



# **Suppressor Application Technical Report**

Aiden Bond - Whistl Fulfilment South West  
Post-Development Version 2 - 09/02/2024

# Contents

Section	Title	Page
1.0.0	<b>Introduction</b>	3
2.0.0	<b>Pre-Development Design &amp; Planning</b>	4
2.1.0	<b>Application &amp; Usage</b>	4
2.2.0	<b>Forms</b>	5
2.2.1	<b>Login Form - Design</b>	5
2.2.2	<b>Login Form - Error Handling &amp; Dialog Boxes</b>	6
2.2.3	<b>Login Form - UI Elements</b>	6
2.2.4	<b>Main Form - Design</b>	7
2.2.5	<b>Main Form - Error Handling &amp; Dialog Boxes</b>	8
2.2.6	<b>Main Form - UI Elements</b>	9
3.0.0	<b>SQL Database Construction</b>	10
3.1.0	<b>Tables</b>	10
3.1.1	<b>Suppressor_Details</b>	10
3.1.2	<b>Suppressor_Reasons</b>	11
3.1.3	<b>Suppressor_Clients</b>	11
3.1.4	<b>Suppressor_LogBook</b>	12
3.1.5	<b>Suppressor_Parameters</b>	12
3.2.0	<b>Stored Procedures</b>	13
3.2.1	<b>Suppressor_Get_Details</b>	13
3.2.2	<b>Suppressor_Insert_Details</b>	14
3.2.3	<b>Suppressor_Insert_LogBook</b>	18
4.0.0	<b>Development</b>	19
4.1.0	<b>Login Form</b>	19

<b>4.1.1</b>	<b>Login Form - Design</b>	<b>19</b>
<b>4.1.2</b>	<b>Login Form - Errors</b>	<b>20</b>
<b>4.1.3</b>	<b>Login Form - Methods</b>	<b>22</b>
<b>4.2.0</b>	<b>Main Form</b>	<b>27</b>
<b>4.2.1</b>	<b>Main Form - Design</b>	<b>27</b>
<b>4.2.2</b>	<b>Main Form - Errors</b>	<b>28</b>
<b>4.2.3</b>	<b>Main Form - Methods</b>	<b>36</b>
<b>5.0.0</b>	<b>Demo &amp; Testing</b>	<b>53</b>
<b>5.0.1</b>	<b>Logging Onto the Application</b>	<b>53</b>
<b>5.0.2</b>	<b>Selecting a Client</b>	<b>54</b>
<b>5.0.3</b>	<b>Completing the required Fields</b>	<b>55</b>
<b>5.0.4</b>	<b>Selecting a reason</b>	<b>56</b>
<b>5.0.5</b>	<b>Accepting a record</b>	<b>57</b>
<b>5.0.6</b>	<b>Clearing the form</b>	<b>58</b>
<b>5.0.7</b>	<b>Exiting the Application.</b>	<b>59</b>
<b>5.0.8</b>	<b>Post Demo Review</b>	<b>60</b>
<b>6.0.0</b>	<b>Summary</b>	<b>61</b>

# 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.

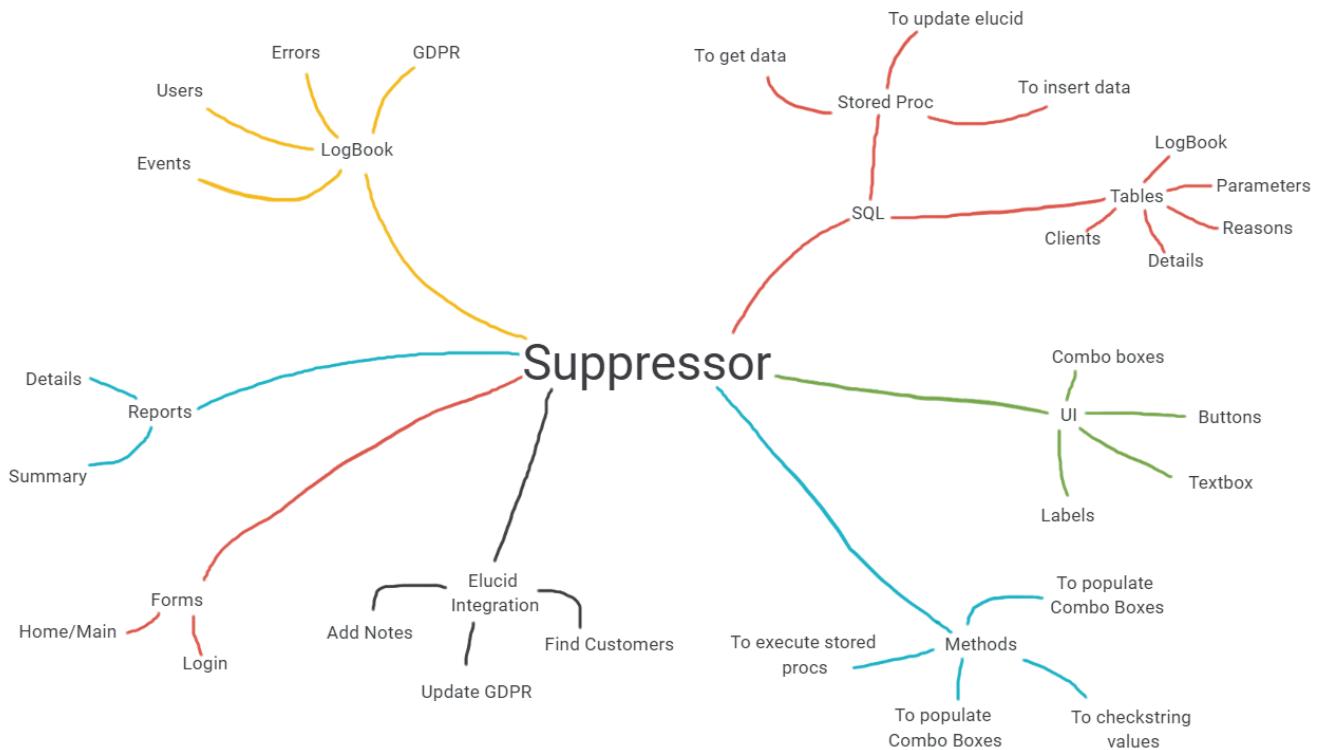
To ensure comprehensive tracking, the application incorporates an embedded logging process. This process captures instances such as application usage, errors, each saved entry, and every connection made to the Elucid system.

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



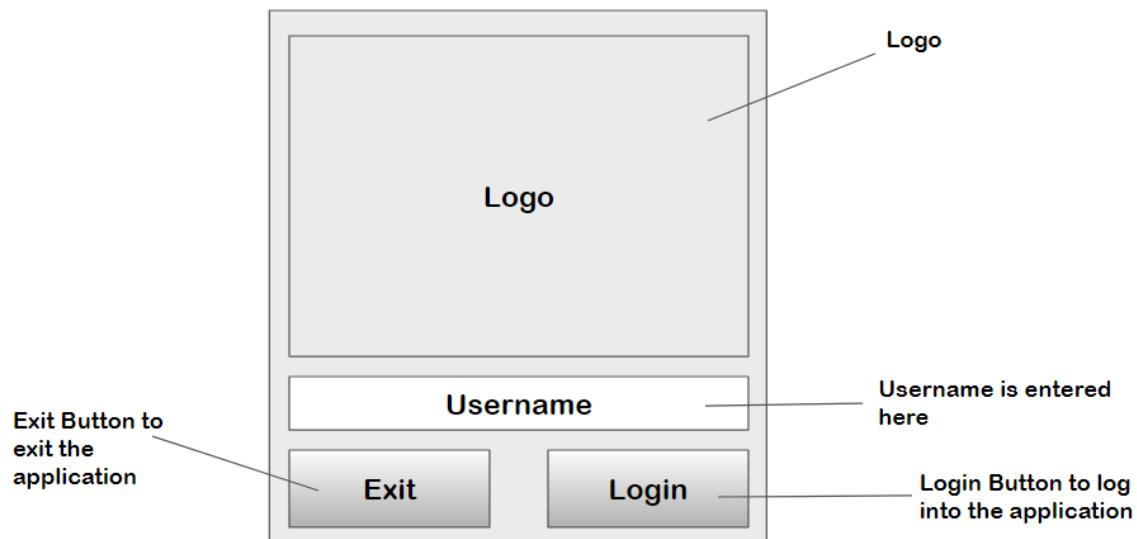
- A user needs to be able to input data based on the information they receive as no-mails or mailing suppressions
- The user will have personal data such as a recipient's name, address, city and postcode which the recipient would like to be removed from mailing lists.
- A user needs to be able to choose 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 **lblUsername** and will be 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

The diagram illustrates the Main Form - Design with the following components and descriptions:

- Username pulled through from login screen**: Points to the "Username" field.
- Dropdown menu for Client to be selected here**: Points to the "Client" dropdown menu.
- Name goes here**: Points to the "Name" text box.
- City goes here**: Points to the "Address" text box.
- Postcode goes here**: Points to the "City" text box.
- Email Address goes here**: Points to the "Postcode" text box.
- Customer Number goes here (Optional)**: Points to the "Telephone" text box.
- Address goes here**: Points to the "Email" text box.
- Telephone Number goes here**: Points to the "Reason" dropdown menu.
- Dropdown menu for reason code**: Points to the "Accept" button.
- Accept button to accept the record**: Points to the "Accept" button.
- Exit button to Exit Application**: Points to the "Exit" button.
- Clear button to clear all fields**: Points to the "Clear" button.

- 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.
- **ST\_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\_Updated\_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_Updated_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\_Updated\_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_Updated_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

```

14 CREATE TABLE [dbo].[Suppressor_LogBook](
15     [tKey] [varchar](255) NOT NULL,
16     [Domain_User] [varchar](48) NULL,
17     [User_Created] [varchar](48) NULL,
18     [Form] [varchar](255) NULL,
19     [Event] [varchar](255) NULL,
20     [Notes] [varchar](4000) NULL,
21     [DT_Created] [datetime] NULL,
22 
23 PRIMARY KEY CLUSTERED
24 (
25     [tKey] ASC
26 )WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
27 ) 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

```

14 CREATE TABLE [dbo].[Suppressor_Parameters](
15     [ID] [varchar](3) NOT NULL,
16     [Description] [varchar](100) NULL,
17     [String_1] [varchar](4000) NULL,
18     [String_2] [varchar](4000) NULL,
19     [String_3] [varchar](4000) NULL,
20     [Int_1] [int] NULL,
21     [Int_2] [int] NULL,
22     [Int_3] [int] NULL,
23 
24 PRIMARY KEY CLUSTERED
25 (
26     [ID] ASC
27 )WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
28 ) 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)

```

1 USE [CrossDatabase]
2 GO
3 /****** Object: .StoredProcedure [dbo].[Suppressor_Get_Details]    Script Date: 16/01/2024 15:28:34 *****/
4 SET ANSI_NULLS ON
5 GO
6 SET QUOTED_IDENTIFIER OFF
7 GO
8
9
10 ALTER PROC [dbo].[Suppressor_Get_Details]
11
12 |@Customer VARCHAR(24)
13 |,@Client VARCHAR(255)
14
15 AS
16
17 DECLARE @FROM_CLAUSE VARCHAR(MAX)
18 DECLARE @Site VARCHAR(12)
19 DECLARE @SQL VARCHAR(MAX)
20
21 SET @Site = (SELECT [Site] FROM Suppressor_Clients WHERE [Description] = @Client)
22
23 SET @FROM_CLAUSE =
24 (
25 SELECT
26 'FROM [' + RTRIM([Server]) + '].[[' + RTRIM([Database]) + '].dbo.cust'
27 FROM
28 Suppressor_Clients
29 WHERE
30 [Description] = @Client
31 )
32
33 SET @SQL =
34 '
35 SELECT
36 ISNULL(RTRIM(customer), ''No Customer'') as Customer
37 ,ISNULL(RTRIM(initials) + ' ', '') + ISNULL(RTRIM(full_name), '') as [Name]
38 ,ISNULL(RTRIM([address]), '') as [Address]
39 ,ISNULL(RTRIM([city]), '') as [City]
40 ,ISNULL(RTRIM([postcode]), '') as [Postcode]
41 ,ISNULL(RTRIM([email_address]), '') as [Email]
42 ,ISNULL(RTRIM([phone_day]), '') as [Telephone]
43 '
44 + @FROM_CLAUSE +
45 '
46 WHERE
47 level_1 = '' + @Site + ''
48 AND customer = '' + @Customer + ''
49 '
50
51 PRINT (@SQL)
52 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)

