6

AQA COMP4: The computing practical project

The Laptops Clinic Software coursework

Riddlesdown collegiate

By: Amanuel Yosief-Mussie

Contents:

[Analysis 3](#_Toc449432680)

[Definition, Investigation & Analysis 3](#_Toc449432681)

[Background to the computer based problem: 3](#_Toc449432682)

[Observation of Existing System 3](#_Toc449432683)

[Uses of the Current System 4](#_Toc449432684)

[Interview with Mo Ibrahim- Owner of The Laptops Clinic 4](#_Toc449432685)

[Programming Language Chosen 5](#_Toc449432686)

[Potential Solutions 6](#_Toc449432687)

[Proposed Solutions (Reading off the Chart) 7](#_Toc449432688)

[User needs and limitations 9](#_Toc449432689)

[Numbered general and specific objectives of the project 9](#_Toc449432690)

[Data Source & Data Destination 11](#_Toc449432691)

[Data Volumes 13](#_Toc449432692)

[Data Dictionary 14](#_Toc449432693)

[Data Flow Diagram 17](#_Toc449432694)

[Object Orientation planning (for Object Orientation) 19](#_Toc449432695)

[Entity Relationship Diagram 19](#_Toc449432696)

[Design 20](#_Toc449432697)

[The User’s Interface 20](#_Toc449432698)

[User Interface Design 0](#_Toc449432699)

[IOPS Chart 0](#_Toc449432700)

[Form Navigation Design 1](#_Toc449432701)

[Sample of planned SQL Queries 2](#_Toc449432702)

[Storage Media and Format 3](#_Toc449432703)

[Identification of suitable algorithms for data transformation, pseudocode of these algorithms 4](#_Toc449432704)

[Class definitions and details of object behaviours and methods 7](#_Toc449432705)

[UI sample of planned data capture and entry designs 10](#_Toc449432706)

[UI sample of planned valid output designs 14](#_Toc449432707)

[Description of measures planned for security and integrity of data 18](#_Toc449432708)

[Description of measures planned for system security 19](#_Toc449432709)

[Overall test strategy 19](#_Toc449432710)

[Navigation Testing Design 20](#_Toc449432711)

[Testing 21](#_Toc449432712)

[Input and Output Testing 21](#_Toc449432713)

[Black Box Testing (Images) 28](#_Toc449432714)

[Navigation Testing 86](#_Toc449432715)

[Maintence 86](#_Toc449432716)

[Form Navigation Overview 0](#_Toc449432717)

[Class Overview 0](#_Toc449432718)

[Class DataQueries: 1](#_Toc449432719)

[Class Hashing: 6](#_Toc449432720)

[Variables Class: 8](#_Toc449432721)

[Form Overview 9](#_Toc449432722)

[TemplateUC 9](#_Toc449432723)

[CreateSystem UC: 12](#_Toc449432724)

[Login Form 17](#_Toc449432725)

[Register UC 21](#_Toc449432726)

[ForgotUC 27](#_Toc449432727)

[MainForm Form 36](#_Toc449432728)

[CustomerUC 39](#_Toc449432729)

[AddCustomer Form 46](#_Toc449432730)

[MakeAnOrder 48](#_Toc449432731)

[Employee UC 51](#_Toc449432732)

[Update Employee Form 58](#_Toc449432733)

[Email Form 65](#_Toc449432734)

[Calendar UC 68](#_Toc449432735)

[ReceiptAndFinance UC 73](#_Toc449432736)

[Appraisal 78](#_Toc449432737)

[Comparison of Project Performance against Objectives 78](#_Toc449432738)

[Analysis of User Feedback 79](#_Toc449432739)

[Extensions 80](#_Toc449432740)

# Analysis

## Definition, Investigation & Analysis



**The Laptops Repair Clinic  
Owner: Mo Ibrahim  
Contacts:** 020 8688 4167

**Address:**

**20 Selsdon Road**

**South Croydon**  **CR2 6PA**

**Link:**

[www.thelaptopsclinic.co.uk/](http://www.thelaptopsclinic.co.uk/)

## Background to the computer based problem:

**During my first work experience, I was fortune enough to work with a computer repair and fix store called “The Laptops Clinic”. The Laptops Clinics is store based in Selsdon in Croydon, which first opened in 2009 by the entrepreneur Mo Ibrahim who only had £84 and an idea leading to a well earning store 6 years later.**

**The Laptops Clinic deals with repairing computer/laptop, updating and installing softwares and devices. Mo Ibrahim is looking to expand this business and buy more stores across London while also improving his services and providing more services to his customers.**

**Currently the software the Laptops Clinic uses is not bespoke and efficient. They are collection of different softwares;**

**Word office to write online receipts, Microsoft Outlook for customer’s details and contacting, Excel to handle and store finical date and charges.  
This isn’t very efficient and a stress as having to manoeuvre between programs each time by “saving, closing, opening another software, save and exit” each time they manoeuvre between programs and they also need to locate the right document which is hard as their folders consists of multiple folders and documents, ones that deal with customers, finance and extra documentations. This is a hassle and time is being wasted which can be used to deal with the customers instead and there is a risk of editing/accessing wrong documents.**

**To decrease this; The Laptop Clinic seeks to unite all the features of all the software they use into one which can deals with all the documents; it shall deal with the finance, customers and also handle receipts, contacting and also some extra features.**

## Observation of Existing System

Description of the Current System:

The current systems used by the TLC (The Laptops Clinic) is composed for three separate programs, each with their own purpose. Once a customer arrives and buys an offer or service from the TLC, their contact details e.g. Name, address, phone, email, is entered in the Microsoft Outlook as a new contact. Before buying the service/offer, the TLC switches to Excel to calculate the expense and charge, then also include this data into their finical earning report of the month. Once the expenses are calculated for that individual customer, the prices is added to the receipt form, which is a basic receipt template in a word office document or a paper receipt made on the spot. TLC will add the customer’s details and charges in the receipt form and will either email them with the attachment to their email address or print off in store and hand it to the customer.

|  |  |
| --- | --- |
| Advantages of the Current System | Disadvantages of the Current System |
| Each program is designed for their own specific task;  Word designed for making forms and document. Excel designed for calculation data and organizing figures. Outlook is designed for storing customer’s data, contacts, emailing with added features such as contacts, task. | Manoeuvring between each program is time consuming and is open to corruption. |
| All the three programs contain all the features required by the TLC and provide extra features. | The TLC requires not all the features. Bespoke software is required for the store to work efficiently. |
| Documents are all organized into folders.  E.g. Finical report folder into broken down into separate folders corresponding to each month of the year.  E.g. Finical report -> Months of 2015 -> September  While Receipts will contain, receipts of the year then the consumer.  e.g. Receipts-> Receipt Of 2015 -> “Mark Johnson’s Receipt” | It is rather time consuming and hard to find the right document inside this large collection of folders & files. Files and Folders are also not backed-up and are prone to corruption or even misuse. Securing files and folders would only make it even more time consuming.  Deleting one folder deletes all the other files and folders inside of it. |

## Uses of the Current System

The system is currently used and will be used (in the proposed system) by all the employees of The Laptops Clinic. There will be two employee roles; Administrator and Employees. Administrators have more administrator power e.g. accessing/permitting and deleting features while Employees will have the basic power e.g. adding customers, making receipts, emailing.

## Interview with Mo Ibrahim- Owner of The Laptops Clinic

**For my interview, I chose to speak with Mo Ibrahim, who is the owner of TLC. I believe he would be the best person to interview since their the owner of the TLC (The Laptops Clinic) and he was the one who proposed bespoke software to be developed and will be developed to meet the requirements of the store and his requirements also, thus his the best person to interview.**

**The method I chose to interview him was a straight “in person” interview which allows me to be fully interactive with him, ask him questions and be able to create a plan/design with him.   
An “in person” interview is far better than those by email or telephone since emails interviews require times to read , format a reply and send back then waiting for the reply which is rather long and less interactive.**

**Question: Who will be using the program?**

**Reply: All the employees of The Laptops Clinic stores will use the software. Each store will mostly consist four employees at max, thus all employees will need access and should be very secure. Only me, the owner will be able to create/delete an employee account, I should have access to all data and edit it however I cannot modify Employee due to Data protections.**

**Question: Who will have access to the program?**

**Reply: I require two types of access accounts; Administrator for the storeowner’s and Employee for the employees of The Laptops Clinic.  
None else expect those two accounts will have access to it and should very secure.  
Both will have different administrator access, for example, Employee can delete Customers but Employee cannot delete Administrators. Some of the features such adding an Employee will require an administrator however Employee can add customers e.g.**

**Question: What are the programs that The Laptops Clinic currently use?**

**Reply: Currently we use Microsoft Outlook for creating and storing customer details.**

**Excel to store finical reports of each month and store in a folder.  
Microsoft office word to produce receipts that we email to customers via outlook or print out.**

**Question: What are the issues associated with these programs that you currently use?**

**Reply: The main issue is that none of the programs that we use, are not linked or interconnected together, therefore we have to manually switch between each program, making sure that we saved and exited it correctly. There are few features that could be improved in the current programs for our likings and some feature we do not require at all, that is why we would like a bespoke software designed for all our stores, with the features we require and that is all.**

**Question: Any specifications that you require?**

**Reply: The Laptops Clinic requires a bespoke software for all our stores which consists all of the basic features of the softwares; Microsoft Outlook, Excel and Word office. E.g.  
Microsoft Outlook for editing, storing and searching customer’s data. I also require a built in calculation, calendar, sales report, notice board and emailing to customers.  
One feature that I desperately want is to be able to link a customer to all the files; to be able to click on customers and check for receipts and finances that are linked to them. Similar thing for employee would be useful.**

**Question: What should be the layout be like?**

**Reply: After all we are technology store and I am a tech enthusiast so I require my software to be update to date with the latest design thus I require a layout similar to the window 7 and onwards and should be very simple and appealing.**

## **Programming Language Chosen**

For this system I will be using the programming language C# in the visual studio program, this is mostly due to my desire to learn a new language instead of VB.NET, as VB.NET I consider to be a “beginners and basic language”. It is time that I expand myself and learn a powerful and popular language in the computing industry, which is C#.

I will be programming using C# in Visual Studio software, which an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as web sites, web applications, web services and even android apps. Visual Studio program will allow to me to create an .exe (executable) program, which are more executable than those of e.g. (.jar) which require constant update. Visual Studio consists of a sleek and efficient interface so that I can navigate between my works and use tools efficiently.

Another reason for picking C# was that due to my experiences of searching for possible answers to fixing my errors. I would be faced with answer that were mostly answered in C#, especially for complex codes were written in C#, I can use this advantages to help myself when learning C# as there is a language C# community and support available online compared to VB.Net.

In the end, VB.NET and C# are both similar to each other, they both use the .NET library and syntax however VB.NET provides no control over memory while C# is also faster at executing commands.  
I have decided to use C# as the programming language to create the new system, due to my desire to learn a new language and wide range of support available in C#.

## Potential Solutions

Potential solutions to the Current System can be either a manual system or computerised system.

**A manual database** is a hard-file storage system that consists of paper records, folder and filing cabinets or storage boxes. While a **computerised** system is a huge compilation of electronic files stored at a single location. A computerised system is automated and handed by softwares thus eliminating possible errors and saving time. This makes it easier for everyone to access data than searching through filing cabinets and paper records.   
Computerized database can handle complex database and show relationship with primary keys and foreign keys.

Using Manuel system is not efficient and would not upgrade the Laptops Clinic current system, which use a variety of different softwares and using manual system to handle receipts and some finical reports, e.g. handing customer’s paper receipts made in a notepad.

A manual system would be much more time-consuming on a day-to-day basis, wasting lesson time, all data would have to be manually entered which would involve a lot of unnecessary repetition of entry for fields such as Contacts, Receipts and Reports. This could also lead to typographic errors.   
The whole point of this project is to develop a database system for the Laptops Clinic, which uses a single software, and that everything is organized and computerised. After all the laptops clinic is a computer store so makes sense to have its details computerised.

## Proposed Solutions (Reading off the Chart)

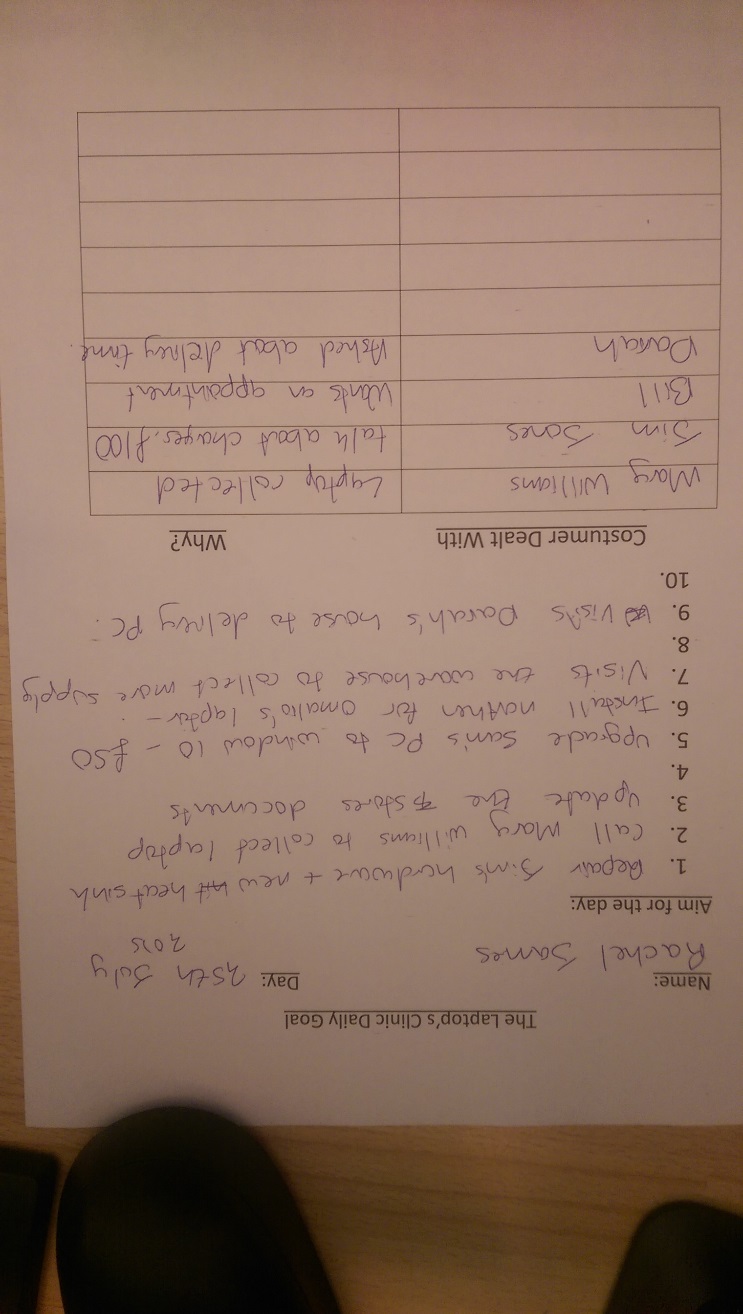
Due the data collected and shown in the Radar chart. I will produce a computerised system that will address the issues faced with the Employees and Customers of the Laptops Clinic.

Employee of the TLC have issues with mostly the software, tasks and receipts in descending order. Software issue is due to having to use multiple programs at the same time and accessing folders to update precise documents.

The software issue that the Employee faces, will be addressed by having one program, which contains all the features of the multiple programs used.

It will include shortcuts and access to the separate folders and receipts folders e.g. program will have a finance interface which will shows this month’s finance and provides access for other months, this can be done by a simple search (in a textbox). Instead of searching for it by searching through many documents.  
(TLC -> Finance -> Months of 2014 -> December) which is repetitive and open to corruption.

Tasks, the Employee of TLC (The Laptops Clinic) have a daily routine and aims for each day in which they specify the aim of the day, how will they achieve it, time frame and also “customer dealt with “which informs the store manager who they “dealt with today” and why. For example (Image).

This daily routine is beneficial to the employee and offers an efficient working place. However, daily means that there would be lots of sheets which they have produced, even though some as disposed of, the others are kept as reminder, which may hold a customer’s numbers or a reminder e.g. “Call Diana in 2 days”, thus the sheet will need to be kept for more 2 extra days.

This is not efficient as it leaves more paper works in TLC to handle, the sheet might accidently get damaged (coffee) or throw in the bin, and easy to lose. Thus, a build-in task planner in the software with calendar would be efficient as it will remind anyone of what tasks were sent in any specific day and the calendar allows an alarm/reminder to occur, reminding the employee than depending on fragile piece of sheet or human memory to remind you.

Finally, the third(least) important for the employees are receipts; receipts are given into two forms; paper or online receipts.

Paper receipts aren’t printed by rather are written down on paper and signed, online receipts are written on the Microsoft office document and are far more details than paper receipts. The online receipt are written on a default template (which contains the logo and contact details of the store), this online receipt can then be printed or emailed. Both are time consuming since you have to fill in the details or write them down. Therefore, a feature in the system that fills in the details automatically for the user will be very beneficial and time saving.

However, for Customers, their priority is receipts, contacting and time management, these priority will be addressed by improving the system with the features I suggested.

As long as all the objectives prove to be realistic and achievable, this is the most feasible potential solution. It meets the user’s needs as outlined by the objectives for the proposed system.

The existing staff should not have a problem using the system skill-wise(since it’s a technology/computer store which requires knowledge and experience with computers), which means I can make it as efficient as possible without having to keep the interface at the most basic of complexity levels. This solution should be achievable when taking into account my skill level and the time constraints.

## Justification of Chosen Solution

The solutions I have chosen is to have is to have a computerised database system in which the employees of the The Laptops Clinic can use. This system will be run on a centralised computer in which all the Employee have access to and can only use the system through this computer since the database and system only exists in that central computer. The system will provides all the features the employee of the TLC required from the different systems they previously used into a single program, so they will not have to navigate from different program but rather just use my proposed system, which contains all the features they required. Administrators and Employees will have different administrator accesses and administrator will have adminkey which is used to register any employee, promote employees to administrator e.g. both the employee and administrator be able to modify their own details and retrieve any forgotten details (username, password and adminkey), and they are both allowed to add, delete and modify customers. The system will allow the users of the software to check for any receipts or finance data associated to any customer or employee and be able to print them out. The system will allow a form of communication to customer/employees by emailing and have a build in calendar so that employee can input calendar events.

The solution will also come with an easy to use interface and be specific to the requirement of the TLC so employees cannot be confused, espically if their new and lack knowledge of the program. The solution itself is an executeable program and will be able to run on any PC with built in window 7 and up to date with the Microsoft updates, I will ensure that the program works in the central computer.

## User needs and limitations

The new system will allow TLC to store customer’s details and employee details. It will have a login system only for the employee. It will allow TLC to manage their customers and will consist of all the computable features they require.

Limitation are

* The Laptops Clinic already have a system for their customer details; Microsoft Outlook, thus when starting with the new system, it will have no customer details available to itself. However, it is possible to turn the Microsoft Outlook customer’s details into a database and include it in the new system. This can also include the finical reports which are down in separate excel files.
* Some of the feature of the system will require internet connection to function correctly. For example, emailing an employee or sending an online receipt to customer.
* The system will accessible and compatible with PCs and Laptops that can meet the requirement of the system. However, such things as tablet, mobiles cant access it. Only system that can execute .exe files and can process the graphic design of a window 7 to 10.
* The amount of space used by the database and software.
* Time constraints - The system needs to be completed by the end March.
* My skills and knowledge - The problem cannot be too complex for me to be able to solve using the resources I have available (the internet and google). I will be creating the solution in VB.NET and so could write the program in a number of different paradigms.
* Hardware and software constraints - There is only a limited amount of hardware in the physics department for the teachers to use, and a limited amount of software available for me to design the new system. This rules out the use of mobile applications as not all teachers have smart phones or tablets.
* The program will require internet connection to send email

## Numbered general and specific objectives of the project

Objectives are

1. The new system will allow the TLC to save customer’s details e.g. Name, Address, Phone & employee‘s details e.g. Username, Password. Also, to be able to edit the database, remove and search accounts, also allowing a way of contacting them by email.
   * + Specific: Yes
     + Measureable: This is measureable as you will be able to see customers and employees being saved in the database along with their details and will be able to search and modify them the ability to email them. Screenshots will show all of this.
     + Attainable: This is defiantly attainable since it is rather simple and the system evolves around the employee and customer interactions in the system; thus if this was not attainable then the system will not be able to work at all and render it useless.
     + Relevant: It is very relevant to the objectives of the project, as the owner of the TLC (The Laptops Clinic) requires a system that includes a database, which the employee can log into and modify customer’s details, without this objective there is no database or system, which the TLC can use.
     + Time-bound: this would be the first implementation I will include in the system as the system starts by the employee logging in, so it won’t be possible to add the features of the systems without finishing the this objective so it is time-bound.
2. The system will use steganography to encrypt the password for security purposes, also allowing the employee to modify their own details and being able to retrieve their
   * + Specific: Yes
     + Measureable: Yes
     + Attainable: Yes, should be simple to encode the password and enter a security system.
     + Relevant: Yes, it is important that the database and the system is password protected so they cannot be hacked.
     + Time-bound: Yes
3. Employee are able to retrieve their AdminKey, Username if they forgotten them and be able to change their details.
   * + Specific: Yes
     + Measureable: Yes, screenshots will show that it is possible to retrieve forgotten details.
     + Attainable: Yes, important that employee would be able to retrieve any forgotten details so they can have access to the system again.
     + Relevant: Yes
     + Time-bound: Yes
4. The system will have a clear and understandable user interface with a slick design
   * + Specific: Yes
     + Measureable: This is measured at the end of the project, by how simple and easy the program is and how many clicks are required to navigate around the program. Therefore, the program will not consists of tons of buttons and information that will confuse the user, but rather have few buttons with icons or labels to make it clear to their purpose.
     + Attainable: Yes
     + Relevant: Yes, user requested for a clear and understandable design.
     + Time-bound: Yes
5. It will provide different administration access for the Employees and Administrators, they should be able to exit anytime and data edited will be saved correctly and data is not lost.
   * + Specific: Yes
     + Measureable: Yes, screenshot in testing will show messagebox/ alert messages to employee that they can’t do specific task due to not having administrator access. Screenshot will also show a exit button at the top right for the user to exit, if database is saved, user is alerted and when failed to is also alerted as shown in messagebox.
     + Attainable: Yes
     + Relevant: Yes,
     + Time-bound: Yes
6. It will have a task planner and a calendar to remind the TLC of meetings, deadlines and alarm system.
   * + Specific: Yes
     + Measureable: Yes, screenshots will show the calendar and events on calendar created.
     + Attainable: Yes, but very time consuming due to the fact, the Visual Studio doesn’t have a slick calendar components for me to use, therefore I might be required to create a calendar from the start or find solutions online.
     + Relevant: Yes, the employee of the Laptops Clinic required a calendar to manage their appointments, deadlines and events.
     + Time-bound: Yes
7. It will show receipts, finance and allow receipts to be printed out and emailed.
   * + Specific: Yes
     + Measureable: Yes, Screenshot would be shown in testing, showing a table made up different attributes from different tables.
     + Attainable: Yes, The hard bit might be the fact I will have to dynamically design (on run time) the look(company logo, map e.g.) of the receipt document template, which is a word document and need to fill the word document with the information collected from the different tables
     + Relevant: Yes
     + Time-bound: Yes

## Data Source & Data Destination

For the current system:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Source  (input) | Process | User | Output | Destination |
| Customer’s Details | Employee stores and adds the customer’s data to Microsoft Outlook and uses Outlook to search, edit and remove customer’s details. | TLC’s employee, TLC’s administrator | Data about the customer is saved in Microsoft Outlook and displayed | Microsoft Outlook |
| Finance reports | Employee adds expenses to the right Microsoft Excel document corresponding to the month and year of the expense, then saves the document | TLC’s employee, TLC’s administrator | Excel document is updates on the latest reports of the day, month and year. These data can be used to represent data on tables, graphs. | Microsoft Excel |
| Receipts | Employee either writes on a paper receipts or makes a word document (which already has a design template) and adds manually the required information, saves them then emails it or prints it. | TLC’s employee, TLC’s administrator | Receipt on paper is easily handed out and signed to confirm or receipt is printed/emailed. | Microsoft Word Office.  Receipt that is written on paper or the receipt is printed is handed to the customer. Receipt that is emailed is sent to the customer. |
| Emails | Employee uses Outlook to email individuals. | TLC’s employee, TLC’s administrator | Email is sent along with/without an attachment | Email is sent to the individual |
| Reminders | Employee writes on sticky note for reminders instead of the calendar provided on Outlook | TLC’s employee, TLC’s administrator | Reminder is written on paper | none |
|  |  |  |  |  |

For the current system:

The programs and folders the Laptops Clinic uses aren’t secure or password protected therefore it is clearly easy for anyone to be able to access the documents, folders and the programs ( Outlook, Word, Excel) without any restrictions ( computer must be logged on ). As stated Microsoft Outlook has a calendar and alarm system in place already but it is rather confusing and are not used by the employees, instead sticky notes and human memory is used to remind the user.

For the proposed system:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Source (Input) | Process | User | Output | Destination |
| Customer’s Details | Employee adds the customer’s details to the new system | TLC’s employee, TLC’s administrator | Details about the customer is saved on the database. Customer added can now be searched and edited in the new system. | Database  (Customer Table) |
| Employee’s Details | Employee of the TLC must register to access the new system. Registration requires administrator access (AdminKey) to register new employees in the new system. Once registered, their details are saved in the database. | TLC’s employee, TLC’s administrator | Details about the employee is saved along with their username and encrypted password in the database. This employee can also be searched, login into the new system and can modify his or her own details. | Database  (Employee Table) |
| AdminKey | AdminKey is used to permit features of the new system e.g. Registration of a new employee. At the first start of the new system, the first user to be registered will be an Administrator and have an AdminKey.  EmployeeID are saved in the Administrator Table, AdminKey is a field in Administrator Table which must be filled in. | TLC’s administrator | Permits access for different features of the new system. | Database  (Administrator Table) |
| Navigational Buttons and Labels | Buttons with icons or labels once clicked will direct the user to the correct interfaces | TLC’s employee, TLC’s administrator | Displays the interface corresponding to the button or label click. | Displayed Interface |
| Calendar Event | Employee will add the required details of the Calendar Event ( Title of Event, Date, Time, Frequency, EventLength and the colour of the event). This will be displayed on the calendar with the colour picked. Calendar events can be deleted just by providing the correct Title and Time of the Event. Calendar Event are saved in the database. | TLC’s employee, TLC’s administrator | Displays the calendar event on the calendar with the colour picked. | Displays the Calendar and Database (Calendar Table) |
| Receipts and Finances | Any purchase done in the new system, the system will automatically save the details of the product bought in the Finance Table which contains details of the purchase (Product, DescriptionOfProduct, Quantity, Price, Date) while in the Receipt Table, it will store the customer who bought the product, the employee that assisted with the purchase and the details of the product.  Receipt table just contains the IDs of each of the tables (CustomerID,FinanceID, EmployeeID) that contain the required information, instead of copying data. | TLC’s employee, TLC’s administrator | Saves the receipts and finances data on the database | Database (Finance Table, Receipt Table) |

## Data Volumes

The external portable database I am currently using offers me a storage capacity of 5 gigabytes.  
3 gigabyte will used on this whole program and everything related to the software and an extra 2 gigabyte of storage will be available just to insure there is enough storage for the next 5 years to 10 years and in case they expand, hire employee and deal with more customers.

Each customer taking up 25 kilobytes of storage, while employee is 18 kilobytes. Employee is limited to 100 due to the fact there is less employee than customers are. Last year, only 16 employees were hired throughout the whole year of 2015.   
Thus 18 x 100 employees = 1800 kilobytes of storage for employees, if there is a lack of space, employee can be deleted or a request for more storage can be made.  
However for customers, I estimate that a capacity of 2000 customers which is far more enough thus 25 x 2000 = 5000. The proposed program itself has a limit of 50 megabytes

3 gigabyte of storage offers us with 3000 megabytes of storage, which is 3000000 kilobytes, 3 gigabyte minus the employee storage and customer storage and program storage limit equals 2948200 kilobytes of storage available for customer and receipts, calendar, and financial reports, which is far more than enough storage for the proposed system which is 2.9 gigabyte of storage.

## Data Dictionary

For Customer and Employee Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field Name | Field Purpose | Field Type | Start Value | Example Data | Validation |
| Full Name | Stores the first name and username of the Employee or Customer | TEXT | NOT NULL | James Roades | Not Null (Presence Check) |
| Email | Stores the Email of the Employee or Customer | TEXT | NULL | [Jamesroades@gmail.com](mailto:Jamesroades@gmail.com) | Presence Check (Not Blank) |
| Phone Number | Stores the Customer’s or Employee’s Phone Number | TEXT | NULL | 0708659695 | Presence Check and Checks the data entered is integer but saves the data as text, this is due to the fact, when saving as integer it removes the 0 at the beginning of the phone number. |
| Address | Contains the first line of Address of the Employee or Customer | TEXT | NULL | 53 London Road | No validation is required, customer or employee may not want to provide or have a permanent residence address |
| Postcode | Contains the postcode of the Address of the Employee or Customer | TEXT | NULL | E20 2ST | No validation is required, customer or employee may not want to provide or have a permanent residence address |
| Username | Username of the Employee used to login | TEXT | NOT NULL | JamesRoades | Presence Check (Not Blank) |
| Password | Password for the Employee to login | TEXT | NOT NULL | James123 displayed in a \* format when entering in the system | Presence Check (Not Blank), password will be encrypted and the password must match to the password of the provided Username |
| ImageDirectory | Stores the image location of the Employee’s profile picture | TEXT | Null | N:\Downloads\ExampleImage.png | Presence Check. If empty, saves null on database or else saves the location of the Image. |

Fields that are available in both Employee and Customer have an identifier, which shows in which table it belongs to. E.g. FullName is a field available in both Customer and Employee Table, to make it more easy to identify it’s remained to eFullName in Employee Table and cFullName in Customer table. This is required for a later purpose, which is explained in ….

For Finance Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field Name | Field Purpose | Field Type | Start Value | Example Data | Validation |
| Product | The product name of the purchase | TEXT | NULL | MACBOOK PRO 13 |  |
| DescriptionOfProduct | Description of the product | TEXT | NOT NULL | 2.53 GHz Intel Core 2 Duo  4GB RAM  250GB Hard Drive |  |
| Quantity | Quantity of the product bought | INTEGER | NOT NULL | 1 |  |
| Price | Price of the product bought | TEXT | NOT NULL | £250 |  |
| Date | Date of when the product was bought | TEXT | NOT NULL | 03/03/2016 |  |

For Calendar Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field Name | Field Purpose | Field Type | Start Value | Example Data | Validation |
| Title | Title of the calendar event | TEXT | NOT NULL | “Visit John G house to repair laptop” | Presence Check (Not Blank) |
| Time | Time of the calendar event | TEXT | NOT NULL | “15:30” | Drop down lists is used so the input is limited and less like to cause an error. Drop down list contains (0 to 24) for hours; another one contains (0 to 59) for minutes. Presence Check(user must pick an value in each drop down list) |
| Date | Date of the Event | TEXT | NOT NULL | 05/03/2016 | Requires none, the data picker used will always have a data (current day) or the day picked by other |
| Frequency | The frequency in which the event occurs e.g. weekly, monthly , yearly | TEXT | NOT NULL | Weekly | Drop down list is used so the input is limited and less like to cause an error. Presence Check(user must pick an value in each drop down list) |
| EventLength | The duration of the calendar event | INTEGER | NOT NULL | 2 | Drop down list is used so the input is limited and less like to cause an error. List contains (0 to 24) for hours.Presence Check(User must pick a value in drop down box) |
| Colour | Colour of the calendar event, the colour will be displayed on the calendar interface | TEXT | NOT NULL | Red | Drop down list is used so the input is limited and less like to cause an error. Presence Check(User must pick a value in drop down box) |

For Administrator Table

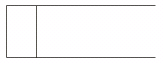
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field Name | Field Purpose | Field Type | Start Value | Example Data | Validation |
| EmployeeID | Gets the EmployeeID of the employee who is an administrator | INTEGER | NOT NULL | 1 | Not Null |
| AdminKey | Stores the AdminKey for the employee that is an administrator | TEXT | NOT NULL | James91key | Not Null |

For Receipt Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field Name | Field Purpose | Field Type | Start Value | Example Data | Validation |
| FinanceID | Used to link to the Finance Table, to get the product details | INTEGER | NOT NULL | 1 | Not Null |
| EmployeeID | Used to link to the Employee Table, to get the Employee’s detail (Username) | INTEGER | NOT NULL | 1 | Not Null |
| CustomerID | Used to link to the Customer Table, to get the Customer’s detail (cFullName and cEmail) | INTEGER | NOT NULL | 1 | Not Null |

## **Data Flow Diagram**

**For the current system:**



Microsoft Outlook

D1

Customer’s details

1

Locates the Customer via search

Locates the Customer

Employee talks to the Customer

2

Writes receipt on word document or paper

Uses the customer’s details to write up a purchase

Using Customer’s details provided

If customer doesn’t exist in the system (Microsoft Outlook)

Employee writes up the expense of the purchase in an Excel document

2

Writes on the Excel document and saves it in the right directory (folder)

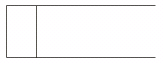
Customer’s details

3

Emails or handles the receipt to the customer

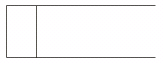
1

Adds a Customer in Outlook



Microsoft Excel

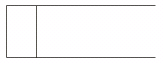
D2



Microsoft Outlook

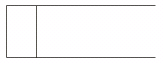
D1

**For the** proposed **system:**



Customer Table

D3



Employee Table

D1

Customer’s details

2

Adds a New Customer

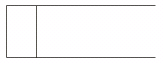
Customer’s details is used to locate them and present them

Employee’s details

Employee’s Username & Password

Employee deals with   
a new customer and adds them

Receipts and product bought is shown to the employee



Receipt Table

D4

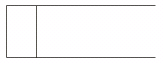
2

1

Adds a New Employee

D5

Searches for a customer



Finance Table

Employee takes in existing customer’s details.

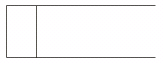
Employee takes in the customer’s purchase

1

3

Customer details is shown

Employee Logs in



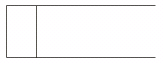
Customer Table

D3

Employee modifies (Update or Deletes) customer’s details

Employee details &  
AdminKey

D6



Calendar Table

Navigates to the calendar

Employee navigates through the program

Calendar Event’s details

Details on all the calendar events

2

Navigate   
(Menu) Page

3

Shows calendar event and allows add/delete them

Username, Password

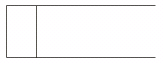
Program starts up.

Navigates to employee Interface

Navigates to the Receipt & Finance

3

Shows all the employees



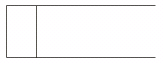
Receipt Table

D4

Once Administrator is created, returns back.

3

Shows all the receipts, Finance together



Finance Table

D5

Program loads for the first time

Only Administrator is able to delete Employees

Employee modifies his or her own details.

Administrator is able to delete the receipt/finance

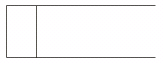
Employee is able to save and print selected receipt as pdf

3

Receipt is saved and printed

1

Creates the first account (administrator)

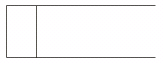


Employee Table

D1

Creates the database & their tables

Administrator details saved in  
Employee table



AdminKey

Employee Table

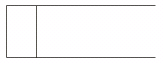
D1

Process

Data coming out

Data going in

Stores the EmployeeID of the Admininstrator in Administrator Table

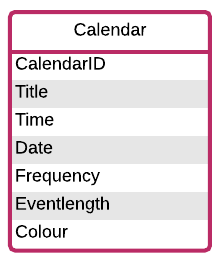
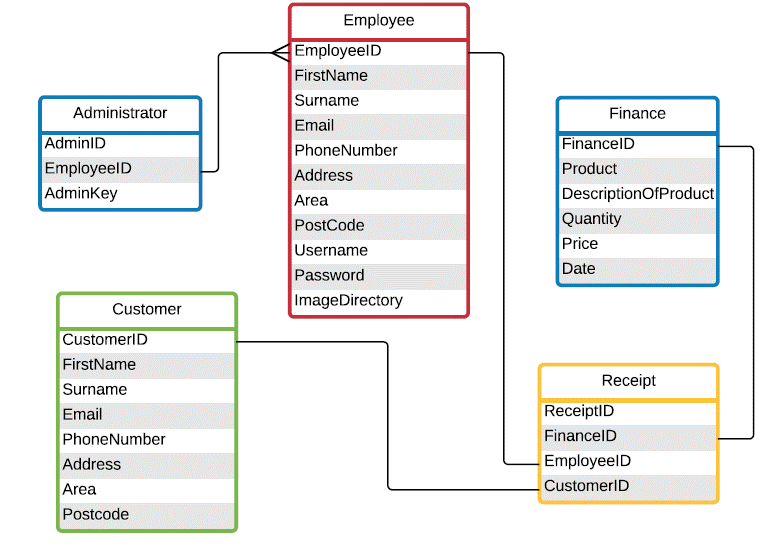


Administator Table

D2

## Entity Relationship Diagram

For the proposed system:



How the Laptops Clinic Works (using ER Diagram):

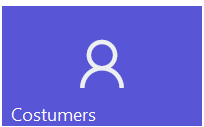
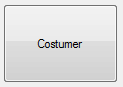
* When the program first loads up, the first person to be registered would be an Administrator, their details are added in the Employee Table while their AdminKey and along with their EmployeeID is saved in the Administrator Table (to verify their Administrator).
* Administrator uses their AdminKey to register new employees in the system with their details provided (check Employee Table on diagram) and is also able to delete employees and another administrators.
* Program starts and the employee login providing their correct Username and Password.
* Customer enters the store and talks to the employee. If the customer is not registered, they will be added in the system with the fields provided (check Customer on ER Diagram).
* Once the customer is added or already exists in the program, the employee is able to search them up using their customer’s details and is able to modify their customer details or delete the customer.
* If the customer seeks to make a purchase, the details of the product to be bought will be saved in the Finance Table with the fields provided (check Finance Table on ER).
* To print the receipt, employee will navigate to the Receipt & Finance Interface which will show the Receipt Table, the receipt table contains the IDs of the three tables (Check on ER diagram). The IDs are used to reference the name & email of the customer that bought the product, the username of the employee that assisted with the purchase and the details of the product. Instead of copying the data, foreign keys are used to reference them. PDF document is made with the receipt details and is automatically saved before printed, this saved document can be used to email it also.
* To check the calendar, employee simply navigates to the Calendar Interface and checks the calendar events which all have different colours to make it easy to distinguish and can add new calendar or delete them if they have provided the fields required.

