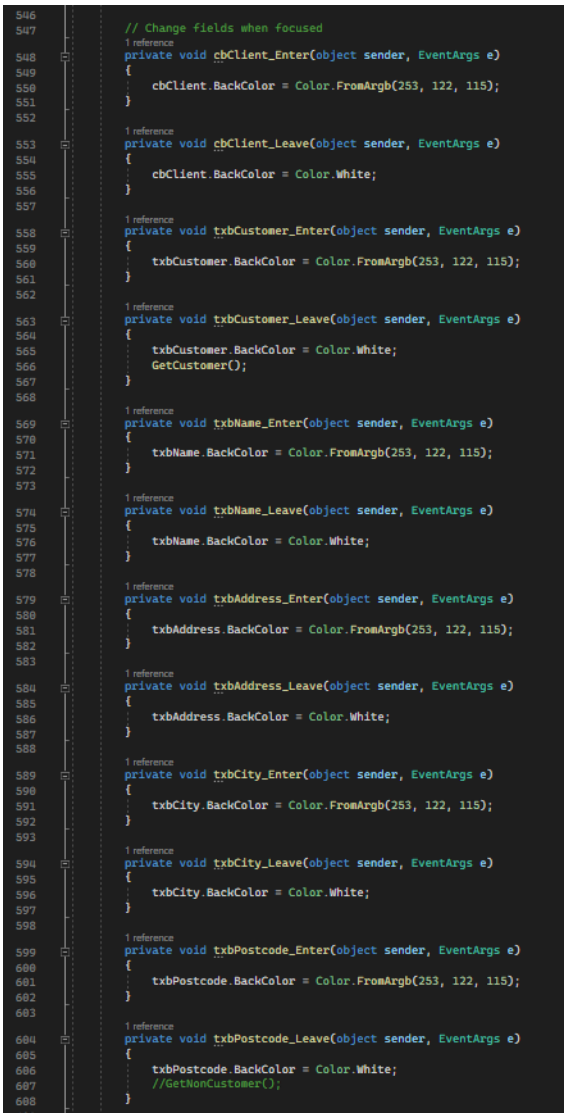
Here is the method used to insert the values entered on the Main Form into the Suppresser_Details database table. This method gathers the values inputted by the user and inserts them into the database table by executing the SQL stored procedure Suppressor_Insert_Details.

Main Form - ClearFields

```
526         // Clear field method
            3 references
527         private void ClearFields()
528         {
529             // Clear all fields
530             cbReason.Text = "";
531             cbReason.SelectedItem = null;
532             txbCustomer.Text = "";
533             txbName.Text = "";
534             txbAddress.Text = "";
535             txbCity.Text = "";
536             txbPostcode.Text = "";
537             txbEmail.Text = "";
538             txbTelephone.Text = "";
539         }
540
```

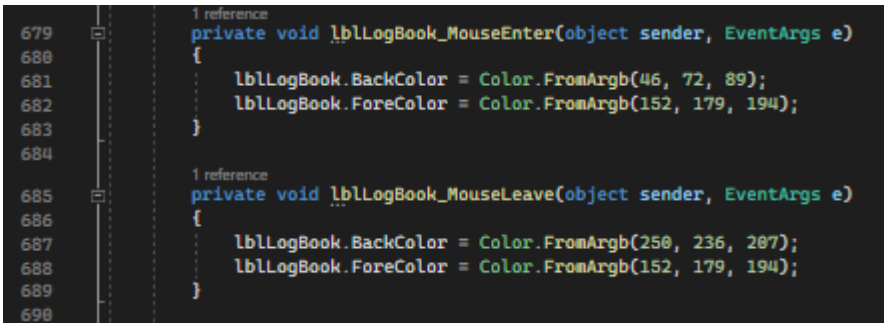
Here is the method used to clear the fields on the main form. This method iterates through all the fields and sets their values to either a blank string or null, effectively clearing the inputted data.

Main Form - Environment Events



```
546
547         // Change fields when focused
            1 reference
548         private void cbClient_Enter(object sender, EventArgs e)
549         {
550             cbClient.BackColor = Color.FromArgb(253, 122, 115);
551         }
552
            1 reference
553         private void cbClient_Leave(object sender, EventArgs e)
554         {
555             cbClient.BackColor = Color.White;
556         }
557
            1 reference
558         private void txbCustomer_Enter(object sender, EventArgs e)
559         {
560             txbCustomer.BackColor = Color.FromArgb(253, 122, 115);
561         }
562
            1 reference
563         private void txbCustomer_Leave(object sender, EventArgs e)
564         {
565             txbCustomer.BackColor = Color.White;
566             GetCustomer();
567         }
568
            1 reference
569         private void txbName_Enter(object sender, EventArgs e)
570         {
571             txbName.BackColor = Color.FromArgb(253, 122, 115);
572         }
573
            1 reference
574         private void txbName_Leave(object sender, EventArgs e)
575         {
576             txbName.BackColor = Color.White;
577         }
578
            1 reference
579         private void txbAddress_Enter(object sender, EventArgs e)
580         {
581             txbAddress.BackColor = Color.FromArgb(253, 122, 115);
582         }
583
            1 reference
584         private void txbAddress_Leave(object sender, EventArgs e)
585         {
586             txbAddress.BackColor = Color.White;
587         }
588
            1 reference
589         private void txbCity_Enter(object sender, EventArgs e)
590         {
591             txbCity.BackColor = Color.FromArgb(253, 122, 115);
592         }
593
            1 reference
594         private void txbCity_Leave(object sender, EventArgs e)
595         {
596             txbCity.BackColor = Color.White;
597         }
598
            1 reference
599         private void txbPostcode_Enter(object sender, EventArgs e)
600         {
601             txbPostcode.BackColor = Color.FromArgb(253, 122, 115);
602         }
603
            1 reference
604         private void txbPostcode_Leave(object sender, EventArgs e)
605         {
606             txbPostcode.BackColor = Color.White;
607             //GetNonCustomer();
608         }
```

```
610         1 reference
            private void txbEmail_Enter(object sender, EventArgs e)
611         {
612             txbEmail.BackColor = Color.FromArgb(253, 122, 115);
613         }
614
            1 reference
615         private void txbEmail_Leave(object sender, EventArgs e)
616         {
617             txbEmail.BackColor = Color.White;
618         }
619
            1 reference
620         private void txbTelephone_Enter(object sender, EventArgs e)
621         {
622             txbTelephone.BackColor = Color.FromArgb(253, 122, 115);
623             // txbTelephone.BackColor = Color.FromArgb(255, 91, 83); Darker shade of red
624         }
625
            1 reference
626         private void txbTelephone_Leave(object sender, EventArgs e)
627         {
628             txbTelephone.BackColor = Color.White;
629         }
630
            1 reference
631         private void cbReason_Enter(object sender, EventArgs e)
632         {
633             cbReason.BackColor = Color.FromArgb(253, 122, 115);
634         }
635
            1 reference
636         private void cbReason_Leave(object sender, EventArgs e)
637         {
638             cbReason.BackColor = Color.White;
639             txbName.BackColor = Color.White;
640         }
641
642         // Change Button Colors
            1 reference
643         private void btnAccept_MouseEnter(object sender, EventArgs e)
644         {
645             btnAccept.BackColor = Color.FromArgb(152, 179, 194);
646             btnAccept.ForeColor = Color.FromArgb(46, 72, 89);
647         }
648
            1 reference
649         private void btnAccept_MouseLeave(object sender, EventArgs e)
650         {
651             btnAccept.BackColor = Color.FromArgb(46, 72, 89);
652             btnAccept.ForeColor = Color.FromArgb(152, 179, 194);
653         }
654
            1 reference
655         private void btnClear_MouseEnter(object sender, EventArgs e)
656         {
657             btnClear.BackColor = Color.FromArgb(152, 179, 194);
658             btnClear.ForeColor = Color.FromArgb(46, 72, 89);
659         }
660
            1 reference
661         private void btnClear_MouseLeave(object sender, EventArgs e)
662         {
663             btnClear.BackColor = Color.FromArgb(46, 72, 89);
664             btnClear.ForeColor = Color.FromArgb(152, 179, 194);
665         }
666
            1 reference
667         private void btnExit_MouseEnter(object sender, EventArgs e)
668         {
669             btnExit.BackColor = Color.FromArgb(152, 179, 194);
670             btnExit.ForeColor = Color.FromArgb(46, 72, 89);
671         }
672
            1 reference
673         private void btnExit_MouseLeave(object sender, EventArgs e)
674         {
675             btnExit.BackColor = Color.FromArgb(46, 72, 89);
676             btnExit.ForeColor = Color.FromArgb(152, 179, 194);
677         }
```



```
            1 reference
679         private void lblLogBook_MouseEnter(object sender, EventArgs e)
680         {
681             lblLogBook.BackColor = Color.FromArgb(46, 72, 89);
682             lblLogBook.ForeColor = Color.FromArgb(152, 179, 194);
683         }
684
            1 reference
685         private void lblLogBook_MouseLeave(object sender, EventArgs e)
686         {
687             lblLogBook.BackColor = Color.FromArgb(250, 236, 207);
688             lblLogBook.ForeColor = Color.FromArgb(152, 179, 194);
689         }
690
```