```

1 USE [CrossDatabase]
2 GO
3 /****** Object: StoredProcedure [dbo].[Suppressor_Insert_Details] Script Date: 07/02/2024 13:35:14 *****/
4 SET ANSI_NULLS ON
5 GO
6 SET QUOTED_IDENTIFIER OFF
7 GO
8
9
10 ALTER PROC [dbo].[Suppressor_Insert_Details]
11
12    @tKey VARCHAR(255)
13    ,@Client VARCHAR(255)
14    ,@Customer VARCHAR(48)
15    ,@Name VARCHAR(255)
16    ,@Address VARCHAR(500)
17    ,@City VARCHAR(255)
18    ,@Postcode VARCHAR(24)
19    ,@Email VARCHAR(48)
20    ,@Telephone VARCHAR(48)
21    ,@Reason VARCHAR(255)
22    ,@User VARCHAR(48)
23
24 AS
25
26 -----
27 -- DECLARE VARIABLES
28 -----
29
30 DECLARE @ReasonCode VARCHAR(12) = (SELECT [Reason_Code] FROM [Suppressor_Reasons] WHERE [Description] = @Reason)
31 DECLARE @FROM_CLAUSE VARCHAR(MAX)
32 DECLARE @Site VARCHAR(12)
33 DECLARE @SQL_GET NVARCHAR(MAX)
34 DECLARE @SQL_UPDATE NVARCHAR(MAX)
35 DECLARE @Client_ID VARCHAR(3)
36 DECLARE @OG_No_Promote VARCHAR(1)
37 DECLARE @OG_Mail VARCHAR(1)
38 DECLARE @OG_Email VARCHAR(1)
39 DECLARE @OG_Phone VARCHAR(1)
40 DECLARE @OG_SMS VARCHAR(1)
41 DECLARE @OG_Survey VARCHAR(1)
42 DECLARE @OG_Rent VARCHAR(1)
43 DECLARE @OG_GDPR_Compliant VARCHAR(1)
44
45 SET @Customer = CASE WHEN @Customer = '' THEN NULL ELSE @Customer END
46 SET @Address = CASE WHEN @Address = '' THEN NULL ELSE @Address END
47 SET @City = CASE WHEN @City = '' THEN NULL ELSE @City END
48 SET @Postcode = CASE WHEN @Postcode = '' THEN NULL ELSE @Postcode END
49 SET @Email = CASE WHEN @Email = '' THEN NULL ELSE @Email END
50 SET @Telephone = CASE WHEN @Telephone = '' THEN NULL ELSE @Telephone END
51

```

```
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 -----  
76 --BEGIN TRY  
77  
78 --DROP TABLE OUTPUT_FLAGS  
79  
80 --END TRY  
81 --BEGIN CATCH  
82 |  
83 --END CATCH  
84  
85 --CREATE TABLE OUTPUT_FLAGS  
86 --(  
87 --  [OG_No_Promote] [varchar](1) null,  
88 --  [OG_Mail] [varchar](1) null,  
89 --  [OG_Email] [varchar](1) null,  
90 --  [OG_Phone] [varchar](1) null,  
91 --  [OG_SMS] [varchar](1) null,  
92 --  [OG_Survey] [varchar](1) null,  
93 --  [OG_Rent] [varchar](1) null,  
94 --  [OG_gdpr_Compliant] [varchar](1) null,  
95 --)  
96 -----  
97 -- GET GDPR FLAG DATA  
98 -----  
99
```

```
96 -----  
97 -- GET GDPR FLAG DATA  
98 -----  
99  
100 SET @SQL_GET = '  
101  
102 INSERT INTO OUTPUT_FLAGS  
103  
104 SELECT  
105     RTRIM(no_promote)  
106     ,RTRIM(mail)  
107     ,RTRIM(email)  
108     ,RTRIM(phone)  
109     ,RTRIM(sms)  
110     ,RTRIM(survey)  
111     ,RTRIM(rent)  
112     ,RTRIM(gdpr_compliant)  
113 FROM  
114     ' + @FROM_CLAUSE + 'cust_gdpr WITH(NOLOCK)  
115 WHERE  
116     customer = ''' + @Customer + ''''  
117  
118 INSERT INTO ' + @FROM_CLAUSE + 'cust_hist VALUES  
119 (  
120     ''' + @tKey + ''''  
121     ,'''!  
122     ,GETDATE()  
123     ,''' + @User + ''''  
124     ,GETDATE()  
125     ,''' + @Site + ''''  
126     ,'''GROUP''  
127     ,CAST(GETDATE() as date)  
128     ,CAST(GETDATE() as time)  
129     ,'''85'''  
130     ,'''Suppressor - No Mail Recorded'''  
131     ,''' + @Customer + ''''  
132     ,NULL  
133     ,NULL  
134     ,'''SUP'''  
135     ,NULL  
136     ,NULL  
137     ,NULL  
138     ,NULL  
139     ,NULL  
140     ,NULL  
141     ,NULL  
142     ,''' + @User + ''''  
143     ,'''This customer has been recorded as a No Mail through the suppressor application.'''  
144 )  
145
```

```

147
148
149
150 PRINT (@SQL_GET)
151 EXECUTE (@SQL_GET)
152
153 SET @OG_No_Promote = (SELECT OG_No_Promote FROM OUTPUT_FLAGS)
154 SET @OG_Mail = (SELECT OG_Mail FROM OUTPUT_FLAGS)
155 SET @OG_Email = (SELECT OG_Email FROM OUTPUT_FLAGS)
156 SET @OG_Phone = (SELECT OG_Phone FROM OUTPUT_FLAGS)
157 SET @OG_SMS = (SELECT OG_SMS FROM OUTPUT_FLAGS)
158 SET @OG_Survey = (SELECT OG_Survey FROM OUTPUT_FLAGS)
159 SET @OG_Rent = (SELECT OG_Rent FROM OUTPUT_FLAGS)
160 SET @OG_GDPR_Compliant = (SELECT OG_gdpr_Compliant FROM OUTPUT_FLAGS)
161
162 --DROP TABLE OUTPUT_FLAGS
163
164 -----
165 -- INSERT THE DATA
166 -----
167
168 IF NOT EXISTS (SELECT Customer FROM Suppressor_Details WHERE Customer = @Customer AND Customer <> '' AND Client = @Client)
169 INSERT INTO Suppressor_Details VALUES
170 (
171     @tKey -- BATCH NUMBER
172     ,GETDATE() -- DATE CREATED
173     ,@Client_ID -- CLIENT ID
174     ,@Client -- CLIENT
175     ,@Customer -- CUSTOMER
176     ,CASE WHEN ISNULL(@Customer, '') = '' THEN 0 ELSE 1 END -- CUSTOMER FLAG
177     ,@Name -- NAME
178     ,@Address -- ADDRESS
179     ,@City -- CITY
180     ,@Postcode -- POSTCODE
181     ,@Email -- EMAIL ADDRESS
182     ,@Telephone -- TELEPHONE
183     ,@ReasonCode -- REASON CODE
184     ,@OG_No_Promote -- NO PROMOTE
185     ,@OG_Mail -- MAIL
186     ,@OG_Email -- EMAIL
187     ,@OG_Phone -- PHONE
188     ,@OG_SMS -- SMS
189     ,@OG_Survey -- SURVEY
190     ,@OG_Rent -- RENT
191     ,@OG_GDPR_Compliant -- GDPR COMPLIANT
192     ,@User -- USER CREATED
193 )
194
195

```

### 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**

