**Initial Design**

**Problem Decomposition**



The reason I am using a top-down design is to take each important part of my program and break it down into smaller parts to manage what I need to focus on a lot easier. This is linked to the computational method, Thinking Logically. This allows me to determine which part of the program needs to be loaded first and the sequence it needs to be executed in. This is also linked to Thinking Ahead, as I will be able to determine my inputs and outputs.

**Explanation of Each Module**

**Login**

The user will be presented with two textboxes at the start of the program, in which they must type in their username and password. The username and password will have to be validated by restricting the username and password to a certain number of characters. When the “Enter” Button is pressed, the program will check if the user’s input is exactly the same as a record in the Log-In database. If the details do not match, an error will be displayed as a label. Otherwise, the user will be takin into the main menu and the login page will be closed.

**Main Menu**

The main menu will present 4 options:

1. Letting the user edit the customer database
2. Letting the user edit the vehicle database
3. Letting the user hire a vehicle to a customer
4. Being able to exit the program

**Customer Database**

The user will be presented with 6 textboxes and 5 buttons, in which they will be enter details about the customer. After entering the details, the 5 buttons will determine what happens to the details. The user will be able to:

1. Add the details to the customer database
2. Delete existing records from the database
3. Search for customers based on their name
4. Update customer records in the database
5. Import/Refresh the customer records

**Vehicle Database**

The user will be presented with 6 textboxes and 5 buttons, in which they will be enter details about the vehicle. After entering the details, the 5 buttons will determine what happens to the details. The user will be able to:

1. Add the details to the vehicle database
2. Delete existing records from the database
3. Search for vehicle based on their name
4. Update vehicle records in the database
5. Import/Refresh the vehicle records

**Hire a Vehicle**

The user will be presented with 6 Buttons, 6 Text Boxes, a Tick Box and 2 Data-Grid-Views. The 6 buttons are for:

1. Searching for a vehicle
2. Searching for a customer
3. Refreshing the vehicle database
4. Refreshing the customer database
5. Calculating the total amount needed to pay
6. Generating an invoice

The 6 Textboxes are for:

1. Searching for a customer
2. Searching for a vehicle
3. Constant read-only value of VAT
4. Entering the number of days, you want to rent the vehicle for
5. Read-only value of the cost per day
6. Read-only value of the total amount including VAT

There will be a tick box for whether the customer wants insurance in the deal and there will also be 2 Data-Grid-Views that are there to link customer records with vehicle records.

**Invoice**

The user will be presented with 3 buttons:

1. Print Invoice
2. Email Invoice
3. Exit Program

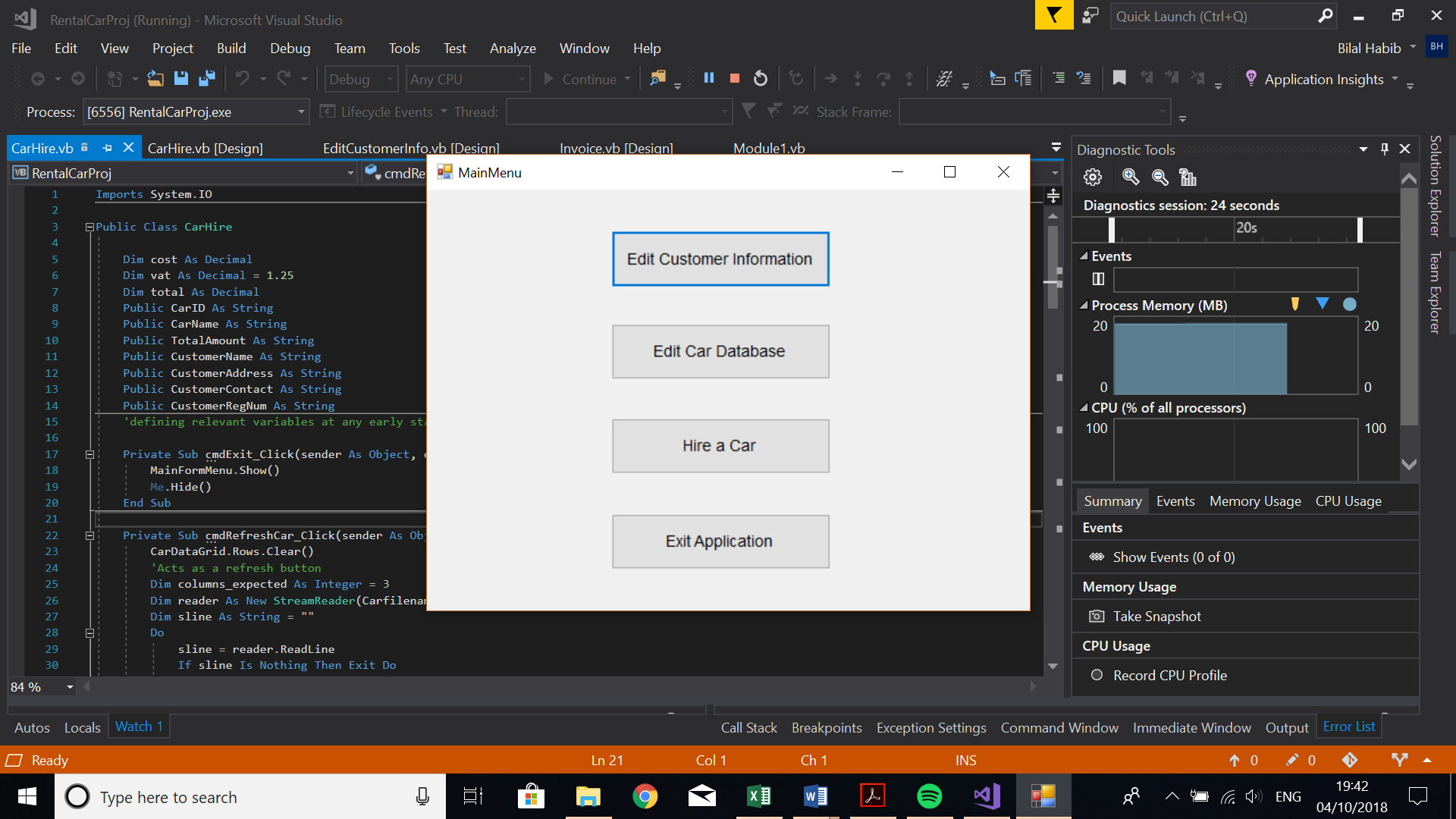
In the invoice, the user will be able to see:

1. The addresses of the company
2. The email of the company
3. The customer’s details
4. The vehicle’s details
5. The date of the transaction

**Usability**

I will need to make my program user friendly so that the user does not have difficulties when using my program. I will determine the extent of my program’s usability using numerous factors.

**Navigation**



I have added a main menu to make it easier for the user to navigate to the different parts of the program that are labeled with obvious and clear names. I have also added an exit button in every module and class so that the user will be able to navigate back to the main menu and therefore have access to the other areas of the program. This will be less time consuming as the user will not have to shut down the whole program to navigate to the other modules.