# Design

## The User’s Interface



Since I am producing a software for a technology-based company, it would make sense to provide them with a slick modern look (User interface design) with the latest frameworks. I have achieved this by using Windows Modern UI for .NET Win Forms Application (Metro framework).  
Metro Framework applications are chrome less; this means every pixel on the window can be used for the metro style application. Metro style applications are fast, resizable, responsive, flow with good animation and great touch experience.   
These applications are scalable to different sizes of your system screen and provide the user with a new window 10 look and experience than an ancient look provided by Window XP or Window 7.   
For example instead of using Buttons; most metro-style application use Tiles which have the same function as a button but are far more flexible and provide images or text to make it attractive, also tiles are alive with status and activity updates encouraging the users to dive into your application. It shows overall personality of your application. During my program, I will be using Tiles and other metro framework components and I will be keeping the same programming terminology to not confuse anyone who reads my program as they may be confused on what a TileButton.Click is, so rather I will just use Button.Click (btn.Click)

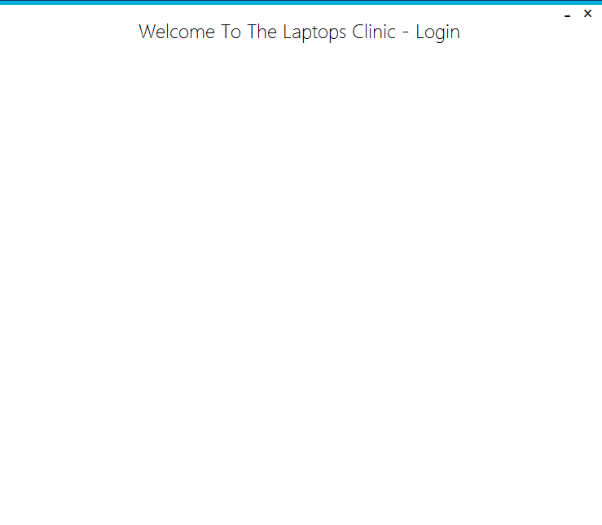
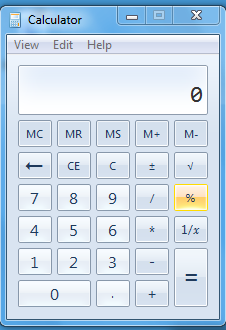


Window 7 Button

Metro Framework   
Tile Button

There is also a difference in design is in the form design. Window XP & 7 have a border around their form for example.   
However in Window 10 their borderless and there is no text in the bar. But rather a title inside the form not outside.

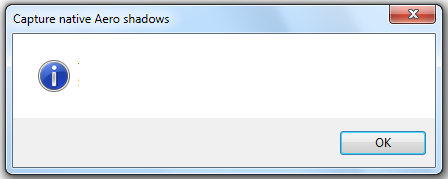
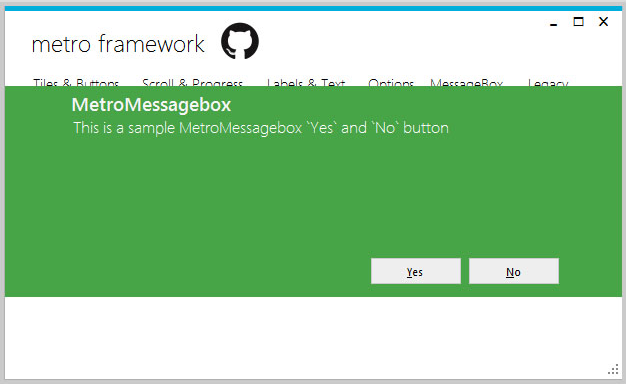
Diagram to show the differences.



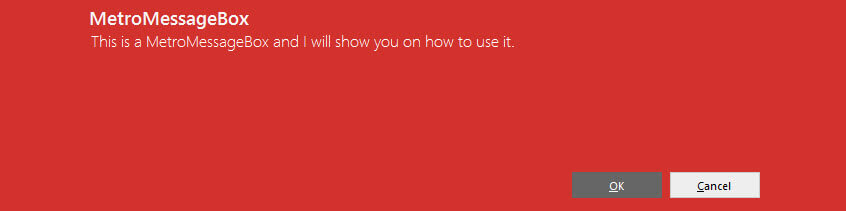
Window 7

Window XP

Metro Framework

The bonuses of using the Window 10 metro framework means that I get a new design for most of the components (tools) and also the message box which covers the whole screen instead of the original “pop up” classical one. Normally we would have a popup message with a border, title and icons inside the form and the warning message but now we get a border, which covers the centre of the form, it also has a heading followed with a subtext and buttons.

It has no icons on its message box but rather the colour of the message box represent the type of message box.

e.g. Red colour for errors

Form vs UC:

To put it very simply:

Windows forms are the container for controls, including UCs. While it contains many similar attributes as a UC, its primary purpose is to host controls (e.g. UCs).

UCs are a way of making a custom, reusable component. A UC can contain other controls but must be hosted by a form (exists inside a form).UCs serve the purpose of reusing controls. Imagine you need a search box in several pages of your application. You can create a search UC and drop it in every form where you want it visible. You use them to group a set of controls and behaviours together in a re-usable way. You cannot show a control on the screen unless a form hosts it.

When navigating between forms, the forms will pop into display while the previous one will either hide or stay visible behind. It will not be disposed, just the act of disposing it will also dispose the form that just popped into view. Therefore, a form (e.g. window) that just popped into view depends on the form that caused it to pop into view and the animation is always a pop up into view.

However for my proposed system, I simply do not want many forms to been pop into view continuously and having to close them all one by one, but rather I want to make a swipe navigating and less popping into view. To do this, I will use multiple UCs that hold the components of Menu Interface e.g. and to make them come into view, they will simply appear into view instead of popping into view.

To create the effect of sliding into view, the UC will appear outside the view area and after every milliseconds, it will move an x coordinate to the centre of the form. This creates the illusion of sliding into view.

## User Interface Design

Login Form (Accessed when the program is loaded each time & also accessed after a database, administrator account is firstly created.-Used to log in to the program.

S and E button represent a label with an Image; I named them S and E to easily reference them in my design, as they will have the same size and mostly the same function. The S is used to swipe a setting UC into view while E is for exiting the program. The empty dashed area is used to represent the area in which the setting UC will swipe into view.

The setting UC allows the user to modify theme, style and provide the email details required. Dimensions of S & E are both 40 by 40.

E

S

IMAGE ( will have a logo here)

e

Login

Register

Forgot Details

Username

Password

Welcome to the Laptops Clinic

.

Dimensions of these textboxes are;   
H: 264 & W: 30  
Textboxes are filled with prompt messages inside them specifying what details are required in the textboxes.  
The message disappears once the user enters a value in the textboxes

E

S

B

Back

Email Theme

Theme:

Dark

White

Style

Once S is clicked.  
Setting UC will be visible

Email tab clicked

Dimensions of Register & Forgot button:   
H: 130 and Width of 40 while Login Button is H: 100 & W: 70

When one of these buttons are clicked, a slide animation will occur where a UC will slide in front of this form. These UC correspond to the button clicked, Register button will slide Registration UC into view when clicked e.g.

Radio buttons have no size. Theme changes due to the selection of radio button

E

S

Back

Email Theme

Client:

Port no:

B

B is label with an image (an arrow pointing back🡨); it has the text “Back” and does the job as returning to previous interface. In this example, it will make setting UC invisible and out of view.

Buttons of size (30, 30)   
each will have a different colour. When colour clicked, will set the style of the program to the colour of the button.

Email:

Email Username:

Test and Save

Email Password:

Dimensions of H: 33 and W: 186  
Once clicked checks if there is Internet Connection and saves these Email details as global variables, which will later be used to email.

The dimensions of the text boxes in Email Tab are H: 23 & W: 186 will also consist of prompt messages alerting the user.

TemplateUC (Temple UC)  
 ( This is a template for all the UCs and causes the animation of the UCs swiping in view and out, handles events and disposes anything related to UC after use.

E

Even though this UC (UserControl) is mostly empty, it is very important, as it is a template for all the UCs. All the UCS will inherit this Template and all its behaviour. This includes the swiping animations and the exiting button [E] located at the top (Dimension and purpose explained in Login Form, previous page).  
The size of this UC does not matter, as one of its behaviour is that it resizes to the size of the form that hosts it.

All the UC will reference this and inherits its behaviour. This TemplateUC will have a metro style design and therefore all the other UCs will have the metro style design.

CreateSystemUC (First to appear when the program is firstly loaded) –used to create the database and will not appear once database and admin account is created.

Clear All

Get Started!

Basic Information:

Address:

Login Details:

Create an Administrator Account to get started.

Welcome to the Laptops Clinic

Image:

Clear

Modify

These textboxes are filled with prompt messages indicating what is required in the text (fields) boxes.  
Dimensions;   
-Width of 264  
-Heights of 30

These small textboxes have the dimensions;   
-Width of 131  
-Heights of 30

Dimensions for these buttons are  
-Width of 128  
-Heights of 50  
Clear All is used to clear all the textboxes while Get Started! Will create the database and the first administrator account.  
This UC won’t appear at the launch of program anymore since the program has a database and an administrator now.

Once Modify is clicked causes a file explorer to allow the User to pick an Image.  
Once Clear is clicked removes the image.  
Both have a size of   
63 Width, 60 Height

E

FullName

Create an Admin Password

Create an Admin Username

Email

Confirm Admin Password

Phone Number

AdminKey

House Number, Road, Street

PostCode

Area

RegisterUC (Register UC) (Accessed when user clicks “Register” button in Login Form) – used to create Employee/Administrator account.   
RegisterUC will slide in front of Login Form

All the textboxes in this UC have the dimensions of width 213 and height of 30.  
Most of all of them have prompt messages indicated what is required (if it wasn’t obvious already)

Basic Information:

Login Details:

Address:

Admin Key:

Clear

Register

Employee

Administrator

Registration an Account

E

FullName

Email

Phone Number

Username

Password

Confirm Password

House Number, Road, Street

Area

PostCode

Image:

Clear

Modify

Dimensions of Register Button. Height of 213 and Width of 144. When clicked will register the employee or administrator account to the database.

Back

B

Modify clicked causes a file explorer to allow the User to pick an Image.  
Clear once clicked removes the image.  
Both have a size of   
 Width 63 and Height of 30.

Dimensions of Clear Button: Width of 213 and Height of 100. Once clicked clears the data in all textboxes.

Radio buttons. User will select from two options; Employee or Administrator.

When creating an account, an Admin key is required from an Administrator to approve the registration.

If employee is selected and admin key is provided it will create the employee account.

However if Administrator is selected, each administrator requires to have their own admin key so a textbox will appear so that this new administrator can enter their own admin key

Address:

Admin Key:

Register

Employee

Administrator

Create an Admin Key:

If Administrator is selected, an animation will occur where the Register button will shorten to the same height as the Clear button and make space for a textbox to appear so that the new administrator that is registering can create their own administrator key.

Input your Admin Key

ForgotUC (Accessed when user click “Forgotten Details” button in Login Form) – Used to retrieve forgotten details from the system.

Once administrator radio button is clicked, a moving animation where these hidden components will animate into view, this is similar to the one in the previous page. They will slide into view and move on top of the corresponding text boxes.

Clear

Emai

B

FullName

Email

Username Or Password

E

Employee

Administrator Key

Basic Information:

Login Details:

Enter New Password

Confirm New Password

Dimensions of Clear Button: Width of 80 and Height of 80. Once clicked clears the data in all textboxes.

Forgotten Details

Dimensions of Email Button: Width of 80 and Height of 80.  
Once clicked it will act according to the Radio button (rb) chosen.

Employee (rb) + Forgot Password (rb) = will update the Employee with the new password.

Employee (rb) + Retrieve Username (rb) = will email the their Username to the Employee.

Administrator key (rb) = Will email the Admin Key to the Employee.

Each Process, it will email the User that their Login Details have changed.

FullName

Email

Administrator Username

Basic Information:

Login Details:

Administrator Password

Clear

Retrieve Username

Forgot Password

Email

All the text boxes in this UC have the dimensions:  
Width of 340 and Height of 30.  
Radio Buttons don’t have sizes.

All Employees including Administrators are stored in the Employee Table, only difference is that administrators are Employees who have their id stored in the Administrator Table, and also have an AdminKey.

Menu Form (Accessed when the user login in the system) –used to show all the various interfaces and features in the system in a simple design.

Hi! Currently Logged in as:

[ICON]

[ICON]

[ICON]

[ICON]

 [ICON]   
[ICON]

Options

Sign Out

[ICON]

Customers

Calendar

Email

Employee]

Receipt & Finance

Welcome to the Laptops Clinic

Width: 130, Height: 140

Picturebox with the Employee Picture.

(full name of the logged in user)

Width: 177  
Height: 195

Width: 177   
Height: 369

All these metro tiles, are event driven so when clicked. They will slide a UC in front of this form and disable this form so it the user can’t interact with it unless the UC is closed and focus is return to this form.

These MetroTiles also pass down the theme and style of the form.

Width: 95  
Height: 90

Width: 360  
Height: 158

Width: 360  
Height: 158

Customer’s Database UC (Accessed from the Menu Form) –used to show Customers, to edit and also set purchases and reminders

Search For Customer

FullName

Email

Number

Address

Area

Postcode

Unlock

Customer’s Details

Modify

Make an Order

Customer’s Receipt

Add Customer

Delete Customer

B

Customer Interface

E

130, 40

Once a customer is selected, it will update Customer’s Details to show a profile of that customer.

Email

Dimensions for these two buttons are W80, H40

Provides with options to modify the profile of customers and make a purchase for the customer.

Btn Unlock is used to disable and enable the text boxes.

Custumer’s Data Table.

Show all the customers in the system in ascending (alphabetically) order.

Columns are FullName, Email, Phone Number, Address, Area, Postcode

Size of W588, G526

id

Customer’s Receipt Data Table shows all the receipts of this selected customer, in descending order (dates of receipts).  
Columns are Customer FullName, Email,Fullname Of Employee, Product , DescriptinOfProduct, ,Quantity, Price and Date. W424, H252

W80,H40  
Opens the MakeAnOrder Form

W360, H40, used to search for in the Customer Table

Employee Pane (Accessed from the Menu Form) –used to show employee, to edit, update employees and links to Email Form.

W120, H40, one removes employee and the other opens Email Form to email people

Search For Employee

FullName

Email

Number

Address

Area

PostCodee

Selected Employee Details

Modify

Linked Receipt and Finance Data

Remove Employee

These labels are empty with a small size of W52, H25, it resizes to a different length when text is inputted into it

Employee Interface

B

W360, H40, used to search for in the Customer Table

E

Email

(Label)

Employee Data Table.

Show all the Employees in the system in ascending (alphabetically) order.

Columns are FullName, Email, Phone Number, Address, Area, Postcode

Size of W442, H526

id

W80, H40

Once clicked opens the Update Employee Form

(Label)

(Label)

(Label)

(Label)

Status

(Label)

(Label)

(Label)

Shows all the Receipt & Finance where the Employees ID is included  
Columns are FullName Of Customer, Email, Employee FullName, Product , DescriptinOfProduct, ,Quantity, Price and Date. W754, H278

150,150. Contains the Picture of the Selected Employee

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 |
| 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| 28 | 29 | 30 |  |  |  |  |

Day

Today

Saturday

Thursday

Friday

Month/Year

Sunday

Tuesday

Monday

Wednesday

<

>

>

Calendar UC (Accessed from the Menu Form) –used calendar event and display them.

Add Calendar Event

E

Calendar Interface

B

Event Title:

W225, H21

Combo boxes(drop down list) with selected options.

Recurring:

How an event will look

Event Colour:

Date Title:

Hour, Minute & Duration of Event

W225,H23

W71, H23

Add Calendar Event

W225, H37, creates the calendar event and stores it

Delete Calendar events, removes event and update the calendar

Delete Calendar Event

Event Title:

Event Time:

Event Date:

Delete Calendar Event

The Calendar dll comes with this design. Only allowed to resize the calendar. W1223, H579

W139, H43

Changes the calendar to month or day view

Add a Customer Form (accessed by Customer’s Database UC)-used to add customers to the system.

Make an Order Form (accessed by Customer’s Database UC when user clicks the button “Purchase”)-used to add a Purchase a customer’s account to the system.

40x40 return button (B) . Same as the return button is UCs however, this doesn’t swipe since it’s not UCs but just closes the forms.

B

Make an Order

Custumer’s Information:

Purchase:

Clear

[ICON]

[ICON]

FullName

Email

Product

Description of Product

Quantity

Employee’s Information:

Username

Price

Date

Purchase

B

Add a Customer

Basic Information:

Address:

Clear

Add Customer

[ICON]

[ICON]

FullName

Email

Phone Number

Line 1

Area

PostCode

These textboxes are automatically filled in using the information provided in Customer’s Details in Customer UC and the information of the current User (employee) logged in.

Clears all the fields (textboxes).  
Height of 72 and Width of 213.  
However clear button in the “Make an order Form”, only wipes out Purchase textboxes not Customer’s Information or Employee’s Information textboxes.

Adds a Customer to Customer database. Height of 112 and Width of 213

Adds a Purchase of a customer to Receipt database. W213, H144

All the textboxes have a promo text which tells what is required in the field. All fields must be fields correctly all or a warning message box appears. Dimensions for text boxes are: Width: 213 and Height: 30.

Search

Receipt & Finance UC (Accessed from the Menu Form) –used display all the Orders in the system, order them to the user’s choice also print them out.

W58, H29, combo boxes(Drop down list) lets the user modify the order of the Datatable by either ASC (Asecnding) or DESC(descending)

W120, H40, removes receipts and prints receipts

H317, W39

Receipt & Finance Interface

B

E

Print Receipt

Remove Receipt

Receipt & Finance form show all the data stored in the Receipt Table in the Database and Finance Table, combining them together into one table.

It will show all the customer ( FullName,Email, Product ( Product they bought), DescriptionOfProduct(Describes the product e..g size, colour, model of the product), Quantity (how many they bought), Price and Date of the Purchase.  
  
The Employee that dealt with customer is also included into the table to show which employee was present when this purchase was done.

W140, H29, combo boxes(Drop down list) lets the user modify the order of the Datatable.

W360, H40, used to search for in the Customer Table

UpdateEmployee Form accessed via Employee UC after you select a user to Modify. It will get the selected user and fills the basic information and Address along   
with their image.

Modify an Employee

Basic Information:

 Modify Employee

Yes

Login Details:

ID

Picturebox of size W129, H138, shows the selected Employee Image, and changes if user picks another image.

All the texboxes in this form of the same size. They contain watermark which are prompts to instruct the user what to input. W213, H30

B

Unlock

W87, H37. Change Image lets the user change the image of the Picturebox while Clear sets it to null.

Change Image

Email

FullName

Clear

Phone Number

Address:

House Number, Road, Street

Area

PostCode

Password

UserName

Make Administrator?

Change Login Details?

The border shows the visible to the user. While these outside the visible area.

When user picks Yes Radio Button for Make Administrator. Animation occurs were these components will slide into view. No slides out of view

Create an AdminKey:

Admin Key

Radio button sizes depend on the inputted text example. Yes Radio button are size of W45, H19. While No Radio buttons are W43, H19. Due to the extra charater is Yes. “s”.

W213, H91. Updates the Employee

Confirm New Password

New Password

New UserName

Yes

No

No

Confirm your Admin Key

Email Form. Used to send emails also available to add up three attachments.

Even though it is a form, not UC so therefore doesn’t interfert the Pnslide and it components. I have the back button in each form as an option of exiting the form.

These textboxes have sizes of W200, H30

Sent Email To:

Attachments:

W546, H217

Send Email

Email

B

Email From:

Email Subject:

Email TextBox

These buttons will have images of a Tick. Sizes are 35, 35.

## IOPS Chart

This chart outlines what happens to the data in the new system at the most basic level, in terms of input/output, processing and storage.

|  |  |
| --- | --- |
| Input   * Customers: full name, email, phone number, area and postcode * Employees: full name, email, phone number, area and postcode, username and password. * Administrators: EmployeeID, administrator key * Purchases * Receipts | Process   * Encrypt Passwords * Save , delete, update and insert data into databases * User login details are compared to the ones in the database and check if they match. * System checks if the user logged in has the correct administrator access when performing actions e.g. deleting * When inputting new records in the database, the records are checked and compared with the ones already stored data in the database to stop duplication. * Receipt shows details of the customer, employee and the product details (finance). * Receipt is turned into a pdf file, which can be printed or emailed. * Stores Administrator’s details, Employee’s details and Customer’s details. * Calendar details are stored and can be retrieved to show calendar events. |
| Storage   * 1800 kilobytes of information about Employees and administrators in saved in an external database(3gigabytes) * 5000 kilobytes of information about Customers in saved in an external database(3gigabytes) | **Output**   * Registration successful/unsuccessful message * Login Successful/unsuccessful messages. * Error messages everyone in the program to ensure it runs smoothly and is perfect without errors. * Receipts printed or emailed. * Shows table results of all customers or employee or receipts stored in the database |

## Form Navigation Design

The following chart illustrates how users would navigate between all of the forms and UC (UC) in the new system, starting from the Login Form, which is the main start up interface (after the database is first created).

CreateSystem UC

Forgot UC

Register UC

Login Form

Main Form

Customer UC

Email Form

Calendar UC

Receipt & Finance UC

Employee UC

AddCustomer Form

MakeAnOrder

Form

Update Employee Form

To fit with my modern and aesthetic appreciation of my program, I wanted to create a slide in animation between transitions between forms and UCs.

In my program, it will consist of Forms and UCs, which will contain both objects/controls inside them. Both of them have a metrostyle (dll) look that I have installed into the system to give them a modern and window 10 , UC are just UCs which will also contain objects that relate to their function. E.g. Forgot UC will contain objects, controls related to helping the user retrieve their details from the system.

For an UC to exist, a form must host it, this means it must call upon it and then present it. When UC are called upon, they won’t appear right in front of the user but rather will outside the view of the user, then they will be animated to slide into view. This is achieved by x coordinates of the UC every miliseconds (makes a swift smooth animation slide since its moving every miliseconds (not specified atm)) until the UC comes into view.

The program starts with the Login form, however if there is no administrator account and a database is not created, the CreateSystem UC will slide into Login form covering every object inside Login Form. Once an administrator account and the database is created in CreateSystem UC, the CreateSystem will slide out of view and dispose, revealing every objects that it was covering (in the login form). The Login Form is able to navigate to 3 UC ( CreateSystem (UC), Register (UC) and Forgot (UC)) and also into Main Form.

The Main Form provides all the shortcut to other features in the system for navigation. All these navigation are UC and are (Email UC, Calendar UC, Customer UC, Receipt and Finance UC, Employee UC), once a UC is open, it will cover the Main Form and a “back” button is provided in all the UCs so they can navigate back to the form, the UC will take into account the size of the Main form and adjust to its size. Another aspect of my program, which I have not spoken about it’s the theme and style available. Theme is black and white while style varies. (show in the Login Form User Interface Design). Once a user picks a theme and style of the program(in the login form), it will be passed between all the UCs and forms, so all the controls and interfaces will have the theme and style the user has picked.

## Sample of planned SQL Queries

Due to the use of an external database in the software, a range of SQL queries are needed.

INSERT INTO Administrator (EmployeeID,AdminKey)

SELECT \* FROM Employee WHERE Username = “ExampleUsername”

This SQL query selects values from the table called “Employee”. It selects rows where the field “ExampleUsername” matches the variable. This SQL query is performed during registration, to check if a username is already taken or not.

INSERT INTO Employee ( FullName , Email, Address, PostCode, Username, Password)

This SQL query inserts values into the table called “Employee”. It inserts values into the fields “FullName”,” Email”,”Address”,”PostCode”,”Username”, “Password” What these values are depends on the user input and are inserted during registration.

SELECT \* FROM Employee ORDER BY [a-z]  
SELECT \* FROM Customer ORDER by [a-z]

Update Employee SET FullName = ? , Email = ?  
Where EmployeeID = ?

This SQL query selects all the values in the table used to get all the values in the table so that they can be represented in a table or listview in ascending alphabetic order.

An example of updating a table, this is updating Employee and changing the FullName and Email where the EmployeeID exists.  
Update is used to update employees and customers, you can’t update Receipt/Finance Tables

This SQL is used to create the Employee table, it sets its properties (string, text e.g.) and Null or Not Null. If its Null, it will start with the value of 0. If its not Null. It means that attribute that are inserted into the table can’t be Null, if it is. It will create an exeption.

" CREATE TABLE [Employee] ([EmployeeID] INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,[eFullName] TEXT NULL,[eEmail] TEXT NULL,[ePhoneNumber] TEXT NULL,[eAddress] TEXT NULL,[eArea] TEXT NULL,[ePostCode] TEXT NULL,[Username] TEXT NULL,[Password] TEXT NULL,[ImageDirectory] TEXT NULL)"

## Storage Media and Format

As the software is designed for a specific purpose, it should therefore not be large, I calculate that at max, the program will use of around 100 MB of storage without any data stored inside. Once downloaded, it will be taking up at max 25 Kilobytes of data per Customer stored, and a combined max of around 100 Kilobytes per Customer stored (since Customer’s receipt and details and purchases will use up data), this data will also be added to Receipt’s date (takes Customer and employee details and their purchases and date of purchase) and Finance Data (takes the numerical purchases, profits and stores them). However these data being used up is rather small which provides with me with varies ways of distributing the system.

One way of disturbing the program, would be that the user will have to download the software from their website using the internet, they would require access to the internet to use some parts of the software in the first place (e.g. need internet so that the program can email employee their details if they forgot them or just send emails to customer with receipts) but it is possible for the software to work without using the internet but features such as emailing and forgotten details retrieval won’t be possible. The Laptops Clinic store has multiple laptops and computers that use the internet provided however it doesn’t have any strict web filtering so there is access to most of sites available to everyone so no issue using this method.

Another method would be to put the software on a disk. Out of all the disk types, a CD-ROM would be the most sensible due to the disk only having to be read from and not written to, and because CDs have the least storage space (194MB) in them compared to DVDs and Blu-Rays (as the software takes up a small amount of space). However, some computers do not have a disk drive and this may leave some users unable to access the software. Another problem would be that a large amount of space would be wasted, as the software should only take up around a megabyte in space out of the available 194MB on the disk.

Another method would be to use a USB flash-drive, for several reasons. The file itself wouldn’t create any wasted space on the flash drive, as the rest of the drive could still be read to and written from as usual. The Laptops Clinic tend to already own flash drives and have a storage full of empty USBs and even sell USBS, so therefore they can keep backup copies of the executable on these instead of having to store physical CDs, and no internet connection is required during the installation process. As USB ports are an industry standard on laptops and desktops for the foreseeable future, as long as the .NET framework is kept updated, the software can still be installed. It is also possible for some laptops and computer to not have a CD/DVD drive, and so from a programmer’s perspective it is also a great deal easier to store the executable files on a USB flash drive. Installation speeds from USB 2.0 and 3.0 are both faster than installation speeds from a CD-ROM, which is another advantage.

In the end, I have decided to store the software on a USB stick. This is due to the fact that all computers have a USB drive, and it would not result in any wasted space as the USB can still be used for other reasons due to the fact that they can always be read from and written to. All users already have their own USB sticks. As USB drives are standard on computers, even if the laptops or computers are updated, they will still have access to the software – provided they have required software installed on their machine. Therefore, I believe that USB sticks are the best method of distributing the software in this instance. Storing on the internet, leaves the software open to the general public, making it accessible to another similar stores and companies which is a good thing however it also makes it accessible to hackers which can download it to find exploits in the system or people might distribute/copy the program for their own profit. This is a program that is specific to the Laptops Clinic and references the store everywhere.

**Visual Display Unit (VDU) –** In order to see the software, they’re using

**Keyboard -** In order to provide input to the system

**Mouse –** In order to navigate through the system  
**Universal Serial Bus (USB) -** In order to store the software and access it  
**Network –** this is a suggestion as if the software was stored in a network, all computers and laptops will have access to it and run it, instead of manually downloading it in all the laptops or the computer they need. Storing on a network allows you to access all the data in the software and keeps the program connected. For example if the program was downloaded manually in each computer, it will work however, the database will not be connected but rather be separated.

## Identification of suitable algorithms for data transformation, pseudocode of these algorithms

|  |
| --- |
| Title: |
| Explanation: |
| Pseudo Code: |

|  |
| --- |
| Salting and Hashing Password |
| Explanation: Hasing is encryption based on math and because it encodes the data in a secure manner it can not be reversed. Even though we cannot reverse engineer the password from the hash, Hackers can create a single rainbow table (a precomputed table for reversing cryptographic hash functions) which is used for cracking password hashes. Therefore the reason Salting is important cause it adds a random generated piece of information (characters, symbols e.g.) to the password to create a unique hash results. This forces the hacker to create a rainbow table for each user to figure out the password. This makes it computiontionally too expensive and time consuming. |
| Pseudo Code:  Salt = RandomSize(Min,Max) – The salt has a random length within the min and max e.g (4,12) implies it has a random salt of values and needs to be length of 4 to 12.  RNGCryptoServiceProvider GenerateSaltValue = New RNGCryptoServiceProvider  GenerateSaltValue.GetNonZero(Salt) –fills the salt with random values which isn’t null within the size  PlainAndSalt = PasswordText + Salt  Sha256Manged HashThePassword = new Sha25Managed – Used to hash, computes a hash that is almost unique.  EncryptedPassword = HashThePassword.ComputeHash(PlainAndSalt) – computes a hash for the provided value(PlainAndSalt) |

|  |
| --- |
| Validating User Registration |
| Explanation: Users’ registration details must be controlled in such a way so their input does not cause any issues for the system. Also, it must be controlled for the user’s own security  This function checks all of the user’s input during registration. Employee Registration Fields are (FullName, Email,Phone Number, Address, PostCode, Username, Password) |
| If FullName.Text = Null OR Email = Null, Or Phone Number = Null ……… Password = Null then  Show messagebox to tell user to required Fields  Else  If an User with that Email or Username exist ) then  Show messagebox to tell the user an User already exists with that email or username  Else  If Password.Text == ConfirmPassword.Text then  Messagebox to alert the user that the password doesn’t match!    Else    Inserts the User into the databae  Messagebox to show stating “The User has been added to the system”  End if  End if |

|  |
| --- |
| Presenting the Tables in the Database into a Table in the System |
| Explanation: This gets all the values in a Database Table e.g. Customer Table and saves them into a DataTable in the system which is similar to an array and holds all the values. Now we obtain the values we want from the DataTable and show them in a ListView |
| DataTable CustomerTable = SELECT \* FROM Customer  ListView CustomerListView = new ListView  CustomerListView .AddColumn = “FullName”  CustomerListView.AddColumn = “Email”  CustomerListView.AddColumn = “PhoneNumber”  ^Sets the ListView with Columns. Don’t require in programming as you can design it with the columns instead of dynamically programming it  Foreach (DataRow CustomerRow in CustomerTable) – For Each row in Customer Row.  ListViewItem Item = new ListViewItem(CustomerRow[“FullName”] –Gets the items from that specifc row and gets the column value of “FullName”  Item.subitems.Add(CustomerRow[“Email”] –Gets colum “Email” for this row  Item.subitems.Add(CustomerRow[“PhoneNumber”]  CustomerListView.Items.Add(Item) - ListView CustomerListView then adds this items into the ListView.  this loops until it has got all the rows.  CustomerListView.AutoResizeColumns – to resize Columns and items it contains.  Example. If it had gotten “Barry Allen” From the CustomerRow[“FullName”] then for [“Email”] was [BarryAllen@theflash.com](mailto:BarryAllen@theflash.com) and the [“PhoneNumber”] it got was 079….  The items will be added in the order they are collected into the list view.  The first column in the ListView we made was FullName and the first items we got from the DataTable from our query was [“FullName”]  Output example  Fullname l Email l PhoneNumber  ---------------------------------------------------------------------------------------  Barry Allen l [BarryAllen@theflash.com](mailto:BarryAllen@theflash.com) l 079….. |

|  |
| --- |
| Querying an SQL through a class |
| Explanation: Instead of writing long lines to declare connection, set up variables and setting up command and executing for everytime you need to run a query. It is faster to run a query through a class which has everything set up already. |
| **Class RunMyQueries :**  **Bolean RunExecuteNonQuery (string Query) – Needed to be provided with a query**  Connection p\_Connection = new Connection (TheDatabase.txt)  try   p\_Connection.Open()  Catch  Adds the error to the Debug e.g. “Connection Failed”  Return false  Command p\_Command = new Command ( Query, p\_Connection)  Try    p\_Command.ExecuteNonQuery()  Return true    Catch    Adds the error to the Debug e.g. “Command Execute Failed”  Return false    p\_Connection.Close  This is will run a query and try to execute it, any errors it will add it to the debug for the programmer to see.  At the start of my project I had to do this for every time I run a query, most times I had 7 lines for each query I needed.  However making a Class and putting the query for ExecuteNonQuery, Since its bolean it returns true for false if it executed it or not. Now back to my form.  **Form** :  User click RegisterButton to add the new Customer to the Database  RegisterButton.Click  Bolean **RegisterACustomer** = **RunMyQueries.RunExecuteNonQuery** ( INSERT INTO Employee ( “Barry Allen, [BarryAllen@theflash.com](mailto:BarryAllen@theflash.com) , 07908348495)  To check if it worked, it will return true or false to the variable boolean RegisterACustomer  If (RegisterACustomer == true)  messagebox ( “Customer has been added)  Else  Messagebox (“Failed to add customer”)  All it took me was one small line to add an employee. This saves up time and space in my code. |

## Class definitions and details of object behaviours and methods

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Class Name | Overview | Inheritance | Overiding | Public/Private Variables | Public/Private Method |
|  |  |  |  |  |  |
| Hashing | Used to hash and confirm the password entered. | None | None | None however, there are variables within the Methods. | (Boolean) ExecuteNonQuery,  (String) ExecuteReader,  (String) ExecuteReaderToGetUserID,  (DataTable) ShowDataResults,  (Bolean) ExecuteReaderBolean,  (String)  ExecuteReaderToGetUserImage |
| Variables | Used to hold all the variables in the system | None | None | CurrentEmployeeUserID  EmployeeUsername  EmployeeFullName  SmtpClient  Port  CompanyEmail  CompanyEmailUsername  CompanyEmailPassword | No methods, consists of get /set accessors.  Which is similar to a method however, they retrieve values for the variables and set them.  set {Variable Name }  get {Variable Name} |
| DataQueries | Used to handle all the SQL execution of the Database | None | None | None; however, there are variables uses in the Methods. | (String)  ComputeHash  (Bool) Confirm |

Class: Hashing

Attributes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Accessor Type | Field Name | Field Type | Initial Value | Description |
|  |  |  |  |  |

Methods:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Accessor Type | Method Name | Paramaters | Return Values | Description |
| Public | (Boolean) ExecuteNonQuery | a\_SQLString | True/False | Executes Insert, Delete and Update Queries |
| Public | (String) ExecuteReader | a\_SQLString,  SQLcolumn | Returns string of the value found. | Uses Readers to get values from the Database, “a\_SQLString” is the query while “SQLcolumn  “is the column value. |
| Public | (String) ExecuteReaderToGetUserID | a\_SQLString | Returns string of the value found. | Similar to ExecuteReader however explicitly returns the value of the ID of affected value. |
| Public | (DataTable) ShowDataResults | a\_SQLString | A datatable containing all the returns in an array | Used to return table results of our queries. |
| Public | (Bolean) ExecuteReaderBolean | a\_SQLString | True/False | Similar to ExecuteReader however, this is used to check where a value exists in the system not to retrieve values |
| Public | (String)  ExecuteReaderToGetUserImage | a\_SQLString | Returns string of the value found. | Similar to ExecuteReader however explicitly returns the Image of The User. |

Class: Variables

Attributes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Accessor Type | Field Name | Field Type | Initial Value | Description |
| Private | CurrentEmployeeUserID | string | Null (“”) | Stores the ID of the Employee currently logged in |
| Private | EmployeeUsername | string | Null (“”) | Stores the Username of the Employee  currently logged in |
| Private | EmployeeFullName | string | Null (“”) | Stores the FullName of the Employee  currently logged in |
| Private | SmtpClient | string | Null (“”) | Stores the SmtpClient used for emailing |
| Private | Port | string | Null (“”) | Stores the Port number for the email provider |
| Private | CompanyEmail | string | Null (“”) | Stores the Email address of the Laptops Clinic |
| Private | CompanyEmailUsername | string | Null (“”) | Username of the email used by the Laptops Clinic |
| Private | CompanyEmailPassword | string | Null (“”) | Stores the password for the email used by the Laptops Clinic |

Methods:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Accessor Type | Method Name | Paramaters | Return Values | Description |
|  |  |  |  |  |

Class: DataQueries

Attributes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Accessor Type | Field Name | Field Type | Initial Value | Description |
|  |  |  |  |  |

Methods:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Accessor Type | Method Name | Paramaters | Return Values | Description |
| Public | (String)  ComputeHash | string plainText, byte[] salt | Returns the hash and salted password for the inputted plaintext, | Used by the system to retrieve the password entered and return back the hashed and salted version. Byte[] salt is used when a specific salt is provided otherwise its null and random will be used. A specific salt is provided when user is logining into the system. |
| Public | (Bool) Confirm | string plainText, string hashvalue | Returns True/False to confirm whether the password entered is correct. | Used by the system to retrieve plaintext (password user entered) and hashvalue(password stored in the system), system hashes plaintext in ComputeHash with a specific salt and then compare the hashed plaintext with the hashvalue to see if the password entered is correct. |

## UI sample of planned data capture and entry designs

\*Screenshot provided are not of the final completed design, still in development\*

When the user is confronted in the program with options, depending on the options available, I will either pick between radio buttons or drop down list. No tick boxes are used in this software. If the options were between two things then radio buttons are used (e.g. selecting whether the new employee that your registering will either have an employee account or administrator account) else more than two options, drop down list would be used instead.  
The position of the element is important for a better experience while the colour scheme is depending on the theme the user picked at the Login buttons.