```
1 USE [CrossDatabase]
2 GO
3 /****** Object: StoredProcedure [dbo].[Suppressor_Insert_LogBook]
4 SET ANSI_NULLS ON
5 GO
6 SET QUOTED_IDENTIFIER OFF
7 GO
8
9
10 ALTER PROC [dbo].[Suppressor_Insert_LogBook]
11
12     @tKey VARCHAR(255)
13     ,@Domain_User VARCHAR(48)
14     ,@User_Created VARCHAR(48)
15     ,@Form VARCHAR(255)
16     ,@Event VARCHAR(255)
17     ,@Notes VARCHAR(255)
18
19
20
21 AS
22
23 INSERT INTO Suppressor_LogBook VALUES
24 (
25     @tKey
26     ,@Domain_User
27     ,@User_Created
28     ,@Form
29     ,@Event
30     ,@Notes
31     ,GETDATE()
32 )
33
34
35
```

# 4.0.0 - Development

## 4.1.0 - Login Form

### 4.1.1 - Login Form - Design

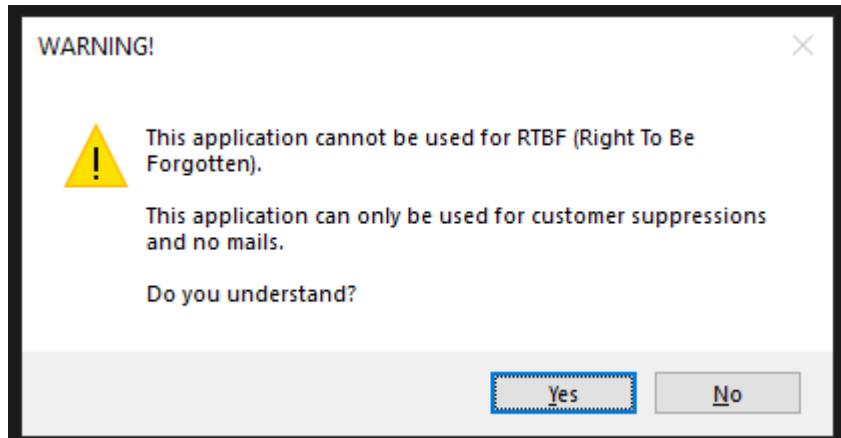


Here we have the Login Form, as specified in the pre-development and planning stages. It features a text box for users to enter their username, buttons for exiting and logging into the application, and a label displaying the application name and the currently open form. The application logo is centred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

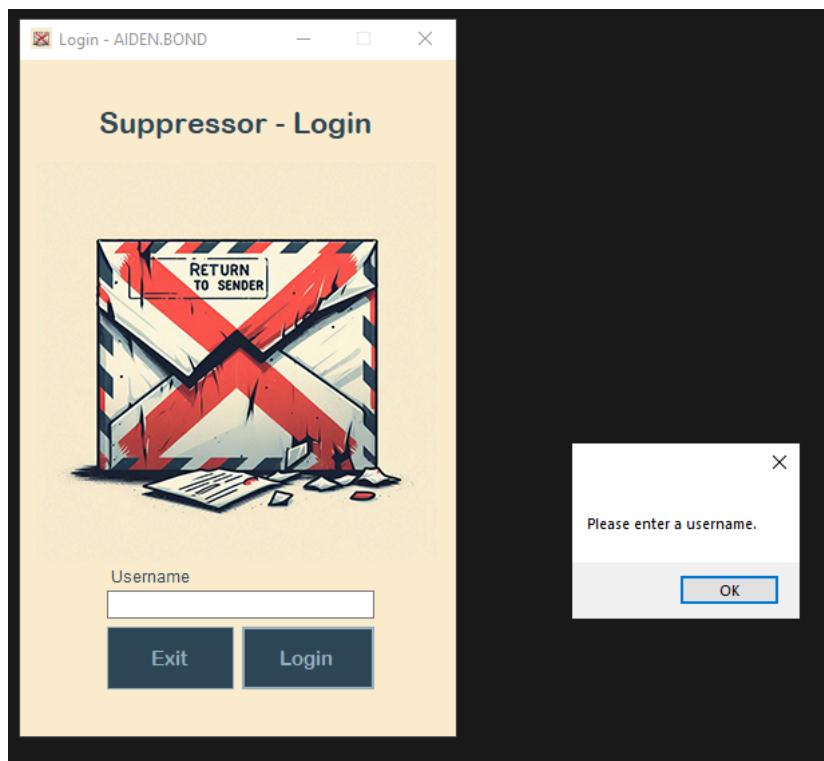
To enhance user experience and streamline design, the window of the application is non-resizable. This decision was made to prevent complications when scaling UI elements for various resolution displays, ultimately reducing the application's complexity and design time.

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.

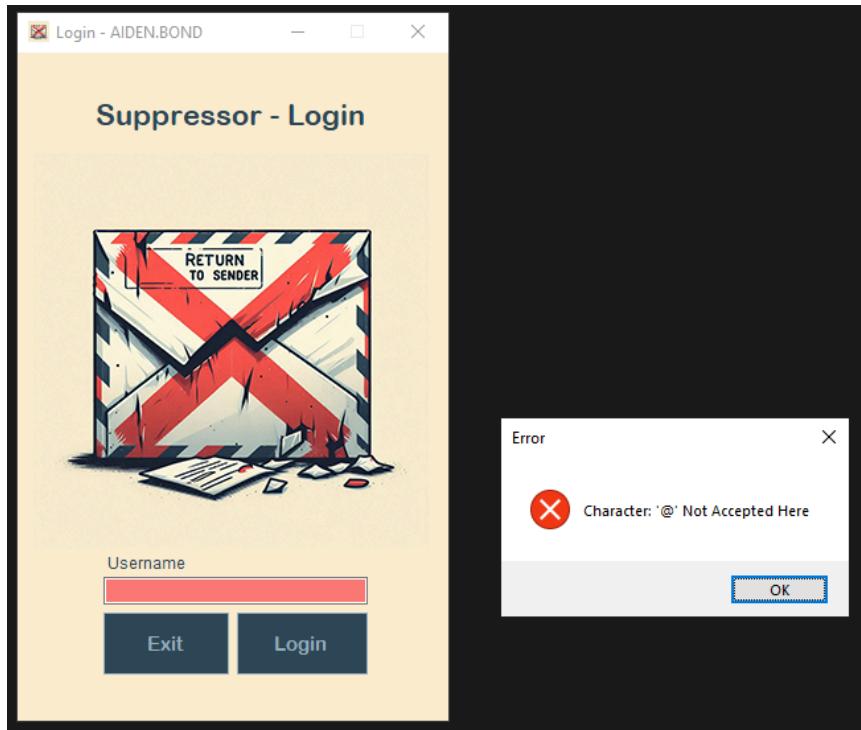
As specified in the pre-development design planning, here is the message which pops up when a user logs in, this informs the user that this application cannot be used for RTBF (Right to be forgotten) and is only to be used for no mails and customer suppressions



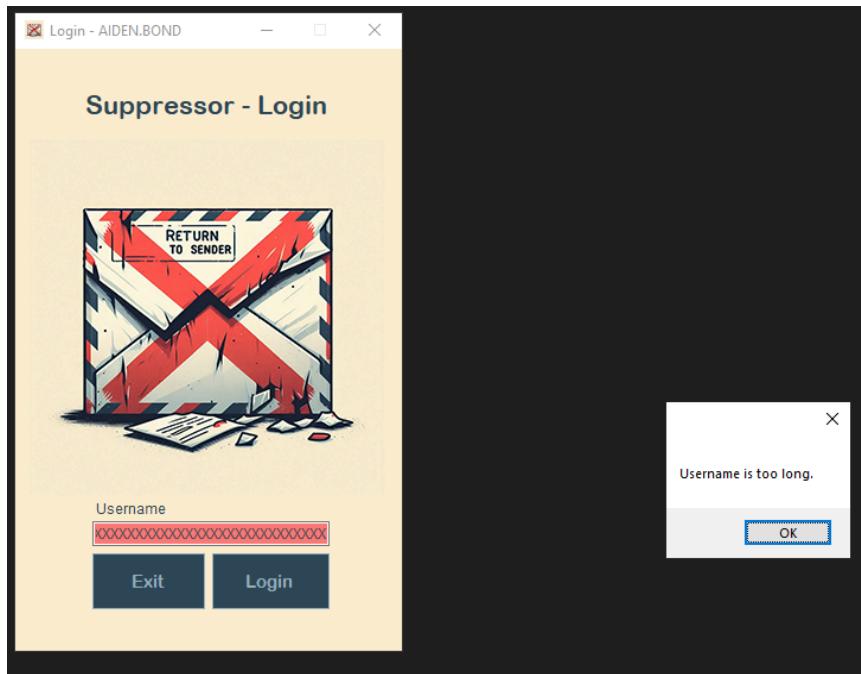
#### 4.1.2 - Login Form - Errors



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

```

11     | 1 reference
12     | public LoginForm()
13     |
14     | {
15     |     InitializeComponent();
16     |     this.FormClosing += MainForm_FormClosing;
17     |     this.MaximizeBox = false;
18     |     this.KeyPreview = true;
19     |     this.KeyDown += LoginForm_KeyDown;
20     |     txbUsername.KeyPress += txbUsername_KeyPress;
21     |     txbUsername.KeyDown += txbUsername_KeyDown;
22     |     Text = $"Login - {Environment.UserName.ToUpper()}";
23

```

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

```

23     | // Exit Application Method
24     | 1 reference
25     | private void MainForm_FormClosing(object sender, FormClosingEventArgs e)
26     | {
27     |     if (e.CloseReason == CloseReason.UserClosing)
28     |     {
29     |         DialogResult result = MessageBox.Show("Are you sure you want to Exit the application?", "Confirmation", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
30     |         if (result == DialogResult.Yes)
31     |         {
32     |             Application.Exit();
33     |         }
34     |         else if (result == DialogResult.No)
35     |         {
36     |             e.Cancel = true;
37     |         }
38     |     }

```

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