Here, we can see multiple methods that are invoked when UI elements are interacted with. These methods, when triggered, change the appearance of the UI elements to provide visual feedback to the user. This has been implemented to enhance the user experience and make the application more engaging.

Main Form - txbCustomer_TextChanged

```
691         // Set Character Limit for Customer
            1 reference
692         private void txbCustomer_TextChanged(object sender, EventArgs e)
693         {
694             string customerString = txbCustomer.Text;
695             int customerLength = customerString.Length;
696
697             if (customerLength > 12)
698             {
699                 MessageBox.Show("Customer is too long.");
700                 txbCustomer.Text = customerString.Substring(0, Math.Min(customerString.Length, 12));
701                 txbCustomer.SelectAll();
702                 return;
703             }
704         }
705
```

Here is the method that validates the length of the string being entered into ***txbCustomer***. Each time the text value in the textbox changes, this method is invoked to count the current length of the string. If the string reaches its limit, a message box is triggered, and the last character entry is undone.

Main Form - txbName_TextChanged

```
706         // Set Character Limit for Name
            1 reference
707         private void txbName_TextChanged(object sender, EventArgs e)
708         {
709             string nameString = txbName.Text;
710             int nameLength = nameString.Length;
711
712             if (nameLength > 255)
713             {
714                 MessageBox.Show("Name is too long.");
715                 txbName.Text = nameString.Substring(0, Math.Min(nameString.Length, 255));
716                 txbName.SelectAll();
717                 return;
718             }
719         }
```

Here is the method that validates the length of the string being entered into ***txbName***. Each time the text value in the textbox changes, this method is invoked to count the current length of the string. If the string reaches its limit, a message box is triggered, and the last character entry is undone.

Main Form - txbAddress_TextChanged

```
721         // Set Character Limit for Address
            1 reference
722         private void txbAddress_TextChanged(object sender, EventArgs e)
723         {
724             string addressString = txbAddress.Text;
725             int addressLength = addressString.Length;
726 
727             if (addressLength > 255)
728             {
729                 MessageBox.Show("Address is too long.");
730                 txbAddress.Text = addressString.Substring(0, Math.Min(addressString.Length, 255));
731                 txbAddress.SelectAll();
732                 return;
733             }
734         }
```

Here is the method that validates the length of the string being entered into ***txbAddress***. Each time the text value in the textbox changes, this method is invoked to count the current length of the string. If the string reaches its limit, a message box is triggered, and the last character entry is undone.

Main Form - txbCity_TextChanged

```
736         // Set Character Limit for City
            1 reference
737         private void txbCity_TextChanged(object sender, EventArgs e)
738         {
739             string cityString = txbCity.Text;
740             int cityLength = cityString.Length;
741 
742             if (cityLength > 48)
743             {
744                 MessageBox.Show("City is too long.");
745                 txbCity.Text = cityString.Substring(0, Math.Min(cityString.Length, 48));
746                 txbCity.SelectAll();
747                 return;
748             }
749         }
```

Here is the method that validates the length of the string being entered into ***txbCity***. Each time the text value in the textbox changes, this method is invoked to count the current length of the string. If the string reaches its limit, a message box is triggered, and the last character entry is undone.

Main Form - txbPostcode_TextChanged

```
751         // Set Character Limit for Postcode
            1 reference
752         private void txbPostcode_TextChanged(object sender, EventArgs e)
753         {
754             string postcodeString = txbPostcode.Text;
755             int postcodeLength = postcodeString.Length;
756
757             if (postcodeLength > 10)
758             {
759                 MessageBox.Show("Postcode is too long.");
760                 txbPostcode.Text = postcodeString.Substring(0, Math.Min(postcodeString.Length, 10));
761                 txbPostcode.SelectAll();
762                 return;
763             }
764         }
```

Here is the method that validates the length of the string being entered into ***txbPostcode***. Each time the text value in the textbox changes, this method is invoked to count the current length of the string. If the string reaches its limit, a message box is triggered, and the last character entry is undone.