The only UCs that have their own colour scheme is Register and Forgotten UCs which are only accessed via Login Form. After the user picks his theme and is logged on to the Main. Any forms and UCs that are open via the Main form will have the same colour scheme as the one the user picked in the login form.

Basic Information:

Login Details:

Address:

Admin Key:

Clear

Register

Employee

Administrator

Registration an Account

E

FullName

Email

Phone Number

Username

Password

Confirm Password

House Number, Road, Street

Area

PostCode

Image:

Clear

Modify

I have placed the radio buttons at important places within the interface.  
For example when registering an account, you will need to provide the details, it does not matter if the radio button was at the front top next to the title, as you would still have to fill in the details. Therefore, I placed it at the top of the Register Button, as it would easily grab the user’s attention as they about to click it and lets them think about it. If it was placed at the top, user can just forget and rush to fill in the details and click register.   
However, when faced with two options as you about to click register, it will slow them down and remind the user to pick. (Employee is default selected) and have a purple colour to indicate selected   
In addition, I have an animation under The Register Button, which was explained in the User interface design.

Clear

Emai

B

FullName

Email

Username Or Password

E

Employee

Administrator Key

Basic Information:

Login Details:

Enter New Password

Confirm New Password

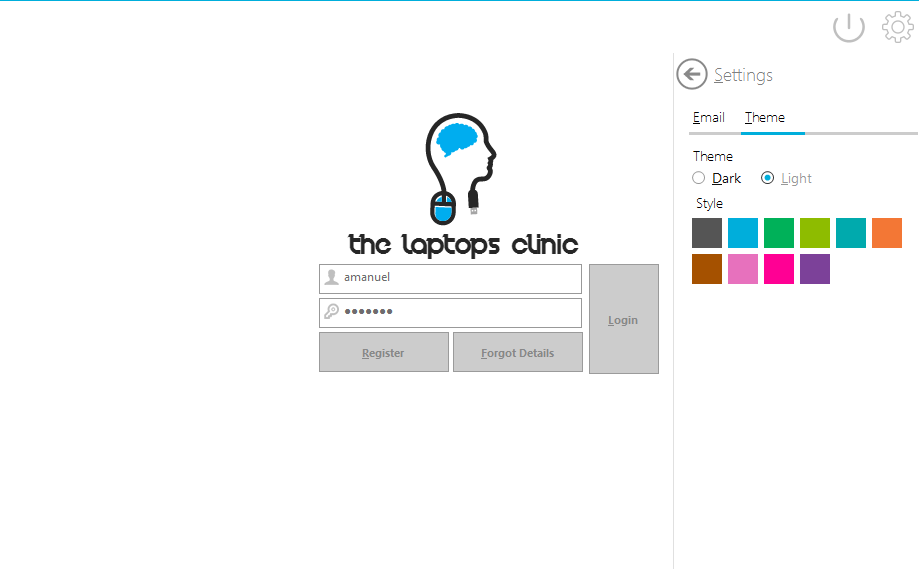
In the Forgot UC, which has its own red colour scheme, the radio button is placed in a Hierarchy system of importance.

Does the user want to get the Details for Employee or the Admin Key? – Which is the most important question and quick navigation to the option you choose.

If the user wants to get employee details instead of admin key, then there is a choice of getting password (“Forgot Password” ) or getting their username (“Retrieve Username”).

Two radio buttons are provided once the user picks Employee. Three textboxes are provided under the radio buttons, depending on the radio button picked, they will become enabled/disabled or visible/invisible. E.g. User forgot their password and wants a new password, therefore the user will select “Employee Account” then “Forgot Password” and fill in all the 3 textboxes while if they seeks to “Retrieve Username” then all they will require is the first textbox instead all the 3 textboxes. The prompt message changes depending on what is required.

Radio buttons and drop down lists that exists in UC or forms after the user has logged on will have the same theme and style as the style and theme picked in the Login form.

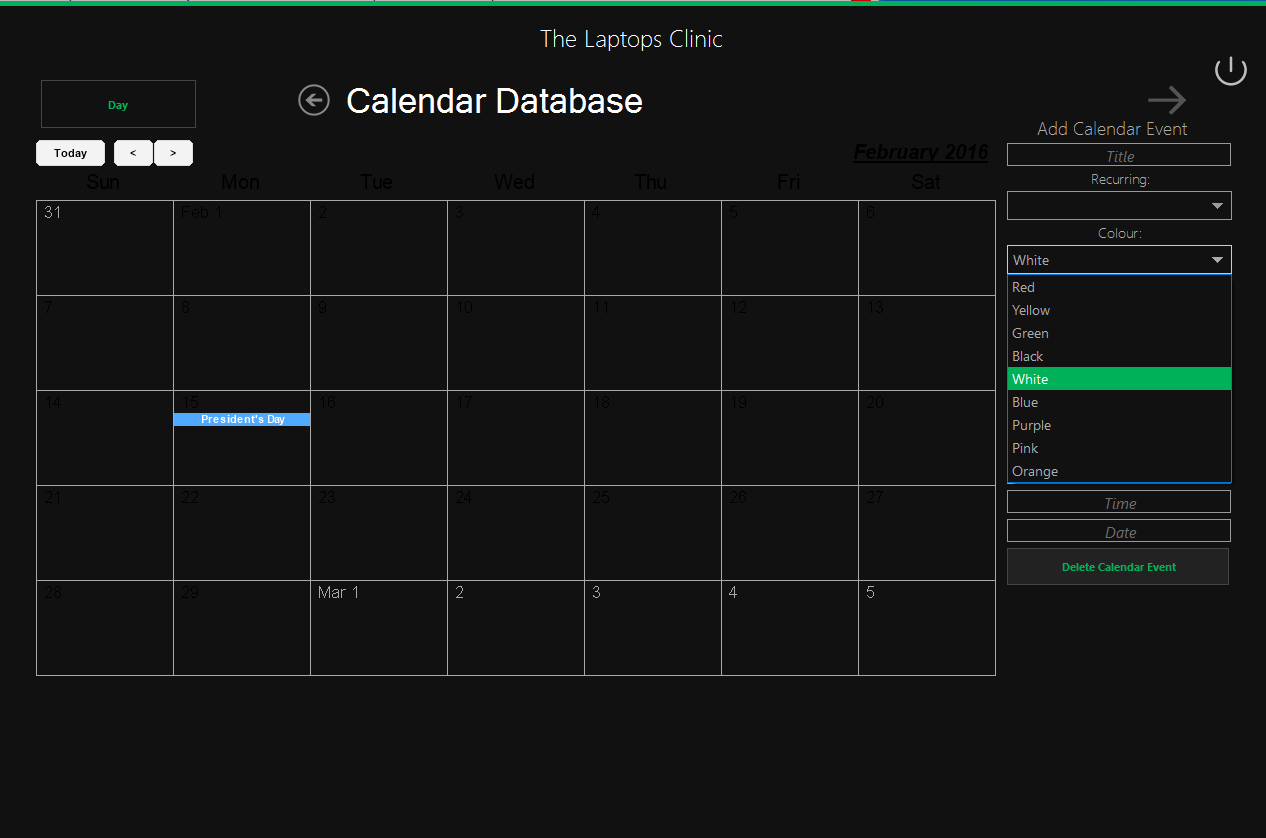


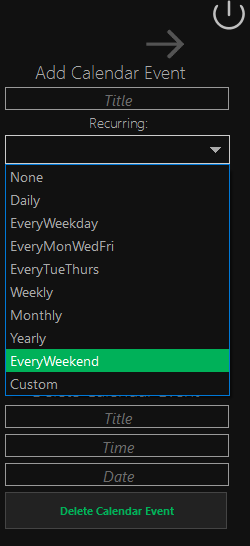
When adding a calendar event, you are provided with textboxes and drop down boxes, the drop down boxes are required as the options are vast.

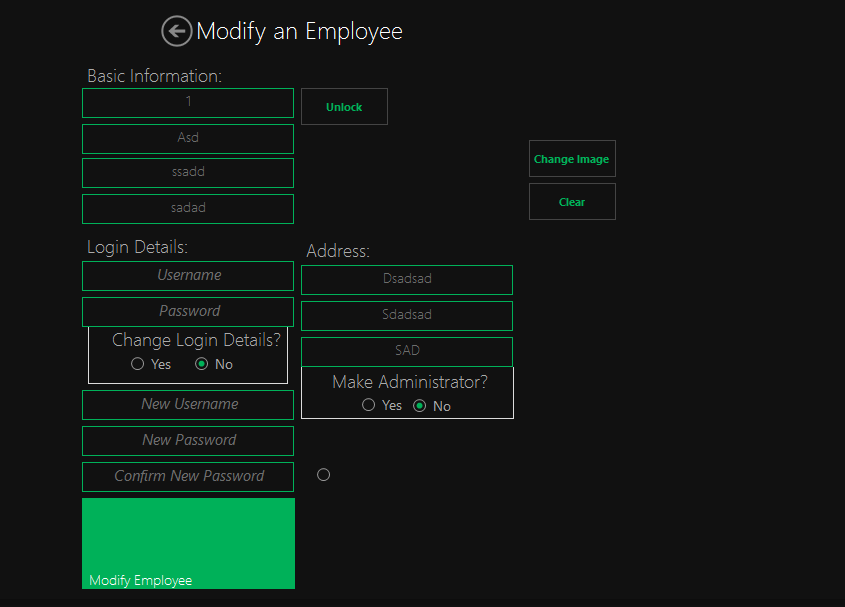
E.g., in the picture, there is 9 options of colours and on top of that drop box there is another drop box.  
Colour drop down list stores the colour of the event, on the calendar there is a blue event.

It has an options of 10, so drop down are used. Radio buttons can’t be used as that would require 10 radio buttons which is far too much. This drop down box will store how recurring the event is, is it weekly, monthly?

If the user picked black theme and Green style then logins in. Then then theme and style colour will passed on to all forms, UCs, and their components example.



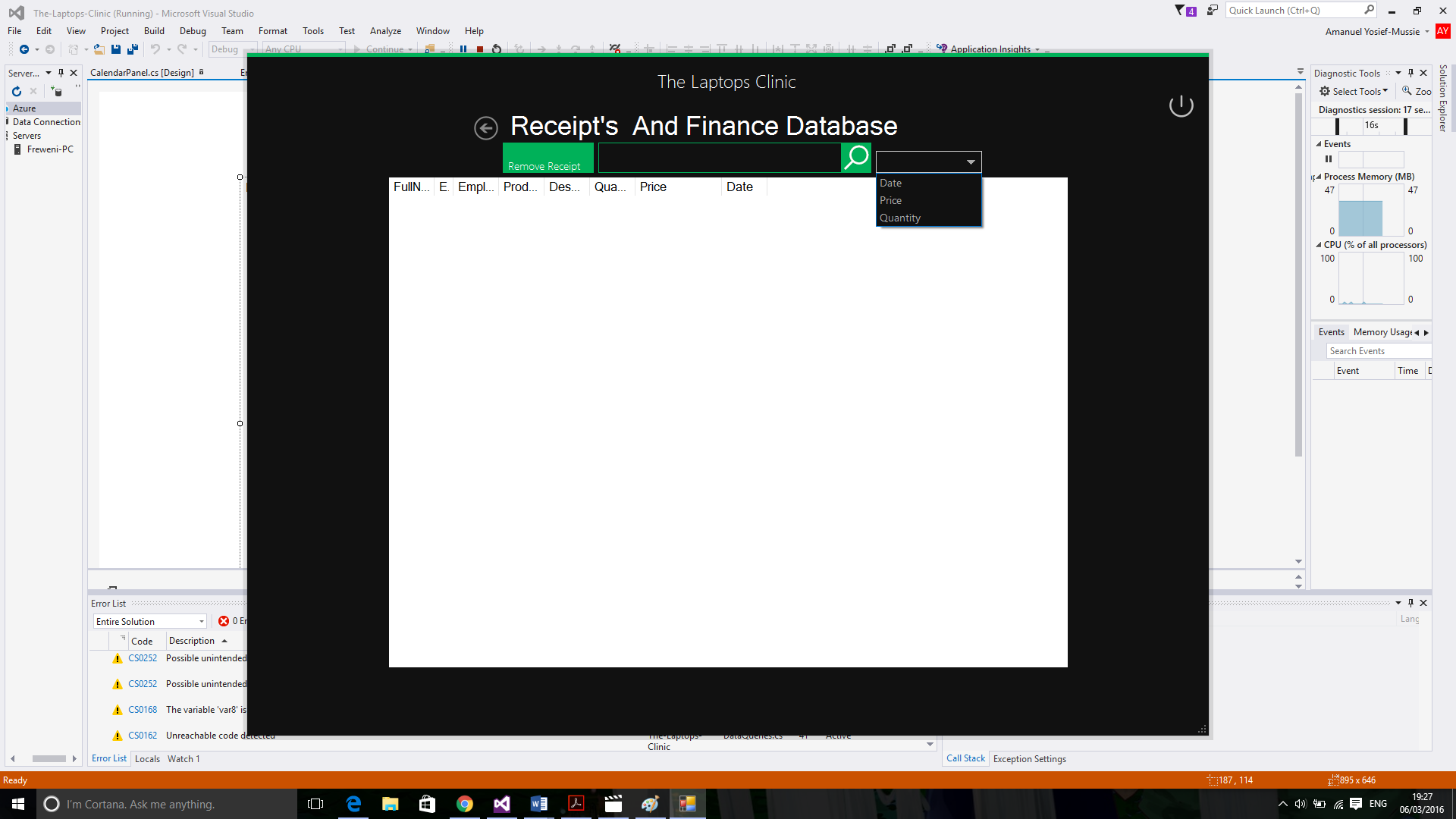




Using the colour scheme, which I logged in with to show how it is passed through all the components.

A drop down list is provided in the Receipts and Finance Database, to allow the user to manipulate and change the order of the ListView provided.  
They are able to order it by Date, Price, Quantity bought, more options will be provided. It is possible to another drop down list button here to represent if they want to order it by ascending or descending

Similar to the Register and Forgot UCs, there radio buttons are placed at the end of the Interface before the user clicks The Modify button, to let the user think and decide before clicking.  
colour scheme is passed on and animation is also provided here.

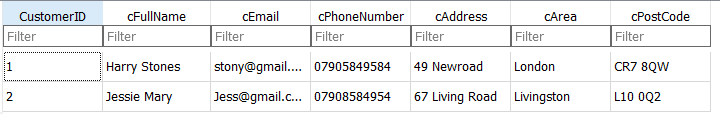


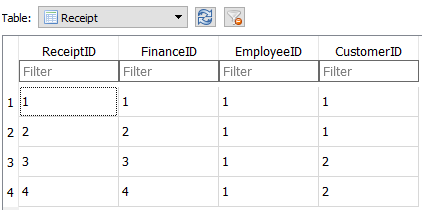
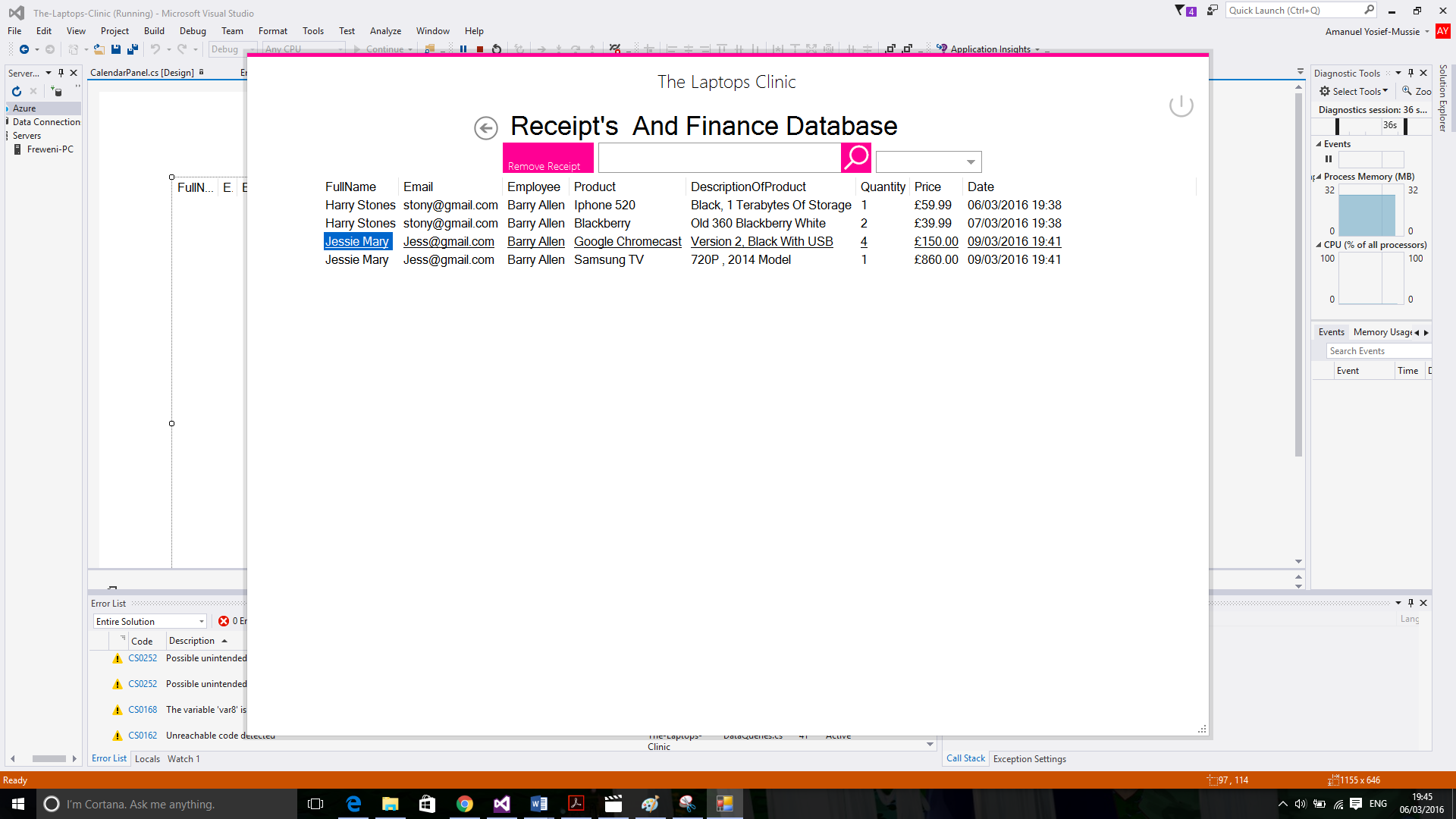
## UI sample of planned valid output designs

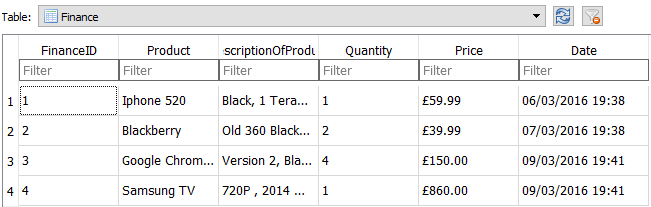
The Output provided in my program consists of mostly Tables and customer/employee details however, one of my objectives was to be able to create a receipt file and let the User Print it out.

I filled in example data to show how the Outputs will be on tables.

In the Receipt and Finance UC, it will show both the Receipt and Finance Table combined together to make one table. Receipt Table contains the ID of the Employee, Of The Finance and ReceiptID as Primary. The EmployeeID tells us who the user is and FinanceID shows us the

product they bought.

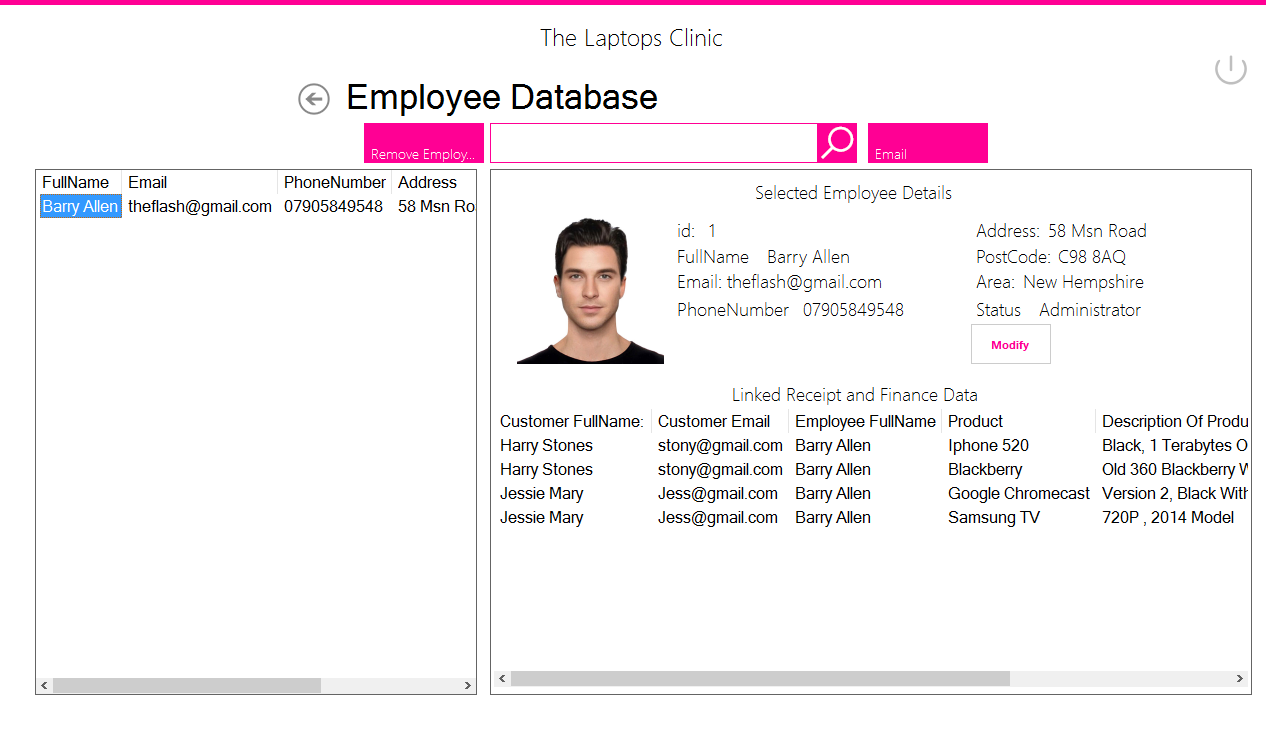




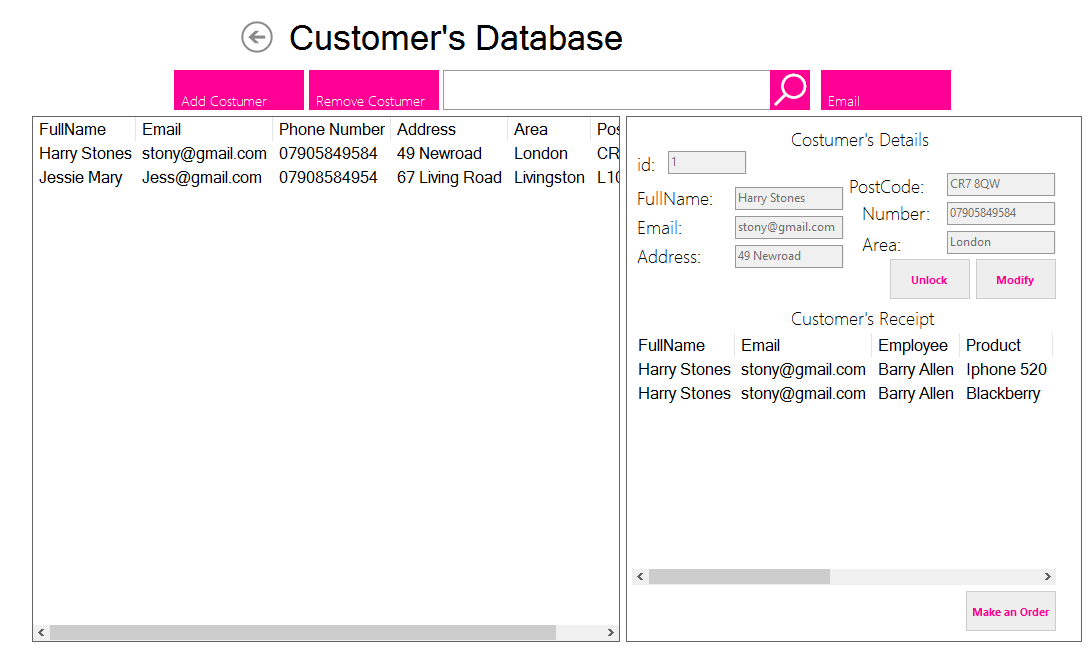
As you can see from the Receipt table in the database, it only consists of FinanceID, CustomerID, EmployeeID. In EmployeeID we have only 1s. That’s the EmployeeID of BarryAllen who is the only current registered employee in this example and we have 4 Barry Allens on ReceiptAndFinance Table as he was the user who conducted the Orders.  
The Other ID gets the Customer’s FullName and Email, while FinanceID gets the Product, DescriptionOfProduct, Quantity, Price, Date. (White and Pink Colour Scheme)

However, in the Customer, Employee UCs. It outputs their details and shows them to the User. Employee have their own Images. In this example. The Employee Profile of BarryAllen in the program will be:

Outputs all the Employees stored in the System



Displays the selected Employee details and any receipts, finance data related to the selected employee



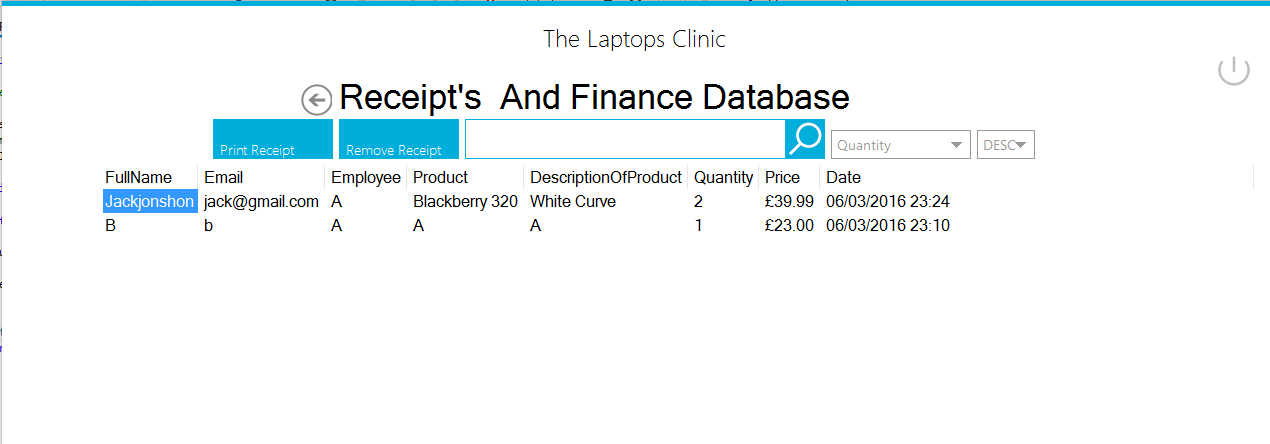
Displays the selected Customer details and any receipts, finance data related to the selected Customer

Outputs all the Customers stored in the System

As stated one of my objects was to be able to print/output receipts. Which I have been able to make during the time bound of my course. The colour scheme in the next screenshots are White Theme and Blue style as the user picked.

It’s a fairly good positions on the top bar of the pane, with buttons two one side and drop down list to the other side. I was also able to add a drop down list for Ascending(ASC) and DESC.   
Its easy to navigate to the button as well.

The user must pick a selected a receipt at first:

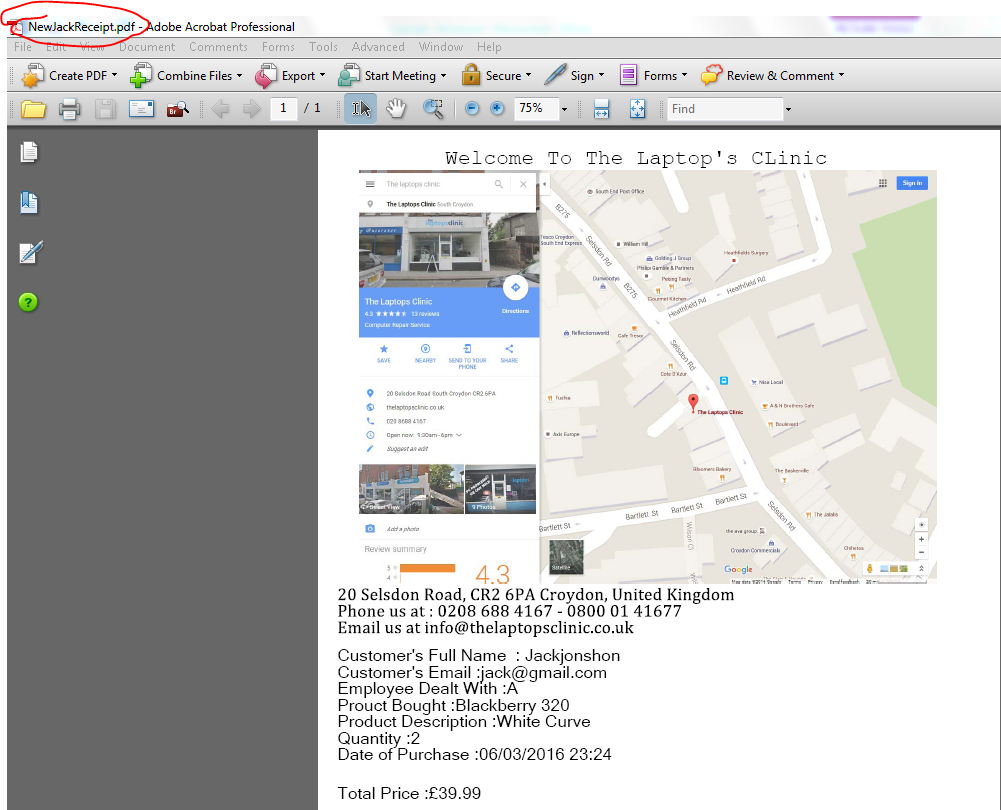


Then upon clicking the options provided on top, they will be able to produce a print and even store the document as a reminder in their own receipts folders if they wish to.

Clicking PrintReceipt causes the PrintDialog to show to let you choose the printer and have properties, however before printing you have to save the document. In the next picture I created a folder for the receipt and saved the new receipt.



Once the PDF containing the receipt is saved, it will automatically open showing the Receipt. This all correlates to the value we had in the ListView, is now printed and saved and can be used again to email which can be send as an attachment in the Email Form.





## Description of measures planned for security and integrity of data

Door

Store counter

PC

Customer

Employee

\*Simplified illustration of the store used to explain\*

**Security** means preventing unauthorised people accessing areas of your work.  
The Laptops Clinic deals with multiples customers daily and deals with them on the store counter; the store counter is just a long table that separate the customers in one side and the employee to the other side. However, in the counter, Laptop and computers (PC) are placed in the store counter, just as any store that has to deal with customers; they talk to them while they use the Laptop/computer to store their information. There is a potential risk of a Customer turning the Laptops/Computer around to “hack” or check for details however that fairly impossible as there is always an Employee in the counter and will be spotted.

The potential security risk comes from the Employee mostly, for example a corrupt Employee might seeks to trash the software with information or delete them, they could for example; delete all the receipts, add tons of unauthorised people as “Employee”, delete customers and employees and maybe even admins.

To limit this security threat, I made sure Administrator have an adminkey, this key is used to give access to features like Adding a New Employee, deleting a Receipt, deleting an Employee, therefore making it hard for corrupt employee to trash the system, also made it so that no Administrator can edit any Employee details. Only Employee can edit their own details.

Other than that, my program meets the strict rules called “the data protection principles”, all the details of the Customer and Employee are used fairly and lawfully, used in a relevant and no excessive way, kept safe and secure, used for a limited, specifically stated purpose. The information we store concerning the Customer is required as a form of contact and delivery to them for the Employees. The program does not store irrelevant information such as Religion, health, background e.g.

**Integrity:** To ensure any data that is stored or inputted is not corrupt, I have used IF/Else condition to catch the data that violate certain rules e.g. Data entered cannot be blank, IF/ELSE checks that the data is blank and acts upon on it. Furthermore, the attribute in my tables have the correct field type (e.g. IDs are integer, Quantity is Integer), so if the user tries to enter a text field type into the Quantity or ID attributes, the software will ignore the text and set the them to 0 or null instead since they don’t match the field type. All these IF/Else and field types ensure the right format is entered in the database.

Referential integrity is also maintained in my program, this means that my program does not allow user to add or manipulate ids for any tables and upon the deletion of a user. When deleting a customer from the database, the program will remove all it details in the customer table and other tables that use that customer’s CustomerID, e.g. Receipt table uses CustomerID to retrieve information about the customer to show in a receipt form and present them in a table to show. If they aren’t deleted to from all the other tables, it will cause error when presenting tables as the customerID won’t exist and will cause the program to crash.

## Description of measures planned for system security

To further prevent unauthorised access, the end user is going to store the program in a secure location or behind operating system passwords. The end user can place the program in a secure file location that is protected and requires Administrator Password to open.

In the event, that the system becomes corrupt or lost. If the system becomes corrupt. If the database is lost, the program at startup will check if the Database exist, if it doesn’t it will create the database again and will function normally. If the system does become corrupt, the user can just reinstall the program that I made and move the database so that they can use it on the reinstalled program.

## Overall test strategy

There are several different ways that you could test my system, i will be testing my system through a combination of black box and white box testing and using break point. There are some parts of the code that can only be tested via black box testing only, for example when navigating before different forms and user controls.

Visual Stuido consists of debugging tools such as breakpoints, which allow the programmer to half the program at a certain line of code, this would be used for white box testing to test the code at the line and check their values. For example, if a breakpoint was made at an SQL Query, it would half the query and allow the programmer to see the code and values of the SQL Query before it is executed. This is white box and will be used to test all the components of the system.

A combination of white Box with breakPoints and black box testing will be used to ensure each part of my program works as designed. Although some of my codes can only be tested by black box testing. However I would use all possible debugging tools offered by Visual Stuido to test my codes, so that no issues would arise causing the program to crash and the system to fail, it is essential each part of my code be tested to minimise the chances of the system crashing when completed and used by the end user.