```

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
91      1 reference
92      private void txbUsername_TextChanged(object sender, EventArgs e)
93      {
94          string username = txbUsername.Text;
95          int usernameLength = username.Length;
96
97          if (usernameLength > 100)
98          {
99              MessageBox.Show("Username is too long.");
100             txbUsername.Text = username.Substring(0, Math.Min(username.Length, 100));
101             txbUsername.SelectAll();
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
111  1 reference
112  private void btnLogin_Click(object sender, EventArgs e)
113  {
114      string userName = txbUsername.Text.ToUpper();
115
116      if (string.IsNullOrEmpty(userName))
117      {
118          MessageBox.Show("Please enter a username.");
119          return;
120      }
121
122      // Display T&Cs
123      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);
124
125      if (result == DialogResult.Yes)
126      {
127          MainForm MainForm = new MainForm();
128          MainForm.userName = userName;
129          MainForm.Show();
130          this.Hide();
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 suppressions 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
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)
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
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      }
179
```

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

The screenshot shows a Windows application window titled "Home". Inside, there's a title label "Suppressor - Home". Below it are several input fields and labels: "Client\*" with a dropdown arrow; "Customer" with a text input field containing placeholder text "Customer Exists False?"; "Name\*" with a text input field; "Address" with a text input field; "City" and "Postcode" with separate text input fields; "Email" and "Telephone" with separate text input fields; "Reason\*" with a dropdown arrow; and a "Details" section which is currently hidden. At the bottom are three buttons: "Exit", "Clear", and "Accept". A "LogBook" link is visible at the very bottom left.

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

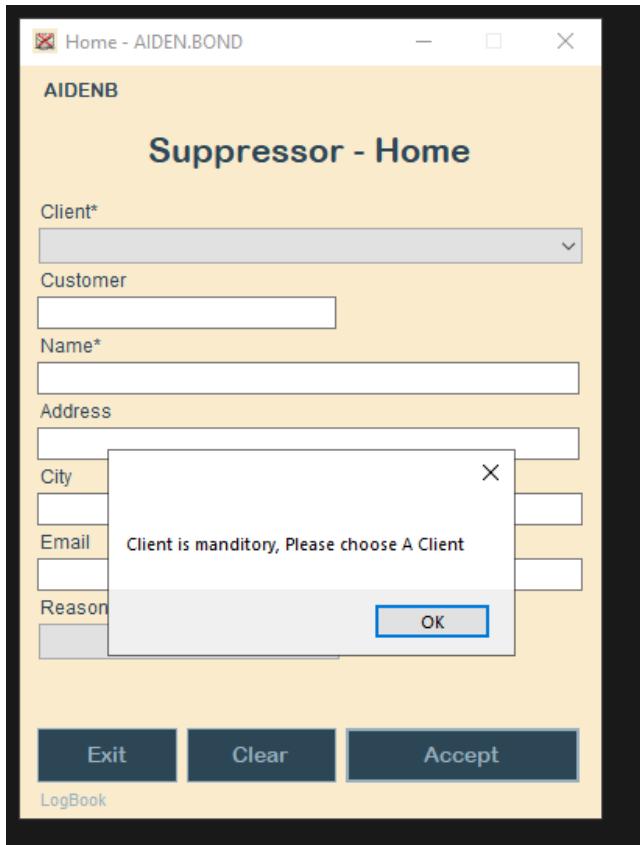
successful operation and provides details of the entered record and timestamp.

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.

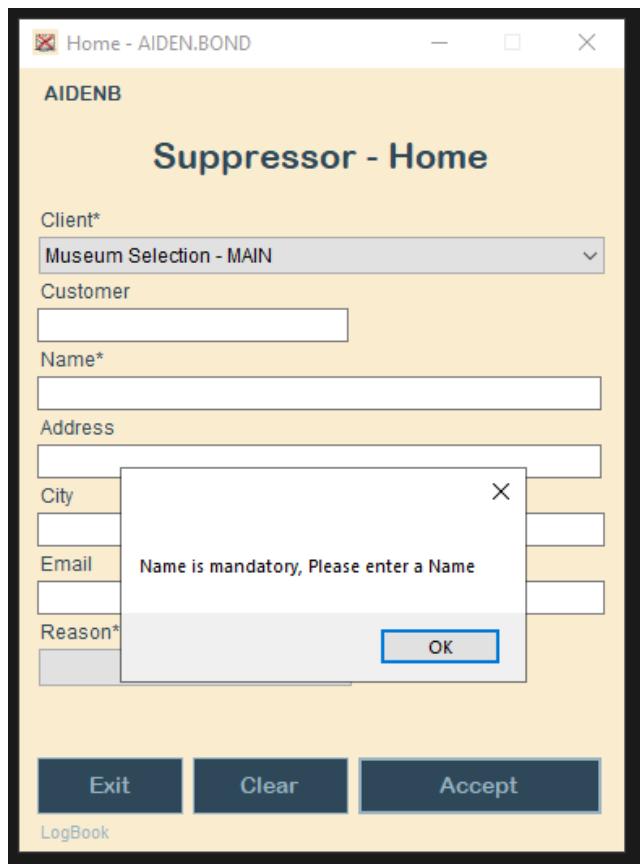
Similar to the Login Form, the window's ability to resize has been disabled to avoid complications when scaling UI elements for different resolution displays, aiming to simplify the application's design.

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.

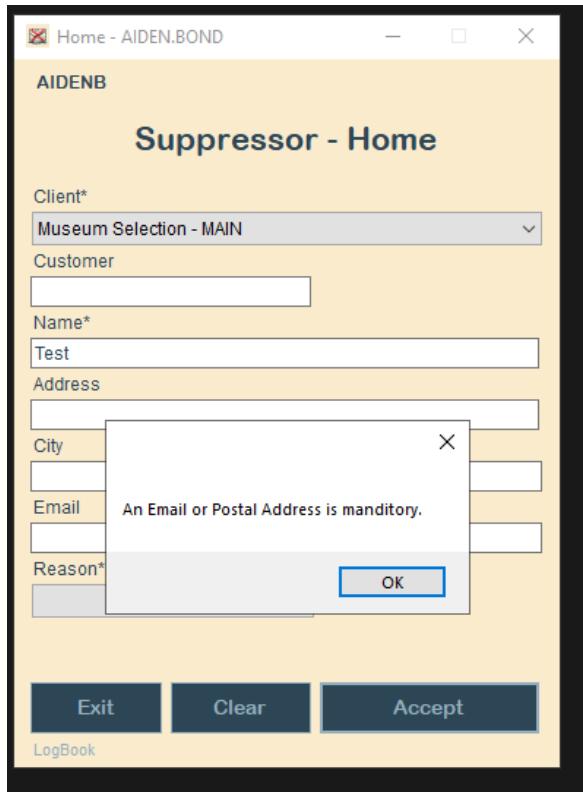
#### 4.1.2 - Main Form - Errors



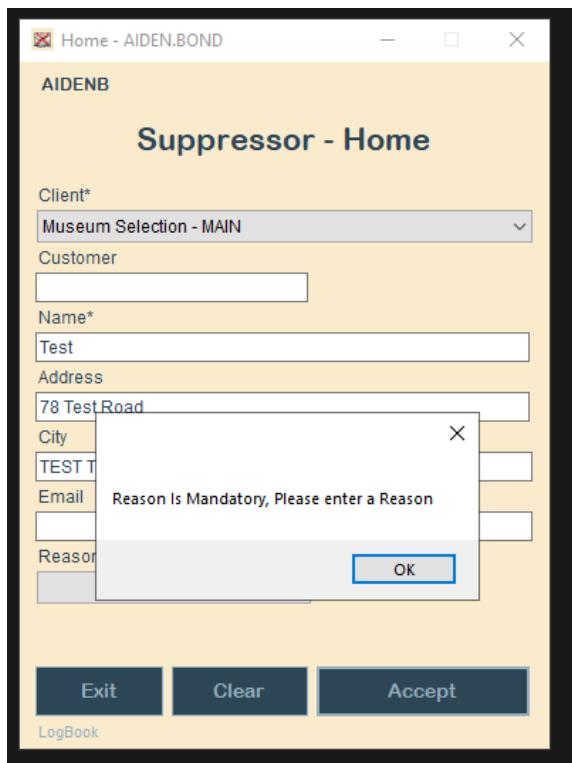
This error indicates that the user attempted to click the "Accept" button without selecting an option from the Client field. In response, a message box is triggered to notify the user of the omission.



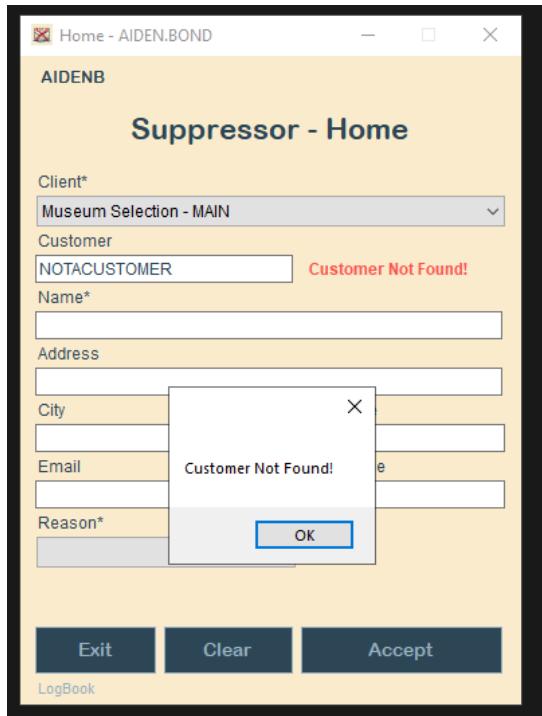
This error indicates that the user attempted to click the "Accept" button without entering a value in the Name field. As a result, a message box is triggered to notify the user of the missing information.



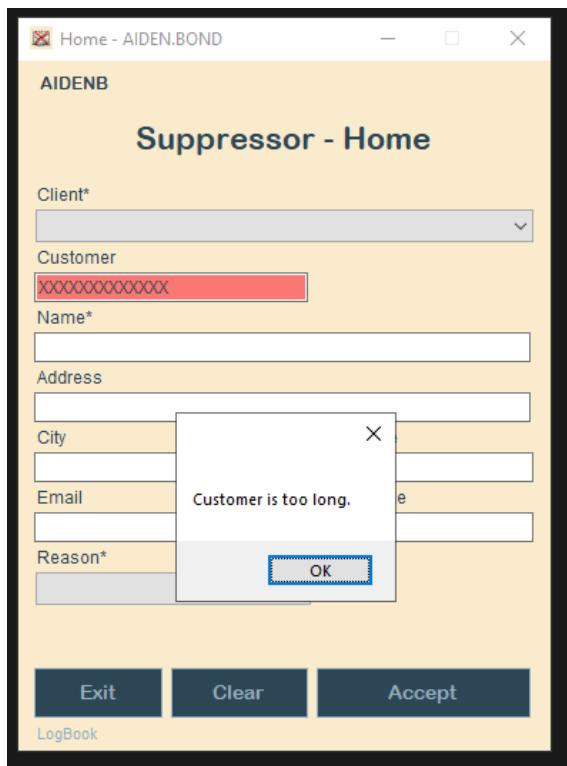
This error signifies that the user tried to click the "Accept" button without entering a value in either the email field or the Address field. Consequently, a message box is triggered to notify the user of the missing information.