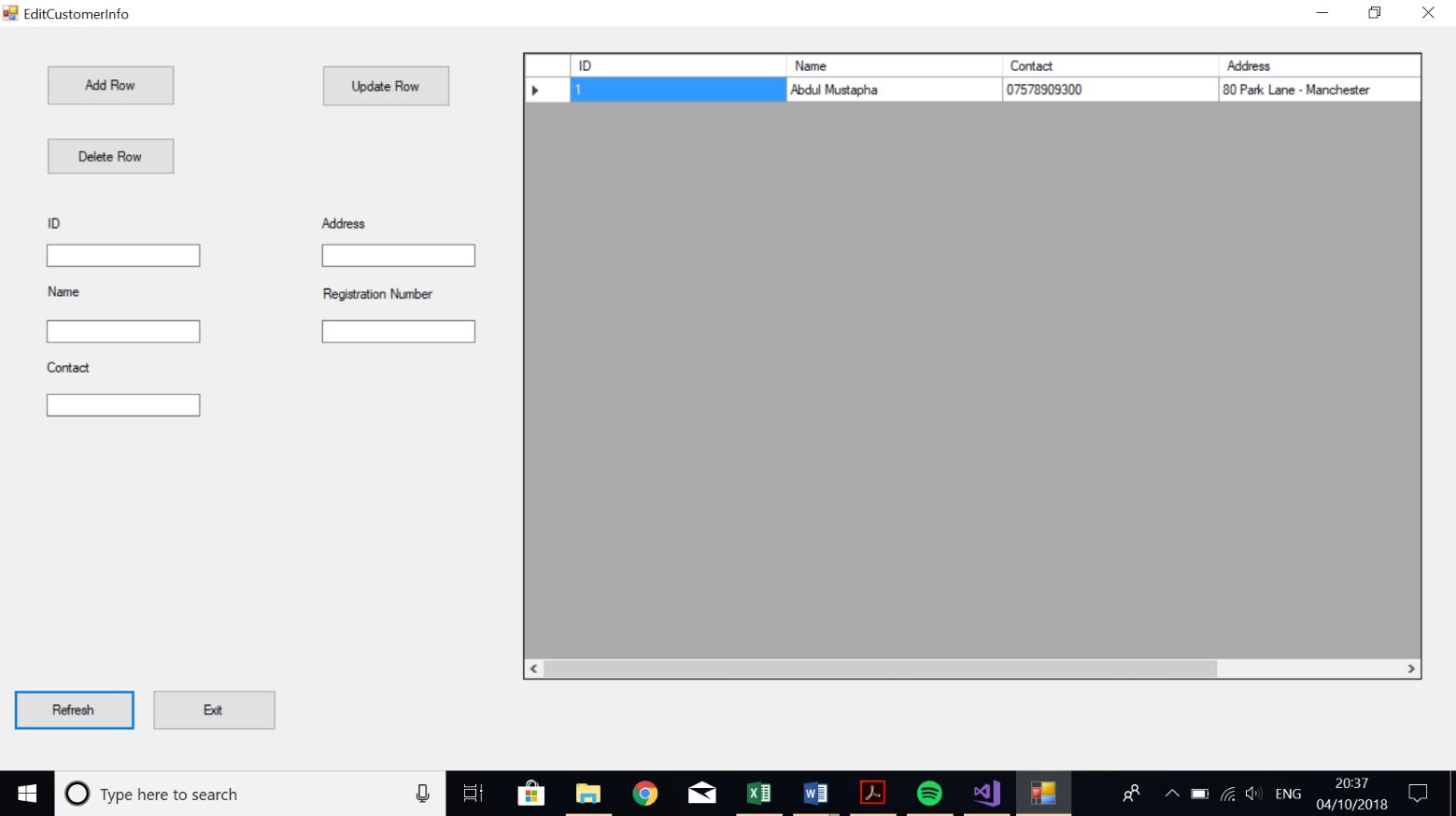
To benefit from my program, the user must be able to know how to navigate their way through my program. I can make this easier for them by making the layout of the screens simplistic and easy to read so that the user does not get confused the first few times they use my program Additionally, I will make the text quite large so that even users who have trouble reading text on monitors can read without difficulty. Above every text box there will be a label telling them what information needs to go into that text box. This is to avoid confusion and so that the user does not put their password into the username text box and vice versa, rendering them unable to log in and use my program.

**Consistency**

When handling my data (for example, importing and exporting information to the databases), the format will need to be the same to prevent errors and crashing. To do this I will need to ensure the user inputs the correct data type when appropriate so that the program can understand what the user has inputted and prevent errors/crashes. Moreover, consistency somewhat ties into navigation as I will make all controls the same colour so that the user knows what a button is and what a label is, etc. Doing this will make navigating my program easier for the user, allowing them to benefit from my program.

**User Feedback**

I will allow some of my stakeholders to test my program so that they can give me feedback on any adaptations I can make or any trouble they came across when using my program. This will allow me to tailor my program to suit the users’ needs so that they can benefit from my program more and have an easier time when using it.

**Visual Clarity**

To make this visually better and a more enjoyable experience, I have made the buttons big with large text inside them to be read more easily. I have also aligned the buttons in the same positions and they are all the same size. I have also included a Data-Grid-View, this is so that when the user is editing the data. It can be visually seen better, as it portrays a data grid rather than using something ugly like a list-box. I will also include a function where, whenever the user searches for a record in the database, it highlights the record for it to be seen more clearly.

**Efficiency**

Making my program as efficient as possible will keep the size of the file at a minimum, allowing for a faster download when the user comes to download/install my program. Keeping the file size low also means there will be less code. This will make my program faster as there will be less code for the computer to process as well as reducing the likelihood of errors due to the reduced amount of code. This aids usability as the user will not have to wait to install my program and use it whilst calculations are being performed. This means that the user will not be wasting time which could be better spent revising. If my program is slow during runtime, then this decreases the usability of my program as they may have better, more efficient methods of revising. This will also reduce the total number of users and in turn, reducing the success of my program.

**Error Prevention**

Preventing errors will mean that the user will be unable to break/crash my program, allowing them to benefit from the program. To do this, I will need to consider when and where the user may be able to cause an error (for example, typing in a letter when there should be a number). Preventing errors from occurring will increase the usability of my program as the program will be unlikely to crash, meaning the user will not lose any unsaved progress/amendments.