## Test Plan

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test No. | Description | TEX (Typical, Erroneous, Extreme) | Expected Outcome | Actual Outcome | Comment and Corrective Actions | Code being tested |
| 1 | Program checks if a database exits. – in Login Form | **Typical:**  System check if the Database exists in the directory. If the Database exists, then CreateSystem UC will slide into view.  Else, nothing will come into view and remain at Login Form. | Database does exist and nothing slides into view (remains in Login Form)  If the database does not exist, therefore CreateSystem slides into view. (not possible to show animation) |  |  |  |
| **Erroneous:** |  |  |  |
| 2 | Creating the Database - in CreateSystems UC | **Typical:**  All the fields must be filled in correctly,  Database is created  and the First Administrator/Account is inserted into it and then CreateSystem will slide out of view and return to Login Form | Notify the user that the database has been created along with the first administrator  CreatedSystems UC slides out view and shows login form. (Not possible to show the animation) – |  |  |  |
| **Erroneous:**  Unfilled Fields, Username, **,** Password are incorrect and Confirm Password & Retry Password don’t match | Message boxes will appear when an erroneous event occurs:  By Notify I mean that a message box will appear with the specific message or warning.  Notify the user their required to fill in the required fields to create an account  Notify the password entered and the retry password doesn’t match  Notify the user that the phone number they have entered is not numeric. |  |  |
| 3 | Register UC – A UC that slides into Login form after Register Button is clicked in Login Form.  -Used to register Administrators and Employees | **Typical:**  All Fields must be filled in correctly this include basic information (FullName, email, address e.g.) and login details (Username and Password) and an admin key to grant access to make register an account. User can also pick an image for their Profile if they wish. | Notify the user that the Employee has successfully been added to the database  Notify the user that the Administrator has been successfully been added to the database |  |  |  |
| **Erroneous:**  Admin Key is incorrect,  Required fields are empty and needs to be filled in.  Can’t make an account with the same Username and Email as another Employee  Password and retry passwords don’t match.  Can’t make an administrator account with the same Admin Key as another administrator. | Notify the user that there is a user already with that Username and Password  Notify the user that the required fields are not filled in  Notify the user that the admin key entered is incorrect  Notify the user that the Password and Retry Password don’t match |  |  |
| 4 | Forgot UC– A UC that slides into Login form after a user click Forgot Button.  -User is able to retrieve their forgotten information from the database. | **Typical:**  User uses Forgot UC to retrieve their lost details; these are either Employee Account details (Password or Username) or retrieving the Admin Key. | Selecting a radio buttons will come with the desired reaction (e.g. Selecting (x) radio button will show all the components that is relevant to (x) instead of (y). ;   * User select Employee Account radio button and Forgot Password button   Notify the user that program has successfully updated the Employee account with the new password – 4b   * User selects Employee Account radio button and Retrieve Username radio button   Notify the user that an email has been sent to their Email with their Username   * User selects AdminKey radio button   Notify the user that an email has been sent to their Email with their AdminKey |  |  |  |
| **Erroneous:**  Incorrect FullName and Email provided.  Incorrect Username is provided when user tries to change their password.  Incorrect Password when user is trying to retrieve their Username(which they forgot)  Can’t email, due to no access to the internet.  Can’t email, due to incorrect user details (FullName, Email) | Notify the user that the details provided are incorrect (FullName and Email)  When user seeks to make a new password but enters the wrong username of that Employee account – Notify that there is no Employee in the system with that Username  Notify the user that the password entered and the retry password doesn’t match  When user seeks to get their Username from the system but however enters the wrong password for that Employee Account – Notify that the password entered doesn’t match the password stored for that Employee  Notify the user that they can’t be emailed since they haven’t full email details at the Login Form  Notify that the user cannot be emailed due to no internet access or network issues. |  |  |
| 5 | Logging in – at Login Form | **Typical:**  User fills in their login details and logs into the Main Form. Before logging in user can change theme and provide email details. | User provides correct login details and logins in Main Form.  User decides to click the setting button and a UC appears with tabs and options.  User decides to change the theme and style colour in the Theme tab  User decides to provides the email details and saves these details. – Notify that their details has been saved. |  |  |  |
| **Erroneous:**  Incorrect Username is inputted  Incorrect Password | Notify the user that there is no Employee with that username  Notify that the password provided is incorrect for the username entered.  Notify if the email details couldn’t be saved due to no internet access. |  |  |
| 6 | Main Form | **Typical:**  User clicks on a button to call upon a UC to come in front view of Main Form, another buttons for signing out and settings. | User is redirected to the form or UC relevant to the button click. |  |  |  |
| **Erroneous:**  Double clicking causes for a UC to be called upon twice-thus making two UCs to be made and slide into view. | Causes multiple UCs to cast on top of each, it is possible disable the button clicked for a few seconds after the first click to stop double clicking. |  |
| 7 | Customer UC | **Typical:**  On load, it will present the user with all the customers stored in a ListView, arranged in an alphabetic order  Customer can be selected from the List of Customer and their details will be shown along with their previous receipts.  Customers details can be modified/updates  Customer can be added to the system or deleted and even searched.  An order can be placed for a customer, if the customer wishes to make an order. | After Customer button is clicked, Customer UC is swiped into view and presents the user with a list view of all the customer stored in the system in an alphabetic order  Shows the selected customer in the list, shows their details along with any previous receipts they have ordered).  Notify if the program has successfully updates the Customer details  .  Notify if it failed to update the Customer Customers.  User clicks add customer and causes Add Customer Form to pop into view.  User clicks delete customer, notifies them of this decision with a Yes or No messagebox then either deletes this customer or not, depending on their input. No = nothing changes (therefore no screenshot is provided), Yes = Deletes customer from the list view and refreshes the list view.  If the customer wishes to make an order, employee selects this customer and clicks on Make an Order button. It will pop MakeAnOrderForm into view with the customer and employee details already filled in.  Show the results of any searches made in the search textbox. |  |  |  |
| **Erroneous:** | Notifies there is no customer is selected whenever the Employee clicks delete customer, or modify or make an Order button since you cannot use these buttons if no customer is selected.  Notifies if it couldn’t update the customer’s details. |
| 8 | Employee UC | **Typical:**  On load, it will present the user with all the Employees stored in a ListView, arranged in an alphabetic order  Employees can be selected from the List of Employees and their details will be shown along with any receipts that they helped with.  Employee details can be modified by selecting the employee and clicking Modify, which navigates them to UpdateEmployee Form to modify their details.  Employees can be deleted, however to be able to delete an Employee, you are required to be an administrator.  An order can be placed for a customer, if the customer wishes to make an order. | After Employee button is clicked, Employee UC is swiped into view and presents the user with a list view of all the Employees stored in the system in an alphabetic order  Shows the selected customer in the list, shows their details along with any previous receipts they have ordered.  User selects an Employee and clicks Modify Employee.  If the User is an Administrator that is logged in and selected an employee then clicks delete, it will warn them and provides a Yes or No option to delete this Employee.  Yes causes employee to be deleted  No – nothing occurs.  Then the Employee ListView is refreshed. -  User can search for a specific Employee , the result of the search will be shown in the list view.  Notifies the user when they have successfully deleted an Employee or Administrator from the system. |  |  |  |
| **Erroneous:** | Clicking Modify or Delete Employee without having selected an Employee will causes a messagebox (notify) the user that an Employee needs to be selected.  Not possible to delete an employee without being an Administrator, notifies the user who is logged in that they can’t delete since their not an Administrator.  Administrator is required for providing access to register new employees and other features, thus therefore it is not possible to delete all the administrator in the database or you won’t be able to add/delete employee e.g.  Due to this, it will notify the user that it ain’t possible to delete the only administrator stored in the system unless they add a new administrator.  Notify the user when their deleting oneself (administrator deleting themselves) as the system will have to delete the user and close the Employee UC, main form , navigating back to the login form.  Notify the user that they can’t modify the details of other accounts since it doesn’t belong to them |
| 9 | Receipt & Finance  UC | **Typical:**  Shows all the receipts that are stored in the system, linking them to the customer, employee and the product (finance) it belongs to, displaying them.  User can search, delete and rearrange the order of how the receipts are displayed in the list view.  User can also print out receipts. | After Receipt & Finance button is clicked, the UC will swipe into view, with a loaded list of all the receipts. It will be in ordered by descending date on load.  User can modify the order of the list.  User can search for a specific receipt  User can have receipt printed out.  User can delete receipts, however required to be an administrator to delete and be notified when the receipt is printed out. |  |  |  |
| **Erroneous:**  No administrator rights  User needs to select an receipt before deleting or modifying it. |  |  |  |
| 9 | Updating Employee at UpdateEmployee Form | Gets the selected employee from the Employee Table in Employee UC, display their value and lets the user modify the value and save it to the database. Employee can only edit their own data so they need to verify the Employee by Username and Password. | Updates the Employee details |  |  |  |
| **Erroneous:**  Details provided to verify that their same Employee that their modifying is incorrect.  Required Fields are empty | Messagebox to show incorrect login details  Messagebox to alert the fields aren’t empty |  |
| 10 | Add a Customer at AddCustomer form | Employee adds the details of the Customer and it is stored in the database.  Required fields must be filled | Notify (messagebox) the user that they customer has been added into the that database . |  |  |  |
| **Erroneous:**  Customer already exists with the same email,  FullName and Email not filled in,  Failed to Add Customer | Notify the user that there is an already a customer with the same email.  Notify the user that they need to fill the FullName and Email to be able to add a customer.  Notify the user that they need to input a correct phone number (integer).  Notify the user if the Insert SQL failed to run. |
| 11 | Make An Order at MakeAnOrder Form | Employee adds the details of the Customer’s order and it is stored in the database.  Required fields must be filled and right format of input e.g. Quantity is integer values  Saves the details of the purchase,  Converts the price into British pound currency | Notify the user when the purchase/order has been saved. |  |  |  |
| **Erroneous:**  Employee needs to enter a correct price and quantity (needs to be integer) no letters or currency symbols needed.  Employee needs to fill in all the fields.  Failed to add the order. | Notifies the user to fill in all the fields and quantity and price must both be integer values.  Notifies if it fails to execute the Insert SQL |
| 12 | Employee adds a Calendar Event in Calendar UC | Calendar event is added to the calendar and inserted into the database.  All the fields must be filled to be able to add calendar events or delete them  Loads the calendar events when the form loads | Notifies the user when program has succeeded in adding the calendar event to the database or deleting it from the database  Loads the calendar events are the start and presents them. |  |  |  |
|  | Notifies the user to provide all the fields, checks that the components aren’t empty E.g. Time can’t be empty due their required to specifying the time of the event.  Notifies the user if the system failed to execute the INSERT or Delete SQL |

## Navigation Testing Design

To be able to see at a glance which forms and UCs are linked to each other and which can only navigate through other forms or UCs, I’ve created a table. It is unidirectional (being able to navigate from form A to B for example does not mean that the user can directly navigate from form B back to A, it may only be able to return focus to A upon closing.)

The table can simply be filled in with ticks or crosses to test the navigation works as expected. Until implementation is complete, I cannot know for sure how many forms or UCs there will be in the system so the following table with forms A-H is merely an estimate. The finished system could have more or less, but not drastically so.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Login | CreateSystems | Forgot | Register | Main | Customer | AddCustomer | MakeAnOrder | Email | Employee | Update Employee | Receipt and Finance | Calendar |
| Login |  |  |  |  |  |  |  |  |  |  |  |  |  |
| CreateSystems |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Forgot |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Register |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Main |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Customer |  |  |  |  |  |  |  |  |  |  |  |  |  |
| AddCustomer |  |  |  |  |  |  |  |  |  |  |  |  |  |
| MakeAnOrder |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Email |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Employee |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Update Employee |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Receipt and Finance |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Calendar |  |  |  |  |  |  |  |  |  |  |  |  |  |

# Testing

## Input and Output Testing

This table is an overview of the basic input and output tests conducted on the system in order to make sure the expected outcome of every action is the actual outcome when using the system.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test No. | Description | TEX (Typical, Erroneous, Extreme) | Expected Outcome & (white box & black box reference) | Actual Outcome | Comment and Corrective Actions | Code being tested |
| 1 | Program checks if a database exits. – in Login Form | **Typical:**  System check if the Database exists in the directory. If the Database exists, then CreateSystem UC will slide into view.  Else, nothing will come into view and remain at Login Form. | Database does exist and nothing slides into view (remains in Login Form) – 1a)  If the database does not exist, therefore CreateSystem slides into view. (not possible to show animation) -1b) | As expected. | None required. |  |
| **Erroneous:** |  |  |  |
| 2 | Creating the Database - in CreateSystems UC | **Typical:**  All the fields must be filled in correctly,  Database is created  and the First Administrator/Account is inserted into it and then CreateSystem will slide out of view and return to Login Form | Notify the user that the database has been created along with the first administrator – 2a)  CreatedSystems UC slides out view and shows login form. (Not possible to show the animation) – 2b | As expected. | None required. |  |
| **Erroneous:**  Unfilled Fields, Username, **,** Password are incorrect and Confirm Password & Retry Password don’t match | Message boxes will appear when an erroneous event occurs:  By Notify I mean that a message box will appear with the specific message or warning.  Notify the user their required to fill in the required fields to create an account – 2c  Notify the password entered and the retry password doesn’t match – 2d  Notify the user that the phone number they have entered is not numeric. -2e | As expected. |  |
| 3 | Register UC – A UC that slides into Login form after Register Button is clicked in Login Form.  -Used to register Administrators and Employees | **Typical:**  All Fields must be filled in correctly this include basic information (FullName, email, address e.g.) and login details (Username and Password) and an admin key to grant access to make register an account. User can also pick an image for their Profile if they wish. | Notify the user that the Employee has successfully been added to the database – 3a  Notify the user that the Administrator has been successfully been added to the database -3b | As expected. | None required. |  |
| **Erroneous:**  Admin Key is incorrect,  Required fields are empty and needs to be filled in.  Can’t make an account with the same Username and Email as another Employee  Password and retry passwords don’t match.  Can’t make an administrator account with the same Admin Key as another administrator. | Notify the user that there is a user already with that Username and Password – 3c  Notify the user that the required fields are not filled in – 3d)  Notify the user that the admin key entered is incorrect – 3e)  Notify the user that the Password and Retry Password don’t match- | As expected. |  |
| 4 | Forgot UC– A UC that slides into Login form after a user click Forgot Button.  -User is able to retrieve their forgotten information from the database. | **Typical:**  User uses Forgot UC to retrieve their lost details; these are either Employee Account details (Password or Username) or retrieving the Admin Key. | Selecting a radio buttons will come with the desired reaction (e.g. Selecting (x) radio button will show all the components that is relevant to (x) instead of (y). ;   * User select Employee Account radio button and Forgot Password button -4a)   Notify the user that program has successfully updated the Employee account with the new password – 4b   * User selects Employee Account radio button and Retrieve Username radio button – 4c   Notify the user that an email has been sent to their Email with their Username – 4d   * User selects AdminKey radio button– 4e   Notify the user that an email has been sent to their Email with their AdminKey – 4f | As expected. | None required. |  |
| **Erroneous:**  Incorrect FullName and Email provided.  Incorrect Username is provided when user tries to change their password.  Incorrect Password when user is trying to retrieve their Username(which they forgot)  Can’t email, due to no access to the internet.  Can’t email, due to incorrect user details (FullName, Email) | Notify the user that the details provided are incorrect (FullName and Email) –4g  When user seeks to make a new password but enters the wrong username of that Employee account – Notify that there is no Employee in the system with that Username – 4h  Notify the user that the password entered and the retry password doesn’t match – 4i  When user seeks to get their Username from the system but however enters the wrong password for that Employee Account – Notify that the password entered doesn’t match the password stored for that Employee – 4j  Notify the user that they can’t be emailed since they haven’t full email details at the Login Form– 4k  Notify that the user cannot be emailed due to no internet access or network issues. – 4l | As expected. |  |
| 5 | Logging in – at Login Form | **Typical:**  User fills in their login details and logs into the Main Form. Before logging in user can change theme and provide email details. | User provides correct login details and logins in Main Form. -5a)  User decides to click the setting button and a UC appears with tabs and options. -5b)  User decides to change the theme and style colour in the Theme tab – 5c)  User decides to provides the email details and saves these details. – Notify that their details has been saved. – 5d) | As expected. | None required. |  |
| **Erroneous:**  Incorrect Username is inputted  Incorrect Password | Notify the user that there is no Employee with that username – 5e)  Notify that the password provided is incorrect for the username entered. – 5f)  Notify if the email details couldn’t be saved due to no internet access. -5g) | 6b)  6c) |  |
| 6 | Main Form | **Typical:**  User clicks on a button to call upon a UC to come in front view of Main Form, another buttons for signing out and settings. | User is redirected to the form or UC relevant to the button click. | As expected. | None required. |  |
| **Erroneous:**  Double clicking causes for a UC to be called upon twice-thus making two UCs to be made and slide into view. | Causes multiple UCs to cast on top of each, it is possible disable the button clicked for a few seconds after the first click to stop double clicking. |  |
| 7 | Customer UC | **Typical:**  On load, it will present the user with all the customers stored in a ListView, arranged in an alphabetic order  Customer can be selected from the List of Customer and their details will be shown along with their previous receipts.  Customers details can be modified/updates  Customer can be added to the system or deleted and even searched.  An order can be placed for a customer, if the customer wishes to make an order. | After Customer button is clicked, Customer UC is swiped into view and presents the user with a list view of all the customer stored in the system in an alphabetic order – 7a)  Shows the selected customer in the list, shows their details along with any previous receipts they have ordered. -7b)  Notify if the program has successfully updates the Customer details.- 7b)  Notify if it failed to update the Customer Customers.-7c)  User clicks add customer and causes Add Customer Form to pop into view. – 7d)  User clicks delete customer, notifies them of this decision with a Yes or No messagebox then either deletes this customer or not, depending on their input. No = nothing changes (therefore no screenshot is provided), Yes = Deletes customer from the list view and refreshes the list view.  If the customer wishes to make an order, employee selects this customer and clicks on Make an Order button. It will pop MakeAnOrderForm into view with the customer and employee details already filled in. -7e)  Show the results of any searches made in the search textbox. – 7f) |  |  |  |
| **Erroneous:** | Notifies there is no customer is selected whenever the Employee clicks delete customer, or modify or make an Order button since you cannot use these buttons if no customer is selected. – 7g)  Notifies if it couldn’t update the customer’s details. -7h) |
| 8 | Employee UC | **Typical:**  On load, it will present the user with all the Employees stored in a ListView, arranged in an alphabetic order  Employees can be selected from the List of Employees and their details will be shown along with any receipts that they helped with.  Employee details can be modified by selecting the employee and clicking Modify, which navigates them to UpdateEmployee Form to modify their details.  Employees can be deleted, however to be able to delete an Employee, you are required to be an administrator.  An order can be placed for a customer, if the customer wishes to make an order. | After Employee button is clicked, Employee UC is swiped into view and presents the user with a list view of all the Employees stored in the system in an alphabetic order – 8a)  Shows the selected customer in the list, shows their details along with any previous receipts they have ordered. -8b)  User selects an Employee and clicks Modify Employee – 8c)  If the User is an Administrator that is logged in and selected an employee then clicks delete, it will warn them and provides a Yes or No option to delete this Employee.  Yes causes employee to be deleted  No – nothing occurs.  Then the Employee ListView is refreshed. -8d)  User can search for a specific Employee , the result of the search will be shown in the list view. – 8e)  Notifies the user when they have successfully deleted an Employee or Administrator from the system. – 8f) |  |  |  |
| **Erroneous:** | Clicking Modify or Delete Employee without having selected an Employee will causes a messagebox (notify) the user that an Employee needs to be selected. -8g)  Not possible to delete an employee without being an Administrator, notifies the user who is logged in that they can’t delete since their not an Administrator. – 8h)  Administrator is required for providing access to register new employees and other features, thus therefore it is not possible to delete all the administrator in the database or you won’t be able to add/delete employee e.g.  Due to this, it will notify the user that it ain’t possible to delete the only administrator stored in the system unless they add a new administrator. -8i)  Notify the user when their deleting oneself (administrator deleting themselves) as the system will have to delete the user and close the Employee UC, main form , navigating back to the login form. -8j)  Notify the user that they can’t modify the details of other accounts since it doesn’t belong to them – 8k) |
| 9 | Receipt & Finance  UC | **Typical:**  Shows all the receipts that are stored in the system, linking them to the customer, employee and the product (finance) it belongs to, displaying them.  User can search, delete and rearrange the order of how the receipts are displayed in the list view.  User can also print out receipts. | After Receipt & Finance button is clicked, the UC will swipe into view, with a loaded list of all the receipts. It will be in ordered by descending date on load. -9a)  User can modify the order of the list. – 9b)  User can search for a specific receipt -9c)  User can have receipt printed out. -9d)  User can delete receipts, however required to be an administrator to delete and be notified when the receipt is printed out. –9e) | As expected. | None required. |  |
| **Erroneous:**  No administrator rights  User needs to select an receipt before deleting or modifying it. |  |  |  |
| 9 | Updating Employee at UpdateEmployee Form | Gets the selected employee from the Employee Table in Employee UC, display their value and lets the user modify the value and save it to the database. Employee can only edit their own data so they need to verify the Employee by Username and Password. | Updates the Employee details |  | None required. |  |
| **Erroneous:**  Details provided to verify that their same Employee that their modifying is incorrect.  Required Fields are empty | Messagebox to show incorrect login details  Messagebox to alert the fields aren’t empty |  |
| 10 | Add a Customer at AddCustomer form | Employee adds the details of the Customer and it is stored in the database.  Required fields must be filled | Notify (messagebox) the user that they customer has been added into the that database -10a) | As expected. | None required. |  |
| **Erroneous:**  Customer already exists with the same email,  FullName and Email not filled in,  Failed to Add Customer | Notify the user that there is an already a customer with the same email -10b)  Notify the user that they need to fill the FullName and Email to be able to add a customer -10c)  Notify the user that they need to input a correct phone number (integer). -10d)  Notify the user if the Insert SQL failed to run.  -10e) |
| 11 | Make An Order at MakeAnOrder Form | Employee adds the details of the Customer’s order and it is stored in the database.  Required fields must be filled and right format of input e.g. Quantity is integer values  Saves the details of the purchase,  Converts the price into British pound currency | Notify the user when the purchase/order has been saved. | As expected. | None required. |  |
| **Erroneous:**  Employee needs to enter a correct price and quantity (needs to be integer) no letters or currency symbols needed.  Employee needs to fill in all the fields.  Failed to add the order. | Notifies the user to fill in all the fields and quantity and price must both be integer values.  Notifies if it fails to execute the Insert SQL |
| 12 | Employee adds a Calendar Event in Calendar UC | Calendar event is added to the calendar and inserted into the database.  All the fields must be filled to be able to add calendar events or delete them  Loads the calendar events when the form loads | Notifies the user when program has succeeded in adding the calendar event to the database or deleting it from the database  Loads the calendar events are the start and presents them. | As expected. | None required. |  |
|  | Notifies the user to provide all the fields, checks that the components aren’t empty E.g. Time can’t be empty due their required to specifying the time of the event.  Notifies the user if the system failed to execute the INSERT or Delete SQL |

## Black Box Testing (Images)

Screenshots showing the output of my white box testing. These are not the official end design of my program.

|  |  |  |
| --- | --- | --- |
| Test No |  | Black Box Image |
| 1 | a) | Break points are used to show my white box testings, breakpoints are a feature provided by the debugger in Microsoft Visual Studio that allows breaking execution of code at runtime when debugging application. However, over an element will show their value at run time.  Using the IF condition, we can work out if the code worked or not.  If its True, then nothing occurs as it implies that the database has been located in that directory.  Else it would have CreateSystem slide into view. |
|  | b) | Using code in 1a), if mypath is false so it hasn’t found the directory, it will cause the UC (CreateSystemUC) to swipe(true), swipe(true) is used to initate the swipe animation and makes the form host the UC.When set to swipe(true), then it will animate into view and come in front of Login Form.  When its swipe(false), then it will come out of the view and show the login form instead. |
| 2 | a) | If the Boolean returns true , then therefore the system has created the tables. It is possible for me to make boolean variables to show they all return true for all the other Tables (Employee e.g.) but I will provide a screenshot of my SQL database browser which shows all the tables. If CreateAdministratorTable = true therefore the first line of code is correct with the CreateFile(“TheLaptopsClinicDatabase.sqlite”) since you can’t make a table in a database that doesn’t exist.    Made another Boolean variable to test if it actually inserted an Adminstrator into the system, as you hover over the Boolean variables, it returned true, so therefore it has successfully inserted the Administrator. Then a MessageBox will appear showing that it has created the database.  and then swipe(false), as explained in 1b), it will cause it to slide out view(2b) and show the Login Form. This leads the user back to 1a)- with the view of the Login Form. |
|  | b | A screenshot of the animation, after the user click “OK” on the message box on 2a), it will save their details and swipe CreateSystems out of view. Red arrow shows the direction of the swiping out. |
|  | 2c | Checks that the txt boxes are not white space or null. If they are then a messagebox will show. |
|  | 2d) | Above screenshot shows that, if the txtAdminPassword.Text and TxtConfirmAdminPassword it will cause a messagebox to appear. (below screenshow). Placed a breakpoint before the message box appears, to show what the values of txtAdminPassword.Text and TxtConfirmAdminPassword were.  Above screenshot shows they are “password1” and “password123” which then caused the messagebox to pop up. |
|  | 2e) | tested that this code works by entering “twentyfive” a string instead of a number into the txtAdminPhoneNumber.Text, since it is not an integer. It will cause the messagebox to appear shown below. |
| 3 | 3a) | The code for inserting Employee and Administrator is similar to the code shown above, in Administrator you have an extra fields (AdminKey) and uses the Administrator Table. However in this example. I made a boolean Variable to test where if the Insert into Employee worked. Its called “DoesitInsertEmployee”. I made a break point before the messagebox appears. When hovering over the boolean variable during run time it shows that its true therefore it has inserted the employee. If its true, as shown in the highlighted yellow code, then a messagebox would appear, shown below. |
|  | 3b) | Screenshot of the code. Similar to 3a), I made a Boolean variable to check if the ExectueNonQuery worked or not (return true or false). To register an Administrator, you insert them into Employee which (DoesitInsertAdministrator) checks. GetEmployeeID is a string which gets the EmployeeID of the employee we just inserted. Then to make the employee an Administrator, we use their EmployeeID and insert it into Administrator. (InsertEmployeeIntoAdministrator) checks if it has inserted it into Adminstrator , by either returning true or false. As shown, in the above screenshot, they returned True, 2- which is the EmployeeID and true also for inserting into Administrator.  The Highlighted Code checks where it has returned true for inserting into Employee and Administrator, so therefore an Administrator has been added and shows the message box. Shown Below. |
|  | 3c) | After the user clicks the Register button, it would check if there is an Employee in the database that has the same email or username. It does it by having a Select Query From Employee and using the OR, to check for either Username or Email.  In this example, CheckIfExists returned true, therefore an Employee already exists with the same email OR username, if its True, then messagebox is box. |
|  | 3d) | After Register button is clicked (btnRegister\_Click). It will check that all the values entered are not Null or WhiteSpace then finally it would try and parse (txtPhoneNumber.Text) – to try and parse is to try and convert a value into an integer. try and parse int64 (txtphone.text), will convert the value of the phone number to string, if it works, then therefore the phonenumber entered is a integer, however in this example txtPhoneNumber.Text is not numeric and would therefore make the program show a messagebox. |
|  | 3e) | To check if the AdminKey is correct and exists, it will run a Select Query in the Administrator Table to check if it exists. Boolean IsAdminkeyCorrect was created to test this, as if the Adminkey existed, it would return true else if it did not exist then it would return false. In this test run example, I entered an AdminKey which would return false. When it returns false, it will return a messagebox to alter the user that the AdminKey is incorrect. |
| 4 | 4a) | As the code below showns, when the Forgot Password is checked, it would cause the prompt text of the textbox to be “Provide your Username” and make the two textboxes below is visible and enabled.  It is due as when Forgotten Details is loaded, Retrieve username is checked instead of Forgot Password. The effects of Retrieve Username is that It makes the text buttons not visible nor enabled. Forgot Password checked counter reacts these, by changing the promt message from “Provide Your Password” into “Provide your Username” and enables the txtboxes by setting their Visible and Enabled to True. |
|  | 4b) | The program checks that the details provided (FullName and Email) is correct, the Variable CheckForDetails runs a boolean SQL Select Query to check if it does, in this example it returned true, so therefore the Employee Exists, else if it returned false, a messagebox would appear as shown in 4g) In this example, the user has selected Forgot Password (radio button) and needs to provide the correct Username. The boolean variable CheckForUsername runs SQL query for that entered Username, it returned true as shown in the pop up message as run time. Else if the username provided doesn’t exists, messagebox is shown altering them that there is no Employee with that Username name as shown in 4h).  However, it returned true as shown and then the user is requried to enter a new password for that Employee account that exists and has the same username. It checks that the text boxes aren’t empty and that “txtConfirmNewPassword.Text == txtPassword.Text” (double confirmation for password). If it returns true messagebox is shown as in 4i). Nevertheless, I provided the correct details and the software now hashes the new password and updates the Employee Account.  Boolean Variable UpdateEmployee is used to check whether the employee account has been Updated, because all the other conditions have returned true, UpdateEmployee should return true unless there is a mistake in the SQL query. As shown in the above image, it returned true so has sucessfully updated Employee, notifies the user with the messagebox shown below. |
|  | 4c) | As explained in 4a). When the ForgotUsername is checked, it would counter act the effects of Forgot Password. In 4a) ForgotPassword makes the two textboxes visible and changes the prompt to “Provide Yoru Username” however, when RetrieveUsername is checked, it would set the text (txtUsernameOrPassword.WaterMark” to “Provide Your Password”, before that it had “Provide your Username as its value as shown. Then sets the textboxes to false, which were originally true. This is because they are not required when Retrieveing Username. |
|  | 4d) |  |
|  | 4e) | Not possible to test this out using breakpoint, it is possible to see rbForgotUsername.checked change from False to true. As explainined in 4c). Checking ForgotUsername would make the textboxes not visible, that’s the whole point in this example as those components are not required to be visible or enabled when dealing with Administrator Key, their only used in Forgot Password.  This screenshot shows as the components required for Administrator Key swipe into view and hover over the other components under it. If rbForgot.Checked isn’t set to true then the two extra textbox would be shown which aren’t required and would look like :  which doesn’t look pleasant at all and are not required. Hence why rbForgotUsername is checked in the code above to hide the components and make it look like this instead: |
|  | 4f) |  |
|  | 4g) | Explained in 4b as well).    If CheckForDetails returns false, then the If( checkForDetails == true) is false and the else of the If Condition would make a messagebox appear as shown in the code above. Messagebox shown below. |
|  | 4h) | Explained in 4b) |
|  | 4i) | Explained in 4b) |
|  | 4j) |  |
|  | 4k) | When the user has forgotten their Username or Administrator key, they are emailed their forgotten details.  In this example, I didn’t fill in all the Email Details – (check 5d) on saving Email details to the program.  The program firstly checks that there is internet available before emailing the Employee.  Bool bb in the 3rd line of code, check if internet access is available. It has returned true , due to internet access was available in this test run, the program then retrieves the email address of the Employee to email to. In this example, the user is retrieving their Username so has provided their password (storedHashedPassword) as a method of confirmation of being the actual account holder. It retrieves the email, in which it was [abonobmussie@gmail.com](mailto:abonobmussie@gmail.com) at this run time example.  Then an If Condition is ran to check the email details stored in Variables (Global Variables) aren’t empty and the email retrieved from the Employee isn’t empty also.  Due to the fact I didn’t provide email details, Variabiles.GetEmail is “” , since I didn’t provide and save the details, this cause the messagebox to appear as shown below.  If there is no Internet connection then bb returns false and shows a messagebox as shown in |
|  | 4i) |  |
| 5 | 5a) |  |
|  | 5b) | Metro framework has 14 avaiable styles with a default, the default style is blue, so therefore 13 colours instead of 14. Each colour corresponds with a number. I will be using this number tags for each colour to change the style colours.    When the Login Form loads, it will create metrotiles(buttons), they will have size of 30,30 and will have a tag value of “i”, since there is 14 colours, it will loop 14 times and assign each button a tag number and creates a click function for each button. At the end of the for loop, it would have made 14 buttons which have different styles colours and their event click handler is called “ \_tile\_Click”, which is used to set the metrostyle of a form into the style of the MetroTile(button) clicked.  At start/default, the style is blue shown in the pop up. As shown in the highlighted code above (msmLogin) , the default(when the programloads each time) is blue, however when you clicked a metrotile which has a tag and colour set to it, it would change the style colour of msmLogin.Style to the tag. In example 5c), Orange metrotile is clicked, orange as a tag number of 8 as shown in the list of colours and number above. This causes the style of the form to change to orange. As shown in 5c)  To change themes,    User will either check a white or dark radio button and then check the theme. In the 5c) example. Black radiobutton was checked, which turned the the white them into the dark theme, as shown during run time.  The screenshot shown all the metrotiles on the side with all their different style colours available. |
|  | 5c) | Explained in 5b) |
|  | 5d) | For the user to be able to use the email function, they are required to provide the system with the email details, client, port and their login details to be able to email from the program. In this example, I filled in the fields and clicked the “Test and Save” button.    Once “Test and Save” (btnEmailTest) button is clicked, it will firstly check for internet access, in this example, internet access is available. Therefore bb returns true, when hovered over durng run time, since internet is available. It would then save the email details to Variables(which stores global variables) and show a messagebox telling them that it has successfully saved their details. Message box is shown below. |
|  | 5e) | Once btnLogin is clicked, at login. To check that the Employee Username exists, we run a select query for the encrypted hashed-salted password of that employee and save it in encryptedPassword. If encryptedPassword is equal to null , it would mean that the Select Query has found no employee with that Username therefore returns null. As shown in this example, I entered an username which isn’t stored in the system and has returned null. This causes the messagebox to appear as shown below. |
|  | 5f) | Similar to 5e), in this runtime, I provided an Employee username that actually exists, so therefore encryptedpassword doesn’t return null, but returns the encrypted password as shown in the image.  This means the Username inputted is correct, the next lines check that the entered password is correct. It gets the user input and compares with the encrypted password using the Hashing class. (It encrypts the input password of the user (txtPassword.Text) and check if it returns encrypted). Boolean match is used for this example to show whethere if the inputted password matches the password stored for the employee.  I provided an incorrect password for this example, so matching has returned false when hovered during run time, thus causing the messagebox shown below to appear.    Else if match is true  In this run time, I provided a existing Username and correct password so match returns true as shown. If Match is true, it will do a Select Query to recheck that the there is an Employee with that Password as return a Boolean to Employee Login. Employee Login returns true at run time as the details are all correct, it would then also run more Select Query which are used to get details of the Employee stored, for example their ID ( EmployeeID), in this example, the employee logged in has the ID of “1” and saves these details along with their Username and FullName to Variables(which stores the global variables). These are required in the system as e.g. when employee is deleteing; we need their ID to confirm if their administrator, when creating receipts, the employee ID is required and Username to link the receipt to employee. Having these details saved, shows us who is currently logged in.  After it has saved these details, it would create a new instance of Main Form and pass the theme (light) and style (blue) to it so that it would have a light theme and blue style. This is how forms and usercontrols have the same theme and style since we are passing the theme and style around when making them appear (creating new instances). Then the login form will “this.hide”, which hides the form and OpenMainForm.ShowDialog() shows the main form which Login form is out of view. 6a) shows the Main Form after the code has been run. |
|  | 5g) |  |
| 6 |  |  |
| 7 | 7a) | Customer UC is swiping into view, infront of the components of the Main Form. Arrow shows the direction. You can also see the list of the customer already has been done quickly while its swipping into view. |
|  | 7b) | In this example, I selected an Customer called Tyrion Lannister, it shows me his details on the right on the screen and any receipts made. Unlock button is used to make the textbox which has Tyrion Lannister’s details to become enabled so that Employee can change them and Modify them by clicking the Modify button. |
|  |  | Continuing with the example, I unlocked the text boxes that contain the details of Tyrion Lannister, I changed his FullName to “Robert Lannister” and modified it. Clicking modify while update this name and ensure that his receipts are also mainted to his account. As you can see the Textboxes in the image are more visible instead of the Id since that can’t be modified. After modifying , it will lock the textboxes and refresh the customer list.  Name is changed, has the same ID and still the same receipts and textboxs are automatically locked . |
|  | 7c) | Deleting Customers  When deleting a customer, there must be a customer picked, hence why CustomerListView.SelectedItem.Count returns in the screenshot above to show that a customer is picked. Since a customer is already picked, and the delete button is clicked, it would show a messagebox (shown below) of the customer their about to delete. The yes or no response in the screenshot below, decide what the value of the DiagonalResult DeleteUser is.    In this example, DeleteUser has a value of Yes since Yes button was clicked to delete the employee. Therefore if DeleteUser returns Yes then, it would do a DELETE SQL to remove them and refresh the Customer Table. (ShowCustomerTable()) |
|  | 7d) |  |
|  | 7e) | In this example, I selected Adrianne Palicki and clicked MakeAnOrder. Pops MakeAnOrder into view, with Adrianne’s details and the current employee (their username e.g. AmanuelManny in this example) logged in. Receipts needs to be able to link to the Customer and Employee that dealt with them.  Behind the MakeAnOrder, we can see Adrianne’s details on the right as she is selected. |
|  | 7f) | In this example I searched for any users that have a phone number that starts with “0790”  The highlighted yellow line shows the breakpoint on which the program is halted to see the variables and components value at that present line of code.  Since I searched “0790” it runs a Select Query using LIKE txtSearchBar.Text which has the value of 0790, showing that my program is running the 0790 search. It returns a DataTable which stores all the values that return true (all the customers that have “0790”) and using the DataTable for every row, we add them to the list. As shown in the first run of this search, “Brett Dalton” is being added to the list” |
|  | 7g) | Texts boxes on the right side are empty due to no customer is selected.    CustoemrListView.SelectedItems.Count returns 0 therefore no customer is selected from the Customer ListView, hence they the messagebox pops up alerting the user and the textboxs are empty on the right side since no customer is selected. |
|  | 7h) |  |
|  | 8a) | Not possible to show the animations but a screenshot showing the UC swiping into view from the side. The Employee UC will hover the Main Form, the list of employee is already loaded but won’t be seen until the UC has fully slided into view. |
|  | 8b) | Selecting an Employee, shows all their details and an Image of their login account. Textboxes are not used like in the Customer UC. Sansa Stark is selected here and as shown, it has a blank list in the Linked Receipt and Finance data , this is due to the fact Sansa Stark has yet done any orders for any customers at all.    Each time a different employee is selected, it grabs their details as shown in the Screenshow above, it has taken Sansa Stark’s FullName and Email, and has gotten their ID (string GetEmployeeID) by running a SELECT Query from the Employee Table using the FullName and Email provided, and uses this GetEmployeeID to run a SELECT Query on Administrator to check whether their an Employee or Administrator, depending on this, it will set lblEmployeeStatus to either Employee or Administrator.  After that, it will call upon the sub routine GetSelectedEmployeeImage() to retrieve the Sansa Stark’s user image and also call upon ShowEmployeeReceiptAndFinanceTable providing them the lblID.text which is the Employee ID, this is used to show any receipts that the Employee has dealt with.    The screenshot shows the Employee Sansa Stark, their details, their status and id also. It has a blank Linked Receipt and Finance Data due to the fact Sanasa Stark hasn’t done any receipts. |
|  | 8c) | Carrying on with the example of using Sansa Stark. I have logged in as Sansa Stark and decided to click Modify Employee.  Since Sansa Stark is selected, EmployeeListView.SelectedItems.Count returns 1 since an Sansa is selected, else a messagebox would have appeared shown in 7g).  The program that I have made doesn’t allow other users to modify other user’s details, hence why it checks the current logged in user’s ID and the employee that is to be modified ID. Sansa Stark is logged in and her ID was saved in Variable.userID – (Check 5c to see how the ID of employee is stored to Variables when login). The program compares the logged in Employee ID (Variable.UserID) and the selected Employee to be modified (lblID.text), if their equal then it means that the current logged in user is modifying their own details. In this example, they both returned 2 (check screenshot above) so its true. It would then call upon UpdateEmployee Form and pass the details of the Employee selected from Employee UC to UpdateEmployee Form.    Screenshot above shows what occurs after Update Employee is clicked and UpdateEmployeeForm shows up with the selected employee (Sansa Stark) with their details and image. |
|  | 8d) | This is similar to 7c) It retrieves the selected Employee to be deleted details and then presents it in a MessageBox as shown below, if the user selects Yes on the messagebox, then DeleteUser would be set to “Yes”. IF DeleteUser is set to Yes, it would then therefore delete the Employee, showing messagebox to inform the user they have deleted the Employee – 8f).  Check 7c) for more details. |
|  | 8e) |  |
|  | 8f) |  |
|  | 8g) | An Employee must be selected before they can be modified or deleted, in this breakpoint. EmployeeListView.SelectedItemsCount returns 0, indicating no employee is selected in the Employee ListView, hence why the lables to the side and image is empty and null and messagebox pops up: |
|  | 8h) |  |
|  | 8i | To ensure and the program doesn’t delete all the administrators in the system, it will do firstly count all the Administrators stored in the database. CountsAdmin is an Int Variable which runs a SQL Query to count the amount of Administrators stored.  If CountsAdmins returned 1 instead of 5, and lblEmployeeStatus.Text == ‘Administrator’ –(meaning the employee to be deleted and is selected is an Administrator) – so the last administrator is being deleted since, it would return a messagebox alerting the user that you can’t delete the only stored administrator in the system.  However in this example, it CountsAdmins retuns 5, therefore there is 5 administrator stored in the database.  Messagebox below shows what the messagebox would show if the selected Employee is Administrator as shown in the screenshot and CountsAdmins is 0.  Administrator Amanuel Yosief-Mussie is selected here, Status Administrator also. |
|  | 8j) | Even though you can see the employee details of “Sansa Stark” on the right, it doesn’t mean their selected, but rather that she was peviously selected. Will make modifcation that when no employee is selected that those fields and image is empty however, it doesn’t affect the program. A user still needs to be selected to delete or modify. Just cause Sansa Stark’s details are shown on the right, it doesn’t mean their selected.  In the example below, I added more employees to the system then searched for employees that have phone numbers that start with “0790”  Above code shows that a run time, it is running an SQL SELECT SQL Query from the Employee Table and uses our input of 0790 as a condition. Variable DataTable sdt stores all the Employee data that it returns from the SQL Query and then the dataTable stores this in an array and row by row, adds the employee to the list view. |
|  | 8k) | This is the same as 8c) however in this example, the user isn’t modifying their own details. Check 8c for more details on this.  However in this example (check code below), lblID.text is the ID of the current employee selected while Variable.UserID is the ID of the employee that is logged into the system. LblID.text returned 2 while Variables.UserID returned 1, meaning that the current employee that is logged on has the ID of 1 while the employee selected has an ID of 2, since their not equal to each other, then the logged in user isn’t modifying their own details and therefore the If condition is false and goes to the else which is highlighted, this shows the messagebox shown below. |
| 9 | 9a) |  |
|  | 9b) | When the User Control (ReceiptAndFinance) loads, it would do a SQL Query to show all the receipts, however it orders it by “Date” and making it in descending “DESC” order (check screenshot) and then calls upon ShowReceiptAndFinanceTable (cbASCDesc, cbOrderBy) which does the SQL Query and orders them.    In this example: cbOrderBy which is a drop down box has been set to Quantity and cbAscDesc also a drop down box is set to ASC (which means ascending). It shows all the receipts with ascending values of Quantity.    Arranged the order by Quantity Descending order: cbAscDesc is set to DESC so rearranges it into Descending order |
|  | 9c) | Looking for receipts where an customer has brought an Asus product, |
|  | 9d) | User must firstly select an Receipt (ReceiptFinanceListView.SelectedItems.Count == 1) to print , else a messagebox will appear. In this example, it returns 1 as shown in the image above so it would call upon PriceReceipt()    PrintReceipt() will show the Print Dialog as shown in the Image below( user can pick on which printer to prick , copies and properties, I will be printing mine as Adobe PDF-which saves the receipt as an Adobe PDF document).    And also call upon PrintDocument\_PrintPage  Which designs the receipt pdf document, how it looks, front and retrieves the customer details. Since I picked Nick Blood in the above image, it retrieves the details of the Receipt, which are the Customer, Employee, Product, Description, Quantity, Date of Purchase and Total price and then saves the design of the pdf document before saving it. Offset is to represent the vertical spacing between each line of words.  e.g. Offset+20 ensures that there is a vertical spacing of 20 pixels down the last line of words.  Printing as a Adobe Document. |
|  |  |  |
| 10 | 10a) | A Boolean Variable “AddCustomer” is used here to check if the Insert SQL Query worked, when the Employee has filled in all the correct details of the Customer, the program will then Insert it. In the screenshot below AddCustomer returns true, hence the SQL query has been executed and managed to insert the customer. AddCustomer returns true in this example, when AddCustomer = true, it would output a messagebox notifying the user that it has successfully added the customer. |
|  | 10b) | It is not possible to add a customer that has the same email with other customers, therefore Boolean CheckEmail checks that the Email entered does or doesn’t exist in the Customer Table, CheckEmail runs an SELECT SQL Query with that email entered ( txtEmail.Text.Trim()), if it returns true, then it already exists so a messagebox appears as shown below since two customers can’t have the same email.  Attempting to add a new customer with the same email as another customer.  MessageBox appears: |
|  | 10c) | Checks that fields required to add a customer aren’t empty, to check this code works, i entered blank values for the fields required and dused a breakpoint. A messagebox appears as I expected and when hovering over the code of the fields, it shows the “” value which I entered. If their null or white space “ “ or PhoneNumber isn’t numeric it would make the messagebox appear. |
| 11 | 11a) | Once btnMakeOrder is clicked, it checks that the inputs provided are valid, that are string and have correct format, in this run time. Product text : HP Pavillion and description were correct however Price.text was incorrect format as it wasn’t integer therefore shows this message box:  Else, if all the fields are correct.  From the provided Email and FullName of the customer, it would get their Customer ID, it would convert the integer of the Price.Text into a decimal and convert it into the correct currency.  Boolean InsertProduct is used to check that the INSERT SQL Query as run, it returns true if it has run while false if it has failed to Insert it.  In this example, it has returned true, so has succeeded in inserting the Product details into Finance, it then gets the FinanceID of the Product we just added and Insert it into the Receipt, along with Customer ID and EmployeeID. Linking all the three tables together so they can be referred with this Receipt.  Boolean SaveReceipt returns true in this example, since all the values I have provided are true, also proving that the CustomerID, FinanceID, EmployeeID are all correct.  If SaveReceipt returns true then Messagebox appears confirming that it has saved the receipt as shown below: else it would notify the other that It hasn’t saved. |
| 12 | 12a) |  |
|  |  | In the code shown below, I have entered a calendar event with the title “Meeting with the Jackson”, none recurring event, orange colour, 23 April 2016, 20:20 and duration of 0 hours.    When AddCalendar even tis clicked. It will create a new calendar event name (addnewdaywithtime).  It takes in the values of my calendar event details, the day e.g. 23, year 2016, title and such and adds the Calendar event CalendarView.AddEvent(addnewdaywithtime).    This screenshot shows the value of the addnewdaywithtime calendar event, which has all the details that we wanted to add on the event.  After the calendar event (addnewdaywithtime) has been added to the CalendarView, it would run a SQL INSERT query to insert the calendar event, as shown below, it returns true to the Boolean AddCalendarEvent, hence it has successfully added the calendar event during run time.    If AddCalendarEvent returns true which it has, then it would notify the user that it has added the calendar event with a messagebox as shown here:    To Delete Calendar Event, in this example, I am deleting the calendar event that I just previously added. Once DeleteCalendar Event is clicked, it would run a Delete SQL Query, Boolean DeleteCalendarEvent check if it has successfully deleted the calendar event and then refresh the calendar and calling upon ShowCalendarDate subrountine to show all the calendar events. |
|  | 12b) | ShowCalendarDate is called upon once the Calendar UC is loaded and when the calendar is needed to be refreshed. To show all the calendar events that is stored in the Database, CalendarDT would do a SELECT Query From the Calendar Table, this stores all the calendar events in the CalendarDT.  For every row that is stored in the DataTable Calendar DT, it will add them to the calendar. Currently the database Calendar has only one row which is the one that was just added. Firstly it will get the individual attributues, titles, dates e.g. as shown and store them to variables made. It would also separate the calendar date “23/04/2016” and time “20:20”, as they are required to be separate when adding calendar events to the Calendar.  After their stored, it will run a calendar event var AddCalendarDate, using the variables stores, it will add the calendar events and then CalendarView.AddEvent(AddCalendarDate) to add them to the Calendar so they can be visible. |