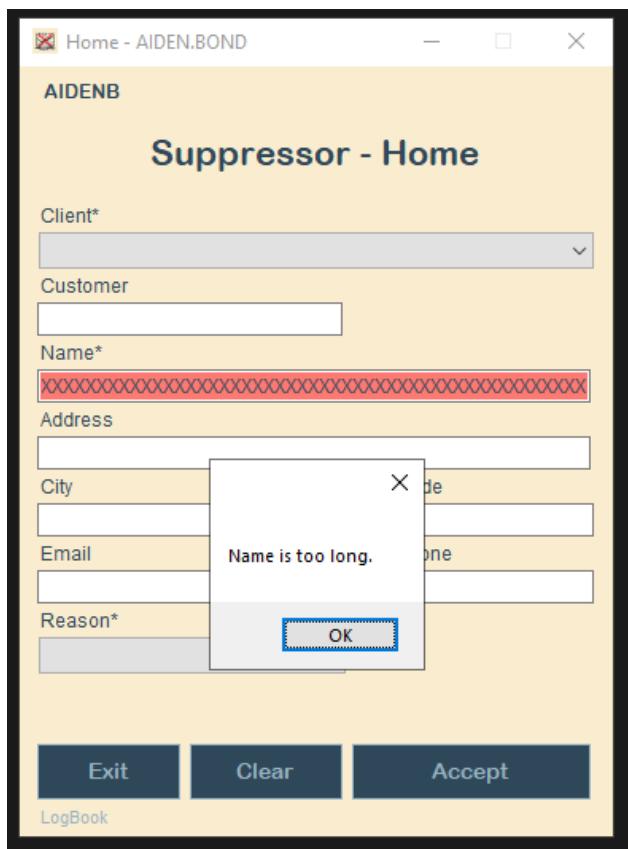
This error indicates that the user tried to click the "Accept" button without selecting an option from the reason field. As a result, a message box is triggered to notify the user of the omission.



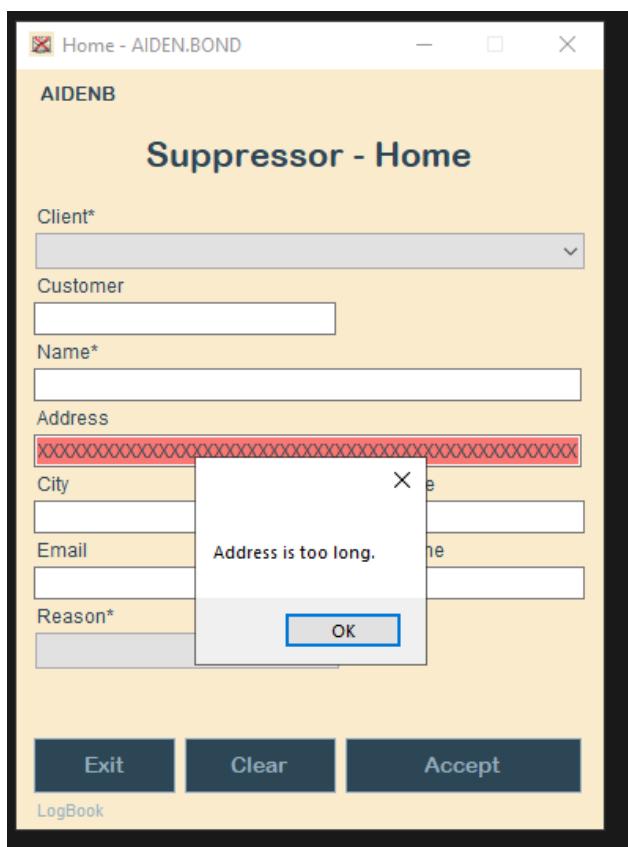
This error signifies that the user entered a customer number that does not exist in the Elucid database. In response, a message box and a label appear, notifying the user of the discrepancy.



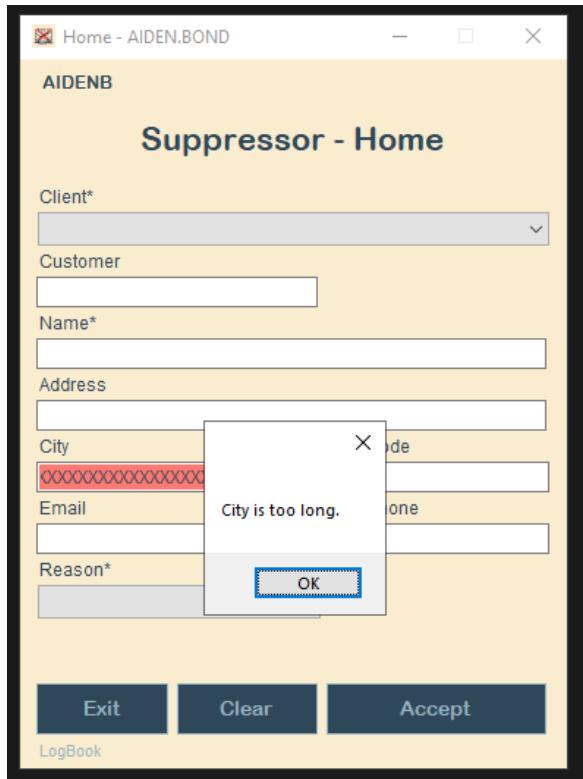
This error indicates that the user attempted to exceed the character limit for the Customer field. As a result, a message box appears, notifying the user of the character limit and preventing the input from exceeding it.



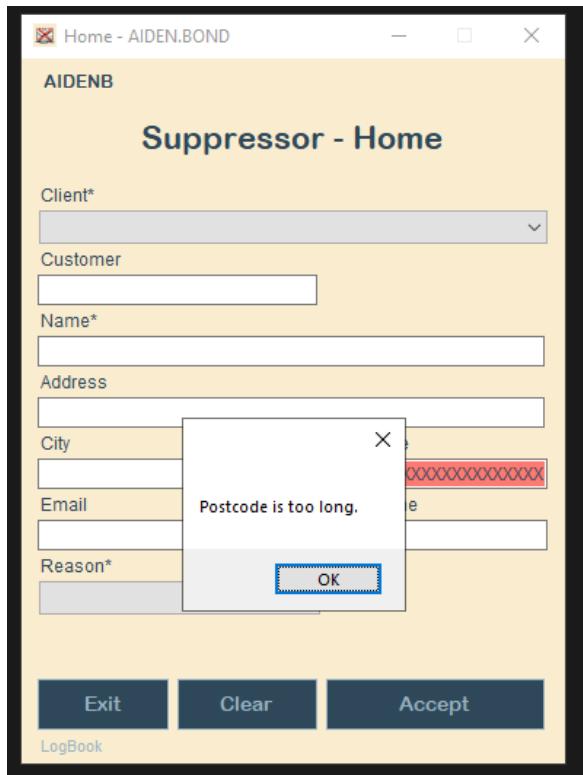
This error indicates that the user attempted to exceed the character limit for the Name field. Consequently, a message box appears, notifying the user of the character limit and preventing the input from surpassing it.



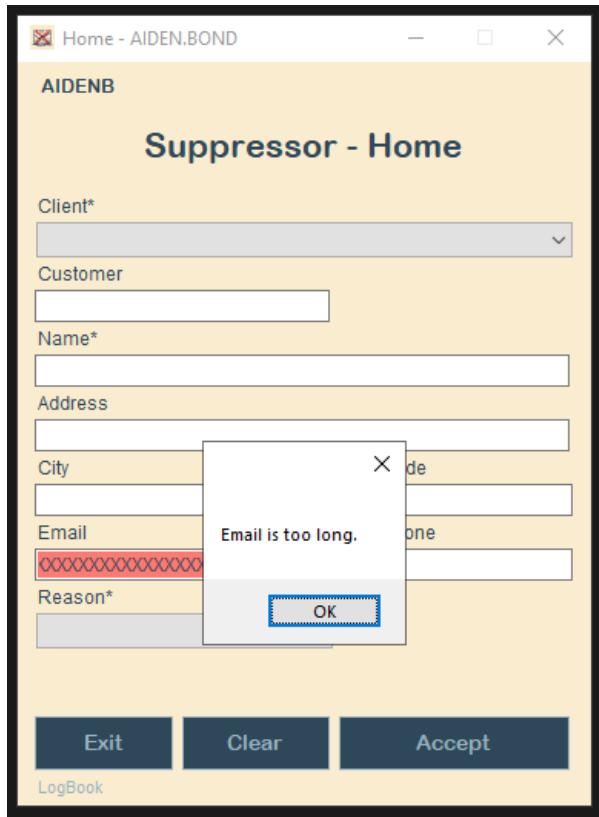
This error indicates that the user attempted to exceed the character limit for the Address field. In response, a message box appears, notifying the user of the character limit and preventing the input from surpassing it.



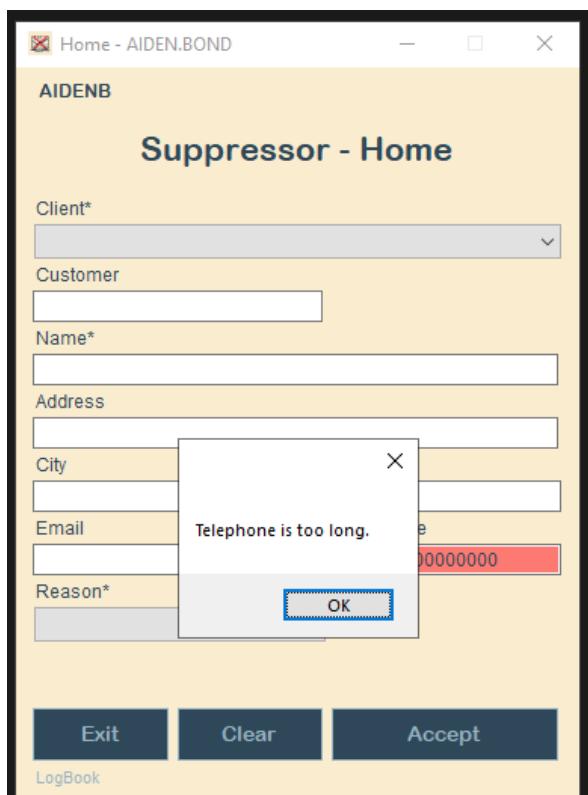
This error indicates that the user attempted to exceed the character limit for the City field. As a result, a message box appears, notifying the user of the character limit and preventing the input from surpassing it.



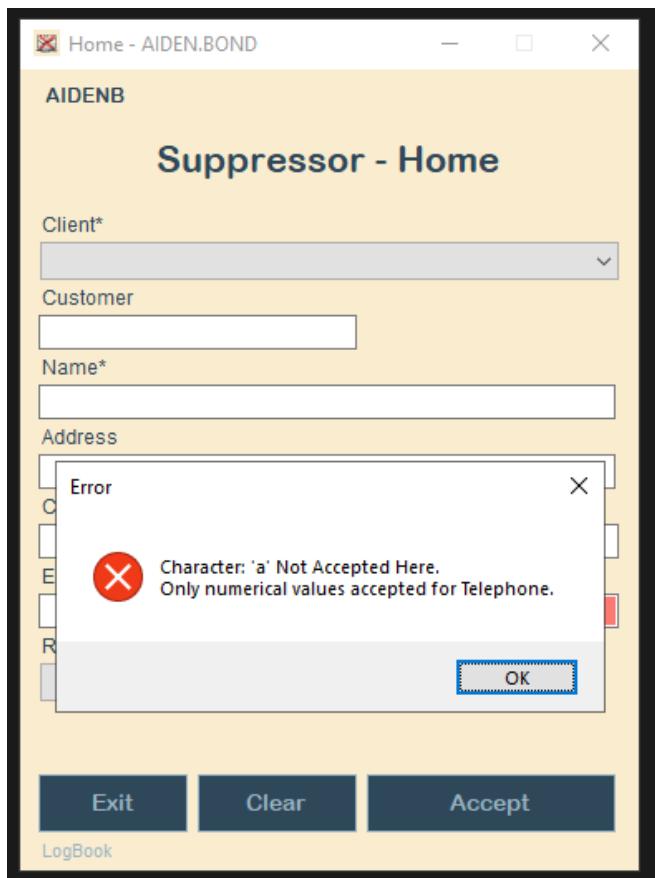
This error indicates that the user attempted to exceed the character limit for the Postcode field. In response, a message box appears, notifying the user of the character limit and preventing the input from surpassing it.



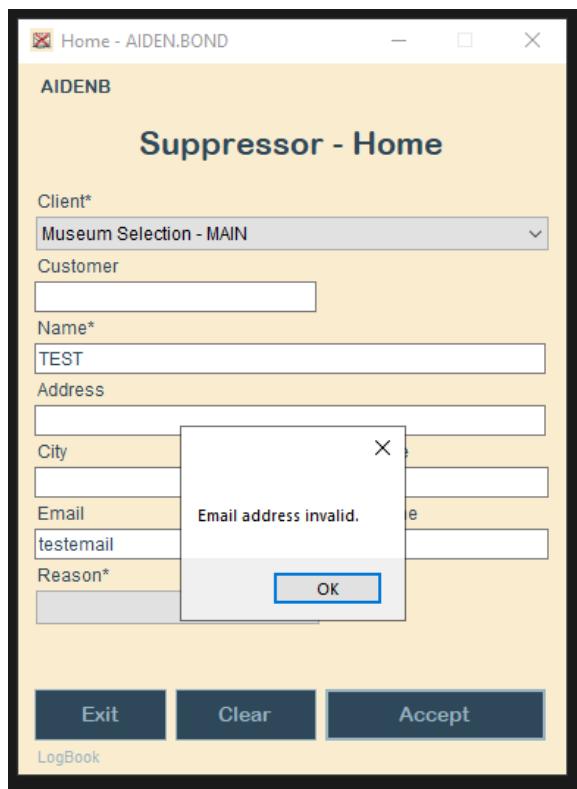
This error indicates that the user attempted to exceed the character limit for the Email field. As a result, a message box appears, notifying the user of the character limit and preventing the input from surpassing it.



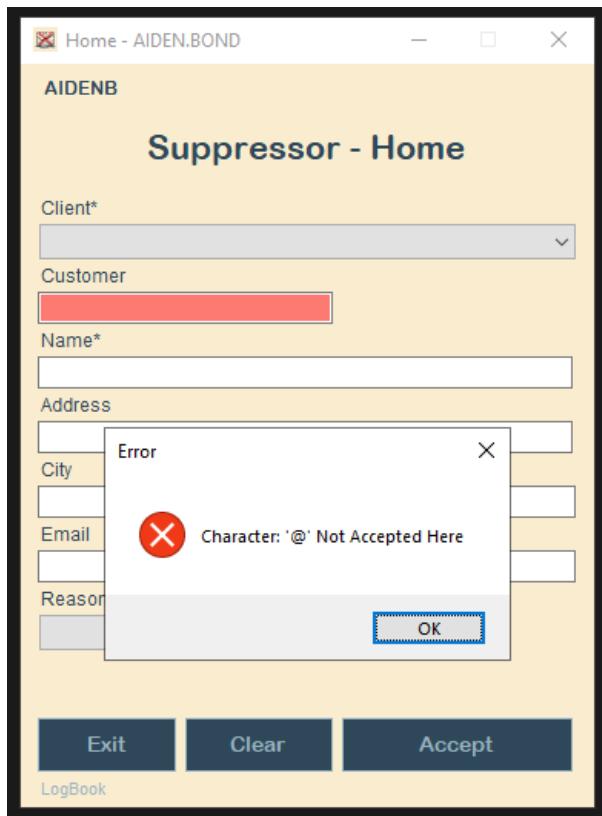
This error indicates that the user attempted to exceed the character limit for the Telephone field. As a result, a message box appears, notifying the user of the character limit and preventing the input from surpassing it.



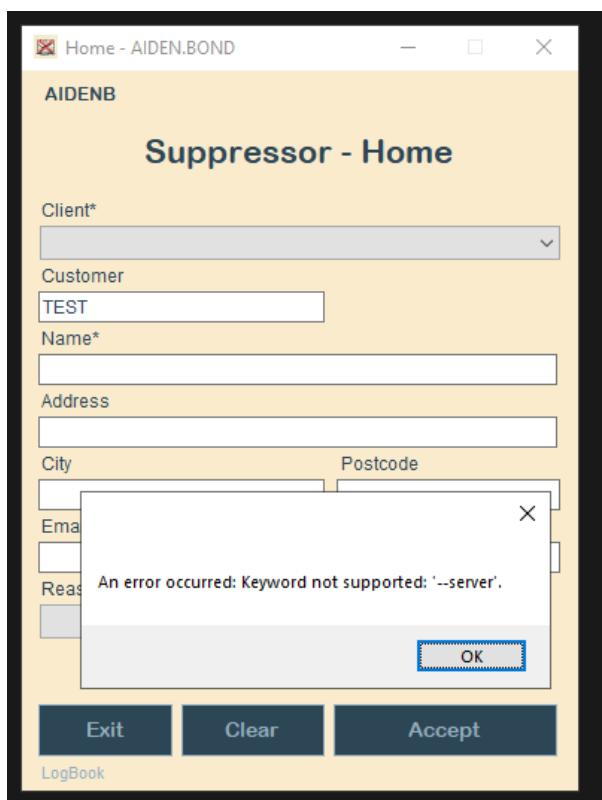
This error indicates that the user attempted to enter a character other than a number or the '+' symbol into the Telephone field. As a result, a message box appears, notifying the user of the invalid input and preventing the entry of non-numeric characters.



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
14     8 references
15     public string userName { get; set; }
16
17     public int cust = 0;
18
19     // Set SQL Connection String
20     private const string connectionString = "Server=SQL-SSRS;Database=CrossDatabase;Integrated Security=True;Encrypt=False;";
21     1 reference
22     public MainForm()
23     {
24         InitializeComponent();
25         this.FormClosing += MainForm_FormClosing;
26         this.MaximizeBox = false; // Disable Maximize window option
27         this.KeyPreview = true;
28         this_KeyDown += MainForm_KeyDown;
29         PopulateComboBoxes(); // Populate the dropdown menus
30         btnClear.CausesValidation = false;
31         btnExit.CausesValidation = false;
32         Text = $"Home - {Environment.UserName.ToUpper()}";
33     }