Main Form - txbEmail_TextChanged

```
766         // Set Character Limit for Email
            1 reference
767         private void txbEmail_TextChanged(object sender, EventArgs e)
768         {
769             string emailString = txbEmail.Text;
770             int emailLength = emailString.Length;
771
772             if (emailLength > 48)
773             {
774                 MessageBox.Show("Email is too long.");
775                 txbEmail.Text = emailString.Substring(0, Math.Min(emailString.Length, 48));
776                 txbEmail.SelectAll();
777                 return;
778             }
779         }
```

Here is the method that validates the length of the string being entered into ***txbEmail***. Each time the text value in the textbox changes, this method is invoked to count the current length of the string. If the string reaches its limit, a message box is triggered, and the last character entry is undone.

Main Form - txbTelephone_TextChanged

```
781         // Set Character Limit for Telephone
            1 reference
782         private void txbTelephone_TextChanged(object sender, EventArgs e)
783         {
784             string telephoneString = txbTelephone.Text;
785             int telephoneLength = telephoneString.Length;
786
787             if (telephoneLength > 14)
788             {
789                 MessageBox.Show("Telephone is too long.");
790                 txbTelephone.Text = telephoneString.Substring(0, Math.Min(telephoneString.Length, 14));
791                 txbTelephone.SelectAll();
792                 return;
793             }
794         }
```

Here is the method that validates the length of the string being entered into ***txbTelephone***. Each time the text value in the textbox changes, this method is invoked to count the current length of the string. If the string reaches its limit, a message box is triggered, and the last character entry is undone.

Main Form - cbClient_TextChanged

```
            1 reference
796         private void cbClient_TextChanged(object sender, EventArgs e)
797         {
798             ClearFields();
799         }
```

Here is the method that is invoked whenever the text changes in the Client field. This method invokes the ClearFields method, ensuring that the form is cleared each time a user changes the client.

Main Form - btnExit_Click

```
805         // Exit Button Event
            2 references
806         private void btnExit_Click(object sender, EventArgs e)
807         {
808             DialogResult result = MessageBox.Show("Are you sure you want to Exit the application?", "Confirmation", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
809             if (result == DialogResult.Yes)
810             {
811                 LogBook("", "[MainForm]", "[FormClosing]", "Application Closed");
812                 Application.Exit();
813             }
814         }
```

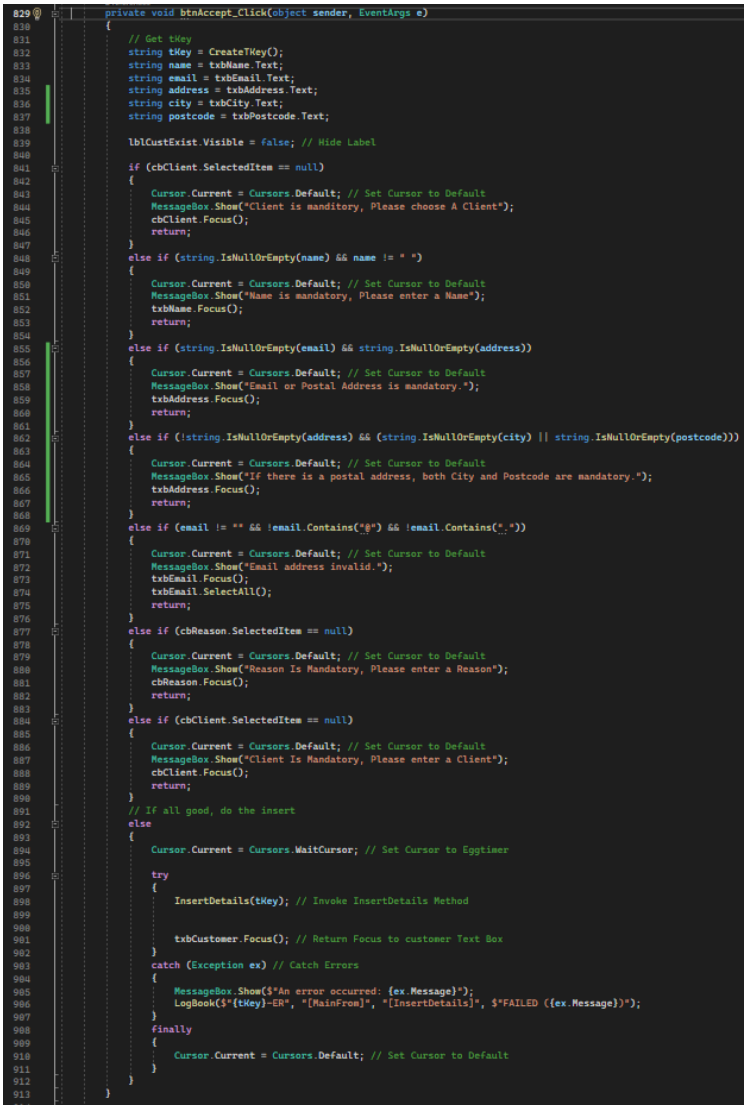
Here is the method invoked when the user clicks the exit button. This method triggers a message box that asks the user if they are sure they want to leave the application. If the user clicks "Yes," the application is closed; if the user clicks "No," the operation is cancelled.

Main Form - btnClear_Click

```
816         // Clear Button Event
            2 references
817         private void btnClear_Click(object sender, EventArgs e)
818         {
819             // Clear all fields
820             txbName.Text = "";
821             ClearFields();
822             cbClient.Focus();
823             cbClient.Text = "";
824             cbClient.SelectedItem = null;
825             lblCustExist.Visible = false;
826         }
```

Here is the method invoked when the user clicks the Clear button. This method invokes the ClearFields method and carries out other processes to reset the form, allowing the user to start data entry again.