## Navigation Testing

The following table tests the unidirectional navigation between forms and UCs which link to each other. Blue cells indicate they are not linked. Black cells indicate there is no need for navigation.

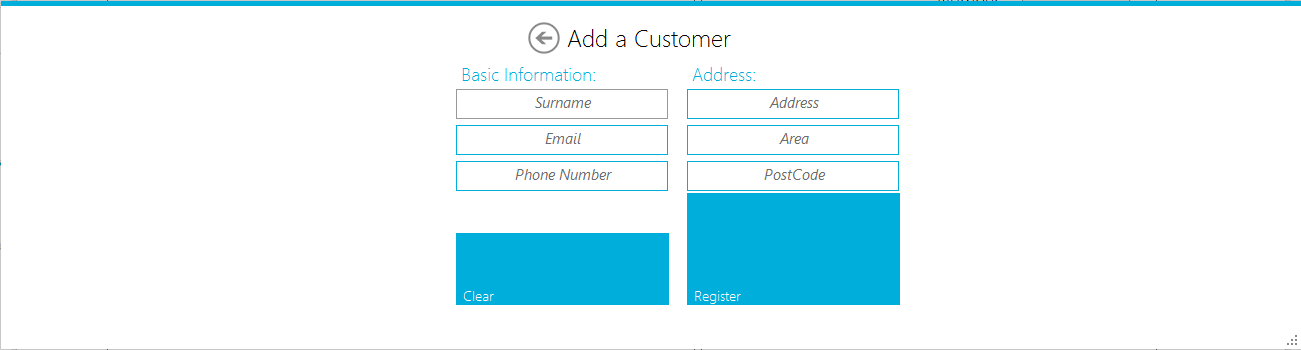
Navigation successful 

No direct navigation, but focus returned to correct form upon closing 

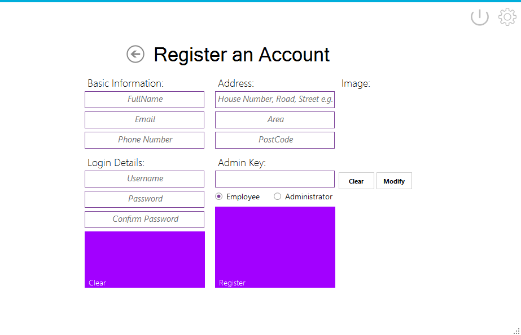
Navigation unsuccessful 

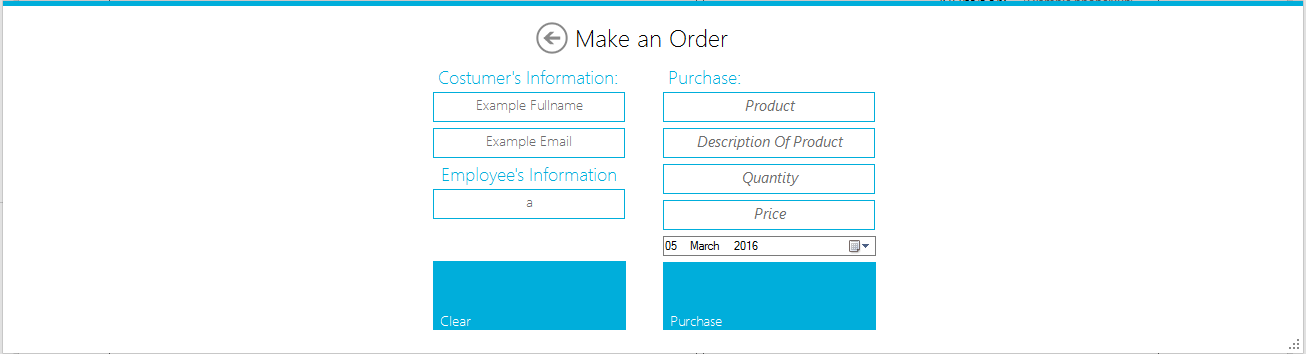
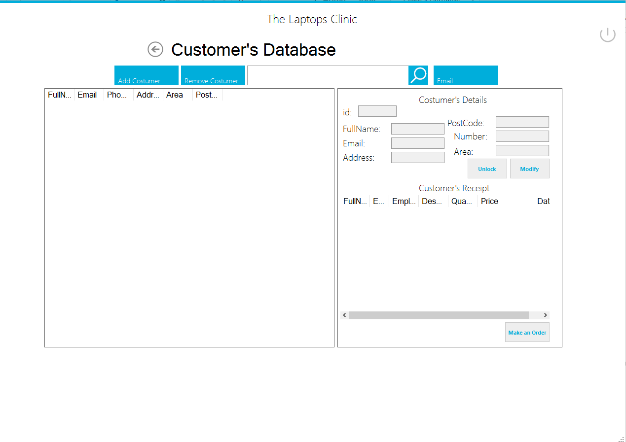
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Login | CreateSystems | Forgot | Register | Main | Customer | AddCustomer | MakeAnOrder | Email | Employee | Update Employee | Receipt and Finance | Calendar |
| Login |  |  |  |  |  |  |  |  |  |  |  |  |  |
| CreateSystems |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Forgot |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Register |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Main |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Customer |  |  |  |  |  |  |  |  |  |  |  |  |  |
| AddCustomer |  |  |  |  |  |  |  |  |  |  |  |  |  |
| MakeAnOrder |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Email |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Employee |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Update Employee |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Receipt and Finance |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Calendar |  |  |  |  |  |  |  |  |  |  |  |  |  |

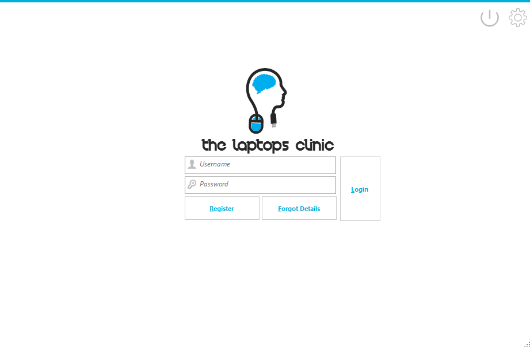
# Maintence

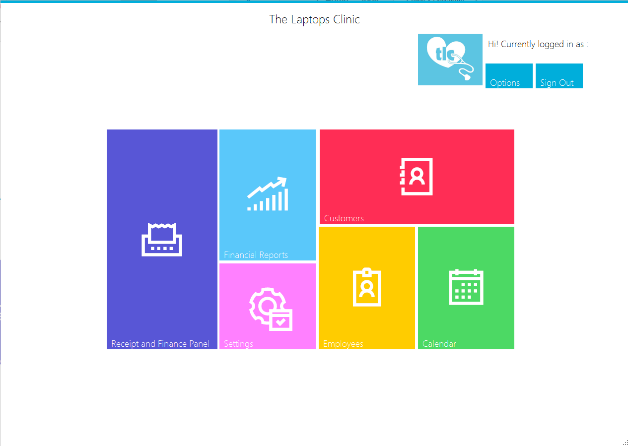
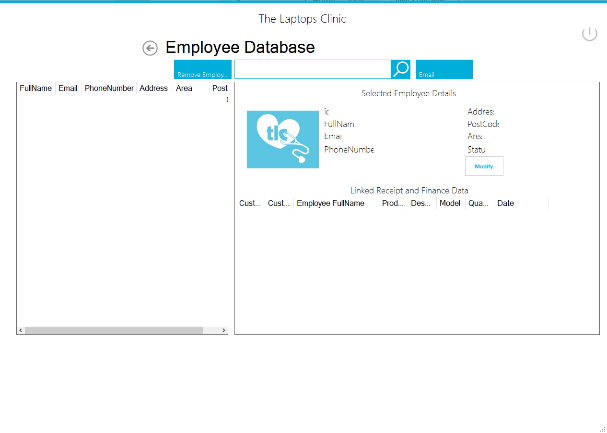
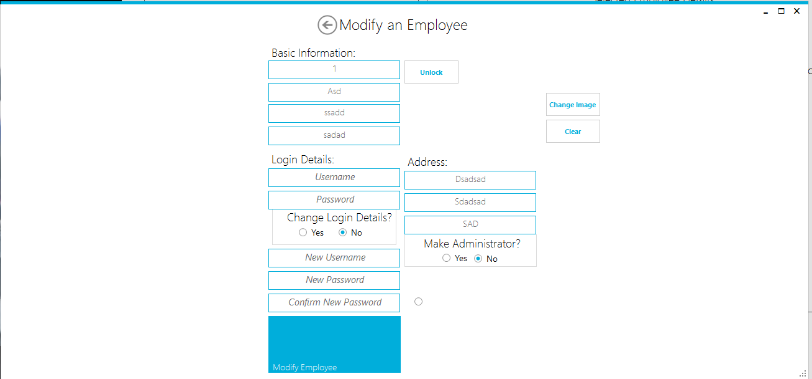


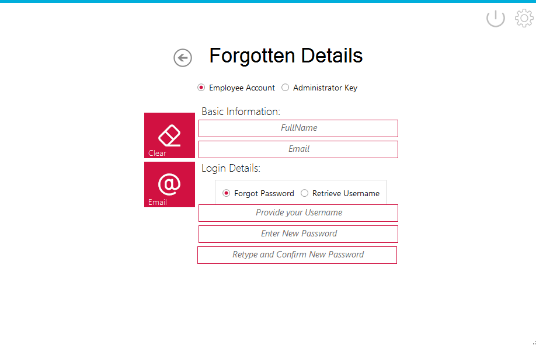
## Form Navigation Overview

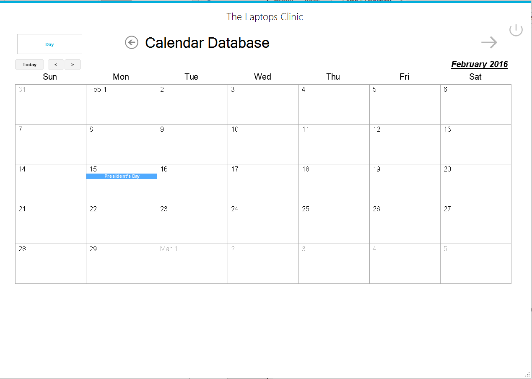
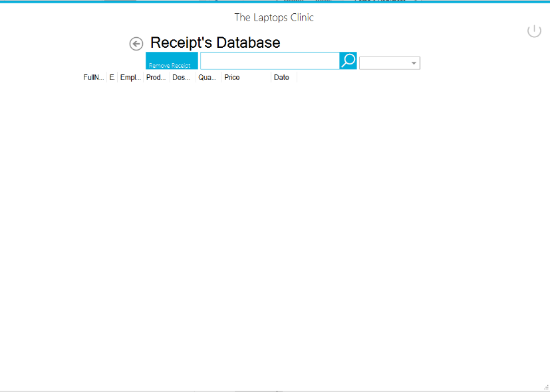
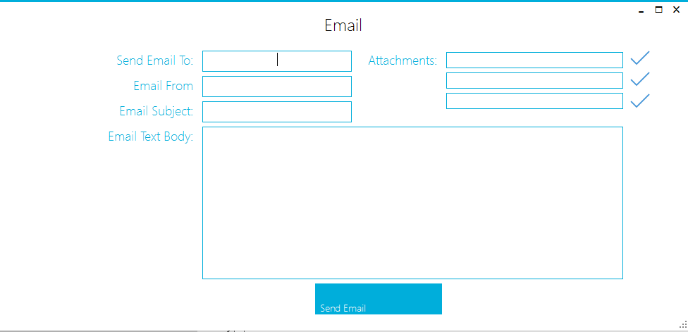
The diagram below illustrates how all the forms of the system link to each other. Compare with navigation design and testing on pages. \*Not actual completed design\*





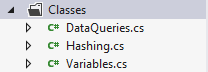






http://www.planetb.ca/syntax-highlight-word

## Class Overview

List of all my classes that I used:

1. Class **DataQueries**: This class is used to handle all SQLite queries, also defining the connection , opening connections, try and catch any errors and also processes the queries and returns the values wanted,.
   * It consist of ExecuteNonQuery to for INSERT, DELETE, Update queries, ExecuteNonQuery, receives the Query and returns a Boolean indicating if it successful e.g. True (if inserted the user, False if it failed to).
   * ExecuteReader, It receives the Query and the Column that is required and returns the value of the column that they seek and returns it in a string format, used by SELECT queries to get specific column values of the row.
   * ExecuteReaderToGetID , receives the Query and uses ExecuteScalar to get the ID of the row that is affected and returns it in string format.
   * ShowDataResults, receives the Query and is used by SELECT queries to select all the rows and column that are affected and returns a datatable, this datatable can be used to present the tables/results in the program. E.g. SELECT \* Customer -> returns a datatable of all the Customers in the system and program will use this datatable to present the results in table.
   * ExecuteReaderBolean is similar to ExecuteReader, it is used for SELECT queries, and however this only returns true/false not a string. It will be used to check the if a value exists in the presence. E.g. Checking if a Employee with the name “Luke” exists in the same when logging in. Returns True if it does, False if it does.   
     These can then be used to make messagebox to alert the user e.g. If it returns false therefore a messagebox can warn the user “There is no Employee in the Database with that Username, Try again”.
2. Class **Hashing**: This is used to encrypting the passwords and confirms the password entered as (Login Form)
   * ComputeHash, receives the password entered by the user, generates a random size of “salt” and adds it to the password. A salt consists characters and symbols. By adding a random amount of size to the password, it ensures that no password in the system will be identical. Having the same amount of size leads to hackers being able to find out the password as it is will have the same password. However, a random size of “salt” ensures no passwords are the same and hackers will not be able to crack the encryption or salt as it will take a polynomial time. ComputeHash, get the password, combines it with the salts, and encrypts about of them. Returns the encrypted salted password which is added to the Database
   * Confirm, it used to confirm the password entered with the encrypted password stored in the system, it does this by retrieving the password that is entered in the login form and also retrieves the encrypted password. It already knows what hash protocol to use and compares them to check if password entered (at login form) can be encrypted to, the one stored in the database. If true, then the password entered is correct or else it is false. Returns a Boolean indicating True if matches, false if it does not.
3. Class **Variables**: This is used to get and set the Global Variables in the system. Firstly defines them and sets them as null (“ “) and these variables will be accessible in the whole program and be can be set anytime. Examples of Global Variable are the Laptop’s Clinic’s email , EmailUsername, EmailPassword as it will be used to send messages. EmployeeID and FullName is also saved as it will be used to add the Employee’s details to the Order e.g. Like a signature, telling the admin what the employee has done.

### Class DataQueries:

using System;

using System.Collections.Generic;

using System.Data;//needed to return a data table (data table stores data in a array

using System.Data.SQLite;

// SQLite is a software library that implements a self-contained,

//serverless, zero-configuration, transactional SQL database engine.

//SQLite is the most widely deployed database engine in the world.

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TheLaptopsClinic.Classes

{

class DataQueries

{

public static Boolean ExecuteNonQuery(string a\_SQLString) //Takes in a string value called a\_SQLString. This will be the QUERY string that is passed to this function

{

// Create the sql connection

SQLiteConnection p\_Connection = new SQLiteConnection("Data source = TheLaptopClinicDatabase.sqlite;");

// Attempt to open the database

try

{

p\_Connection.Open();

}

catch

{ // Failed to open the Connection, returns false to indicate it failed since this function is an bolean which returns true or false.

// Debug.writeLine is used to alert the programmer of what the issue is.

System.Diagnostics.Debug.WriteLine("Failed to open connections");

return false;

}

// Attempt to execute the a\_SQlString command

SQLiteCommand p\_Command = new SQLiteCommand(a\_SQLString, p\_Connection); //Query and Connection is set.

try

{

p\_Command.ExecuteNonQuery(); //Command is executed

return true;

}

catch (SQLiteException p\_Exception)

{

//Failed to execute, so therefore returns false, alerts the programmer in debug. Closes and disposes any connections.

System.Diagnostics.Debug.WriteLine(p\_Exception);

p\_Connection.Close();

p\_Connection.Dispose();

p\_Connection = null;

return false;

}

// After the try catch is completed, closes and disposes the connection.

p\_Connection.Close();

p\_Connection.Dispose();

p\_Connection = null;

return true; //Returns true, due the fact, to get to this points indicates that it hasn't experienced any errors and successfully executed.

}

public static string ExecuteReader(string a\_SQLString, string SQLcolumn) //Takes in the Query String (a\_SQLString) and column that the SQL query will affect.

{

// Create the sql connection

SQLiteConnection p\_Connection = new SQLiteConnection("Data source = TheLaptopClinicDatabase.sqlite;");

// Attempt to open the database

try

{

p\_Connection.Open();

}

catch

{

// Failed to open the Connection, returns Null to indicate it failed since this function is an a string which returns text values.

// Null represents false.

// Debug.writeLine is used to alert the programmer of what the issue is.

System.Diagnostics.Debug.WriteLine("Failed to open connections");

return null;

}

// Attempt to execute the a\_SQlString command

SQLiteCommand p\_Command = new SQLiteCommand(a\_SQLString, p\_Connection); //Sets the query and connection.

try

{

SQLiteDataReader reader = p\_Command.ExecuteReader();

while (reader.Read())

{

string ReaderValue = (string)(reader["" + SQLcolumn + ""]); //Uses reader to get the column value of the SQLcolumn

return ReaderValue; // returns the value collected from the SQLcolumn

}

}

catch (SQLiteException p\_Exception)

{

//Failed to execute the command, alters the programmer in debug, closes and disposes any connection

System.Diagnostics.Debug.WriteLine(p\_Exception);

p\_Connection.Close();

p\_Connection.Dispose();

p\_Connection = null;

}

// close the connection

p\_Connection.Close();

p\_Connection.Dispose();

p\_Connection = null;

return null; //Returns false, due we can't return the value of the column, we can only return true (while Reader.Read()) which gets the specific value

}

public static string ExecuteReaderToGetUserID(string a\_SQLString)

{

// Create the sql connection

SQLiteConnection p\_Connection = new SQLiteConnection("Data source = TheLaptopClinicDatabase.sqlite;");

// Attempt to open the database

try

{

p\_Connection.Open();

}

catch

{

// Failed to open the Connection, returns Null to indicate it failed since this function is an a string which returns text values.

// Null represents false.

// Debug.writeLine is used to alert the programmer of what the issue is.

System.Diagnostics.Debug.WriteLine("Failed to open connections");

return null;

}

// Attempt to execute the a\_SQlString command

SQLiteCommand p\_Command = new SQLiteCommand(a\_SQLString, p\_Connection); //Sets the Query and Connection

try

{

return (string)p\_Command.ExecuteScalar().ToString(); //Executes the command, ExecuteScalar returns an integer value of affected (Mostly used to return the ID of the affected attribute)

//It is converted to string and returned.

}

catch (SQLiteException p\_Exception)

{

//Failed to execute the command, alerts the programmer in debug. Disposes and closes any connection

System.Diagnostics.Debug.WriteLine(p\_Exception);

p\_Connection.Close();

p\_Connection.Dispose();

p\_Connection = null;

}

// close the connection

p\_Connection.Close();

p\_Connection.Dispose();

p\_Connection = null;

//If it has reached the end of this function, therefore it hasn't caught any error or failed, so returns true (by returning the command execute value)

return p\_Command.ExecuteScalar().ToString();

}

public static DataTable ShowDataResults(string a\_SQLString) //using System.DataTable was used to be able to create a Function which returns DataTable instead of strings or boleans.

{

// Create the sql connection

SQLiteConnection p\_Connection = new SQLiteConnection("Data source = TheLaptopClinicDatabase.sqlite;");

//Declares the DataTable and DataSet variables. Used to create the data and also fill it in with values. Currently it is empty.

DataTable dt = new DataTable();

DataSet ds = new DataSet();

ds.Tables.Add(dt);

try

{

p\_Connection.Open();

}

catch

{

//Failed to connect, informs the programmer in debug, returns a new Datable which is empty, same thing as returning Null.

System.Diagnostics.Debug.WriteLine("Failed to open connections");

return new DataTable();

}

// Attempt to execute the a\_SQlString command

SQLiteCommand p\_Command = new SQLiteCommand(a\_SQLString, p\_Connection); //Sets the Query and Connection

try

{

SQLiteDataAdapter da = new SQLiteDataAdapter(p\_Command); //Executes the command, da retrieves all the values affected by the command.

da.Fill(dt); //da fills in the DataTable (dt) which we declared earlier, with the values affected

return dt; //returns dt which will be used to show the data in a Table or ListView in the programmer to the user.

}

catch (SQLiteException p\_Exception)

{

//Failed to execute the command, informs the user in debug of the exception, closes and disposes any connection

// Returns False ( new DataTable) which is the same thing as returning Null.

System.Diagnostics.Debug.WriteLine(p\_Exception);

p\_Connection.Close();

p\_Connection.Dispose();

p\_Connection = null;

return new DataTable();

}

// close the connection

p\_Connection.Close();

p\_Connection.Dispose();

p\_Connection = null;

return dt; //If it has reached the end of this function, therefore it hasn't caught any error or failed, so returns true by returning the dt.

}

public static Boolean ExecuteReaderBolean(string a\_SQLString) //Takes in the SQL Query

{

// Create the sql connection

SQLiteConnection p\_Connection = new SQLiteConnection("Data source = TheLaptopClinicDatabase.sqlite;");

// Attempt to open the database

try

{

p\_Connection.Open();

}

catch

{ // Failed to open the Connection, returns false to indicate it failed since this function is an a boleans which returns true or false.

// Debug.writeLine is used to alert the programmer of what the issue is.

System.Diagnostics.Debug.WriteLine("Failed to open connections");

return false;

}

// Attempt to execute the a\_SQlString command

SQLiteCommand p\_Command = new SQLiteCommand(a\_SQLString, p\_Connection);

try

{

SQLiteDataReader reader = p\_Command.ExecuteReader();

//Count checks if the value is true and exists.

//When count is equal to 1 then the value exists in the system, else when it's 0 then it doesn't exist.

int count = 0;

while (reader.Read())

{

count = 1; //While its reading the database and located the value (its true and exists) then count = 1 to indicate it has located it

}

if (count == 1)

{

return true; //If it has located it and exists, returns true to that it executes. Reader is used to read the database and get values, in this

//function, it used to check if a certain e.g. Customer, Employee or value exists.

}

else //Count must not be equal to 1 therefore is 0 (default value), meaning it hasn't located the value

{

return false; //Returns to indicate it doesn't exist

}

}

catch (SQLiteException p\_Exception)

{

//Failed to execute the command, closes and disposes any connection. Returns false and informs the programmer of the exception faced in the debug.

System.Diagnostics.Debug.WriteLine(p\_Exception);

p\_Connection.Close();

p\_Connection.Dispose();

p\_Connection = null;

return false;

}

// close the connection

p\_Connection.Close();

p\_Connection.Dispose();

p\_Connection = null;

return true; //No error is catched or faced so must be true.

}

public static string ExecuteReaderToGetUserImage(string a\_SQLString) //Takes in the SQL Query, this is just a specific function, with the function of returning image. It could have been done in ExecuteReader string function, with the SQLcolumn being the column this function affects only.

{

// Create the sql connection

SQLiteConnection p\_Connection = new SQLiteConnection("Data source = TheLaptopClinicDatabase.sqlite;");

// Attempt to open the database

try

{

p\_Connection.Open();

}

catch

{ // Failed to open the Connection, returns Null to indicate it failed since this function is an a string which returns text values.

// Null represents false.

// Debug.writeLine is used to alert the program of what the issue is.

System.Diagnostics.Debug.WriteLine("Failed to open connections");

return null;

}

// Attempt to execute the a\_SQlString command

SQLiteCommand p\_Command = new SQLiteCommand(a\_SQLString, p\_Connection);

try

{

SQLiteDataReader reader = p\_Command.ExecuteReader();

while (reader.Read()) //Located the row affected.

{

string imgg = (string)(reader["ImageDirectory"]); //Gets the specific column, to get the user image

return imgg; //Returns the value of ImageDirectory so that it can be used to show the user's image.

}

}

catch (SQLiteException p\_Exception)

{

//Failed to execute command, informs the programmer of the exception, closes and disposes any connection. Returns false

System.Diagnostics.Debug.WriteLine(p\_Exception);

p\_Connection.Close();

p\_Connection.Dispose();

p\_Connection = null;

return null;

}

// close the connection

p\_Connection.Close();

p\_Connection.Dispose();

p\_Connection = null;

return null; //Returns false, due imgg can only be accessed while Reader.Read

}

}

}

### Class Hashing:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security.Cryptography; // provides cryptographic services, including secure encoding and decoding of data, as well as many other operations, such as hashing, random number generation, and message authentication.

using System.Text;

using System.Threading.Tasks;

namespace TheLaptopsClinic.Classes

{

class Hashing

{

public static string ComputeHash(string plaintText, byte[] salt)

{

//The Mininum and Max lenght of the salt to be added to the system. It is random therefore, we won't have a specific amount of salt

int minSaltLength = 4, MaxSaltLength = 16;

byte[] SaltBytes = null;

if (salt != null)

{

//used to specify your own Salt and will just return the plainText + the Salt you defined. This used when the user login.

SaltBytes = salt; //When the user is login in, the system will compare the password entered(x) and the password stored(y),

//calculate the salt and confirm if its possible with the salt used and password used (x) if it can return (y).

//If true, then password entered(x) is true, else it incorrect.

}

else

{

/// Salt is equal to null

///Picks a random a length of the salt using within the size limit. Salt length will be from 4 to 16.

Random r = new Random();

int SaltLength = r.Next(minSaltLength, MaxSaltLength);

//Salt bytes will have the length of the Salt length.

SaltBytes = new byte[SaltLength];

//Used to implement a cryptograhic random number generator

RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();

//rng will fill SaltBytes with a random nonzero values

//SaltBytes go tthe length from SaltBytes = new byte[SaltLength].

// Now rng will fill the saltbytes with random values that aren't of value of 0 to the SaltBytes of the length it has.

rng.GetNonZeroBytes(SaltBytes);

rng.Dispose();/// Don't need it after using it so disposes.

}

byte[] plainData = ASCIIEncoding.UTF8.GetBytes(plaintText); //Fills plaindata with only ASCII characters that are available in a UTF-8 File. UTF-8 encode any text of any language that can be stored in 8 bit.

//ASCII can only represent a limited number of characters at one time. It isn't very useful to represent any language that isn't based on a Latin character set. However, UTF-8 which is an encoding standard for UCS-4 (Unicode) can represent almost any language. It does this by chaining multiple bytes together to represent one character (or glyph to be more correct).

byte[] plainDataAndSalt = new byte[plainData.Length + SaltBytes.Length]; ///Stored Plaindata and SaltBytes into one bit array in PlainDataAndSalt

//Gets all the values in plainData and puts them in PlainDatAndSalt.

for (int x = 0; x < plainData.Length; x++)

plainDataAndSalt[x] = plainData[x];

//Gets all the values in SaltBytes into PlainDatAndSalt

for (int n = 0; n < SaltBytes.Length; n++)

plainDataAndSalt[plainData.Length + n] = SaltBytes[n];

byte[] hashValue = null;

SHA256Managed sha = new SHA256Managed();

///The SHA (Secure Hash Algorithm) is one of a number of cryptographic hash functions. SHA-256 algorithm generates an almost-unique, fixed size 256-bit (32-byte) hash

////Computes the SHA256 hash for the input data

hashValue = sha.ComputeHash(plainDataAndSalt);

sha.Dispose();

byte[] result = new byte[hashValue.Length + +SaltBytes.Length];

for (int x = 0; x < hashValue.Length; x++)

result[x] = hashValue[x];

for (int n = 0; n < SaltBytes.Length; n++)

result[hashValue.Length + n] = SaltBytes[n];

return Convert.ToBase64String(result);

}

public static bool Confirm(string plaintText, string hashvalue)

{

//hashValue is the encrypted password value

//plainText is the inputted password to login.

byte[] hashBytes = Convert.FromBase64String(hashvalue); //Converts it into a byte array

///SHA256 uses 32 bits and adds the salt to the end of the encrypted text

int hashSize = 32;

//Able to get the salt out of the end of the encrypted password.

byte[] saltBytes = new byte[hashBytes.Length - hashSize];

for (int x = 0; x < saltBytes.Length; x++)

saltBytes[x] = hashBytes[hashSize + x];

///produces a new hash for the inputted password using the salt bytes of the database password

///compares that newhash mades is same to the one stored in the system.

string NewHash = ComputeHash(plaintText, saltBytes);

return (hashvalue == NewHash); //Returns true if the password entered (x) is equal to the password stored in the system(y).

}

}

}

### Variables Class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TheLaptopsClinic.Classes

{

class Variables

{

//All the variables are set to null ("") when the program starts and when the program ends

private static string CurrentEmployeeUserID = ""; /// Current User Logged in ID

public static string UserID

{

get { return CurrentEmployeeUserID; } //Returns the variable when is being called on (this is the same for all the other variables in the class)

set { CurrentEmployeeUserID = value; } //Sets the variable when it is being set to (this is the same for all the other variables in the class)

}

private static string EmployeeUsername = "";

public static string GetEmployeeUsername

{

get { return EmployeeUsername; } //Returns the variable when is being called on (this is the same for all the other variables in the class)

set { EmployeeUsername = value; } //Sets the variable when it is being set to (this is the same for all the other variables in the class)

}

private static string EmployeeFullName = ""; // G

public static string GetEmployeeFullName

{

get { return EmployeeFullName; }

set { EmployeeFullName = value; }

}

private static string SmtpClient = ""; //Client that is to be used to email

public static string GetSmtpClient

{

get { return SmtpClient; }

set { SmtpClient = value; }

}

private static string Port = ""; //Port to use to email

public static string GetPort

{

get { return Port; }

set { Port = value; }

}

private static string CompanyEmail = ""; //The Laptop's Clinic's Email

public static string GetEmail

{

get { return CompanyEmail; }

set { CompanyEmail = value; }

}

private static string CompanyEmailUsername = ""; //The Laptop's Clinic's Email Username

public static string GetEmailUsername

{

get { return CompanyEmailUsername; }

set { CompanyEmailUsername = value; }

}

private static string CompanyEmailPassword = ""; //The Laptop's Clinic Email Password

public static string GetEmailPassword

{

get { return CompanyEmailPassword; }

set { CompanyEmailPassword = value; }

}

}

}

## Form Overview

Program loads into Login Form, however there is no database in the system, so UserControl CreateSystems will swipe into view.

However to create the animation and allow for UserControls to slide into the form we need to create UC template that contains the sliding effect and other functions like adding the UC on your form and etc.

### TemplateUC

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using MetroFramework.Controls;

using Transitions;

namespace TheLaptopsClinic.UCs

{

public partial class TemplateUC : MetroUserControl //UC allow us to define properties and

//methods for the control. //MetroUserControl is used as it offers a slicker design instead of the orginal UserControl

{

//The following codes are just the declaration of variables and event for the UC slider template.