```

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
34     1 reference
35     private void MainForm_Load(object sender, EventArgs e)
36     {
37         lblUsername.Text = userName; // Show Username
38         LogBook("", "[MainForm]", "[FormLoad]", "Application Started");
39     }

```

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

```

48     // Exit Application Method
49     private void MainForm_FormClosing(object sender, FormClosingEventArgs e)
50     {
51         if (e.CloseReason == CloseReason.UserClosing)
52         {
53             DialogResult result = MessageBox.Show("Are you sure you want to Exit the application?", "Confirmation", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
54             if (result == DialogResult.Yes)
55             {
56                 LogBook("", "[MainForm]", "[FormClosing]", "Application Closed");
57                 Application.Exit();
58             }
59             else if (result == DialogResult.No)
60             {
61                 e.Cancel = true;
62             }
63         }
64     }

```

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

```

58     // Populate Client Options
59     private void PopulateComboBoxes()
60     {
61         // Declare Variables
62         string ClientQuery = "SELECT [Description] FROM Suppressor_Clients WHERE [Active] = '1' ORDER BY [ID]";
63         string ReasonQuery = "SELECT [Description] FROM Suppressor_Reasons WHERE [Active] = '1' ORDER BY [ID]";
64
65         try
66         {
67             // Execute SQL Command
68             using (SqlConnection conn = new SqlConnection(connectionString))
69             {
70
71                 conn.Open(); // Open SQL Connection
72                 using (SqlCommand cmdClient = new SqlCommand(ClientQuery, conn))
73                 {
74                     using (SqlDataReader reader = cmdClient.ExecuteReader())
75                     {
76                         cbClient.Items.Clear();
77
78                         while (reader.Read())
79                         {
80                             cbClient.Items.Add(reader["Description"].ToString());
81                         }
82                     }
83
84                     using (SqlCommand cmdReason = new SqlCommand(ReasonQuery, conn))
85                     {
86                         using (SqlDataReader reader = cmdReason.ExecuteReader())
87                         {
88                             cbReason.Items.Clear();
89
89                             while (reader.Read())
90                             {
91                                 cbReason.Items.Add(reader["Description"].ToString());
92                             }
93                         }
94                     }
95                 }
96             }
97         catch (Exception ex)
98         {
99             MessageBox.Show($"An error occurred getting combo box values: \n{ex.Message}");
100            Application.Exit();
101        }
102    }
103 }

```

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

```

105 // LogBook Method
106 14 references
107 private void LogBook(string tKey, string Form, string Event, string notes)
108 {
109     string query = "[Suppressor_Insert_LogBook] @tKey, @Domain_User, @User_Created, @Form, @Event, @Notes";
110     string user = $"{Environment.MachineName}.{Environment.UserName}";
111     string tKeySufix = Environment.UserName.Length >= 2 ? Environment.UserName.Substring(0, 2).ToUpper() : Environment.UserName.ToUpper();
112
113     if (string.IsNullOrEmpty(tKey))
114     {
115         tKey = DateTime.Now.ToString("yyyyMMddHHmmssffff") + tKeySufix;
116     }
117     else if (tKey == "ERROR")
118     {
119         tKey = DateTime.Now.ToString("yyyyMMddHHmmssffff") + tKeySufix + "-ER";
120     }
121
122     try
123     {
124         // Execute SQL Query
125         using (SqlConnection conn = new SqlConnection(connectionString))
126         {
127             conn.Open(); // Open SQL Connection
128             using (SqlCommand cmd = new SqlCommand(query, conn))
129             {
130                 cmd.Parameters.AddWithValue("@tKey", tKey);
131                 cmd.Parameters.AddWithValue("@Domain_User", user.ToUpper());
132                 cmd.Parameters.AddWithValue("@User_Created", userName);
133                 cmd.Parameters.AddWithValue("@Form", Form);
134                 cmd.Parameters.AddWithValue("@Event", Event);
135                 cmd.Parameters.AddWithValue("@Notes", notes);
136                 cmd.ExecuteNonQuery();
137             }
138             conn.Close(); // Close SQL Connection
139         }
140         if (Event != "[UpdateElucidGDPR]")
141         {
142             lblLogBook.Text = $"Status: {DateTime.Now} - {Form} - {Event} - {notes};";
143         }
144     }
145     // Catch Errors
146     catch (Exception ex)
147     {
148         MessageBox.Show($"An error occurred: {ex.Message}");
149     }
150 }

```

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

```

156     // Get Customer Method
157     private void GetCustomer()
158     {
159
160         // Declare Variables
161         string customer = txbCustomer.Text;
162         string client = cbClient.Text;
163         string query = "[Suppressor_Get_Details] @Customer, @Client";
164         string isCustomer = null;
165
166         lblCustExist.Visible = false; // Hide Label
167
168         Cursor.Current = Cursors.WaitCursor; // Set Cursor to Eggtimer
169
170         try
171         {
172             // Execute SQL Command
173             using (SqlConnection conn = new SqlConnection(connectionString))
174             {
175                 conn.Open(); // Open SQL Connection
176                 using (SqlCommand cmd = new SqlCommand(query, conn))
177                 {
178                     cmd.Parameters.AddWithValue("@Customer", customer);
179                     cmd.Parameters.AddWithValue("@Client", client);
180                     using (SqlDataReader reader = cmd.ExecuteReader())
181                     {
182                         while (reader.Read())
183                         {
184                             isCustomer = reader["Customer"].ToString();
185                             txbName.Text = reader["Name"].ToString();
186                             txbAddress.Text = reader["Address"].ToString();
187                             txbCity.Text = reader["City"].ToString();
188                             txbPostcode.Text = reader["Postcode"].ToString();
189                             txbEmail.Text = reader["Email"].ToString();
190                             txbTelephone.Text = reader["Telephone"].ToString();
191                         }
192                     }
193                 }
194             }
195         }
196         catch (Exception ex)
197         {
198             MessageBox.Show($"An error occurred: {ex.Message}");
199             LogBook("ERROR", "[MainForm]", "[GetDetails]", $"FAILED ({ex.Message})");
200         }
201         finally
202         {
203             Cursor.Current = Cursors.Default; // Set Cursor to Default
204         }
205
206         // Check if customer exists
207         if (string.IsNullOrEmpty(isCustomer) && !string.IsNullOrEmpty(customer))
208         {
209             lblCustExist.Visible = true;
210             lblCustExist.Text = "Customer Not Found!";
211             cust = 0;
212             txbCustomer.Focus();
213             txbCustomer.SelectAll();
214             MessageBox.Show("Customer Not Found!");
215             return;
216         }
217         else
218         {
219             lblCustExist.Visible = false;
220             lblCustExist.Text = "";
221             cust = 1;
222         }
223     }
224 }
225

```

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

```

227     // Duplicate Checker
228     private void DupeChecker(string tKey)
229     {
230         // Declare Variables
231         string dupeCheck = $"SELECT COUNT(*) FROM Suppressor_Details WHERE Customer = @Customer AND Client = @Client";
232         string customer = txbCustomer.Text;
233         string client = cbClient.SelectedItem.ToString();
234         string name = txbName.Text;
235
236         // Execute SQL Command
237         using (SqlConnection conn = new SqlConnection(connectionString))
238         {
239             conn.Open(); // Open SQL Connection
240
241             using (SqlCommand cmd = new SqlCommand(dupeCheck, conn))
242             {
243                 // Count Number of Customers the same as Parameter
244                 cmd.Parameters.AddWithValue("@Customer", customer);
245                 cmd.Parameters.AddWithValue("@Client", client);
246                 int count = (int)cmd.ExecuteScalar();
247
248                 // Evaluate Results
249                 if (count > 0)
250                 {
251                     lblDetails.Visible = true;
252                     lblDetails.ForeColor = Color.FromArgb(252, 85, 79);
253                     lblDetails.Text = $"Warning! Duplicate Record - {customer} @ {DateTime.Now}";
254                     LogBook(tKey, "[MainForm]", "[InsertDetails]", $"Warning! Duplicate Record - ({client} - {customer})");
255                 }
256                 else
257                 {
258                     lblDetails.Visible = true;
259                     lblDetails.ForeColor = Color.FromArgb(46, 72, 89);
260                     lblDetails.Text = $"Record Accepted - {name} @ {DateTime.Now}";
261                     LogBook(tKey, "[MainForm]", "[InsertDetails]", $"Record Accepted - ({client} - {name})");
262                 }
263             }
264         }
265     }
266 }
```

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

```

267     // Create tKey
268     private string CreateTKey()
269     {
270         string name = txbName.Text;
271         string tKeySufix = name.Length >= 2 ? name.Substring(0, 2).ToUpper() : name.ToUpper();
272         string tKeyTimeStamp = DateTime.Now.ToString("yyyyMMddHHmmssfff");
273         string tKey = tKeyTimeStamp + tKeySufix;
274         return tKey;
275     }
276 }
```

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

```

277 // Update Elucid GDPR Flags
278 private void UpdateElucidGDPR()
279 {
280     LogBook("", "[MainForm]", "[UpdateElucidGDPR]", "Started");
281
282     string getDatabase = "SELECT RTRIM([Server]) as [Server],RTRIM([Database]) as [Database] FROM Suppressor_Parameters WHERE [Description] = @Client";
283     string getQuery = "SELECT String_1, String_2 FROM Suppressor_Parameters WHERE ID = '801'";
284     string customer = tbCustomer.Text;
285     string user = userName;
286     string database = null;
287     string server = null;
288     string queryUpdate = null;
289     string queryGDPRUpdate = null;
290
291     client = cbClient.SelectedItem.ToString();
292
293     try
294     {
295         // Get Database & Query Info
296         using (SqlConnection conn = new SqlConnection(connectionString))
297         {
298             conn.Open(); // Open SQL Connection
299
300             using (SqlCommand cmd = new SqlCommand(getDatabase, conn))
301             {
302                 cmd.Parameters.AddWithValue("@Client", client);
303                 using (SqlDataReader reader = cmd.ExecuteReader())
304                 {
305                     while (reader.Read())
306                     {
307                         server = reader["Server"].ToString();
308                         database = reader["Database"].ToString();
309                     }
310                 }
311             }
312
313             using (SqlCommand cmd = new SqlCommand(getQuery, conn))
314             {
315                 cmd.Parameters.AddWithValue("@Client", client);
316                 using (SqlDataReader reader = cmd.ExecuteReader())
317                 {
318                     while (reader.Read())
319                     {
320                         queryUpdate = reader["String_1"].ToString();
321                         queryGDPRUpdate = reader["String_2"].ToString();
322                     }
323                 }
324             }
325         }
326     }
327     catch (Exception ex)
328     {
329         MessageBox.Show($"An error occurred: {ex.Message}");
330         LogBook("ERROR", "[MainForm]", "[UpdateElucidGDPR]", $"FAILED! Error retrieving database info: ({ex.Message})");
331     }
332
333     // Populate new connection string
334     string updateConnectionString = $"Server={server};Database={database};Integrated Security=True;Encrypt=False;";
335
336     try
337     {
338         // Update cust tables
339         using (SqlConnection conn = new SqlConnection(updateConnectionString))
340         {
341             conn.Open();
342             using (SqlCommand cmd = new SqlCommand(queryUpdate, conn))
343             {
344                 cmd.Parameters.AddWithValue("@Customer", customer);
345                 cmd.Parameters.AddWithValue("@User", userName);
346                 cmd.ExecuteNonQuery();
347             }
348
349             LogBook("", "[MainForm]", "[UpdateElucidGDPR]", $"Update Cust Executed ({customer})");
350
351             using (SqlCommand cmd = new SqlCommand(queryGDPRUpdate, conn))
352             {
353                 cmd.Parameters.AddWithValue("@Customer", customer);
354                 cmd.Parameters.AddWithValue("@User", userName);
355                 cmd.ExecuteNonQuery();
356             }
357             conn.Close();
358
359             LogBook("", "[MainForm]", "[UpdateElucidGDPR]", $"Update GDPR Executed ({customer})");
360
361         }
362     }
363     catch (Exception ex)
364     {
365         MessageBox.Show($"An error occurred: {ex.Message}");
366         LogBook("ERROR", "[MainForm]", "[UpdateElucidGDPR]", $"FAILED! Error Updating Elucid tables ({ex.Message})");
367     }
368
369     LogBook("", "[MainForm]", "[UpdateElucidGDPR]", "Finished");
370 }

```

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