Main Form - btnAccept_Click



```
829     private void btnAccept_Click(object sender, EventArgs e)
830     {
831         // Get tKey
832         string tKey = CreateTKey();
833         string name = txbName.Text;
834         string email = txbEmail.Text;
835         string address = txbAddress.Text;
836         string city = txbCity.Text;
837         string postcode = txbPostcode.Text;
838
839         lblCustExist.Visible = false; // Hide Label
840
841         if (cbClient.SelectedItem == null)
842         {
843             Cursor.Current = Cursors.Default; // Set Cursor to Default
844             MessageBox.Show("Client is manditory, Please choose A Client");
845             cbClient.Focus();
846             return;
847         }
848         else if (string.IsNullOrEmpty(name) && name != " ")
849         {
850             Cursor.Current = Cursors.Default; // Set Cursor to Default
851             MessageBox.Show("Name is mandatory, Please enter a Name");
852             txbName.Focus();
853             return;
854         }
855         else if (string.IsNullOrEmpty(email) && string.IsNullOrEmpty(address))
856         {
857             Cursor.Current = Cursors.Default; // Set Cursor to Default
858             MessageBox.Show("Email or Postal Address is mandatory.");
859             txbAddress.Focus();
860             return;
861         }
862         else if (!string.IsNullOrEmpty(address) && (string.IsNullOrEmpty(city) || string.IsNullOrEmpty(postcode)))
863         {
864             Cursor.Current = Cursors.Default; // Set Cursor to Default
865             MessageBox.Show("If there is a postal address, both City and Postcode are mandatory.");
866             txbAddress.Focus();
867             return;
868         }
869         else if (email != "" && !email.Contains("@") && !email.Contains("."))
870         {
871             Cursor.Current = Cursors.Default; // Set Cursor to Default
872             MessageBox.Show("Email address invalid.");
873             txbEmail.Focus();
874             txbEmail.SelectAll();
875             return;
876         }
877         else if (cbReason.SelectedItem == null)
878         {
879             Cursor.Current = Cursors.Default; // Set Cursor to Default
880             MessageBox.Show("Reason Is Mandatory, Please enter a Reason");
881             cbReason.Focus();
882             return;
883         }
884         else if (cbClient.SelectedItem == null)
885         {
886             Cursor.Current = Cursors.Default; // Set Cursor to Default
887             MessageBox.Show("Client Is Mandatory, Please enter a Client");
888             cbClient.Focus();
889             return;
890         }
891         // If all good, do the insert
892         else
893         {
894             Cursor.Current = Cursors.WaitCursor; // Set Cursor to Eggtimer
895
896             try
897             {
898                 InsertDetails(tKey); // Invoke InsertDetails Method
899
900
901                 txbCustomer.Focus(); // Return Focus to customer Text Box
902             }
903             catch (Exception ex) // Catch Errors
904             {
905                 MessageBox.Show($"An error occurred: {ex.Message}");
906                 LogBook($"{tKey}-ER", "[MainFrom]", "[InsertDetails]", $"FAILED ({ex.Message})");
907             }
908             finally
909             {
910                 Cursor.Current = Cursors.Default; // Set Cursor to Default
911             }
912         }
913     }
```

Here is the method invoked when the user clicks the Accept Button. This method begins by invoking the CreateTKey method to obtain a unique tKey. Afterward, all mandatory fields are checked to ensure they have valid values and are not null or empty. If the values are valid, the method then invokes the InsertDetails method, followed by a function to return focus to the customer field.

Main Form - lblLogBook_Click

```
906     // LogBook Lable Click Event
        1 reference
907     private void lblLogBook_Click(object sender, EventArgs e)
908     {
909         string Message = lblLogBook.Text;
910         MessageBox.Show($"LogBook Activity Recorded For '{Environment.MachineName}.{Environment.UserName}': \n\n{Message}", "Logbook", MessageBoxButtons.OK, MessageBoxIcon.Information);
911     }
```

Here is the method invoked when the user clicks on the Logbook label. This method triggers a message box to appear with the text displaying the last LogBook entry.

Main Form - MainFrom_KeyDown

```
917     // Keyboard Shortcuts
        1 reference
918     private void MainFrom_KeyDown(object sender, KeyEventArgs e)
919     {
920         // ctrl + G
921         if (e.Control && e.KeyCode == Keys.G)
922         {
923             txbName.Text = "";
924             btnClear_Click(sender, e);
925         }
926
927         // ctrl + S
928         if (e.Control && e.KeyCode == Keys.S)
929         {
930             btnAccept_Click(sender, e);
931         }
932
933         // Esc
934         if (e.KeyCode == Keys.Escape)
935         {
936             btnExit_Click(sender, e);
937         }
938     }
```

Here is the method invoked when the user presses any key while the form is open. The method checks to see if the keys pressed are either "Ctrl" and "G", "Ctrl" and "S", or "Esc" and then completes a process depending on which was pressed. If the keys pressed are "Ctrl" and "G", then the name field is set to be an empty string, and the **btnClear_Click** method is invoked, essentially clearing the form. If the user pressed "Ctrl" and "S", the **btnAccept_Click** method is invoked. If the user pressed "Esc", then the **btnExit_Click** method is invoked.

Main Form - txbTelephone_KeyPress

```
940         // Check Values in Telephone Text Box
            1 reference
941         private void txbTelephone_KeyPress(object sender, KeyPressEventArgs e)
942         {
943             if (!char.IsDigit(e.KeyChar) && !char.IsControl(e.KeyChar) && e.KeyChar != '+')
944             {
945                 MessageBox.Show($"Character: '{e.KeyChar}' Not Accepted Here. \nOnly numerical values accepted for Telephone.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
946                 e.Handled = true;
947             }
948         }
```

Here is the method invoked when the user presses any key while focused on the Telephone field. This method checks the characters being entered to ensure they are either numbers or the '+' symbol. If the character entered is not one of these, a message box is triggered, providing an error to the user, and the key press is cancelled.

Main Form - txbEmail_KeyPress

```
            1 reference
949         private void txbEmail_KeyPress(object sender, KeyPressEventArgs e)
950         {
951             if (e.KeyChar == ' ' || e.KeyChar == '"' || e.KeyChar == '\'' || e.KeyChar == '\n' || e.KeyChar == '\t')
952             {
953                 MessageBox.Show($"Character: '{e.KeyChar}' Not Accepted Here.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
954                 e.Handled = true;
955             }
956         }
```

Here we can see the method invoked when the user presses any key while focused in the Email field. This method checks the characters being entered to make sure they are valid characters for an email address. If the character entered is not valid, a message box is triggered giving the user an error and the key press is cancelled.

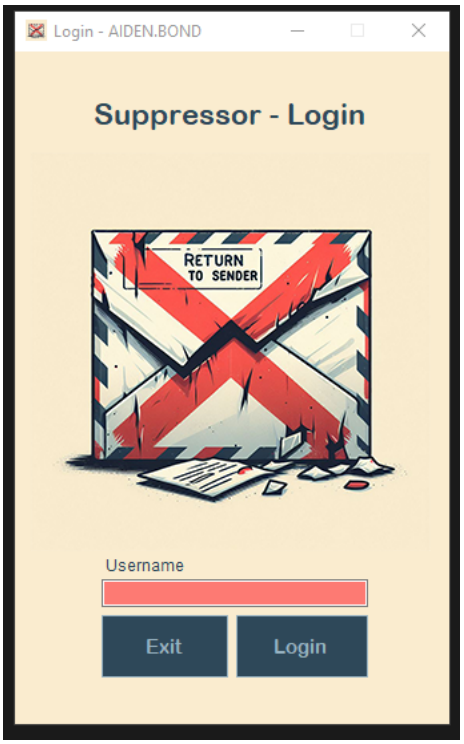
Main Form - txbCustomer_KeyPress

```
            1 reference
958         private void txbCustomer_KeyPress(object sender, KeyPressEventArgs e)
959         {
960             if (!char.IsLetterOrDigit(e.KeyChar) && !char.IsControl(e.KeyChar) && e.KeyChar != '-')
961             {
962                 MessageBox.Show($"Character: '{e.KeyChar}' Not Accepted Here", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
963                 e.Handled = true;
964             }
965         }
966     }
967 }
```

Here is the method invoked when the user presses any key while focused on the Email field. This method checks the characters being entered to ensure they are valid characters for an email address. If the character entered is not valid, a message box is triggered, providing the user with an error, and the key press is cancelled.