**Key Variables and Data Structures**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Type | Explanation | Validation | Justification |
| Class: Log-In | | | | |
| Username | String | Holds staff’s username | Must be minimum 6 characters | Needed to differentiate from other staff |
| Password | String | Holds staff’s password | Must be minimum 9 characters | Need to restrict others from unauthorised access |
| NumOfStaffRecords | Integer | Holds amount of staff records | None | Needed to track of number of customer records |
| StaffFileName | String | Holds the filename of the staff database | None | Needed to avoid repetitive code |
| Class: Customer-Info | | | | |
| ID | Integer | Holds customer’s unique identifier | Must not repeat | Needed to link a customer to a vehicle |
| Name | String | Holds customer’s name | Must be less than 30 characters | Helps identify a customer |
| ContactNumber | Integer | Hold customer’s contact number | Must be 11 characters | Needed to contact a customer |
| Address | String | Hold’s customer’s home address | None |  |
| NumOfCustomerRecords | Integer | Holds amount of customer records | None | Needed to track number of vehicle records |
| CustomerFileName | String | Holds the filename of the customer database | None | Needed to avoid repetitive code |
| Class: Vehicle-Info | | | | |
| ID | Integer | Holds vehicle’s unique identifier | Must not repeat | Needed to link a customer to a vehicle |
| Name | String | Holds vehicle’s name | Must be less than 30 characters | Helps identify a vehicle |
| CostPerDay | Float | Holds the cost of the vehicle per day | None | Needed to calculate total amount |
| Class: Vehicle-Hire | | | | |
| NoOfDays | Integer | Hold the numbers of days to be rented for | None | Needed to calculate total amount |
| VAT | Float | Constant value (e.g. 25%) | None | Needed to calculate total amount |
| TotalAmount | Float | Holds the total amount (including VAT) that the customer needs to pay | None | Needed so that the customer knows how much he/she needs to pay |

**Algorithms**

Before developing my program, the main algorithms that will be crucial to allowing my program to function need to be created. This will make it easier to develop my program when I come to do so.

I will need the following algorithms:

• Login Button

• Add Record

• Delete Record

• Linear Search

• Update Record

• Total Price for Rental

**Login pseudocode:**

01 Open Log-In Database

02 If (Username\_Input = Username\_From\_Database) AND

(Password\_Input = Password\_From\_Database):

03 Load Menu Page

04 Close Log-In Database

05 Else:

06 Output “Login Failed”

07 End If

**Add pseudocode:**

01 Open Database

02 Declare Row = ID + Name + Address + ContactNumber

03 Output Row To Database

04 Close Database

**Delete pseudocode:**

01 Open Database

02 Declare list

03 Read All Lines in Database

04 if Database contains TextboxID

05 Do nothing

06 else

07 list.Add(Line)

08 End IF

09 Close Database

**Search pseudocode:**

01 Open Database

02 found = False​

03 WHILE (found == False):​

04 IF search\_item == array[counter]​

05 DISPLAY “Item at position”

06 found = True​

07 ELSE:​

08 position = position + 1

09 End IF

10 Close Database

**Update pseudocode:**

The update button will just be the delete button code followed by the add button code:

01 Open Database

02 Declare list

03 Read All Lines in Database

04 if Database contains TextboxID

05 Do nothing

06 else

07 list.Add(Line)

08 End IF

09 Declare list

10 if Database contains TextboxID

11 Do nothing

12 else

13 list.Add(Line)

14 End IF

15 Close Database

**Total Price For Rental:**

01 If Insurance is selected:

02 Total = (CostPerDay \* NoOfDays) \* 1.25

03 Else

04 Total = CostPerDay \* NoOfDays

05 End If

06 Output Total

**Post Development Test Data**

Login Form:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Number | What Testing | Input | Expected Result | Justification |
| 1 | User Login | Username: “Admin”  Password: “AdminPass” | User is told they are logged in and taken to the main menu | The login system needs to be tested post-development to ensure that I know the user can login successfully. This is important because for the user to take full advantage for my program, they must be able to login, so that the program can distinguish between users and display relevant information. |
| 2 | User login | Username: “Admin” Password: “adminpaass” | User is told their username and password do not match | “” |
| 3 | Getting login details wrong and then entering them correctly | First Entry  Username: “aDmin”  Password: “adpass”  Second Entry:  Username: “Admin”  Password: “AdminPass” | User is told that their attempt to login was unsuccessful for first entry but when they enter it correctly the second time, they are told they have logged in successfully and taken to the main menu | This needs to be tested so that the user is able to login successfully after entering their username and password incorrectly in previous tries, otherwise they would need to restart the program. This would waste time and could become a tedious task. |