```

374     // Update Elucid GOPR Flags
375     0 references
376     private void GetNonCustomer()
377     {
378         string getDatabase = "SELECT RTRIM([Server]) as [Server], RTRIM([Database]) as [Database], RTRIM(Site) as [Site] FROM Suppressor_Clients WHERE [Description] = @Client";
379         string query = "SELECT COUNT(customer) FROM cust WHERE postcode = @Postcode AND level_1 = @Site";
380         string client = null;
381         string customer = txtCustomer.Text;
382         string user = userName;
383         string database = null;
384         string server = null;
385         string postcode = txtPostcode.Text;
386         string site = null;
387
388         client = cbClient.SelectedItem.ToString();
389
390         try
391         {
392             // Get Database
393             using (SqlConnection conn = new SqlConnection(connectionString))
394             {
395                 conn.Open(); // Open SQL Connection
396
397                 using (SqlCommand cmd = new SqlCommand(getDatabase, conn))
398                 {
399                     cmd.Parameters.AddWithValue("@Client", client);
400                     using (SqlDataReader reader = cmd.ExecuteReader())
401                     {
402                         while (reader.Read())
403                         {
404                             server = reader["Server"].ToString();
405                             database = reader["Database"].ToString();
406                             site = reader["Site"].ToString();
407                         }
408                     }
409                 }
410             }
411             catch (Exception ex)
412             {
413                 MessageBox.Show($"An error occurred: {ex.Message}");
414                 LogBook("ERROR", "[MainForm]", "[GetNonCustomer]", $"FAILED! Error retrieving database info: ({ex.Message})");
415             }
416
417             // Populate new connection string
418             string updateConnectionString = $"Server={server};Database={database};Integrated Security=True;Encrypt=False;";
419
420             try
421             {
422                 // Update cust tables
423                 using (SqlConnection conn = new SqlConnection(updateConnectionString))
424                 {
425                     conn.Open();
426                     using (SqlCommand cmd = new SqlCommand(query, conn))
427                     {
428                         cmd.Parameters.AddWithValue("@Postcode", postcode);
429                         cmd.Parameters.AddWithValue("@site", site);
430                         int count = (Int)cmd.ExecuteScalar();
431
432                         if (count > 0 && string.IsNullOrEmpty(customer) && !string.IsNullOrEmpty(postcode))
433                         {
434                             MessageBox.Show($"{count} Customer(s) were found with this postcode. \nPlease check if this customer exists on Elucid.");
435                             return;
436                         }
437                     }
438                 }
439             }
440             catch (Exception ex)
441             {
442                 MessageBox.Show($"An error occurred: {ex.Message}");
443                 LogBook("ERROR", "[MainForm]", "[GetNonCustomer]", $"FAILED! Error searching Elucid tables ({ex.Message})");
444             }
445         }
446     }

```

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

```

447     // Insert Details Method
448     private void InsertDetails(string tKey)
449     {
450
451         // Declare Variables
452         string query = "[Suppressor_Insert_Details] @tKey, @Client, @Customer, @Name, @Address, @City, @Postcode, @Email, @Telephone, @Reason, @User";
453         string client = null;
454         string customer = null;
455         string name = txbName.Text;
456         string address = txbAddress.Text;
457         string city = txbCity.Text;
458         string postcode = txbPostcode.Text;
459         string email = txbEmail.Text;
460         string telephone = txbTelephone.Text;
461         string reason = null;
462         string user = userName;
463
464         // Update customer string
465         if (cust == 1)
466         {
467             customer = txbCustomer.Text;
468         }
469         else if (cust == 0 || string.IsNullOrEmpty(customer))
470         {
471             customer = "";
472         }
473
474         // Set combo box variables
475         if (cbReason.SelectedItem != null)
476         {
477             reason = cbReason.SelectedItem.ToString();
478         }
479         else
480         {
481             Cursor.Current = Cursors.Default; // Set Cursor to Default
482             return;
483         }
484         if (cbClient.SelectedItem != null)
485         {
486             client = cbClient.SelectedItem.ToString();
487         }
488         else
489         {
490             Cursor.Current = Cursors.Default; // Set Cursor to Default
491             return;
492         }
493
494         DupeChecker(tKey); // Set detail label & Check for Duplicates
495
496         // Execute SQL Query
497         using (SqlConnection conn = new SqlConnection(connectionString))
498         {
499             conn.Open(); // Open SQL Connection
500             using (SqlCommand cmd = new SqlCommand(query, conn))
501             {
502                 cmd.Parameters.AddWithValue("@tKey", tKey);
503                 cmd.Parameters.AddWithValue("@Client", client);
504                 cmd.Parameters.AddWithValue("@Customer", customer);
505                 cmd.Parameters.AddWithValue("@Name", name);
506                 cmd.Parameters.AddWithValue("@Address", address);
507                 cmd.Parameters.AddWithValue("@City", city);
508                 cmd.Parameters.AddWithValue("@Postcode", postcode);
509                 cmd.Parameters.AddWithValue("@Email", email);
510                 cmd.Parameters.AddWithValue("@Telephone", telephone);
511                 cmd.Parameters.AddWithValue("@Reason", reason);
512                 cmd.Parameters.AddWithValue("@User", user);
513                 cmd.ExecuteNonQuery();
514
515                 if (!string.IsNullOrEmpty(customer) || customer != "")
516                 {
517                     UpdateElucidGDPR(); // Update Elucid GDPR Flags
518                 }
519
520                 ClearFields(); // Clear input fields
521             }
522         }
523     }
524 }
```

Here is the method used to insert the values entered on the Main Form into the Supresser\_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 Supressor\_Insert\_Details.

### Main Form - ClearFields

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

```

691     // Set Character Limit for Customer
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
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
722      1 reference
723      private void txbAddress_TextChanged(object sender, EventArgs e)
724      {
725          string addressString = txbAddress.Text;
726          int addressLength = addressString.Length;
727
728          if (addressLength > 255)
729          {
730              MessageBox.Show("Address is too long.");
731              txbAddress.Text = addressString.Substring(0, Math.Min(addressString.Length, 255));
732              txbAddress.SelectAll();
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
737      1 reference
738      private void txbCity_TextChanged(object sender, EventArgs e)
739      {
740          string cityString = txbCity.Text;
741          int cityLength = cityString.Length;
742
743          if (cityLength > 48)
744          {
745              MessageBox.Show("City is too long.");
746              txbCity.Text = cityString.Substring(0, Math.Min(cityString.Length, 48));
747              txbCity.SelectAll();
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
752   □       1 reference
753   □       private void txbPostcode_TextChanged(object sender, EventArgs e)
754   □       {
755   □       string postcodeString = txbPostcode.Text;
756   □       int postcodeLength = postcodeString.Length;
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   □       }
763   □       return;
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
767   □       1 reference
768   □       private void txbEmail_TextChanged(object sender, EventArgs e)
769   □       {
770   □       string emailString = txbEmail.Text;
771   □       int emailLength = emailString.Length;
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   □   }
780

```

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
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          }
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

```

796      private void cbClient_TextChanged(object sender, EventArgs e)
797      {
798          ClearFields();
799      }
800

```

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
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
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

```

828      // Accept Button Event
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
837          lblCustExist.Visible = false; // Hide Label
838
839          if (cbClient.SelectedItem == null)
840          {
841              Cursor.Current = Cursors.Default; // Set Cursor to Default
842              MessageBox.Show("Client is mandatory, Please choose a Client");
843              cbClient.Focus();
844              return;
845          }
846          else if (string.IsNullOrEmpty(name) && name != "")
847          {
848              Cursor.Current = Cursors.Default; // Set Cursor to Default
849              MessageBox.Show("Name is mandatory, Please enter a Name");
850              txbName.Focus();
851              return;
852          }
853          else if (string.IsNullOrEmpty(email) && string.IsNullOrEmpty(address))
854          {
855              Cursor.Current = Cursors.Default; // Set Cursor to Default
856              MessageBox.Show("An Email or Postal Address is mandatory.");
857              txbAddress.Focus();
858              return;
859          }
860          else if (email != "" && !email.Contains("@") && !email.Contains(".")) 
861          {
862              Cursor.Current = Cursors.Default; // Set Cursor to Default
863              MessageBox.Show("Email address invalid.");
864              txbEmail.Focus();
865              txbEmail.SelectAll();
866              return;
867          }
868          else if (cbReason.SelectedItem == null)
869          {
870              Cursor.Current = Cursors.Default; // Set Cursor to Default
871              MessageBox.Show("Reason is Mandatory, Please enter a Reason");
872              cbReason.Focus();
873              return;
874          }
875          else if (cbClient.SelectedItem == null)
876          {
877              Cursor.Current = Cursors.Default; // Set Cursor to Default
878              MessageBox.Show("Client is Mandatory, Please enter a Client");
879              cbClient.Focus();
880              return;
881          }
882          // If all good, do the insert
883          else
884          {
885              Cursor.Current = Cursors.WaitCursor; // Set Cursor to EggTimer
886
887              try
888              {
889                  InsertDetails(tKey); // Invoke InsertDetails Method
890
891                  txbCustomer.Focus(); // Return Focus to customer Text Box
892              }
893              catch (Exception ex) // Catch Errors
894              {
895                  MessageBox.Show($"An error occurred: {ex.Message}");
896                  LogBook($"{tKey}-ER", "[MainFrom]", "[InsertDetails]", $"FAILED ({ex.Message})");
897              }
898              finally
899              {
900                  Cursor.Current = Cursors.Default; // Set Cursor to Default
901              }
902          }
903      }

```

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