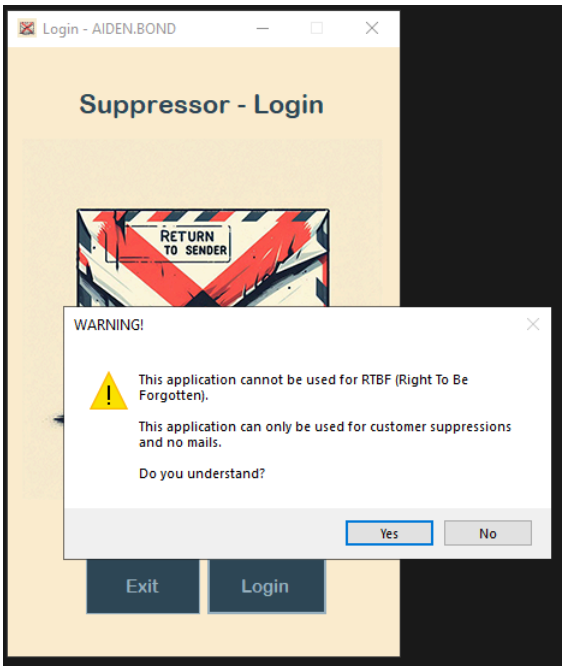
5.0.0 - Demo & Testing

5.0.1 - Logging Onto the Application

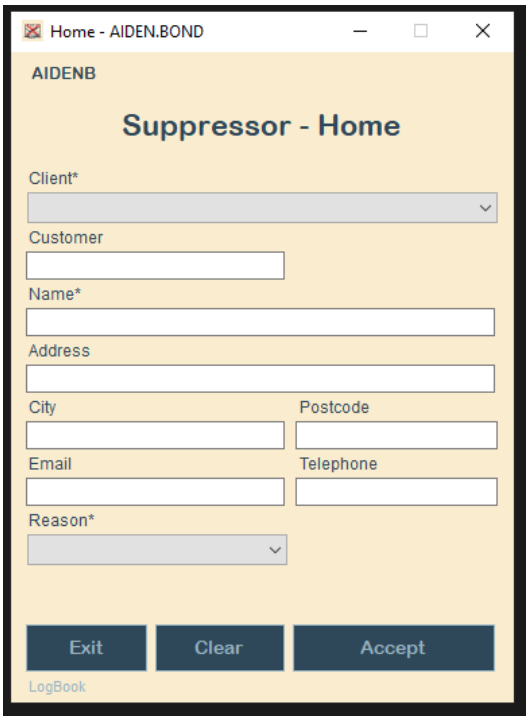


When the user first opens the application, they are greeted with the login form where they can enter a username.

Once a valid username has been entered by the user, they can press "Login" to log into the application.

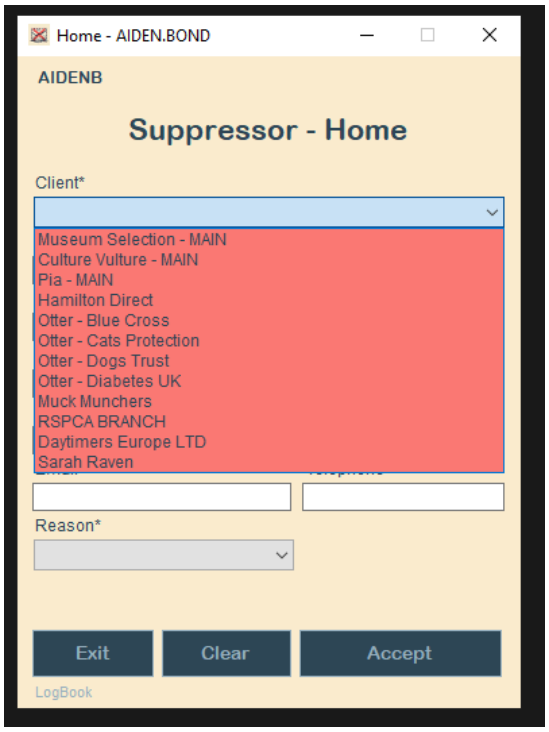


When logging in, the user is presented with a popup advising them that this application is only to be used for customer suppression and cannot be used for RTBF (Right to be forgotten). If the user does not click "Yes," the application will not load the main form.



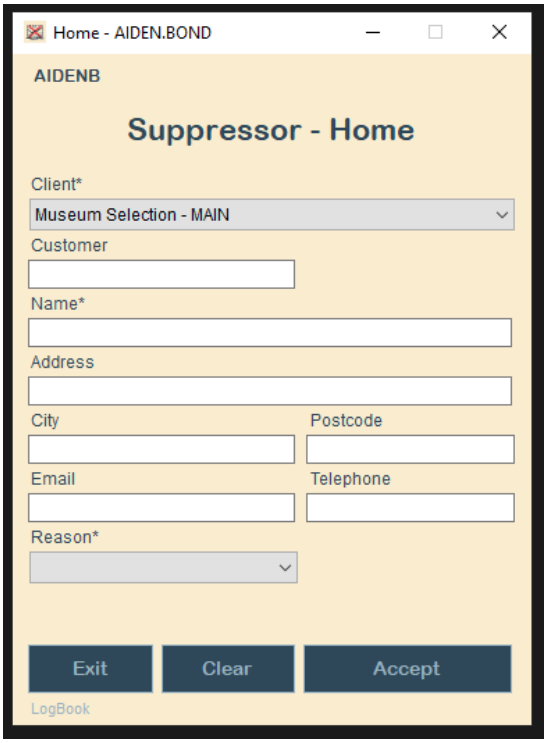
When the user logs in they are met with the main form why they can start data entry

5.0.2 - Selecting a Client

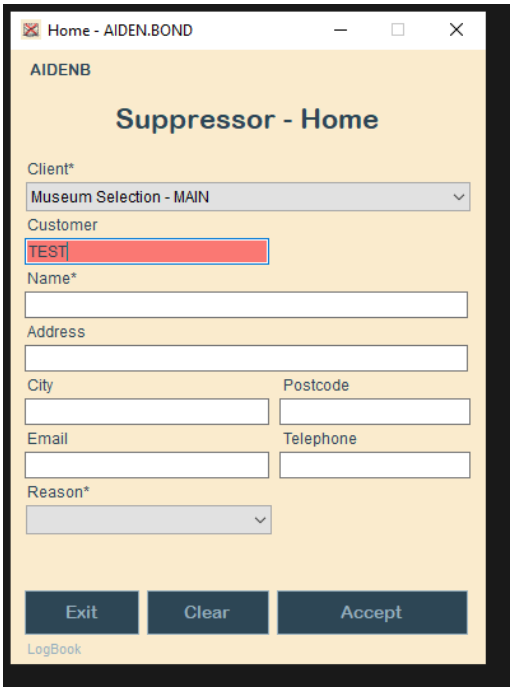


The first field the user needs to complete is the client field, this is a dropdown menu, populated from the data stored in the Suppressor_Clients database table.