//By default I only added two event which is the `Closed` and `Shown`,

// This can be used to disable and enable the button the will trigger the display of the sliding UC.

Form \_owner = null;

bool \_loaded = false;

#region Events

public event EventHandler Closed;

public event EventHandler Shown;

protected virtual void closed(EventArgs e)

{

EventHandler handler = Closed;

if (handler != null) handler(this, e);

}

protected virtual void shown(EventArgs e)

{

EventHandler handler = Shown;

if (handler != null) handler(this, e);

}

#endregion

// Next line of code will be for the constructor of the sliding UC.

//We needed two constructor one with parameter and the other one is without parameter

//so that we will not have any issue when we inherit this custom control

public TemplateUC()

{

InitializeComponent();

}

public TemplateUC(Form owner)

: this()

{

this.Visible = false;

\_owner = owner;

owner.Controls.Add(this);

this.BringToFront();

owner.Resize += owner\_Resize;

ResizeForm();

}

void owner\_Resize(object sender, EventArgs e)

{//Calls to resize the form.

ResizeForm();

}

//This will determine the size of you UC

private void ResizeForm()

{

this.Width = \_owner.Width;

this.Height = \_owner.Height - 77; // -77 is to remove the header and footer

this.Location = new Point(\_loaded ? 0 : \_owner.Width, 50); // -50 is for top position. You can change it depending on your design

}

//This part of the code is the sliding effect.

//I called the function as swipe that can accept a boolean parameter,

//true for slide in and false for slide our.

public void swipe(bool show = true)

{

this.Visible = true;

//using Transitions allows use to create an animation of object moving.

//Transition consists of many transition animation but we will only use EaseinEaseOut which changes

//the x and y coordinate of a specific amount of time to make it seem like it moving.

Transition \_transition = new Transitions.Transition(new TransitionType\_EaseInEaseOut(500)); //EaseInEaseOut(500), the 500 is to represent that it should take to perform the transition.

//500 is 0.5 seconds it wil take to perform the transition

\_transition.add(this, "Left", show ? 0 : this.Width); //Transition allows Left or Top only, which is the direct of the movement. However negative left = moving right

//\_transition.add(takes the UCs name, direction, position it will move into) left = x coordinate, top = y coordinates

\_transition.run(); // Runs the animation which is does by changing the coordinates over time to make it seems its a animation.

while (this.Left != (show ? 0 : this.Width))

{

Application.DoEvents();

}

if (!show)

{

closed(new EventArgs());

\_owner.Resize -= owner\_Resize;

\_owner.Controls.Remove(this);

this.Dispose();

}

else

{

\_loaded = true;

ResizeForm();

shown(new EventArgs());

}

}

private void TemplateUC\_Load(object sender, EventArgs e)

{

}

private void lnlClose\_Click(object sender, EventArgs e)

{

Application.Exit();

}

}

}

The temple for sliding UC has been set, now for all the other UCs to be able to animate and resize e.g. it will require inheriting the template to it as a UC.

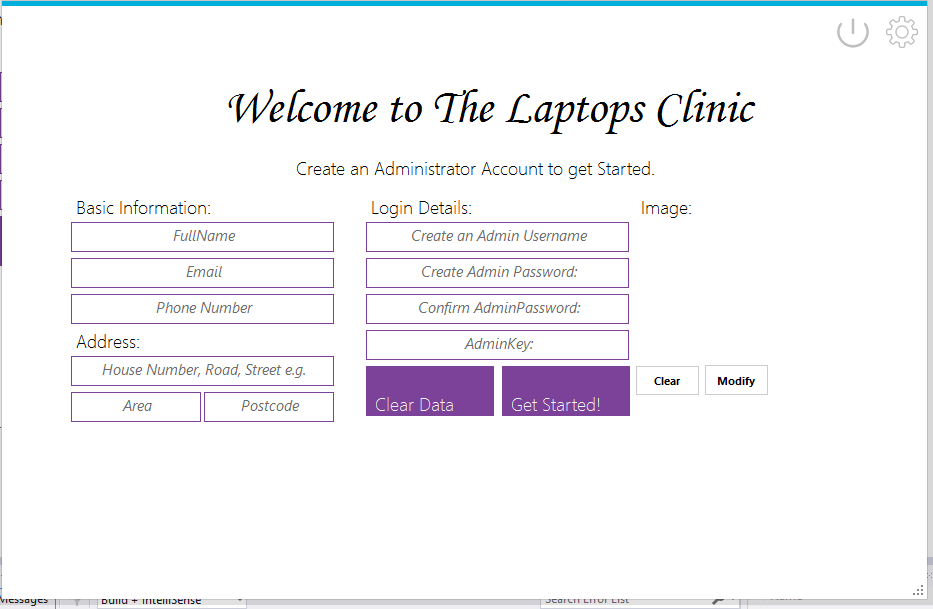
CreateSystem swipes into view if there is no database in the system so the first UC that will swipe and the first interface the user will see the in the CreateSystem UC.

### CreateSystem UC:

lnkSettings

txtConfirmAdminPassword

lnlClose



txtAdminKey

txtAdminPassword

btnGetStarted

btnClearImage

btnModifyImage

pbProfilePicture

txtAdminEmail

txtAdminPhoneNumber

txtAdminAddress

txtAdminPostcode

btnClearFields

txtAdminArea

txtAdminUsername

txtAdminFullName

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using MetroFramework;

using System.Data.SQLite;

using TheLaptopsClinic.Classes;

using System.Globalization;

namespace TheLaptopsClinic.UCs

{

public partial class CreateSystemUC : TemplateUC

{

public CreateSystemUC(Form owner) : base (owner)

{

InitializeComponent();

}

private void CreateSystem\_Load(object sender, EventArgs e)

{

}

private void btnGetStarted\_Click(object sender, EventArgs e)

{

//Users check to get started, create the system and table.

///First checks that all of the required fields aren't blank.

///

if (string.IsNullOrWhiteSpace(txtAdminKey.Text) || (string.IsNullOrWhiteSpace(txtAdminPassword.Text)) || (string.IsNullOrWhiteSpace(txtAdminUsername.Text)) || (string.IsNullOrWhiteSpace(txtConfirmAdminPassword.Text)) || (string.IsNullOrWhiteSpace(txtAdminFullName.Text)) || (string.IsNullOrWhiteSpace(txtAdminEmail.Text)))

{

/// If the required fields are blank. Makes a (Metro) message box to alert the user it's blank

MetroMessageBox.Show(this, "You're requried to fill in all the Fields to create the Account, incorrect format!, E.g. PhoneNumber must be Numeric", "Creation Database Failed", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{

//If none of the required fields are blank,

//Checks that the Password and Confirm Password are equal

if (txtAdminPassword.Text != txtConfirmAdminPassword.Text)

{

// If Password and Confirm Password aren't equal, Makes a message box to alter the user that the Passwords don't match

MetroMessageBox.Show(this, "The Admin Password and Retry Admin Password don't match", "Creation Database Failed", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{

Int64 CheckIfInteger;

if (!Int64.TryParse(txtAdminPhoneNumber.Text.Trim(), out CheckIfInteger))

{

MetroMessageBox.Show(this, "Phone Number entered isn't Numeric, Enter a Numeric Phone Number or enter 0 as Blank", "Creation Database Failed", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{

try

{

//Required Fields are filled. Password and Confirm Password are equal.

//Will Now Create The Database and Tables

//Creates The Database

SQLiteConnection.CreateFile("TheLaptopClinicDatabase.sqlite"); //uses "using.System.IO to create the file"

//Creates Admininistrator Table

DataQueries.ExecuteNonQuery("CREATE TABLE [Administrator] ([AdminID] INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,[EmployeeID] INTEGER UNIQUE NOT NULL,[AdminKey] TEXT NOT NULL)");

//Creates Employee Table

DataQueries.ExecuteNonQuery("CREATE TABLE [Employee] ([EmployeeID] INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,[eFullName] TEXT NULL,[eEmail] TEXT NULL,[ePhoneNumber] TEXT NULL,[eAddress] TEXT NULL,[eArea] TEXT NULL,[ePostCode] TEXT NULL,[Username] TEXT NULL,[Password] TEXT NULL,[ImageDirectory] TEXT NULL)");

//Creates Customer Table

DataQueries.ExecuteNonQuery("CREATE TABLE [Customer] ([CustomerID] INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,[cFullName] TEXT NOT NULL,[cEmail] TEXT NOT NULL,[cPhoneNumber] TEXT NOT NULL,[cAddress] TEXT NOT NULL,[cArea] TEXT NOT NULL,[cPostCode] TEXT NOT NULL)");

//Creates Receipt Table

DataQueries.ExecuteNonQuery("CREATE TABLE [Receipt] ([ReceiptID] INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,[FinanceID] INTEGER NOT NULL,[EmployeeID] INTEGER NOT NULL,[CustomerID] INTEGER NOT NULL)");

//Create Finance Table

DataQueries.ExecuteNonQuery("CREATE TABLE [Finance] ([FinanceID] INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,[Product] TEXT NOT NULL,[DescriptionOfProduct] TEXT NOT NULL,[Quantity] INTEGER NOT NULL,[Price] TEXT NOT NULL,[Date] TEXT NOT NULL)");

//Creates Calendar Table

DataQueries.ExecuteNonQuery("CREATE TABLE [Calendar] ([CalendarID] INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,[Title] TEXT NOT NULL,[Time] TEXt NULL,[Date] TEXT NOT NULL,[Frequency] TEXT NULL,[EventLength] INTEGER NULL,[Colour] TEXT NOT NULL)");

// The First User Registered is an Administrator as you require Administrator Access to Register Employees.

// Salts the Passwords and Hashes the salted Password. -----> this is explained in the Class Hashing.cs

//Creates The Salted Hash for the Password

string hashPassword = Hashing.ComputeHash(txtAdminPassword.Text, null);

//Inserts The Password along all the other User information.

DataQueries.ExecuteNonQuery(string.Format("INSERT INTO Employee (eFullName,eEmail,ePhoneNumber,eAddress,eArea,ePostCode,Username, Password, ImageDirectory) VALUES ('{0}','{1}','{2}','{3}','{4}','{5}','{6}','{7}','{8}')", //string.format as it contain “replacement fields” surrounded by curly braces {n}, n is the number of the field. e.g. in this query '{0}' will be replaced by txtAdminFull.tex as seen below. It also allows organized work, not messy and allows to manipulate strings.

CultureInfo.CurrentCulture.TextInfo.ToTitleCase(txtAdminFullName.Text.Trim()), ///Trim() removes the spaces at the end and Before the text. // CultureInfo......ToTitleCase changes the txt into Title Case. (First letter of every word is a captial)

CultureInfo.CurrentUICulture.TextInfo.ToLower(txtAdminEmail.Text.Trim()), // Culture.info.....ToLower -> makes all the text characters into lower case format.

txtAdminPhoneNumber.Text.Trim(),

CultureInfo.CurrentUICulture.TextInfo.ToTitleCase(txtAdminAddress.Text.Trim()),

CultureInfo.CurrentUICulture.TextInfo.ToTitleCase(txtAdminArea.Text.Trim()),

CultureInfo.CurrentUICulture.TextInfo.ToUpper(txtAdminPostcode.Text.Trim()), //Culuture.info....ToUpper -> Makes all the character to Upper Case ( CAPITAL LETTERS)

txtAdminUsername.Text.Trim(), //Username doesn't require a change in the text format (e.g. to TitleCase)

hashPassword,

pbProfilePicture.ImageLocation.ToString())); //Saves the Location of the Employee's Picture.

//Administrator has been added to the Employee Database.

// To make an Employee Administrator, they require their EmployeeID to be also in the Administrator Database.

//

// Get The EmployeeID of the Selected Employee

String GetID = DataQueries.ExecuteReaderToGetUserID(String.Format("SELECT \* FROM Employee WHERE Username = '{0}' AND Password = '{1}'", txtAdminUsername.Text.Trim(), hashPassword));

// Gets the EmployeeID that we recieved and Inserts it the Administrator Database.

DataQueries.ExecuteNonQuery(String.Format("INSERT INTO Administrator (EmployeeID, AdminKey) VALUES( '{0}', '{1}')", GetID, txtAdminKey.Text.Trim()));

//After Everything is Done.

// This UC which is used to Create the Database, Tables, Insert the First Administrator will "swipe" out of view of the user.

// UC only slides into the Form if the database doesn't Exist.

MetroMessageBox.Show(this, "Welcome To The Laptops Clinic", "Successfully Created The Database and Administrator", MessageBoxButtons.OK, MessageBoxIcon.Information);

swipe(false);

}

catch (Exception ex)

{

MetroMessageBox.Show(this, ex.ToString(), "Unable to Create The Database, Contact Developer", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

}

private void btnModifyImage\_Click(object sender, EventArgs e)

{

//Opens a File Explorer to Allow to User to surf the folders and select the Picture they wanted as their Employee Picture

OpenFileDialog dlg = new OpenFileDialog();

/// A Filter so the User can only see pictures with the specific Formats or all the Files.

dlg.Filter = "JPG Files(\*.jpg)|\*.jpg| PNG Files(\*.png)|\*.png|All Files(\*.\*)|\*.\*";

if (dlg.ShowDialog() == DialogResult.OK)

{

string picLocation = dlg.FileName.ToString();

/// picLocation is the location of the File

pbProfilePicture.ImageLocation = picLocation;

}

}

private void btnClearImage\_Click(object sender, EventArgs e)

{

txtAdminFullName.Clear();

txtAdminEmail.Clear();

txtAdminPhoneNumber.Clear();

txtAdminAddress.Clear();

txtAdminArea.Clear();

txtAdminPostcode.Clear();

txtAdminUsername.Clear();

txtAdminPassword.Clear();

txtConfirmAdminPassword.Clear();

txtAdminKey.Clear();

pbProfilePicture.Image = null;

pbProfilePicture.ImageLocation = "";

}

private void btnClearFields\_Click(object sender, EventArgs e)

{

txtAdminFullName.Clear();

txtAdminEmail.Clear();

txtAdminPhoneNumber.Clear();

txtAdminAddress.Clear();

txtAdminArea.Clear();

txtAdminPostcode.Clear();

txtAdminUsername.Clear();

txtAdminPassword.Clear();

txtConfirmAdminPassword.Clear();

txtAdminKey.Clear();

pbProfilePicture.Image = null;

pbProfilePicture.ImageLocation = "";

}

}

}

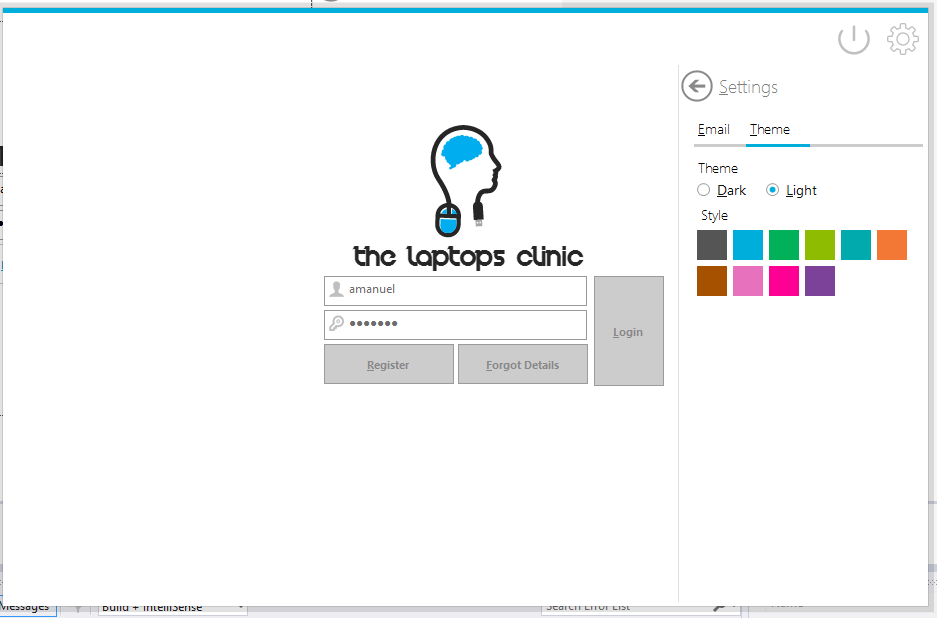
After the database , tables and the first Administrator is saved in the system. The CreateSystems will slide out of view of the Login Form. User can then either login, forgot password or register.

### Login Form

lnlClose

lnkBack  
A label with an image on it

lnkSettings



mrbLight

mtcSettings

mrbDark

LoginPictureBox

txtUsername

btnRegister

txtPassword

btnForgotDetails

btnLogIn

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using MetroFramework.Forms;

using MetroFramework.Controls;

using MetroFramework;

using System.IO;

using TheLaptopsClinic;

using TheLaptopsClinic.Classes;

namespace TheLaptopsClinic

{

public partial class LoginForm : MetroForm

{

public LoginForm()

{

InitializeComponent();

this.StyleManager = msmLogin;

///These are used to dynamically make the colour boxes which i stated in my design.

//They will be of 30,30 and represent all MetroColor Styles available.

for (int i = 3; i < 13; i++)

{

MetroTile \_tile = new MetroTile();

\_tile.Size = new Size(30, 30);

\_tile.Tag = i;

\_tile.Style = (MetroColorStyle)i;

\_tile.Click += \_tile\_Click;

flpSettings.Controls.Add(\_tile); //Will create and add the boxes to the FlowLayoutUC.

}

}

void \_tile\_Click(object sender, EventArgs e)

{

this.msmLogin.Style = (MetroColorStyle)((MetroTile)sender).Tag; //Once a metroTile with a specific colour

//is clicked it will set the component MetroStyleManger to the style picked.

}

private void LoginForm\_Load(object sender, EventArgs e)

{

string mypath = "TheLaptopClinicDatabase.sqlite"; //Paths of our database

if (File.Exists(mypath) == true)

{

//The File Exist, nothing need to be done here. Possible to have a messagebox however, everytime will be a nausence

}

else

{//If the file doesn't exist in the path, it will swipe CreateTheSystem into view so that the Database can be created along with the tables and inserting of administrator.

UCs.CreateSystemUC \_OpenCreateSystem = new UCs.CreateSystemUC(this);

\_OpenCreateSystem.swipe(true);

}

}

private void lnkSettings\_Click(object sender, EventArgs e)

{

pnlSettings.Visible = true;

pnlLoginForm.Enabled = false; //Disable the components of Login Form as it would still be able to click on the component sincce pnl(UC)setting isn't a pnlSlider, it doesn't cover the whole form and doesn't resize. it just appears in the slide as an option.

}

private void lnlClose\_Click(object sender, EventArgs e)

{

Application.Exit(); //Closes the whole program.

}

private void lnkBack\_Click(object sender, EventArgs e)

{

pnlLoginForm.Enabled = true;

pnlSettings.Visible = false; //Ables the components as the pnlSetting is not possible and the pnlCenter is enabled along with its components.

}

private void mrbDark\_CheckedChanged(object sender, EventArgs e)

{

if (mrbDark.Checked)

{

msmLogin.Theme = MetroThemeStyle.Dark; //Radiobutton Dark is checked so Form turns into a Dark MetroThemeStyle.

//Updates msmLogin with the theme of choice so it will pass to other UCs and forms.

}

}

private void mrbLight\_CheckedChanged(object sender, EventArgs e)

{

if (mrbLight.Checked)

{

msmLogin.Theme = MetroThemeStyle.Light;//Radiobutton Light is checked so Form turns into a Light MetroThemeStyle.

// Updates msmLogin with the theme of choice so it will pass to other UCs and forms.

}

}

private void btnLogIn\_Click(object sender, EventArgs e)

{

// An Employee of the The Laptops Clinic Seeks to Login

//Gets the encrypted Password for comparison with the one inputted.

//dt returns null, if the password can't be fine with the Username, therefore Username is incorrect or no registered employee with that Username

string encryptedPassword = DataQueries.ExecuteReader(string.Format("SELECT \* FROM Employee WHERE Username = '{0}'", txtUsername.Text.Trim()), "Password");

if (encryptedPassword == null)

{

//Didn't find the encrypted password for that Username

MetroMessageBox.Show(this, "There is no Employee with that Username, Try Again!", "Incorrect Login Details", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{

//Username found, Encrypted Password Recieved

//Confirms that the password inputted by user (txtPassword.Text) is equal to the encrypted password

Boolean match = Hashing.Confirm(txtPassword.Text, encryptedPassword);

if (match == true)

{

///Hashing.Confirm Confirms that the Passowrd is matched to the Hashed Version OF That Password From The Database.

///Executes a ExecuteReader and checks that the Encrypted Passwords and Username belong to the same Employee

Boolean EmployeeLogin = DataQueries.ExecuteReaderBolean(string.Format("SELECT \* FROM Employee WHERE Username = '{0}' AND Password = '{1}'", txtUsername.Text.Trim(), encryptedPassword));

if (EmployeeLogin == true)

{ //Username and Password both are of the same User.

//Stores the EmployeeID, Username and FullName into the system as global variables to be later used.

String GetEmployeeID = DataQueries.ExecuteReaderToGetUserID(String.Format("SELECT \* FROM Employee WHERE Username IS '{0}' AND Password IS '{1}'", txtUsername.Text, encryptedPassword));//Gets ID

string GetEmployeeFullName = DataQueries.ExecuteReader(string.Format("SELECT \* FROM Employee WHERE Username IS '{0}' AND Password IS '{1}'", txtUsername.Text, encryptedPassword), "eFullName");//Gets FullName

//Sets the Global Variables

Variables.GetEmployeeUsername = txtUsername.Text;

Variables.GetEmployeeFullName = GetEmployeeFullName;

Variables.UserID = GetEmployeeID;

//Login is Complete and the Main Form is brought into view while Login Form hides.

Forms.MainForm OpenMainForm = new Forms.MainForm();

//Sets the StyleManager and components to the Theme and Style Picked in the Login Form.

OpenMainForm.StyleManager = this.StyleManager;

OpenMainForm.msmMainForm.Theme = msmLogin.Theme;

OpenMainForm.msmMainForm.Style = msmLogin.Style;

this.Hide();//Hides this form

OpenMainForm.ShowDialog(); //Shows this form.

}

}

else

{

//Matching has failed, there the Password inputted (txtPassword) doesn't match the password stored in the database.

MetroMessageBox.Show(this, "The Password is Incorrect, Try again", "Incorrect Login Details", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

private void pnlLoginForm\_Paint(object sender, PaintEventArgs e)

{

}

private void btnRegister\_Click(object sender, EventArgs e)

{

//sets registerUC as new RegisterUC which will swipe in this Form.

//while it swipes it into view, it will set the metrostylemanager of the RegisterUC (msmregister) to the theme and style selected in this Login Form.

UCs.RegisterUC registerUC = new UCs.RegisterUC(this);

registerUC.msmRegister.Theme = this.Theme;

registerUC.msmRegister.Style = this.Style;

registerUC.swipe(true);

}

private void btnForgotDetails\_Click(object sender, EventArgs e)

{

//When Forgot Details button is clicked it will swipe (ForgotUC) into view of this Form, this is similar to the Register button,

// it will also pass the theme and style

UCs.ForgotUC forgotUC = new UCs.ForgotUC(this);

forgotUC.msmForgotUC.Theme = this.Theme;

forgotUC.msmForgotUC.Style = this.Style;

forgotUC.swipe(true);

}

private void btnEmailTest\_Click(object sender, EventArgs e)

{

//Checks if the Internet is available

bool bb = System.Net.NetworkInformation.NetworkInterface.GetIsNetworkAvailable();

if (bb == true)

{//Is Available and stores the Login Details for the Email

Classes.Variables.GetSmtpClient = txtSmtpClient.Text;

Classes.Variables.GetPort = txtPort.Text;

Classes.Variables.GetEmail = txtEmail.Text;

Classes.Variables.GetEmailUsername = txtEmailUsername.Text;

Classes.Variables.GetEmailPassword = txtEmailPassword.Text;

MetroMessageBox.Show(this, "Succesfully stored the Email details. Emailing is now possible", "Email Details Saved", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

//Not Available

MetroMessageBox.Show(this, "Internet Is Not Available", "Connection Failed", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

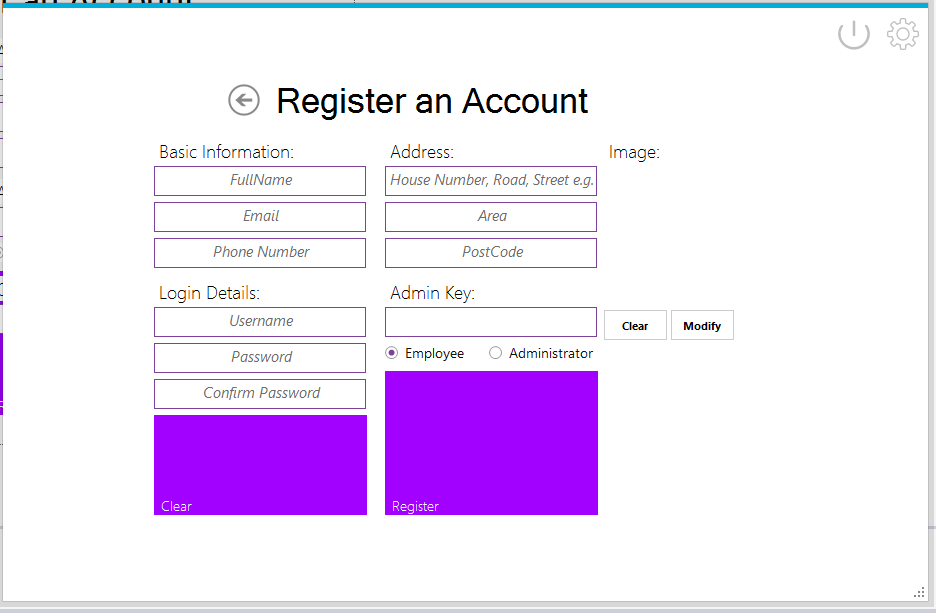
}

}

### Register UC

(which slides in front of Login Form)

txtFullName



btnClear

txtConfirmPassword

txtPassword

txtUsername

txtPhoneNumber

txtEmail

btnRegister

rbAdministrator

rbEmployee

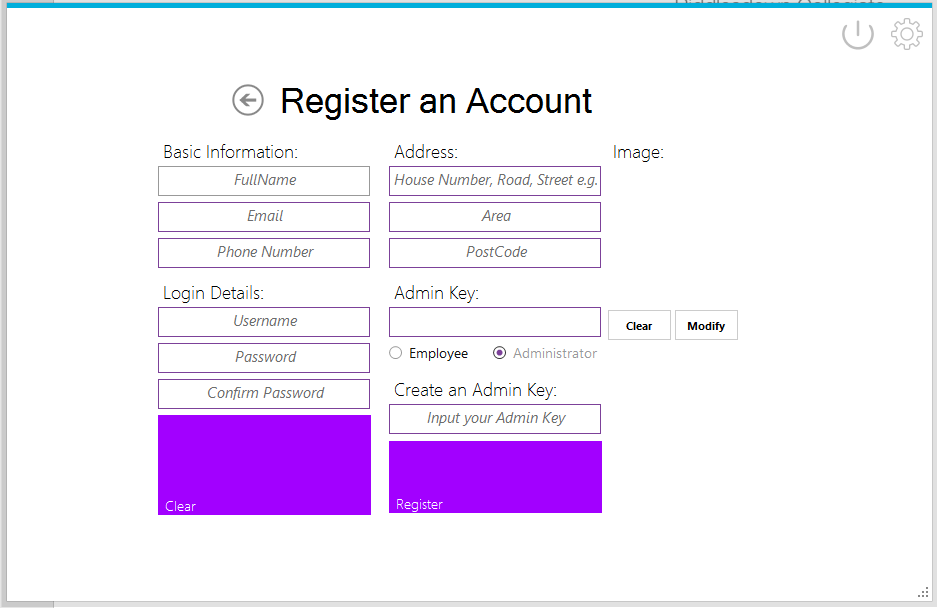
txtAdminKey

txtPostCode

txtArea

txtAddressOne

After Administrator radio button is checked.



txtCreateAdminKey

pbRegisterImage

btnClear

btnModify

pbRegisterImage

I can’t show the animation but the Register Button decreased in size and the components inside the button come into view

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using MetroFramework;

using Transitions;

using TheLaptopsClinic.Classes;

using System.Globalization;

namespace TheLaptopsClinic.UCs

{

public partial class RegisterUC : TemplateUC

{

public RegisterUC(Form owner) : base ( owner)

{

InitializeComponent();

}

private void RegisterUC\_Load(object sender, EventArgs e)

{

if (msmRegister.Theme == MetroThemeStyle.Dark)

lblRegisterTitle.ForeColor = InvertMeAColour(lblRegisterTitle.ForeColor);

}

Color InvertMeAColour(Color ColourToInvert) //This return funciton which i made is used to return the inverse of any colour given to it. I require this as When the Theme is Black, components that aren't part of the MetroFramework will not be seen.

//There InvertMeAColour returns the opposite colour, e.g. Recieves Black and returns white so the components will be visbible.

{

return Color.FromArgb((byte)~ColourToInvert.R, (byte)~ColourToInvert.G, (byte)~ColourToInvert.B);

}

private void btnRegister\_Click(object sender, EventArgs e)

{

//Checks that the required fields aren't Null or just white space ( )

if (string.IsNullOrWhiteSpace(txtFullName.Text) || (string.IsNullOrWhiteSpace(txtEmail.Text)

|| (string.IsNullOrWhiteSpace(txtUsername.Text) || (string.IsNullOrWhiteSpace(txtPassword.Text) ||

(string.IsNullOrWhiteSpace(txtConfirmPassword.Text))))))

{//Throws a message box if the required field are null or white space and alerts user.

MetroMessageBox.Show(this, "You're required to fill the all the Fields (Firstname, Surname, Email, Usernname, Password, Key) to create an account. ", "Registration Failed", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{

//Checks If There Is An Employee With The Same Username And Email.

//Employees can't have the same Username and Email as another Employee

Boolean CheckIfExists = DataQueries.ExecuteReaderBolean(string.Format("SELECT \* FROM Employee WHERE Username = '{0}' OR eEmail = '{1}'",

txtUsername.Text.Trim(),

txtEmail.Text.Trim()));

if (CheckIfExists == true)

{

//throws an Messagebox alerting them that (read below)

MetroMessageBox.Show(this, "An User With Those Exact Username or Email already exists" + Environment.NewLine + "Pick a Different Email or Username or Delete user", "Registration Failed", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{

//Check If Passwords Match And If Administration Key Is Correct

if (txtPassword.Text == txtConfirmPassword.Text)

{

Boolean IsAdminKeyCorrect = DataQueries.ExecuteReaderBolean(string.Format("SELECT \* FROM Administrator WHERE AdminKey is '{0}'", txtAdminKey.Text.Trim()));

if (IsAdminKeyCorrect == true)

{

//Passwords Match, Admin Key Is Correct. Can Now Register An Employee/Administrator Depending On Radio Check

string hashPassword = Hashing.ComputeHash(txtPassword.Text, null);

if (rbEmployee.Checked == true) // if the radio button for Employee is checked

{

//Inserts Employee into the system with the Hash Password along with their details + image

Boolean DoesitInsertEmployee = DataQueries.ExecuteNonQuery(string.Format("INSERT INTO Employee(eFullName, eEmail, ePhoneNumber, eAddress, eArea, ePostCode, Username, Password, ImageDirectory) VALUES('{0}', '{1}', '{2}', '{3}', '{4}', '{5}', '{6}', '{7}', '{8}')",

CultureInfo.CurrentCulture.TextInfo.ToTitleCase(txtFullName.Text.Trim()),

CultureInfo.CurrentUICulture.TextInfo.ToLower(txtEmail.Text.Trim()),

txtPhoneNumber.Text.Trim(),

CultureInfo.CurrentUICulture.TextInfo.ToTitleCase(txtAdress.Text.Trim()),

CultureInfo.CurrentUICulture.TextInfo.ToTitleCase(txtArea.Text.Trim()),

CultureInfo.CurrentUICulture.TextInfo.ToUpper(txtPostCode.Text.Trim()),

txtUsername.Text.Trim(),

hashPassword,

pbRegisterImage.ImageLocation.ToString()));

MetroMessageBox.Show(this, "Successfully added Employee to the Database!", "Successful Registration", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

///// For Administrators

else if (rbAdministrator.Checked == true)

{

///Checks If Admin Key already Exists! Administrators can't have same adminkey

Boolean DoesAdminKeyExist = DataQueries.ExecuteReaderBolean(string.Format("SELECT \* Administrator WHERE AdminKey is '{0}'", txtCreateAdminKey.Text));

MessageBox.Show(DoesAdminKeyExist.ToString());

if (DoesAdminKeyExist == true)

{//Another administrator as the same AdminKey

MetroMessageBox.Show(this, "Admin Key Already Exists" + Environment.NewLine + "Try Again! Choose a Different Admin Key", "Key Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{

if (txtPassword.Text == txtConfirmPassword.Text)

{

//No adminstrator as the same adminkey, so Inserts the Adminstrator into Employee Database as

//All administrator are considered to be Employee

//Only difference between Employee and Administrators are that Administrators are Employees with their EmployeeID linked to the Administrator Table/Database

//Inserts Employee

try

{

//Inserts Employee into the system with the Hash Password along with their details + image

Boolean DoesitInsertAdministrator = DataQueries.ExecuteNonQuery(string.Format("INSERT INTO Employee(eFullName, eEmail, ePhoneNumber, eAddress, eArea, ePostCode, Username, Password, ImageDirectory) VALUES('{0}', '{1}', '{2}', '{3}', '{4}', '{5}', '{6}', '{7}', '{8}')",

CultureInfo.CurrentCulture.TextInfo.ToTitleCase(txtFullName.Text.Trim()),

CultureInfo.CurrentUICulture.TextInfo.ToLower(txtEmail.Text.Trim()),

txtPhoneNumber.Text.Trim(),

CultureInfo.CurrentUICulture.TextInfo.ToTitleCase(txtAdress.Text.Trim()),

CultureInfo.CurrentUICulture.TextInfo.ToTitleCase(txtArea.Text.Trim()),

CultureInfo.CurrentUICulture.TextInfo.ToUpper(txtPostCode.Text.Trim()),

txtUsername.Text.Trim(),

hashPassword,

pbRegisterImage.ImageLocation.ToString()));

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

}

//Gets the EmployeeID so that it can be stored into the Table Administrator

String GetEmployeeID = DataQueries.ExecuteReaderToGetUserID(string.Format("SELECT \* FROM Employee WHERE Username = '{0}' AND Password = '{1}'", txtUsername.Text.Trim(), hashPassword));

//Insert ID into Administrator along with AdminKey, Employee is now considered as an Administrator

Boolean InsertEmployeeIntoAdmin = DataQueries.ExecuteNonQuery(string.Format("INSERT INTO Administrator(EmployeeID,AdminKey) VALUES ('{0}','{1}')", GetEmployeeID, txtCreateAdminKey.Text.Trim()));

MetroMessageBox.Show(this, "Successfully added the Administrator to the Database!", "Successful Registration", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

}

}

else

{

MetroMessageBox.Show(this, "Incorrect Admin Key, No Administrator Exists With That Admin Key " + Environment.NewLine + "Try Again!", "Administraor Key Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

}

private void btnClear\_Click(object sender, EventArgs e)

{

txtFullName.Clear();

txtEmail.Clear();

txtPhoneNumber.Clear();

txtUsername.Clear();

txtPassword.Clear();

txtConfirmPassword.Clear();

txtAdress.Clear();

txtArea.Clear();

txtPostCode.Clear();

txtAdminKey.Clear();

txtCreateAdminKey.Clear();

pbRegisterImage.Image = null;

pbRegisterImage.ImageLocation = "";

}

private void btnModify\_Click(object sender, EventArgs e)

{

//Opens a File Explorer to Allow to User to surf the folders and select the Picture they wanted as their Employee Picture

OpenFileDialog dlg = new OpenFileDialog();

/// A Filter so the User can only see pictures with the specific Formats or all the Files.

dlg.Filter = "JPG Files(\*.jpg)|\*.jpg| PNG Files(\*.png)|\*.png|All Files(\*.\*)|\*.\*";

if (dlg.ShowDialog() == DialogResult.OK)

{

string picLocation = dlg.FileName.ToString();

/// pbRegisterImage is the location of the File, sets the picture box to this location, image will be shown in Picture box of the selected image.

pbRegisterImage.ImageLocation = picLocation;

}

}

private void btnClearImage\_Click(object sender, EventArgs e)

{

pbRegisterImage.Image = null;

pbRegisterImage.ImageLocation = "";

}

private void rbAdministrator\_CheckedChanged(object sender, EventArgs e)

{

//Performs an transition animation in which components will move and make other components visible.