```

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
918 1 reference
919 private void MainFrom_KeyDown(object sender, KeyEventArgs e)
920 {
921     // ctrl + G
922     if (e.Control && e.KeyCode == Keys.G)
923     {
924         txbName.Text = "";
925         btnClear_Click(sender, e);
926     }
927 
928     // ctrl + S
929     if (e.Control && e.KeyCode == Keys.S)
930     {
931         btnAccept_Click(sender, e);
932     }
933 
934     // Esc
935     if (e.KeyCode == Keys.Escape)
936     {
937         btnExit_Click(sender, e);
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
941 1 reference
942 private void txbTelephone_KeyPress(object sender, KeyPressEventArgs e)
943 {
944     if (!char.IsDigit(e.KeyChar) && !char.IsControl(e.KeyChar) && e.KeyChar != '+')
945     {
946         MessageBox.Show($"Character: '{e.KeyChar}' Not Accepted Here. \nOnly numerical values accepted for Telephone.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
947         e.Handled = true;
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

```

949 1 reference
950 private void txbEmail_KeyPress(object sender, KeyPressEventArgs e)
951 {
952     if (e.KeyChar == ' ' || e.KeyChar == '"' || e.KeyChar == '\'' || e.KeyChar == '\n' || e.KeyChar == '\t')
953     {
954         MessageBox.Show($"Character: '{e.KeyChar}' Not Accepted Here.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
955         e.Handled = true;
956     }
957 }

```

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

```

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

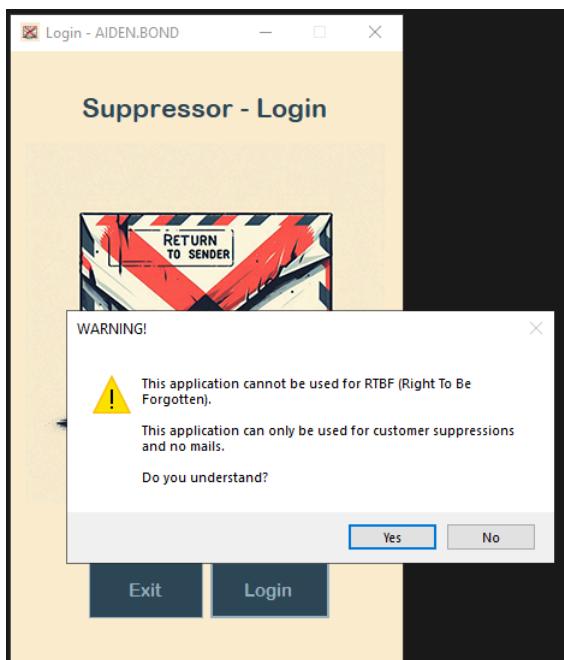
## 5.0.1 - Logging Onto the Application



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.

The screenshot shows the 'Suppressor - Home' application window. At the top, it displays 'Home - AIDEN.BOND' and the user 'AIDENB'. The main area contains several input fields: 'Client\*' (a dropdown menu), 'Customer' (text box), 'Name\*' (text box), 'Address' (text box), 'City' (text box) and 'Postcode' (text box), 'Email' (text box) and 'Telephone' (text box), and a 'Reason\*' dropdown menu. At the bottom are three buttons: 'Exit', 'Clear', and 'Accept', followed by a 'LogBook' link.

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 screenshot shows the 'Suppressor - Home' application window with the 'Client\*' dropdown menu open. The dropdown list contains the following options: Museum Selection - MAIN, Culture Vulture - MAIN, Pia - MAIN, Hamilton Direct, Otter - Blue Cross, Otter - Cats Protection, Otter - Dogs Trust, Otter - Diabetes UK, Muck Munchers, RSPCA BRANCH, Daytimers Europe LTD, and Sarah Raven. The 'Museum Selection - MAIN' option is highlighted. Below the dropdown are other input fields: 'Customer' (text box), 'Name\*' (text box), 'Address' (text box), 'City' (text box) and 'Postcode' (text box), 'Email' (text box) and 'Telephone' (text box), and a 'Reason\*' dropdown menu. At the bottom are three buttons: 'Exit', 'Clear', and 'Accept', followed by a 'LogBook' link.

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.

The screenshot shows the 'Suppressor - Home' application window with the 'Client\*' dropdown menu closed. The selected value 'Museum Selection - MAIN' is now displayed in the dropdown field. Below the dropdown are other input fields: 'Customer' (text box), 'Name\*' (text box), 'Address' (text box), 'City' (text box) and 'Postcode' (text box), 'Email' (text box) and 'Telephone' (text box), and a 'Reason\*' dropdown menu. At the bottom are three buttons: 'Exit', 'Clear', and 'Accept', followed by a 'LogBook' link.

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

### 5.0.3 - Completing the required Fields

The screenshot shows the 'Suppressor - Home' application window. The 'Client\*' dropdown is set to 'Museum Selection - MAIN'. The 'Customer' field contains the value 'TEST'. The 'Name\*' field is empty. The 'Address' field is empty. The 'City' and 'Postcode' fields are empty. The 'Email' and 'Telephone' fields are empty. The 'Reason\*' dropdown is empty. At the bottom are 'Exit', 'Clear', and 'Accept' buttons, and a 'LogBook' link.

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.

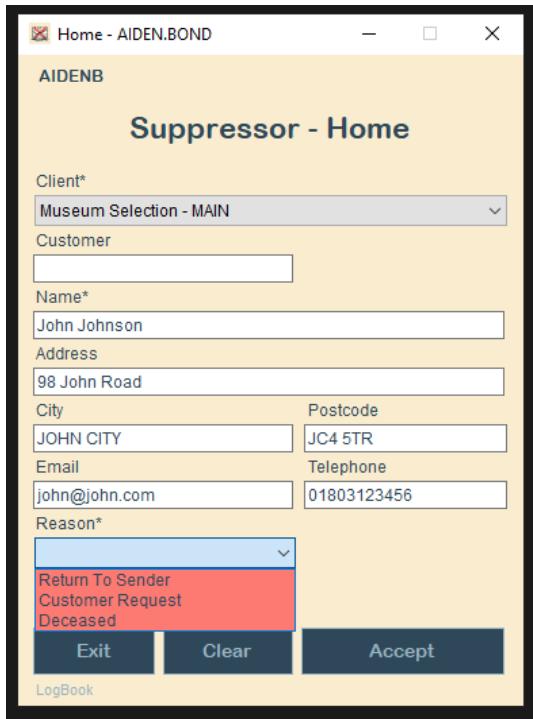
If the customer does exist in the client's database, the application will get that customer's data to pre-populate the rest of the fields, providing the data is available.

The screenshot shows the 'Suppressor - Home' application window. The 'Client\*' dropdown is set to 'Museum Selection - MAIN'. The 'Customer' field contains the value 'TEST'. The 'Name\*' field contains 'Test Testerton'. The 'Address' field contains 'Woodview Road'. The 'City' field contains 'PAIGNTON' and the 'Postcode' field contains 'TQ4 7SR'. The 'Email' field contains 'jimmy.trimble@whistl.co.uk' and the 'Telephone' field is empty. The 'Reason\*' dropdown is empty. At the bottom are 'Exit', 'Clear', and 'Accept' buttons, and a 'LogBook' link.

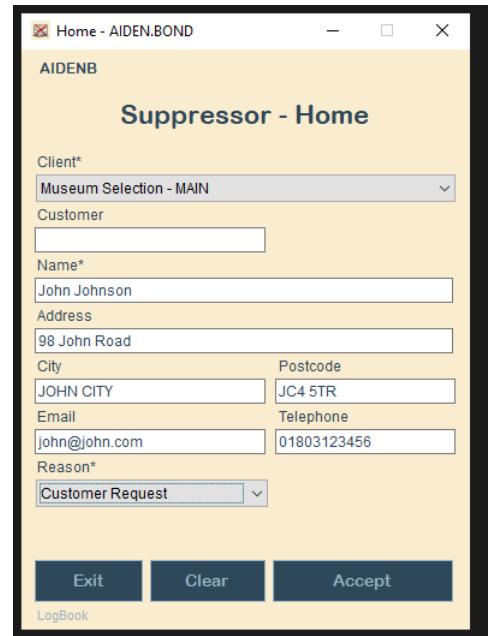
The screenshot shows the 'Suppressor - Home' application window. The 'Client\*' dropdown is set to 'Museum Selection - MAIN'. The 'Customer' field is empty. The 'Name\*' field contains 'John Johnson'. The 'Address' field contains '98 John Road'. The 'City' field contains 'JOHN CITY' and the 'Postcode' field contains 'JC4 5TR'. The 'Email' field contains 'john@john.com' and the 'Telephone' field contains '01803123456'. The 'Reason\*' dropdown is empty. At the bottom are 'Exit', 'Clear', and 'Accept' buttons, and a 'LogBook' link.

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

AIDENB

**Suppressor - Home**

Client\*  
Museum Selection - MAIN

Customer

Name\*  
John Johnson

Address  
98 John Road

City	Postcode
JOHN CITY	JC4 5TR

Email	Telephone
john@john.com	01803123456

Reason\*  
Customer Request

Exit    Clear    Accept

LogBook

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

AIDENB

**Suppressor - Home**

Client\*  
Museum Selection - MAIN

Customer

Name\*  
John Johnson

Address

City	Postcode

Email	Telephone

Reason\*  
Customer Request

Record Accepted - John Johnson @ 08/02/2024 14:09:17

Exit    Clear    Accept

Status: 08/02/2024 14:09:17 - [MainForm] - [InsertDetails] - 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

The screenshot shows the 'Suppressor - Home' application window. It contains a form with the following fields:

- Client\*: Museum Selection - MAIN
- Customer: TEST
- Name\*: Test Testerton
- Address: Woodview Road
- City: PAIGNTON
- Postcode: TQ4 7SR
- Email: jimmy.trimble@whistl.co.uk
- Telephone: (empty)
- Reason\*: Customer Request

At the bottom, there is a status message: Record Accepted - John Johnson @ 08/02/2024 14:09:17 and three buttons: Exit, Clear, and Accept.

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

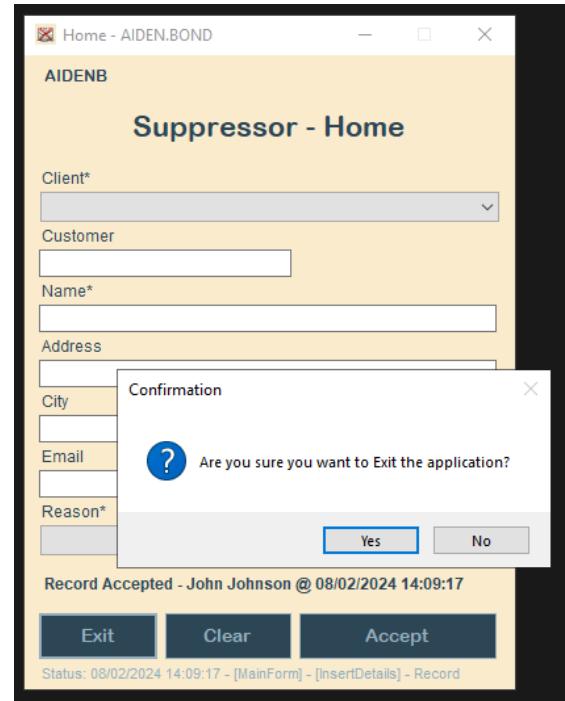
The screenshot shows the same 'Suppressor - Home' application window after the 'Clear' button was clicked. All data entry fields have been cleared, leaving them empty. The status message at the bottom remains the same: Record Accepted - John Johnson @ 08/02/2024 14:09:17 and the three buttons are still present.

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 reentries 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 -...\AIDEN.BOND (65)\* X SQLQuery11.sql -...\AIDEN.BOND (69)\* SQLQuery6.sql - ...\\AIDEN.BOND (177)\*

1 | 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.

SQLQuery12.sql -...\AIDEN.BOND (65)\* X SQLQuery11.sql -...\AIDEN.BOND (69)\* SQLQuery6.sql - ...\\AIDEN.BOND (177)\*

1 | SELECT\* FROM Suppressor\_Details WHERE CAST(DT\_Created as date) = CAST(GETDATE() as date) AND User\_Created = 'AIDENB'

100 %

Results Messages

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	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	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	20240118112612101AI	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	20240124115946567TE	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	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.