Once selected we can see that the option selected is displayed in the field.

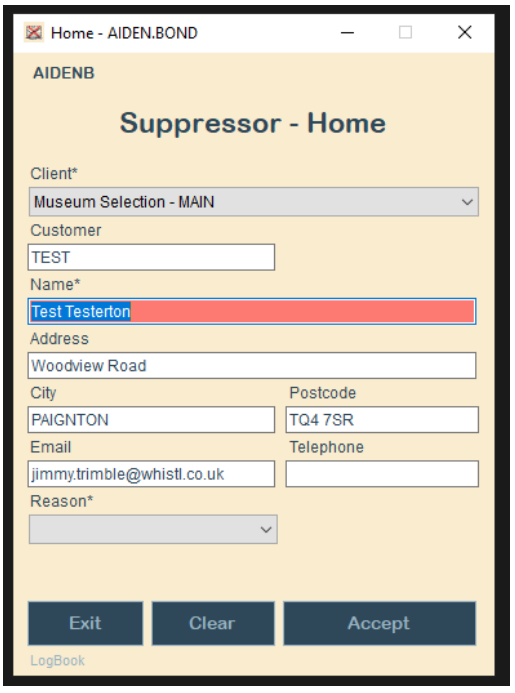


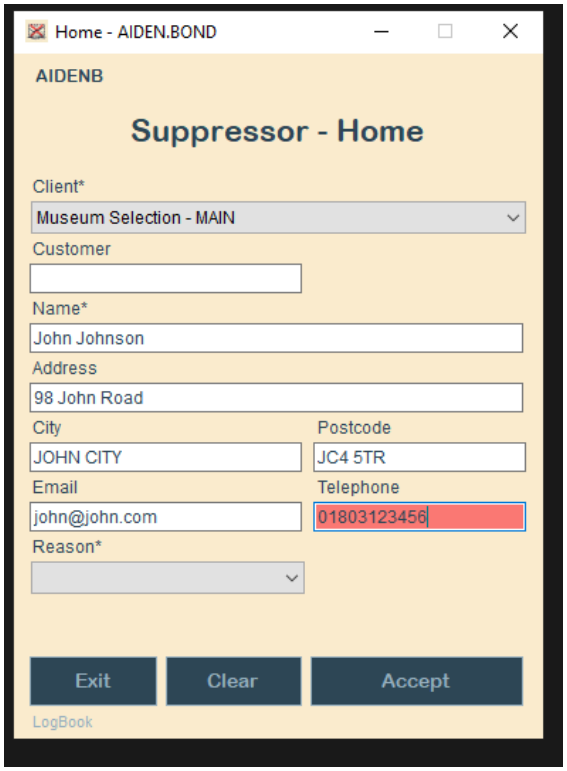
5.0.3 - Completing the required Fields



The second field on the form is the customer field, this field is not mandatory but will error if a customer number is entered which does not exist in the database for the selected client.

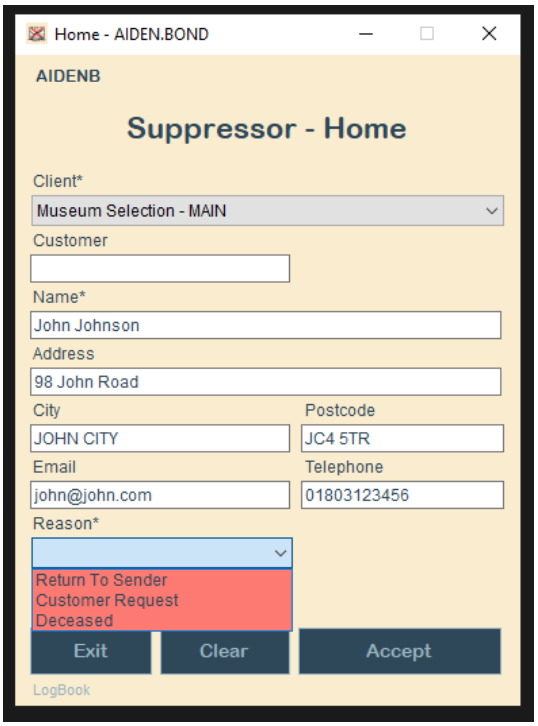
If the customer does exist in the client's database, the application will get that customer's data to pre-populate the rest of the fields, providing the data is available.





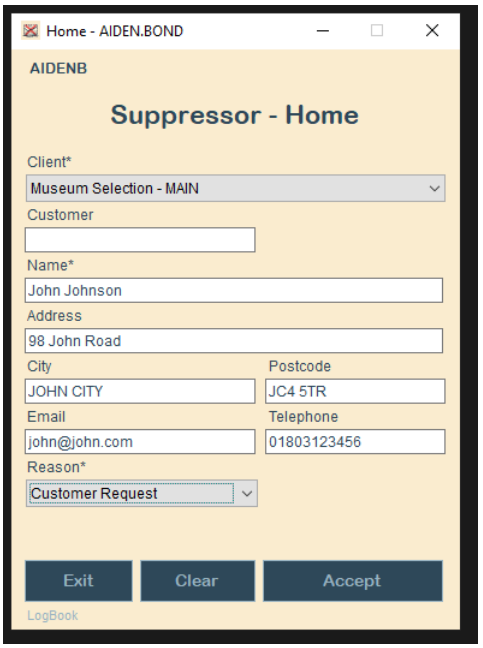
If the user does not have a customer number to enter, they can leave the field blank and manually enter the customer's details into the relative fields. The mandatory fields are Name and either Email address or postal address.