Customer Database Form:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Number | What Testing | Input | Expected Result | Justification |
| 4 | Adding a record | **ID:** 0  **Name**: Bilal Habib  **Contact:** 07123456789  **Address:** 1 Baker Street | The record (customer) gets added to the csv file | This is so that the business can keep track of customers and their purchases |
| 5 | Adding a record | **ID:** 0  **Name**: Bilal Habib  **Contact:**  **Address:** 1 Baker Street | Due to a blank textbox, the record should not be added to the database and the user will get a message saying that a textbox is empty | This is to avoid errors and to make sure that every record gets imported from the csv file |
| 6 | Adding a record | **ID:** bilal  **Name**: Bilal Habib  **Contact:** 07123456789  **Address:** 1 Baker Street | Due to the text in the ID textbox being a string instead of an integer, the record should not be added to the csv file | This is to avoid errors in which when it comes to linking a customer ID with a vehicle ID, the program recognises which customer we’re dealing with |
| 7 | Adding a record | **ID:** 0  **Name**: Bilal Habib  **Contact:** 071234  **Address:** 1 Baker Street | The record should not be added due to the contact number not being 11 numbers long | This is to avoid situations in which the company would want to contact the customer and would fail in doing so due to missing information |
| 8 | Deleting a record | **ID:** 0 | The record in the csv file with ID number 0 should get removed from the file | This is to remove a customer who no longer wants their data to be in the business’s database |
| 9 | Deleting a record | **ID:** b | As all ID’s are integers, none of the records in the csv file should get deleted | This is to avoid errors and missing customers |
| 10 | Searching for a record | **ID:** 0 | The data-grid-view should be cleared and only show records that have an ID of 0 (which should only be one record) | This gives the user the option to search for customers based on their ID |
| 11 | Searching for a record | **Name:** Bilal Habib | The data-grid-view should be cleared and only show records that have a name of Bilal Habib | This gives the user the option to search for customers based on their name |
| 12 | Searching for a record | **Contact:** 07123456789 | The data-grid-view should be cleared and only show records that have a contact number of 07123456789 | This gives the user the option to search for customers based on their contact number |
| 13 | Searching for a record | **Address:** 1 Baker Street | The data-grid-view should be cleared and only show records that have an address of 1 Baker Street | This gives the user the option to search for customers based on their home address |
| 14 | Searching for a record | **ID:** 0  **Name**: Bilal Habib  **Contact:** 07123456789  **Address:** 1 Baker Street | The data-grid-view should be cleared and should only records that have:  **ID**: 0 OR  **Name**: Bilal Habib OR  **Contact**: 07123456789 OR  **Address**: 1 Baker Street | This is to give the user the option to search for any of the customer’s details. Meaning they can search for a customer based on their ID, name, contact number or address |
| 15 | Editing a record | **ID:** 0 | The program will delete the record with ID 0 in the database and add it back in with the textboxes values we’ve changed | This is to change a customer’s piece of information so that the business is up to date with the customers and does not have missing information |
| 16 | Refreshing the data-grid-view | **none** | The data-grid-view should be filled with records from the csv file | This is to be able to see the customers information and therefore make appropriate changes |
| 17 | Exit Button | **none** | The program should close the customer database form and open up the main menu | This is to ease navigation and avoid the user having to restart the program |

Vehicle Hire Form:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Number | What Testing | Input | Expected Result | Justification |
| 18 | Calculating the rental price | No. of days: 2d | It should not generate the total price and produce a message due to it being a string and not a decimal number | The total price will not be able to be calculated due to invalid input and therefore the customer will get a false invoice |
| 19 | Calculating the rental price | No. of days: 20.67 | The total price should be calculated with no errors | To avoid errors and provide a suitable invoice |
| 20 | Exit Button | None | The program returns back to the main menu | This is to ease navigation and avoid the user having to restart the program |

Invoice Form:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Number | What Testing | Input | Expected Result | Justifications |
| 21 | Print Button | None | The program prints the invoice | This is so that the customer gets a copy of the invoice so that they can refer back to it |
| 22 | Email Button | None | The program emails the invoice to the user’s email account | This is to ensure that the customer gets a secure copy of the invoice |
| 23 | Exit Button | None | The program returns back to the main menu | This is to ease navigation and avoid the user having to restart the program |