//THIS IS EXPLAIN IN!!!

if (rbAdministrator.Checked == true)

{

btnRegister.Height = 72;

Transition t1 = new Transition(new TransitionType\_Linear(1000));

t1.add(btnRegister, "Top", 388);

t1.run();

lblCreateKey.Visible = true;

txtCreateAdminKey.Visible = true;

}

else

{

lblCreateKey.Visible = false;

txtCreateAdminKey.Visible = false;

btnRegister.Height = 144;

Transition t1 = new Transition(new TransitionType\_Linear(1000));

t1.add(btnRegister, "Top", 316);

t1.run();

}

}

private void rbEmployee\_CheckedChanged(object sender, EventArgs e)

{

}

private void lnkBack\_Click(object sender, EventArgs e)

{

swipe(false);

}

}

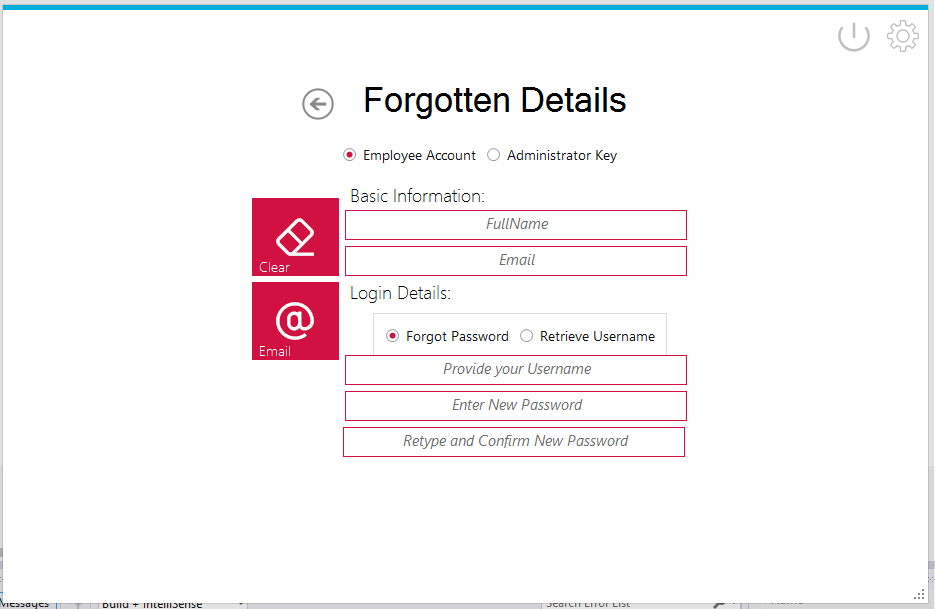
}

### ForgotUC

(slides in front of Login Form)

Employee Account and Forgot Password Check.

rbEmployee



btnClear

btnEmailForgottenDetails

txtConfirmNewPassword

rbAdministratorKey

rbForgotUsername

rbForgotPassword

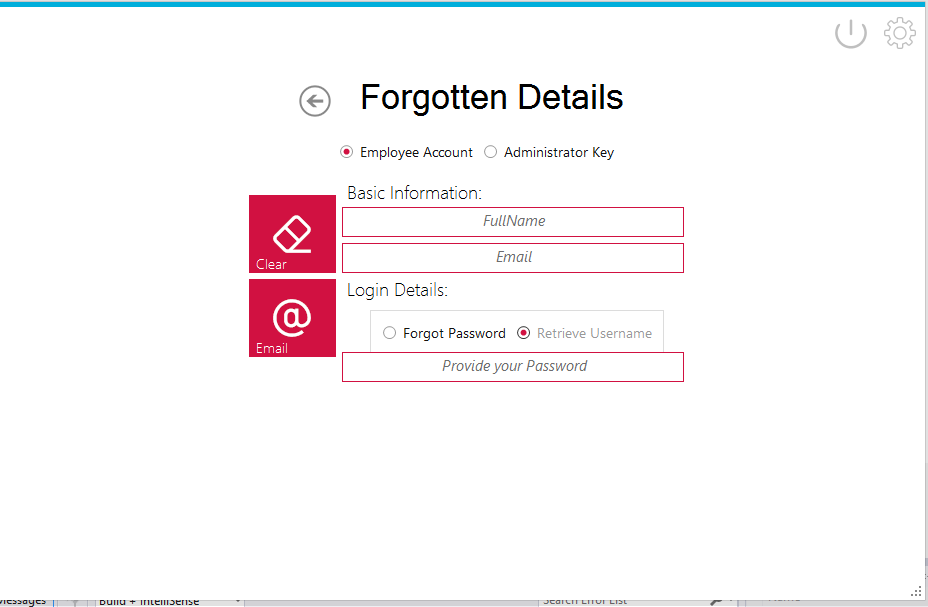
txtNewPassword

txtUsernameOrPassword

txtEmail

txtFullName

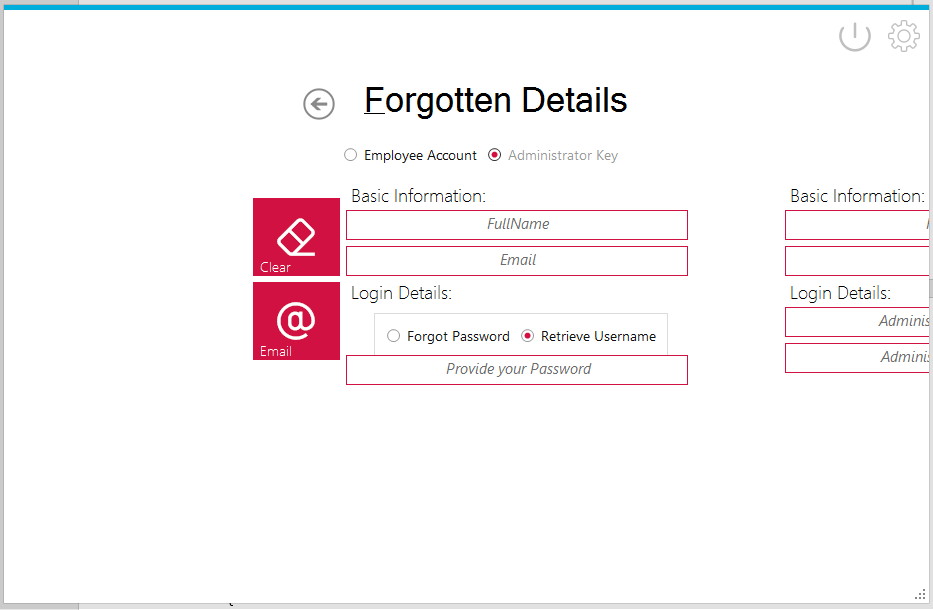
Employee Account and Retrieve Username Checked.

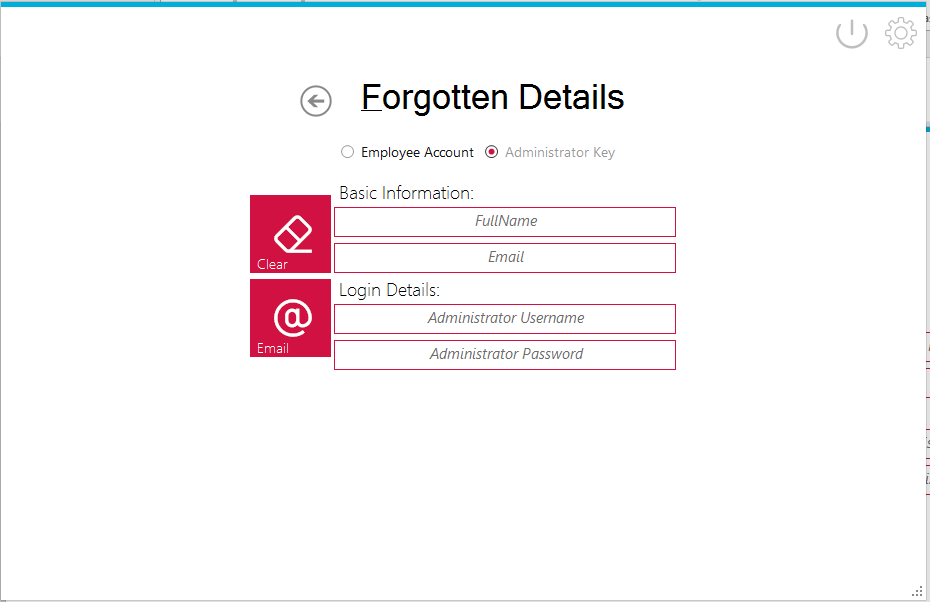


txtConfirmNewPassword

Administrator Key Checked.

Swiping into view





txtAdminKeyPassword

txtAdminKeyUsername

txtAdminKeyEmail

txtAdminKeyFullName

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Transitions;

using MetroFramework;

using System.Net.Mail;

using TheLaptopsClinic.Classes;

using System.Globalization;

namespace TheLaptopsClinic.UCs

{

public partial class ForgotUC : TemplateUC

{

public ForgotUC(Form owner) : base (owner)

{

InitializeComponent();

}

private void ForgotUC\_Load(object sender, EventArgs e)

{

if (msmForgotUC.Theme == MetroThemeStyle.Dark)

lblForgottenDetails.ForeColor = InvertMeAColour(lblForgottenDetails.ForeColor); ///Forecolour is set to InvertMeAColour(the colour of the forecolour) InvertMeAColour will invert the ForeColor and return the colour back.

rbForgotPassword.Checked = true;

rbEmployee.Checked = true; //This ensure that the program starts with these radio buttons checked.

}

Color InvertMeAColour(Color ColourToInvert) //Already Explained, Used to return the Inverted colour of the value inputtted.

{

return Color.FromArgb((byte)~ColourToInvert.R, (byte)~ColourToInvert.G, (byte)~ColourToInvert.B);

}

private void rbAdministratorKey\_CheckedChanged(object sender, EventArgs e)

{

//uses transition to move the components to get AdminKey in view.

//Explains in ....

//Transition is explained in pnlSlide, Transition\_Linear just moves diagonally or vertical.

Transition t1 = new Transition(new TransitionType\_Linear(1000));

t1.add(UC1, "Left", 316);

t1.run();

rbForgotUsername.Checked = true; //This is checked so that textboxes we don't need won't be visible and disabled

}

private void rbEmployee\_CheckedChanged(object sender, EventArgs e)

{

///Hides the components used to get AdminKey into view away.

///Moves it diagonally to the x position 1000 which is out of the view. 1000 in TransitionType\_Linear represent time as states in pnlSlide.cs

Transition t1 = new Transition(new TransitionType\_Linear(1000));

t1.add(UC1, "Left", 1000);

t1.run();

}

private void rbForgotPassword\_CheckedChanged(object sender, EventArgs e)

{

//WaterMark is the same thing as a Prompt Text. It informs the user what to input. Since we are using One TextBox for the User to provide Either their Username or password. WaterMark just informs them depending on what RadioButton they checked.

txtUsernameOrPassword.WaterMark = "Provide your Username";

txtNewPassword.Enabled = true; ///Enables the components to allow the user to input their new password , confirm it also, along with making them visible and enabled.

txtNewPassword.Visible = true;

txtConfirmNewPassword.Enabled = true;

txtConfirmNewPassword.Visible = true;

}

private void rbForgotUsername\_CheckedChanged(object sender, EventArgs e)

{

txtUsernameOrPassword.WaterMark = "Provide your Password"; //WaterMark is the same thing as a Prompt Text. It informs the user what to input. Since we are using One TextBox for the User to provide Either their Username or password. WaterMark just informs them depending on what RadioButton they checked.

txtNewPassword.Enabled = false;

txtNewPassword.Visible = false;

txtConfirmNewPassword.Enabled = false;

txtConfirmNewPassword.Visible = false;

}

private void btnClear\_Click(object sender, EventArgs e)

{

txtFullName.Clear();

txtEmail.Clear();

txtUsernameOrPassword.Clear();

txtNewPassword.Clear();

txtConfirmNewPassword.Clear();

}

private void btnForgot\_Click(object sender, EventArgs e)

{

if (rbEmployee.Checked == true) //Forgotten Details for Employee

#region

{

if (rbForgotPassword.Checked == true) //Employee wants to get their password

{

//Employee who Forgot their password will have a new password made for them. rbForgot.checked will make textboxes ( txtNewPassword) and (txtConfirmNewPassword) to be enabled so user can input their new details.

//To Make a new password, first need to find out what Employee Username we are dealing with and confirm their details is accurate

//Check the details provided is correct, so that it can locate an Employee with those details

Boolean CheckForDetails = DataQueries.ExecuteReaderBolean(string.Format("SELECT \* FROM Employee WHERE eFullName = '{0}' AND eEmail = '{1}'",

CultureInfo.CurrentCulture.TextInfo.ToTitleCase(txtFullName.Text.Trim()), txtEmail.Text.Trim()));

if (CheckForDetails == true) //Employee with those FullName and Email exists

{

//Checks that the Employee Username provided is correct

Boolean CheckForUsername = DataQueries.ExecuteReaderBolean(string.Format("SELECT \* FROM Employee WHERE Username = '{0}'", txtUsernameOrPassword.Text.Trim()));

if (CheckForUsername == false) //Incorrect Username

{

MetroMessageBox.Show(this, "There is no Employee in the Database with that Username", "Forgotten Details Failed ", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{//Located Username and exists in the Database

if (txtConfirmNewPassword.Text == txtNewPassword.Text && !(string.IsNullOrWhiteSpace(txtConfirmNewPassword.Text))) //New Password and Confirm Password Match

{

///Checks the Details user entered is correct!

//Encrypts The New Password.

string hashNewPassword = Hashing.ComputeHash(txtNewPassword.Text, null);

//Updates the Employee Data ( Password)

Boolean UpdateEmployee = DataQueries.ExecuteNonQuery(string.Format("UPDATE Employee SET Password = '{0}' WHERE eFullName = '{1}' AND eEmail = '{2}' AND Username = '{3}'", //Updates Password if the conditions are true

hashNewPassword,

CultureInfo.CurrentCulture.TextInfo.ToTitleCase(txtFullName.Text.Trim()),

txtEmail.Text.Trim(),

txtUsernameOrPassword.Text.Trim()));

if (UpdateEmployee == true) //Succeed in Updating the Details

{

MetroMessageBox.Show(this, "Successfully Update The Employeee Account, The New Password has been saved!", "Forgotten Details Successful", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{//Only reason, that UpdateEmployee is incorrect, is if by a chance ExecuteNonQuery returned false, since all the if/else conditions has all checked the values entered is correct.

//For example, ExecuteNonQuery could return false cause of the timeout it has.

MetroMessageBox.Show(this, "Failed To Update The Employeee Account,Try Again", "Forgotten Details Failed", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

else

{

//If the New Password and Confirm new Password Don't Match

MetroMessageBox.Show(this, "New Password And Confirm Password Doesn't Match", "Forgotten Details Failed", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

else //CheckDetails is incorrect

{

MetroMessageBox.Show(this, "User details provided is incorrect. An Employee with that FullName and Email Doesn't Exist!", "Forgotten Details Failed", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

else

// rbForgotUsername must be checked. User seeks to Retrieve their Username.

{

//Retrieve the Username by using the Employee's Password

//Firstly Employee's Provided Password needs to be hashed first and confirmed to the one stored in the Database.

//Recieves the stored Hashed Password, using the FullName and Email provided ( username can't be used as Employee has forgotten it)

Boolean CheckForDetails = DataQueries.ExecuteReaderBolean(string.Format("SELECT \* FROM Employee WHERE eFullName = '{0}' AND eEmail = '{1}'",

CultureInfo.CurrentCulture.TextInfo.ToTitleCase(txtFullName.Text.Trim()), txtEmail.Text.Trim()));

if (CheckForDetails == false)

{

MetroMessageBox.Show(this, "User details provided is incorrect. An Employee with that FullName and Email Doesn't Exist!", "Forgotten Details Failed", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{

string storedHashedPassword = DataQueries.ExecuteReader(string.Format("SELECT \* FROM Employee WHERE eFullName = '{0}' AND eEmail = '{1}'",

CultureInfo.CurrentCulture.TextInfo.ToTitleCase(txtFullName.Text.Trim()),

txtEmail.Text.Trim()), "Password");

MessageBox.Show(storedHashedPassword);

//Recieves the Hashed Stored Password

//Confirm the provided password by the Employee to the Hashed Stored Password.

Boolean PasswordMatch = Hashing.Confirm(txtUsernameOrPassword.Text, storedHashedPassword);

if (PasswordMatch == true)

{

//Provided Password matches to the stored database password

string RetrieveUsername = DataQueries.ExecuteReader(string.Format("SELECT \* FROM Employee WHERE Password = '{0}'", storedHashedPassword), "Username");

//Emails the Employee that his being dealt with, thats their User Details have been modified and provides them their new Username. Not Password.

//

// Before Emailing the Employee. Checks For Internet Connection

MessageBox.Show(RetrieveUsername);

bool bb = System.Net.NetworkInformation.NetworkInterface.GetIsNetworkAvailable();

if (bb == true)

{

//Internet Access is available

string RetrievedUserEmail = DataQueries.ExecuteReader(string.Format("SELECT \* FROM Employee WHERE Password = {0}'", storedHashedPassword), "eEmail");

//Creates a message to alert the Employee that their login Details have been modified also including the new email

//The Global variables in which i stored Email details for the Company is referenced here.

if (string.IsNullOrWhiteSpace(Variables.GetEmail) || string.IsNullOrWhiteSpace(Variables.GetEmailPassword) || string.IsNullOrWhiteSpace(Variables.GetEmailUsername) || string.IsNullOrWhiteSpace(Variables.GetEmployeeFullName) ||

string.IsNullOrWhiteSpace(Variables.GetEmployeeUsername) || string.IsNullOrWhiteSpace(Variables.GetPort) || string.IsNullOrWhiteSpace(Variables.GetSmtpClient))

{

MetroMessageBox.Show(this, "Unable to send the message, make sure you have provided the details for the Email at the Login Form", "Forgotten Details Failed", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{

MessageBox.Show(RetrievedUserEmail);

MessageBox.Show(Variables.GetEmail);

MessageBox.Show(RetrievedUserEmail);

MessageBox.Show(RetrieveUsername);

MailMessage mail = new MailMessage(Variables.GetEmail, RetrievedUserEmail, "The Laptops Clinic, Your Login Details have been modified", "Your Login Details have been modified" + Environment.NewLine + "Your Username now is " + RetrieveUsername);

SmtpClient client = new SmtpClient(Variables.GetSmtpClient);

client.Port = int.Parse(Variables.GetPort);

client.Credentials = new System.Net.NetworkCredential(Variables.GetEmailUsername, Variables.GetEmailPassword);

client.EnableSsl = true;

client.Send(mail);

MetroMessageBox.Show(this, "An Email has been sent provided with your Username", "Forgotten Details Successful", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

else

{

MetroMessageBox.Show(this, "No Internet Connection, Try Again with an Internet Connection", "Forgotten Details Failed", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

else

{

MetroMessageBox.Show(this, "The password inputted doesn't match the password stored for that User. Try Again", "Forgotten Details Failed", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

#endregion

if (rbAdministratorKey.Checked == true)

{

// User Seeks to Retrieve Their AdminKey

//

// Check if this user is an Administrator, First get EmployeeID and check if it exists in the Administrator Database

string GetEmployeeID = DataQueries.ExecuteReaderToGetUserID(string.Format("SELECT \* FROM Employee WHERE Username = '{0}' AND Password = '{1}'", txtAdminKeyUsername.Text.Trim(),

txtAdminKeyPassword.Text)); ///Got the EmployeeID stored in the string "GetEmployeeID"

MessageBox.Show(GetEmployeeID);

Boolean IsAdminOrNot = DataQueries.ExecuteNonQuery(string.Format("SELECT \* FROM Administrator WHERE Username = '{0}'", GetEmployeeID));

if (IsAdminOrNot == true)//The User is an Administrator so True

{

//Gets the AdminKey and Emails the User

string GetAdminKey = DataQueries.ExecuteReader(string.Format("SELECT \* FROM Employee WHERE Username = '{0}' AND Password = '{1}'", txtAdminKeyUsername.Text.Trim()), "AdminKey");

bool bb = System.Net.NetworkInformation.NetworkInterface.GetIsNetworkAvailable();

if (bb == true)

{

if (string.IsNullOrWhiteSpace(Variables.GetEmail) || string.IsNullOrWhiteSpace(Variables.GetEmailPassword) || string.IsNullOrWhiteSpace(Variables.GetEmailUsername) || string.IsNullOrWhiteSpace(Variables.GetEmployeeFullName) ||

string.IsNullOrWhiteSpace(Variables.GetEmployeeUsername) || string.IsNullOrWhiteSpace(Variables.GetPort) || string.IsNullOrWhiteSpace(Variables.GetSmtpClient))

{

MetroMessageBox.Show(this, "Unable to send the message, make sure you have provided the details for the Email at the Login Form", "Forgotten Details Failed", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{

//Internet Access is available

string RetrievedUserEmail = DataQueries.ExecuteReader(string.Format("SELECT \* FROM Employee WHERE EmployeeID = {0}'", GetEmployeeID), "EmployeeID");

//Creates a message to alert the Employee that their login Details have been modified also including the new email

//The Global variables in which I stored Email details for the Company is referenced here.

MailMessage mail = new MailMessage(Variables.GetEmail, RetrievedUserEmail, "The Laptops Clinic, Your Login Details have been modified", "Your Login Details have been modified" + Environment.NewLine + "Your AdminKey now is " + GetAdminKey);

SmtpClient client = new SmtpClient(Variables.GetSmtpClient);

client.Port = int.Parse(Variables.GetPort);

client.Credentials = new System.Net.NetworkCredential(Variables.GetEmailUsername, Variables.GetEmailPassword);

client.EnableSsl = true;

client.Send(mail);

MetroMessageBox.Show(this, "An Email has been sent provided with your AdminKey", "Forgot Details Sucessful", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

else

{

MetroMessageBox.Show(this, "Message can’t be sent, make sure internet access is available", " Forgotten Details Failed", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

private void lnkBack\_Click(object sender, EventArgs e)

{

swipe(false);

}

}

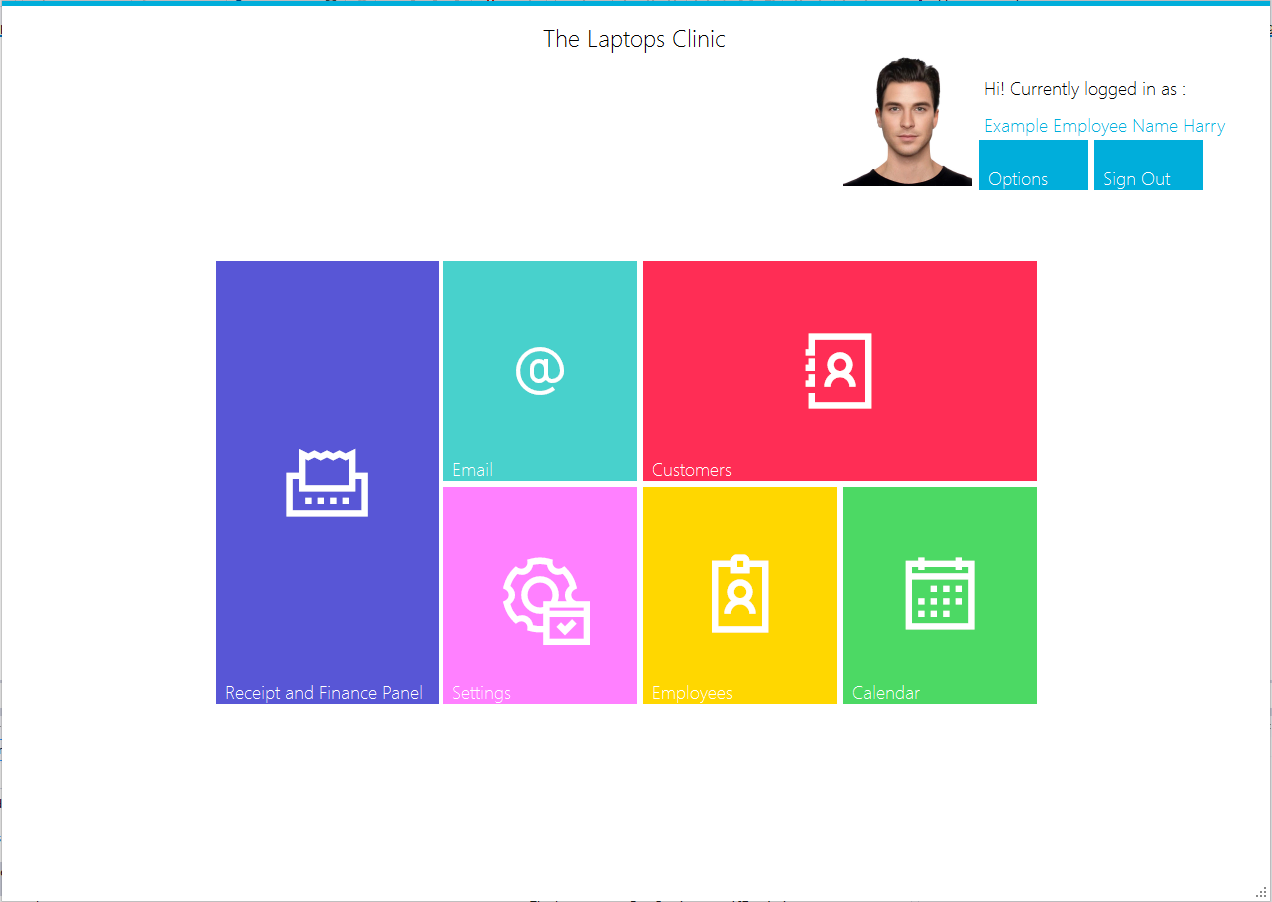
}

After the User has finished with Register UC or Forgot UC and decides to Login in using Login Form.   
Main Form will appear, it used to direct to all the other UCs in the form, so technically it’s just a graphic interface menu to guide to different UCs.

### MainForm Form

pbProfilePicture

btnOptions



btnEmail

btnReceiptAndFinance

btnSignOut

btnCustomers

btnCalendar

btnEmployee

btnSetting

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using MetroFramework.Forms;

using System.IO;

namespace TheLaptopsClinic.Forms

{

public partial class MainForm : MetroForm

{

public MainForm()

{

InitializeComponent();

}

private void MainForm\_Load(object sender, EventArgs e)

{

GetUserImage();

lblCurrentEmploy.Text = Classes.Variables.GetEmployeeFullName; //Gets the User's FullName that is saved in Global Variables

if (msmMainForm.Theme == MetroFramework.MetroThemeStyle.Dark)

lblCurrentEmploy.ForeColor = InvertMeAColour(lblCurrentEmploy.ForeColor);

}

Color InvertMeAColour(Color ColourToInvert)

{

return Color.FromArgb((byte)~ColourToInvert.R, (byte)~ColourToInvert.G, (byte)~ColourToInvert.B);

}

private void btnSignOut\_Click(object sender, EventArgs e)

{

this.Hide(); //Hides this form, if user logs out and shows the Login Form.

LoginForm showmainform = new LoginForm();

showmainform.ShowDialog();

}

private void btnReceiptAndFinance\_Click(object sender, EventArgs e)

{

UCs.ReceiptAndFinanceUC receiptandfinance = new UCs.ReceiptAndFinanceUC(this);

receiptandfinance.msmReceiptAndFinanceUC.Style = this.msmMainForm.Style;

receiptandfinance.msmReceiptAndFinanceUC.Theme = this.msmMainForm.Theme;

receiptandfinance.swipe(true);

}

private void btnCustomers\_Click(object sender, EventArgs e)

{

UCs.CustomerUC customerUC = new UCs.CustomerUC(this); //Creates a new instance of the UC, set the Owner of the UC to (this) Form.

//Passes down the Style and Theme to the MetroStyleManger in that UC

customerUC.msmCustomerUC.Style = this.msmMainForm.Style;

customerUC.msmCustomerUC.Theme = this.msmMainForm.Theme;

MessageBox.Show(msmMainForm.Theme.ToString());

MessageBox.Show(customerUC.msmCustomerUC.Theme.ToString());

customerUC.swipe();

}

private void btnEmployee\_Click(object sender, EventArgs e)

{

//Creates a new instance of the UC, sets the Owner of the UC to (this) Form.

//Passes down the Style and Theme to the MetroStyleManger in that UC

UCs.EmployeeUC employeeUC = new UCs.EmployeeUC(this);

employeeUC.msmEmployeeUC.Theme = this.msmMainForm.Theme;

employeeUC.msmEmployeeUC.Style = this.msmMainForm.Style;

employeeUC.swipe(true);

}

private void btnCalendar\_Click(object sender, EventArgs e)

{

//Creates a new instance of the UC, sets the Owner of the UC to (this) Form.

//Passes down the Style and Theme to the MetroStyleManger in that UC

UCs.CalendarUC calendar = new UCs.CalendarUC(this);

calendar.msmCalendarUC.Style = this.msmMainForm.Style;

calendar.msmCalendarUC.Theme = this.msmMainForm.Theme;

calendar.swipe(true);

}

void GetUserImage() //Once the MainForm loaded, it will call upon the Method GetUserImage

{

//Uses the Global Variable UserID to get the ID of the current logged in User, uses their ID to get the imageDirectory

string imagelocation = Classes.DataQueries.ExecuteReaderToGetUserImage(string.Format("SELECT ImageDirectory FROM Employee WHERE EmployeeID = '{0}'", Classes.Variables.UserID));

try

{

if (imagelocation == "")

{//the user has saved no images to their account

}

else

{

byte[] imageBt = null;

//File Stream locates the image directory/location of the file , open and read its data

FileStream fstream = new FileStream(imagelocation, FileMode.Open, FileAccess.Read);

BinaryReader br = new BinaryReader(fstream); //Converts the image into binary

imageBt = br.ReadBytes((int)fstream.Length); //Stores the image binary into the imageBt array

MemoryStream mstream = new MemoryStream(imageBt); //Creates a memory from the provided byes from imageBt

//MemoryStream allows you to use in-memory byte arrays or other data as though they are streams. Instead of storing data in files, you can store data in-memory for additional performance and control over the behavior of your program.

pbProfilePicture.Image = System.Drawing.Image.FromStream(mstream); ///Draws the image From the memory steam provided.

fstream.Dispose();

br.Dispose();

mstream.Dispose();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

}

}

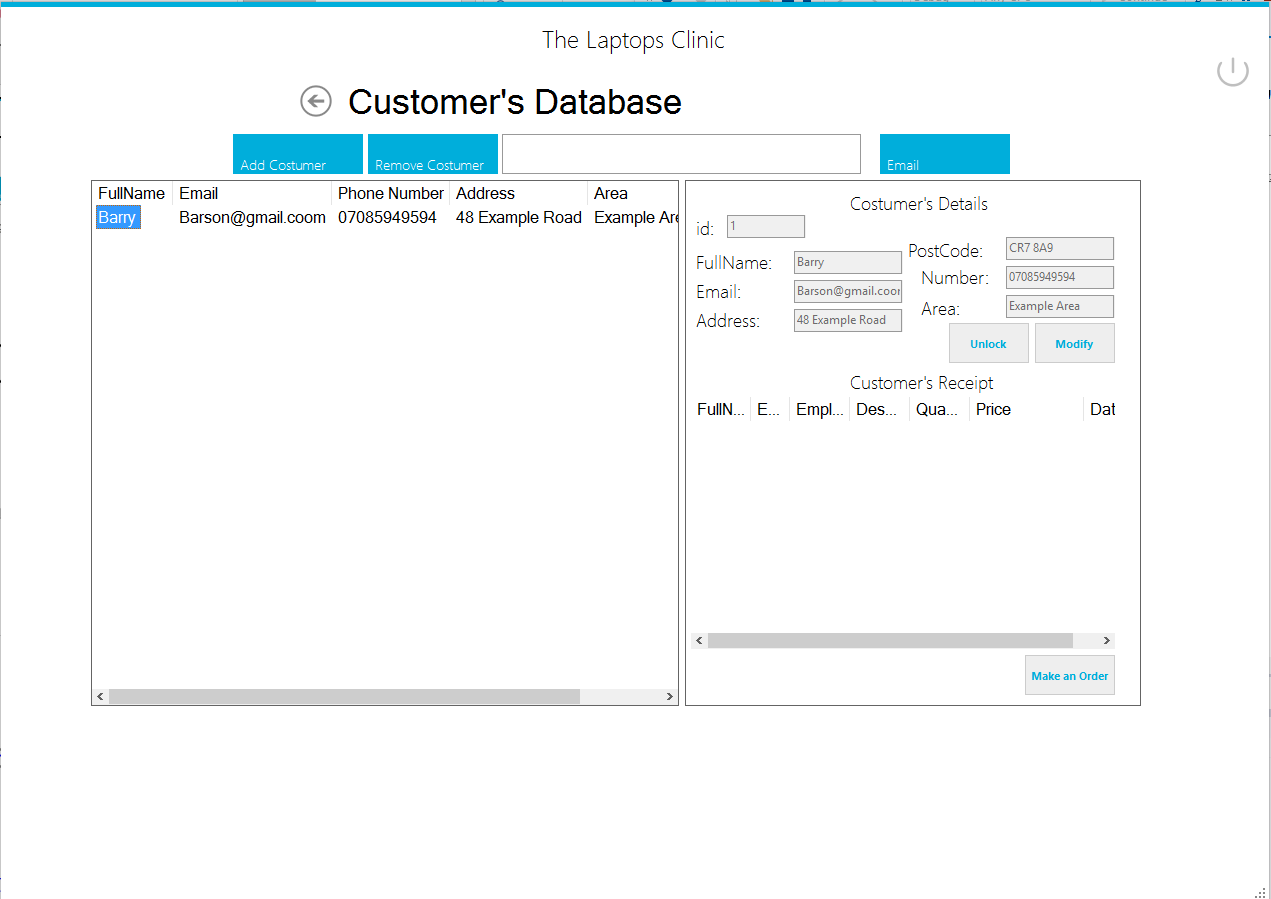
}

}

### CustomerUC

accessible via clicking button in MainForm

btnRemove



btnMakeAnOrder

btnCustomerModify

Txtboxes are named after their lables. E.g. txtID, txtFullName

ReceiptTableListView

btnCustomerLock

btnEmail

txtSearchBar

CustomerListView

btnAddCustomer

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using TheLaptopsClinic.Classes;

using MetroFramework;

using System.Globalization;

namespace TheLaptopsClinic.UCs

{

public partial class CustomerUC : TemplateUC

{

public CustomerUC(Form owner)

: base(owner)

{

InitializeComponent();

}

Color InvertMeAColour(Color ColourToInvert)

{

return Color.FromArgb((byte)~ColourToInvert.R, (byte)~ColourToInvert.G, (byte)~ColourToInvert.B);

}

private void CustomerUC\_Load(object sender, EventArgs e)

{

///Once this UC swipes in front of the Form. It will load and show the Customer Table.

// And Change Fore and Background Colours of components that need to be changed. If the user selected a Black Theme, they won't be able to see any labels that are coloured black.

ShowCustomerTable();

if (msmCustomerUC.Theme == MetroThemeStyle.Dark)

lblCustomerTitle.ForeColor = InvertMeAColour(lblCustomerTitle.ForeColor);

}

private void lnkBack\_Click(object sender, EventArgs e)

{

swipe(false);

}

void ShowCustomerTable() //Method to show the Customer Table

{

CustomerListView.Items.Clear();

try

{

DataTable dt = DataQueries.ShowDataResults("SELECT \* FROM Customer ORDER BY cFullName ASC"); //this returns a dt as explained in DataQueries.cs

foreach (DataRow myRow in dt.Rows) //the returned dt is then used to add column values for each row and present it.

{

CustomerListView.Items.Add(myRow[1].ToString());

CustomerListView.Items[CustomerListView.Items.Count - 1].SubItems.Add(myRow[2].ToString());

CustomerListView.Items[CustomerListView.Items.Count - 1].SubItems.Add(myRow[3].ToString());

CustomerListView.Items[CustomerListView.Items.Count - 1].SubItems.Add(myRow[4].ToString());

CustomerListView.Items[CustomerListView.Items.Count - 1].SubItems.Add(myRow[5].ToString());

CustomerListView.Items[CustomerListView.Items.Count - 1].SubItems.Add(myRow[6].ToString());

//Resizes the Table to be able to show all values

CustomerListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.ColumnContent);

CustomerListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.HeaderSize);

}

}

catch (Exception ex)

{

System.Diagnostics.Debug.WriteLine(ex);

//I don't need to use try catch on every query i do as DataQueries.cs handles all the expections and shows the error on system.diagonistic.

}

}

private void txtSearchBar\_Click(object sender, EventArgs e)

{

}

private void txtSearchBar\_TextChanged(object sender, EventArgs e)

{

CustomerListView.Items.Clear();//Removes all the data inside the Customer Table as it will add new data to it.

try

{

string Searchtext = txtSearchBar.Text;

DataTable sdt = DataQueries.ShowDataResults("SELECT \* FROM Customer WHERE (cFullName LIKE '%" + txtSearchBar.Text + "%') OR (cEmail LIKE '%" + txtSearchBar.Text + "%') OR (cPhoneNumber LIKE '%" + txtSearchBar.Text + "%') OR (cAddress LIKE '%" + txtSearchBar.Text + "%') OR (cArea LIKE '%" + txtSearchBar.Text + "%') OR (cPostCode LIKE '%" + txtSearchBar.Text + "%') ORDER BY cFullName ASC "); //WildCard " & " is used to search to find a specific customer

foreach (DataRow myRow in sdt.Rows)

{//This then present the results of the search in the table. Only showing the Customers affected

CustomerListView.Items.Add(myRow[1].ToString());

CustomerListView.Items[CustomerListView.Items.Count - 1].SubItems.Add(myRow[2].ToString());

CustomerListView.Items[CustomerListView.Items.Count - 1].SubItems.Add(myRow[3].ToString());

CustomerListView.Items[CustomerListView.Items.Count - 1].SubItems.Add(myRow[4].ToString());

CustomerListView.Items[CustomerListView.Items.Count - 1].SubItems.Add(myRow[5].ToString());

CustomerListView.Items[CustomerListView.Items.Count - 1].SubItems.Add(myRow[6].ToString());

CustomerListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.ColumnContent);

CustomerListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.HeaderSize);

}

}

catch (Exception ex)

{

System.Diagnostics.Debug.WriteLine(ex);

//I don't need to use try catch on every query i do as DataQueries.cs handles all the expections and shows the error on system.diagonistic.}

}

}

void ShowReceiptAndFinanceTable() //Once an Customer is picked, it will show the all the receipts of the Customer, so any receipts linked to that Customer is shown

{

ReceiptAndFinanceListView.Items.Clear();

DataTable dt = DataQueries.ShowDataResults(string.Format("SELECT cFullName, cEmail, eFullName, Product, DescriptionOfProduct, Quantity, Price, Date From Customer AS C, Receipt AS R, Employee AS E, Finance AS F WHERE R.EmployeeID = E.EmployeeID AND C.CustomerID = '{0}' AND R.FinanceID = F.FinanceID AND R.CustomerID = C.CustomerID", txtId.Text)); //Dinstinct ensures there isn't any copies, the table it will Full Name of Customer, Email of Customer, FulName of Employee, Product and decription of the product the customer bought, how many they bought (quantity) price and the date.

foreach (DataRow myRow in dt.Rows)

{

ReceiptAndFinanceListView.Items.Add(myRow[0].ToString());

ReceiptAndFinanceListView.Items[ReceiptAndFinanceListView.Items.Count - 1].SubItems.Add(myRow[1].ToString());

ReceiptAndFinanceListView.Items[ReceiptAndFinanceListView.Items.Count - 1].SubItems.Add(myRow[2].ToString());

ReceiptAndFinanceListView.Items[ReceiptAndFinanceListView.Items.Count - 1].SubItems.Add(myRow[3].ToString());

ReceiptAndFinanceListView.Items[ReceiptAndFinanceListView.Items.Count - 1].SubItems.Add(myRow[4].ToString());

ReceiptAndFinanceListView.Items[ReceiptAndFinanceListView.Items.Count - 1].SubItems.Add(myRow[5].ToString());

ReceiptAndFinanceListView.Items[ReceiptAndFinanceListView.Items.Count - 1].SubItems.Add(myRow[6].ToString());

ReceiptAndFinanceListView.Items[ReceiptAndFinanceListView.Items.Count - 1].SubItems.Add(myRow[7].ToString());

ReceiptAndFinanceListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.ColumnContent);

ReceiptAndFinanceListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.HeaderSize);

}

}

private void CustomerListView\_SelectedIndexChanged(object sender, EventArgs e)

{

//When no Customer is selected form the listview returns nothing.

if (CustomerListView.SelectedItems.Count == 0)

return;

//Else a customer is selected

//Item stores all the listview items for the selected Customer

ListViewItem item = CustomerListView.SelectedItems[0];

//These textboxes then refereer to the items in the listview and get their values

//We need to display these values and then be able to update them.

txtFullName.Text = item.SubItems[0].Text;

txtEmail.Text = item.SubItems[1].Text;

txtPhoneNumber.Text = item.SubItems[2].Text;

txtAddress.Text = item.SubItems[3].Text;

txtArea.Text = item.SubItems[4].Text;

txtPostCode.Text = item.SubItems[5].Text;

string GetCustomerID = DataQueries.ExecuteReaderToGetUserID(string.Format("SELECT \* FROM Customer WHERE cFullName = '{0}' AND cEmail = '{1}'", item.SubItems[0].Text, item.SubItems[1].Text)); //Using the values from ListView, we get the ID of the Customer

txtId.Text = GetCustomerID; //present it in a textbox

ShowReceiptAndFinanceTable();//Then it call upon ShowReceiptTableResults which gets the textbox which contain the customerID and show all the receipts with that CustomerID

}

private void btnCustomerLock\_Click(object sender, EventArgs e)

{

//When the Form is loaded, the textboxes with the customer information are disabled.