5.0.4 - Selecting a reason

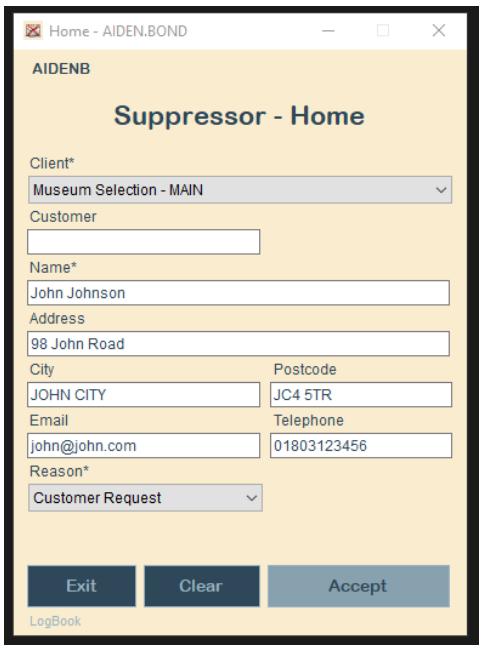


Once the customer's details have been inputted into the relative fields, the last field required is the Reason field. This is a drop down menu populated from the Suppressor_Reasons database table.

Once selected we can see that the option selected is displayed in the field.

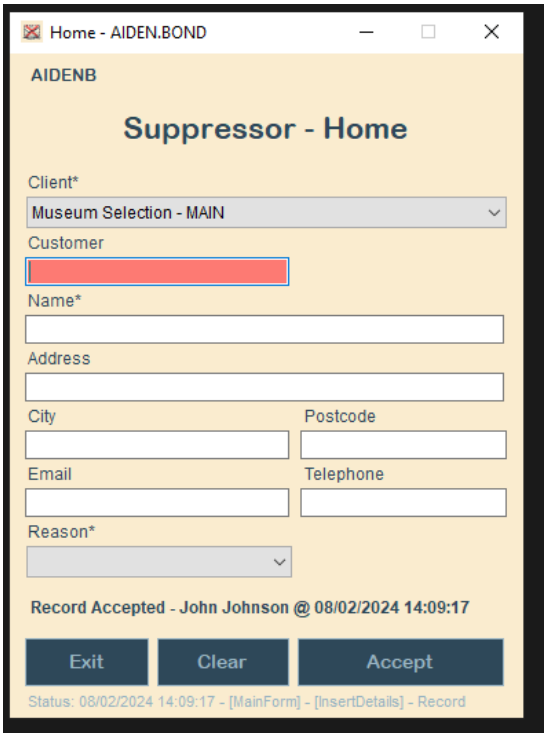


5.0.5 - Accepting a record

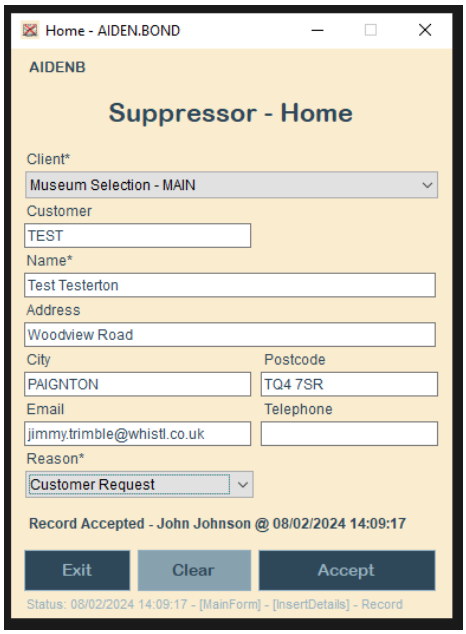


When all the required fields are complete the user can click the accept button to accept the record.

Once clicked, the application will insert the details into the Suppressor_details database table and clear all fields except for the client. This is done so the user can continue with their data entry. There is visual feedback in the form of a label above the buttons, showing the user that they have just accepted a record. The label displays information about when it was accepted and the customer's name. Additionally, there is a status label at the bottom of the form that mirrors the latest LogBook entry.

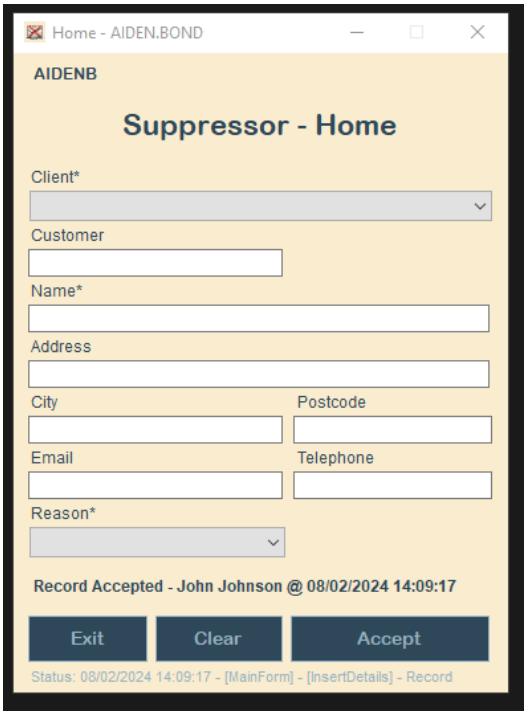


5.0.6 - Clearing the form

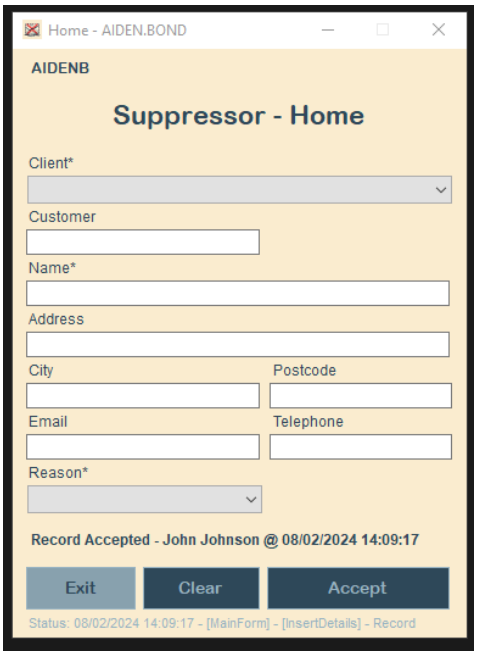


If the user wishes to clear the screen they can do so by clicking the clear button at the button of the form.

By clicking this all fields are cleared, but the notification label and the LogBook label are not. These UI elements are not included in the clear process so that if the user needs to check the last entry they did in order to prevent rentries being missed or duplicated they are able to do so without running any reports.

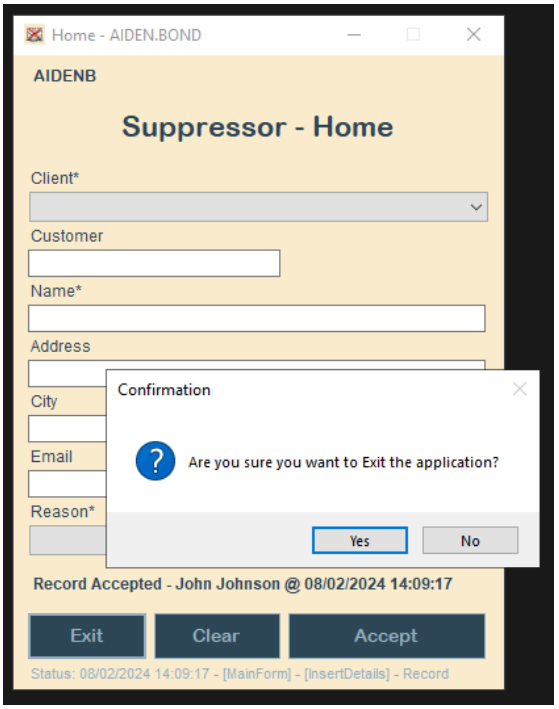


5.0.7 - Exiting the Application.



If the user wishes to exit the application they can do so by clicking the exit button at the bottom of the form.

Once clicked, a message box will appear asking the user if they are sure they want to exit the application. If the user clicks “Yes”, the application will close, if the user clicks “No” the operation will be cancelled.



5.0.8 - Post Demo Review

SQLQuery12.sql -...E\AIDEN.BOND (65))* | SQLQuery11.sql -...E\AIDEN.BOND (69))* | SQLQuery6.sql - S...\AIDEN.BOND (177))*

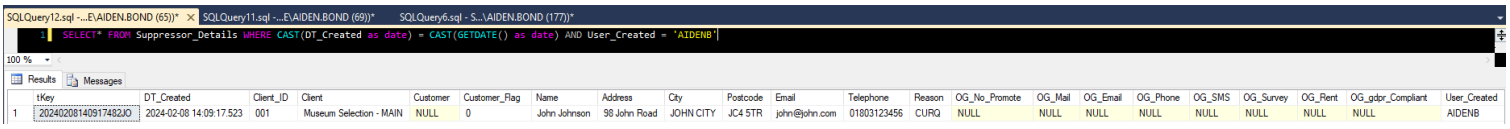
```
SELECT* FROM Suppressor_LogBook WHERE CAST(DT_Created as date) = CAST(GETDATE() as date) AND User_Created = 'AIDENB'
```

100 %

Results | Messages

	tKey	Domain_User	User_Created	Form	Event	Notes	DT_Created
1	20240208140516602AI	VDIS.AIDEN.BOND	AIDENB	[MainFrom]	[FormLoad]	Application Started	2024-02-08 14:05:03.520
2	20240208140917482JO	VDIS.AIDEN.BOND	AIDENB	[MainForm]	[InsertDetails]	Record Accepted - (Museum Selection - MAIN - Jo...	2024-02-08 14:09:17.520
3	20240208141552153AI	VDIS.AIDEN.BOND	AIDENB	[MainForm]	[FormClosing]	Application Closed	2024-02-08 14:15:52.147

After the application has been used, we can examine the SQL database tables to observe the results. In the Suppressor_LogBook database table, we can see the data inserted. A record has been entered for the suppressed customer, and there are also records for when the application has been closed and opened.



	tKey	DT_Created	Client_ID	Client	Customer	Customer_Flag	Name	Address	City	Postcode	Email	Telephone	Reason	OG_No_Promote	OG_Mail	OG_Email	OG_Phone	OG_SMS	OG_Survey	OG_Rent	OG_gdpr_Compliant	User_Created
1	20240208140917482JO	2024-02-08 14:09:17.523	001	Museum Selection - MAIN	NULL	0	John Johnson	98 John Road	JOHN CITY	JC4 5TR	john@john.com	01803123456	CURQ	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	AIDENB

Here we can see that a record has been entered into the Suppressor_Details database table. This record has the same tKey as the one for the LogBook record, allowing the logbook entries to be matched back to the entries in the details table.

	tKey	Domain_User	User_Created	Form	Event	Notes	DT_Created
1	202401181126121O1AI	VDIS.AIDEN.BOND	AIDENB	[MainFrom]	[FormLoad]	Application Started	2024-01-18 11:26:12.240
2	20240118112617558TE	VDIS.AIDEN.BOND	AIDENB	[MainForm]	[InsertDetails]	Record Accepted - (Museum Selection - MAIN - Te...	2024-01-18 11:26:17.560
3	20240118112617643AI	VDIS.AIDEN.BOND	AIDENB	[MainForm]	[UpdateElucidGDPR]	Started	2024-01-18 11:26:17.623
4	20240118112617658AI	VDIS.AIDEN.BOND	AIDENB	[MainForm]	[UpdateElucidGDPR]	Update Cust Executed (TEST)	2024-01-18 11:26:17.640
5	20240118112617660AI	VDIS.AIDEN.BOND	AIDENB	[MainForm]	[UpdateElucidGDPR]	Update GDPR Executed (TEST)	2024-01-18 11:26:17.640

Here we can see LogBook entries for a customer that exists in the Elucid database. These entries indicate that the Elucid GDPR and customer tables have been updated. These events are recorded to show that this part of the database has been amended.

	tKey	DT_Created	Client_ID	Client	Customer	Customer_Flag	Name	Address	City	Postcode	Email	Telephone	Reason	OG_No_Promote	OG_Mail	OG_Email	OG_Phone	OG_SMS	OG_Survey	OG_Rent	OG_gdpr_Compliant
1	20240124115946667TE	2024-01-24 11:59:46.703	001	Museum Selection - MAIN	TEST	1	Test Testerton	Woodview Road	PAIGNTON	TQ4 7SR	jimmy.trimble@whistl.co.uk	NULL	RTS	1	0	0	0	0	0	0	1

Here we can see a record in the Suppressor_Details table where the customer does exist in the Elucid database. This record stores the previous GDPR flags for the customer so that if the suppression needs to be reverted or if historic GDPR flags need to be reported on, it is possible.

6.0.0 - Summary

All criteria established during pre-development planning have been successfully achieved. The application allows users to log in, record customer details for those opting to be suppressed from mailing lists, and gathers data for subsequent transmission to mailing houses, enabling them to halt mailings to the respective customers.

Emphasising robustness and stability, the application incorporates multiple error-checking processes and exception handlers. Its design prioritises user-friendliness and intuitiveness, aiming for an engaging experience. The chosen colour scheme, decided upon during development, adds a unique and visually pleasing aspect. Combined with the user-friendly layout, these features ensure the application meets the client's needs and preferences.