//Only used to show the details

//However if the Employee seeks to Update the details of the customer,

//The textboxes must be enabled

if (btnCustomerLock.Text == "Unlock") //btnCustomerLock originally start with text "Unlock"

{//Once user clicks on the button, it changes text to "Lock"

//enables the textboxes in which the Employee can change the values for then click the button Modify to update the customer

txtFullName.Enabled = true;

txtEmail.Enabled = true;

txtPhoneNumber.Enabled = true;

txtAddress.Enabled = true;

txtArea.Enabled = true;

txtPostCode.Enabled = true;

btnCustomerLock.Text = "Lock";

}

else //If the btnCustomerLock.text isn't Unlock, so therefore is "Lock"

{

//It will disable all the components

txtFullName.Enabled = false;

txtEmail.Enabled = false;

txtPhoneNumber.Enabled = false;

txtAddress.Enabled = false;

txtArea.Enabled = false;

txtPostCode.Enabled = false;

//changes it back to unlock"

btnCustomerLock.Text = "Unlock";

}

}

private void btnModifyCustomer\_Click(object sender, EventArgs e)

{

if (CustomerListView.SelectedItems.Count == 0)

{

MetroMessageBox.Show(this, "Must select a Customer First to perform a deletion or an order!", "Selection Error", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

else

{

Boolean UpdateCustomer = DataQueries.ExecuteNonQuery(string.Format("UPDATE Customer SET cFullName = '{0}' , cEmail = '{1}', cPhoneNumber = '{2}', cAddress = '{3}', cArea = '{4}', cPostCode = '{5}' WHERE CustomerID = '{6}'",

CultureInfo.CurrentCulture.TextInfo.ToTitleCase(txtFullName.Text.Trim()),

CultureInfo.CurrentUICulture.TextInfo.ToLower(txtEmail.Text.Trim()),

txtPhoneNumber.Text.Trim(),

CultureInfo.CurrentUICulture.TextInfo.ToTitleCase(txtAddress.Text.Trim()),

CultureInfo.CurrentUICulture.TextInfo.ToTitleCase(txtArea.Text.Trim()),

CultureInfo.CurrentUICulture.TextInfo.ToUpper(txtPostCode.Text.Trim()),

txtId.Text

));

if (UpdateCustomer == true)

{

btnCustomerLock.Text = "Lock";

btnCustomerLock.PerformClick();

MetroMessageBox.Show(this, "Succesfully Updated Customer!", "Update Successful", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MetroMessageBox.Show(this, "Couldn't Update Customer!", "Update Unsucessful", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

//After updating the customer Refreshes the Customer Table

ShowCustomerTable();

}

}

private void btnEmailCustomer\_Click(object sender, EventArgs e)

{

Forms.EmailForm email = new Forms.EmailForm();

email.msmEmailForm.Theme = msmCustomerUC.Theme;

email.msmEmailForm.Style = msmCustomerUC.Style;

MessageBox.Show(msmCustomerUC.Style.ToString());

MessageBox.Show(msmCustomerUC.Theme.ToString());

email.ShowDialog();

}

private void btnAddCustomer\_Click(object sender, EventArgs e)

{

Forms.AddCustomerForm addcostumerform = new Forms.AddCustomerForm();

addcostumerform.msmAddCustomerForm.Theme = this.msmCustomerUC.Theme;

addcostumerform.msmAddCustomerForm.Style = this.msmCustomerUC.Style;

MessageBox.Show(msmCustomerUC.Theme.ToString());

MessageBox.Show(addcostumerform.msmAddCustomerForm.Theme.ToString());

addcostumerform.ShowDialog();

}

private void btnRemoveCustomer\_Click(object sender, EventArgs e)

{

// If No Customer is Selected from the CustomerList View. It returns null as you are required to pick an Customer to Delete.

if (CustomerListView.SelectedItems.Count == 0)

{

MetroMessageBox.Show(this, "Must select a Customer First to perform a deletion or an order", "Selection Error", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

else

{

// Creates a ListViewItem which represents values in the Customer List View

ListViewItem item = CustomerListView.SelectedItems[0];

string FullName = item.SubItems[0].Text;

string Email = item.SubItems[1].Text;

//Shows a (Metro) Messagebox altering that the User will Delete this Customer with his details. If User Clicks YES = deletes while No Closes The Message box and does't nothing which is equilvant to returning null.

DialogResult DeleteUser = MetroMessageBox.Show(this, "Are you sure you want to delete this Customer?" + Environment.NewLine + "Customer FullName: " + FullName + Environment.NewLine + "Email:" + Email, "Delete Customer", MessageBoxButtons.YesNo, MessageBoxIcon.Hand);

if (DeleteUser == DialogResult.Yes)

{

Classes.DataQueries.ExecuteNonQuery(string.Format("DELETE FROM Customer WHERE cFullName = '{0}' AND cEmail = '{1}'", FullName, Email));

ShowCustomerTable(); //Calls upon the method ShowCustomerTable to Show the Customer ListView again as a Customer has been removed.

}

}

}

private void btnMakeAnOrder\_Click(object sender, EventArgs e)

{

//Employee is now seeking to conduct a order for the customer selected.

if (CustomerListView.SelectedItems.Count == 0)

{

MetroMessageBox.Show(this, "Must select a Customer First to make an order!", "Selection Error", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

else

{

//No customer selected returns null.

//If a customer is selected, it will show the MakeAnOrder Form and pass the values of the Customer to the Form which are needed.

//also passing the theme and style.

//.ShowDialog ensure the form is show in front of all the forms/UCs and the user can't focus on the others or click on them.

Forms.MakeAnOrderForm makeanorder = new Forms.MakeAnOrderForm();

makeanorder.GetCustomerFullName = txtFullName.Text;

makeanorder.GetCustomerEmail = txtEmail.Text;

makeanorder.msmMakeAnOrderForm.Theme = this.msmCustomerUC.Theme;

makeanorder.msmMakeAnOrderForm.Style = this.msmCustomerUC.Style;

makeanorder.ShowDialog();

}

}

}

}

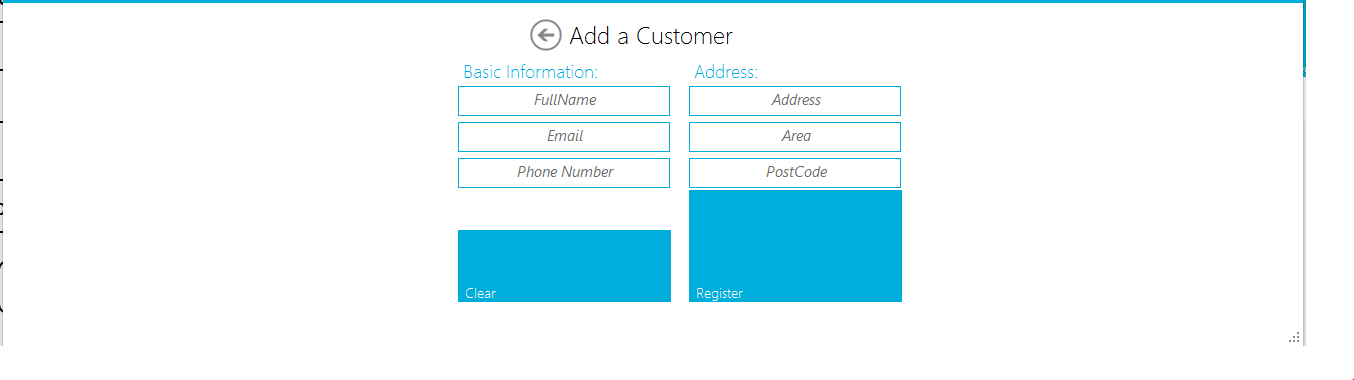
AddCustomer Form and MakeAnOrder Form are accessible via Customer UC.

### AddCustomer Form

used to add Customers.

txtFullName

txtAdressOne



txtEmail

txtPhoneNumber

txtPostCode

txtArea

btnAddCustomer

btnClear

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using MetroFramework.Forms;

using MetroFramework;

using System.Globalization;

using TheLaptopsClinic.Classes;

namespace TheLaptopsClinic.Forms

{

public partial class AddCustomerForm : MetroForm

{

public AddCustomerForm()

{

InitializeComponent();

}

private void AddCustomerForm\_Load(object sender, EventArgs e)

{

this.Theme = msmAddCustomerForm.Theme;

this.Style = msmAddCustomerForm.Style;

}

private void btnClear\_Click(object sender, EventArgs e)

{

txtFullName.Clear();

txtPhoneNumber.Clear();

txtPhoneNumber.Clear();

txtAddress.Clear();

txtArea.Clear();

txtPostCode.Clear();

}

private void btnAddCustomer\_Click(object sender, EventArgs e)

{

//Checks that the required fields are filled in first and in correct format(Phone Number integer)

Int64 checkIfInteger;

if (string.IsNullOrWhiteSpace(txtFullName.Text.Trim()) || string.IsNullOrWhiteSpace(txtEmail.Text.Trim()) || !Int64.TryParse(txtPhoneNumber.Text.Trim(), out checkIfInteger))

{

MetroMessageBox.Show(this, "Fill in the FullName , Email and Phone Number in correct format, enter 0 as Phone Number if blank", "Failed to Add Customer", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{

//FullName, Email are filled, Phone Number is in integer format

//Check if an Customer exists with the exact email

Boolean CheckEmail = DataQueries.ExecuteReaderBolean(string.Format("SELECT \* FROM Customer WHERE cEmail = '{0}'", txtEmail.Text.Trim()));

if (CheckEmail == true)

{

//A Customer Already Exists with the Same Email

MetroMessageBox.Show(this, "A Customer already exists that has the same Exact Email, try again", "Failed to Add Customer", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{

//No Customer exists that has the same Email Address

// Required Fields are filled in, Phone Number is in correct Format, Email is Unique

//Adds Customer to the Database

Boolean AddCustomer = Classes.DataQueries.ExecuteNonQuery(string.Format("INSERT INTO Customer (cFullName, cEmail, cPhoneNumber, cAddress, cArea, cPostCode) VALUES('{0}', '{1}', '{2}', '{3}', '{4}', '{5}')",

CultureInfo.CurrentCulture.TextInfo.ToTitleCase(txtFullName.Text.Trim()),

txtEmail.Text.ToString().Trim(),

txtPhoneNumber.Text.Trim(),

CultureInfo.CurrentCulture.TextInfo.ToTitleCase(txtAddress.Text.ToString().Trim()),

CultureInfo.CurrentCulture.TextInfo.ToTitleCase(txtArea.Text.Trim()),

txtPostCode.Text.ToUpper()));

if (AddCustomer == true)

{

MetroMessageBox.Show(this, "Successfully Added The Customer", "Added the Customer", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MetroMessageBox.Show(this, "Couldn't Add The Customer, Try Again", "Failed to Add The Customer", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

private void lnkBack\_Click(object sender, EventArgs e)

{

this.Close();

}

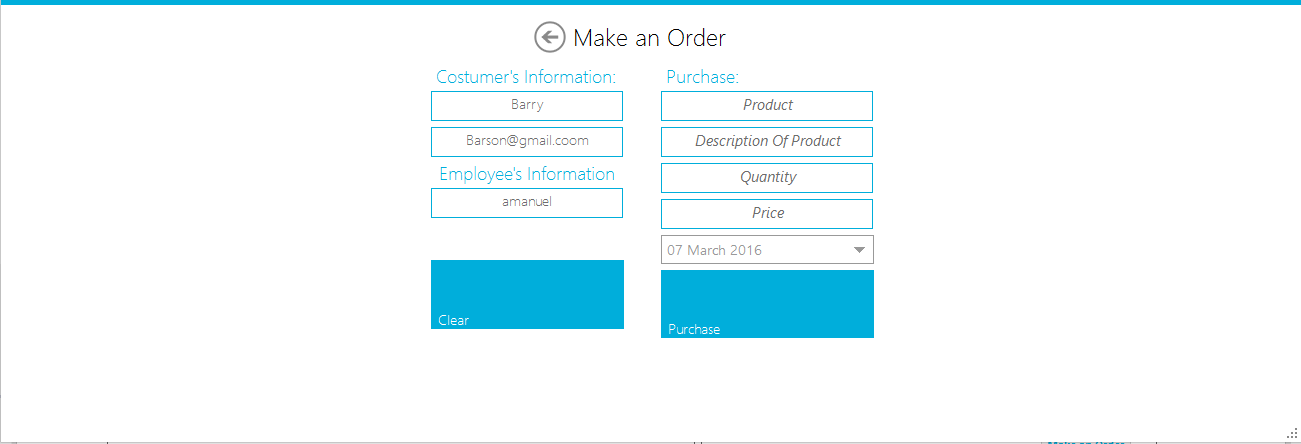
}

}

### MakeAnOrder

is also accessible via Customer UC to make an Order.

txtProduct



txtDescriptionOfProduct

txtEmployeeUsername

txtCustomerFullName

btnPurchase

dataPicker

txtPrice

txtQuantity

btnClear

txtCustomerEmail

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using MetroFramework.Forms;

using System.Globalization;

using MetroFramework;

namespace TheLaptopsClinic.Forms

{

public partial class MakeAnOrderForm : MetroForm

{

/// <summary>

// Get and Set is used to recieve and change these variables.

// Before this UC is loaded, when clicked btnMakeAnOrder in Customer Form, the values for there variables below are set and when this the UC is loaded, they are stored in text boxes to show the Employee.

/// </summary>

private string CustomerFullName;

private string CustomerEmail;

public string GetCustomerFullName

{

get { return CustomerFullName; }

set { CustomerFullName = value; }

}

public string GetCustomerEmail

{

get { return CustomerEmail; }

set { CustomerEmail = value; }

}

public MakeAnOrderForm()

{

InitializeComponent();

}

private void MakeAnOrderForm\_Load(object sender, EventArgs e)

{

//Setting the textbxoes to the Variables

txtCustomerFullName.Text = CustomerFullName;

txtCustomerEmail.Text = CustomerEmail;

txtEmployeeUsername.Text = Classes.Variables.GetEmployeeUsername;

}

private void btnMakeOrder\_Click(object sender, EventArgs e)

{

//Checks that the fields required aren't empty and price, product is integer

Int64 checkIfInteger;

if (string.IsNullOrWhiteSpace(txtProduct.Text.Trim()) || string.IsNullOrWhiteSpace(txtDescriptionOfProduct.Text.Trim()) || !Int64.TryParse(txtPrice.Text.Trim(), out checkIfInteger) || !Int64.TryParse(txtQuantity.Text.Trim(), out checkIfInteger))

{

MetroMessageBox.Show(this, "Fill in the Product & Description of Product, Make sure Quantity and Price is in correct format, enter 0 if their blank", "Failed to Make Order", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{

//Required fields are filled in , Price and Quantity is inputted

string CustomerID = Classes.DataQueries.ExecuteReaderToGetUserID(string.Format("SELECT \* FROM Customer WHERE cFullName = '{0}' AND cEmail = '{1}' ", txtCustomerFullName.Text, txtCustomerEmail.Text));

//Gets ID

//Represent Money in Decimals

Decimal Price = decimal.Parse(txtPrice.Text);

//Inserts the Order into Finance Table

//CreateSpecificCulture sets the Currency to be used ( British Pound)

Boolean InsertProduct = Classes.DataQueries.ExecuteNonQuery(string.Format(CultureInfo.CreateSpecificCulture("en-GB"), "INSERT INTO Finance (Product, DescriptionOfProduct, Quantity, Price, Date) VALUES ( '{0}' , '{1}', '{2}', '{3:C}', '{4:g}')", //{3:C) converts the value of the 4th position (0 is considered as the 1st place) into a currency we picked, It converts into the SpecificCulture i picked (British Pound) while {4:g} converts into a ShortDate+LongTime e.g. // "3/9/2008 4:05:07 PM"

CultureInfo.CurrentCulture.TextInfo.ToTitleCase(txtProduct.Text.TrimEnd()),

CultureInfo.CurrentCulture.TextInfo.ToTitleCase(txtDescriptionOfProduct.Text.TrimEnd()),

txtQuantity.Text,

Price,

dataPicker.Value));//The Data of the Order

string GetFinanceID = Classes.DataQueries.ExecuteReaderToGetUserID("SELECT \* FROM Finance ORDER BY FinanceID DESC LIMIT 1");

//Gets the item we stored in FinanceID and we insert that into Receipt Table along with CustomerID and EmployeeID.

Boolean SaveReceipt = Classes.DataQueries.ExecuteNonQuery(string.Format("INSERT INTO Receipt (FinanceID, EmployeeID, CustomerID) VALUES ('{0}', '{1}', '{2}')",

GetFinanceID,

Classes.Variables.UserID,//Global Variable

CustomerID));

if (InsertProduct == true || SaveReceipt == true)

{

MetroMessageBox.Show(this, "Successfully Saved The Order into Finance and Receipt", "Successfully Added Order", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MetroMessageBox.Show(this, "Couldn't save and insert the Order", "Failed to Add Order", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

private void lnkBack\_Click(object sender, EventArgs e)

{

this.Close();

}

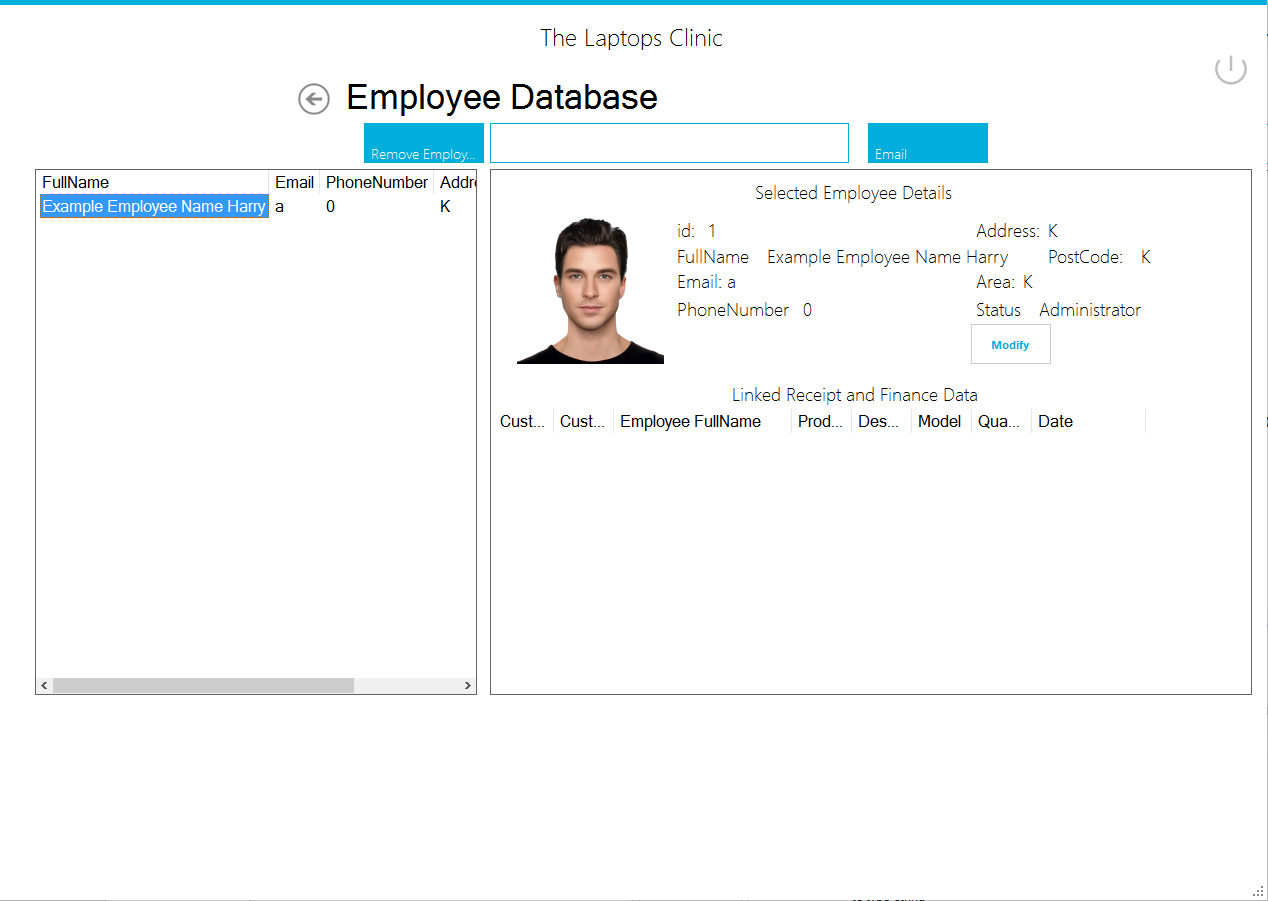
}

}

Done with Customer UC. Now Employee UC, similar to Customer UC, used for Employee to see the receipts they were also involved with, delete employees and also update Employees.

### Employee UC

(accessible via Main Form) – Update Employee and Email is accessible via this pane.



lvEmployeeTable

lvEmployeeReceipt

btnChangeDetails

btnEmail

BtnRemove

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using MetroFramework;

using TheLaptopsClinic.Classes;

using System.IO;

namespace TheLaptopsClinic.UCs

{

public partial class EmployeeUC : TemplateUC

{

public EmployeeUC(Form owner)

: base(owner)

{

InitializeComponent();

}

private void lnkBack\_Click(object sender, EventArgs e)

{

swipe(false);

}

Color InvertMeAColour(Color ColourToInvert)

{

return Color.FromArgb((byte)~ColourToInvert.R, (byte)~ColourToInvert.G, (byte)~ColourToInvert.B);

}

private void EmployeeUC\_Load(object sender, EventArgs e)

{

if (msmEmployeeUC.Theme == MetroThemeStyle.Dark)

lblEmployeeTitle.ForeColor = InvertMeAColour(lblEmployeeTitle.ForeColor);

ShowEmployeeTable(); //Calls upon ShowEmployeeTable to show the Employees in the Table

}

void ShowEmployeeTable() //Shows all the Employees in the system

{

EmployeeListView.Items.Clear(); //Clears the table first before inserting new data into it

try

{

DataTable dt = DataQueries.ShowDataResults("SELECT \* FROM Employee");

foreach (DataRow myRow in dt.Rows)

{

EmployeeListView.Items.Add(myRow[1].ToString());

EmployeeListView.Items[EmployeeListView.Items.Count - 1].SubItems.Add(myRow[2].ToString());

EmployeeListView.Items[EmployeeListView.Items.Count - 1].SubItems.Add(myRow[3].ToString());

EmployeeListView.Items[EmployeeListView.Items.Count - 1].SubItems.Add(myRow[4].ToString());

EmployeeListView.Items[EmployeeListView.Items.Count - 1].SubItems.Add(myRow[5].ToString());

EmployeeListView.Items[EmployeeListView.Items.Count - 1].SubItems.Add(myRow[6].ToString());

EmployeeListView.Items[EmployeeListView.Items.Count - 1].SubItems.Add(myRow[7].ToString());

EmployeeListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.ColumnContent);

EmployeeListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.HeaderSize);

}

}

catch (Exception ex)

{

System.Diagnostics.Debug.WriteLine(ex);

}

}

private void EmployeeListView\_SelectedIndexChanged(object sender, EventArgs e)

{

if (EmployeeListView.SelectedItems.Count == 0) //If no Employees have been selected

return;

ListViewItem item = EmployeeListView.SelectedItems[0]; //item contains all the items in Employeetable

lblFullName.Text = item.SubItems[0].Text; //SubItem[n] represents the column number of the row selected, gets the value of that column

lblEmail.Text = item.SubItems[1].Text;

lblPhoneNumber.Text = item.SubItems[2].Text;

lblAddress.Text = item.SubItems[3].Text;

lblArea.Text = item.SubItems[4].Text;

lblPostCode.Text = item.SubItems[5].Text;

string GetEmployeeId = DataQueries.ExecuteReaderToGetUserID(string.Format("SELECT \* FROM Employee WHERE eFullName = '{0}' AND eEmail = '{1}'", lblFullName.Text, lblEmail.Text));

lblID.Text = GetEmployeeId;

//Checks Statues of The Employee. If their Administrator or Not.

Boolean IsAdmin = DataQueries.ExecuteReaderBolean(string.Format("SELECT \* FROM Administrator WHERE EmployeeID = '{0}'", GetEmployeeId));

MessageBox.Show(IsAdmin.ToString());

if (IsAdmin == true)

{

//Used to indicates the statues of the Employee

lblEmployeeStatus.Text = "Administrator";

}

else

{

//Used to indicates the statues of the Employee

lblEmployeeStatus.Text = "Employee";

}

GetSelectEmployeeImage();

ShowEmployeeReceiptAndFinanceTable(lblID.Text); //Calls upon this method, passing the Employee ID so it display the Receipt And Finance Table where the EmployeeID is in. //Basically finds all the receipts and finance order the Employee has done for customers and show this under thier profile.

}

void GetSelectEmployeeImage() //Gets the user Image as explained before and then present it along with his profile.

{

try

{

string imagelocation = DataQueries.ExecuteReaderToGetUserImage(string.Format("SELECT ImageDirectory FROM Employee WHERE EmployeeID = '{0}'", lblID.Text));

if (imagelocation == "")

{

pbEmployeePicture.Image = null;

}

else

{

byte[] imageBt = null;

FileStream fstream = new FileStream(imagelocation, FileMode.Open, FileAccess.Read);

BinaryReader br = new BinaryReader(fstream);

imageBt = br.ReadBytes((int)fstream.Length);

MemoryStream mstream = new MemoryStream(imageBt);

pbEmployeePicture.Image = System.Drawing.Image.FromStream(mstream);

pbEmployeePicture.ImageLocation = imagelocation;

}

}

catch (Exception ex)

{

System.Diagnostics.Debug.WriteLine(ex);

}

}

void ShowEmployeeReceiptAndFinanceTable(string id) //Recieves the Employee ID

{

EmployeeReceiptAndFinancecListView.Items.Clear();

try

{ //Presents the Receipt And Finance Table together where the Employee has his ID stored in and shows it under his profile

DataTable dt = DataQueries.ShowDataResults(string.Format("SELECT cFullName, cEmail, eFullName, Product, DescriptionOfProduct, Quantity, Price, Date From Customer AS C, Receipt AS R, Employee AS E, Finance AS F WHERE R.EmployeeID = '{0}' AND R.CustomerID = C.CustomerID AND R.FinanceID = F.FinanceID", lblID.Text));

foreach (DataRow myRow in dt.Rows)

{//adds the column values for each row

EmployeeReceiptAndFinancecListView.Items.Add(myRow[0].ToString());

EmployeeReceiptAndFinancecListView.Items[EmployeeReceiptAndFinancecListView.Items.Count - 1].SubItems.Add(myRow[1].ToString());

EmployeeReceiptAndFinancecListView.Items[EmployeeReceiptAndFinancecListView.Items.Count - 1].SubItems.Add(myRow[2].ToString());

EmployeeReceiptAndFinancecListView.Items[EmployeeReceiptAndFinancecListView.Items.Count - 1].SubItems.Add(myRow[3].ToString());

EmployeeReceiptAndFinancecListView.Items[EmployeeReceiptAndFinancecListView.Items.Count - 1].SubItems.Add(myRow[4].ToString());

EmployeeReceiptAndFinancecListView.Items[EmployeeReceiptAndFinancecListView.Items.Count - 1].SubItems.Add(myRow[5].ToString());

EmployeeReceiptAndFinancecListView.Items[EmployeeReceiptAndFinancecListView.Items.Count - 1].SubItems.Add(myRow[6].ToString());

EmployeeReceiptAndFinancecListView.Items[EmployeeReceiptAndFinancecListView.Items.Count - 1].SubItems.Add(myRow[7].ToString());

EmployeeReceiptAndFinancecListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.ColumnContent);

EmployeeReceiptAndFinancecListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.HeaderSize);

}

}

catch (Exception ex)

{

System.Diagnostics.Debug.WriteLine(ex);

}

}

private void txtSearchBar\_Click(object sender, EventArgs e)

{

}

private void txtSearchBar\_TextChanged(object sender, EventArgs e)

{

EmployeeListView.Items.Clear();//Removes all the data inside the Employee Table as it will add new data to it.

try

{

string Searchtext = txtSearchBar.Text;

DataTable sdt = DataQueries.ShowDataResults("SELECT \* FROM Employee WHERE (eFullName LIKE '%" + txtSearchBar.Text + "%') OR (eEmail LIKE '%" + txtSearchBar.Text + "%') OR (ePhoneNumber LIKE '%" + txtSearchBar.Text + "%') OR (eAddress LIKE '%" + txtSearchBar.Text + "%') OR (eArea LIKE '%" + txtSearchBar.Text + "%') OR (ePostCode LIKE '%" + txtSearchBar.Text + "%') "); //WildCard " & " is used to search to find a specific customer

foreach (DataRow myRow in sdt.Rows)

{//This then present the results of the search in the table. Only showing the Customers affected

EmployeeListView.Items.Add(myRow[1].ToString());

EmployeeListView.Items[EmployeeListView.Items.Count - 1].SubItems.Add(myRow[2].ToString());

EmployeeListView.Items[EmployeeListView.Items.Count - 1].SubItems.Add(myRow[3].ToString());

EmployeeListView.Items[EmployeeListView.Items.Count - 1].SubItems.Add(myRow[4].ToString());

EmployeeListView.Items[EmployeeListView.Items.Count - 1].SubItems.Add(myRow[5].ToString());

EmployeeListView.Items[EmployeeListView.Items.Count - 1].SubItems.Add(myRow[6].ToString());

EmployeeListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.ColumnContent);

EmployeeListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.HeaderSize);

}

}

catch (Exception ex)

{

System.Diagnostics.Debug.WriteLine(ex);

//I don't need to use try catch on every query i do as DataQueries.cs handles all the expections and shows the error on system.diagonistic.}

}

}

private void btnEmailEmployee\_Click(object sender, EventArgs e)

{

Forms.EmailForm email = new Forms.EmailForm();

email.ShowDialog();

}

private void btnModifyEmployee\_Click(object sender, EventArgs e)

{

//Used to Update Employee Details

if (EmployeeListView.SelectedItems.Count == 0) //Needs to select an employee form the Employee Table

{

MetroMessageBox.Show(this, "Select an Employee From The Table First before Modifying them!", "Selection Error", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

else

{

if (lblID.Text == Variables.UserID) //The current user logged in user is looking to modify their own account details

{

//Employee is selected from Employee Table and User Clicked Modify. -> this is similar to the Modifying Details in Customer Form however this btnModifyEmployee will show a form as Employee/Administrator are more more detailed in Updating.

Forms.UpdateEmployeeForm updateemployee = new Forms.UpdateEmployeeForm();

//changes the theme of the form and Style

updateemployee.msmUpdateEmployeeForm.Theme = msmEmployeeUC.Theme;

updateemployee.msmUpdateEmployeeForm.Style = msmEmployeeUC.Style;

//Passes the details of the Seleced Employee to the Variables stated in UpdateEmployee

updateemployee.GetFullName = lblFullName.Text;

updateemployee.GetEmail = lblEmail.Text;

updateemployee.GetAddress = lblAddress.Text;

updateemployee.GetArea = lblArea.Text;

updateemployee.GetPostCode = lblPostCode.Text;

updateemployee.GetPhoneNumber = lblPhoneNumber.Text;

updateemployee.GetID = lblID.Text;

if (pbEmployeePicture.ImageLocation == null)

updateemployee.GetImage = null;

else

updateemployee.GetImage = pbEmployeePicture.ImageLocation.ToString();

updateemployee.ShowDialog();

}

else //Current user logged is trying to modify an account that doesn't belong to them which is illegal.

{

MetroMessageBox.Show(this, "You can't modify other employee's details, only allowed to modify your own account detail", "Failed to Modify", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

private void BtnRemove\_Click(object sender, EventArgs e)

{

if (EmployeeListView.SelectedItems.Count == 0) //Needs to select an employee form the Employee Table

{

MetroMessageBox.Show(this, "Select an Employee From The Table First before Modifying them!", "Selection Error", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

else

{

ListViewItem item = EmployeeListView.SelectedItems[0];

string FullName = item.SubItems[0].Text;

string Email = item.SubItems[1].Text;

int CountsAdmins = int.Parse(DataQueries.ExecuteReaderToGetUserID("SELECT COUNT(\*) FROM Administrator"));

MessageBox.Show(CountsAdmins.ToString());

///Checks How many Administraors there are

if (CountsAdmins == 1 && lblEmployeeStatus.Text == "Administrator")

{

MetroMessageBox.Show(this, "Not possible to delete the only Administrator stored in the system, add a new administrator before deleting another Administrator", "Failed to Delete", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else //More than one Adminstrator and is deleteing Employee.

{

DialogResult DeleteUser = MetroMessageBox.Show(this, "Delete this Employee? FullName = " + FullName + " and Email =" + Email, "Delete Employee", MessageBoxButtons.YesNo, MessageBoxIcon.Hand);

if (DeleteUser == DialogResult.Yes)

{

//Checks if the User logged in an Administrator ( Only admins can delete)

Boolean CheckIfAdmin = DataQueries.ExecuteNonQuery(string.Format("SELECT \* FROM Administrator WHERE EmployeeID = '{0}'", Variables.UserID));//gets the logged in user's ID from Global variables

if (CheckIfAdmin == true)

{

if (lblEmployeeStatus.Text == "Employee")

{

DataQueries.ExecuteNonQuery(string.Format("DELETE FROM Employee WHERE EmployeeID = '{0}' AND eEmail = '{1}'", lblID.Text, Email));

MetroMessageBox.Show(this, "Succesfully Deleted Employee", "Succesful Deletion", MessageBoxButtons.OK, MessageBoxIcon.Information);

ShowEmployeeTable(); //Refreshes the Employee table as an Employee has been deleted

}

else //deleting Administrators will also delete them from Administrator Table.

{

if (lblID.Text == Classes.Variables.UserID) //The current user is deleting themself.

{

DialogResult Yes = MetroMessageBox.Show(this, "You are deciding to delete yourself from the system?", "Deleting Oneself", MessageBoxButtons.YesNo, MessageBoxIcon.Information);

if (Yes == DialogResult.Yes)

{

DataQueries.ExecuteNonQuery(string.Format("DELETE FROM Employee WHERE EmployeeID = '{0}' AND eEmail = '{1}'", lblID.Text, Email));

DataQueries.ExecuteNonQuery(string.Format("DELETE FROM Administrator WHERE EmployeeID = '{0}'", lblID.Text));

MetroMessageBox.Show(this, "Succesfully Deleted Administrator", "Succesful Deletion", MessageBoxButtons.OK, MessageBoxIcon.Question);

swipe(false); //UC swipes out view

//Main Form is closed and diposed.

MainMenu closemainform = new MainMenu();

closemainform.Dispose();

//Login Form is shown, deleting oneself means their no longer an logged in so are automatically logged out.

LoginForm showmainform = new LoginForm();

showmainform.ShowDialog();

}

}

else // Current logged in user, isn't deleting themself

{

DataQueries.ExecuteNonQuery(string.Format("DELETE FROM Employee WHERE eFullName = '{0}' AND eEmail = '{1}'", FullName, Email));

DataQueries.ExecuteNonQuery(string.Format("DELETE FROM Administrator WHERE EmployeeID = '{0}'", lblID.Text));

MetroMessageBox.Show(this, "Succesfully Deleted Administrator", "Succesful Deletion", MessageBoxButtons.OK, MessageBoxIcon.Information);

ShowEmployeeTable();

}

}

}

else //CheckIfAdmin == False

{

MetroMessageBox.Show(this, "Failed to Delete Employee, Only Administrators can delete Employees of The Laptop's Clinic", "Failed To Delete", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

}

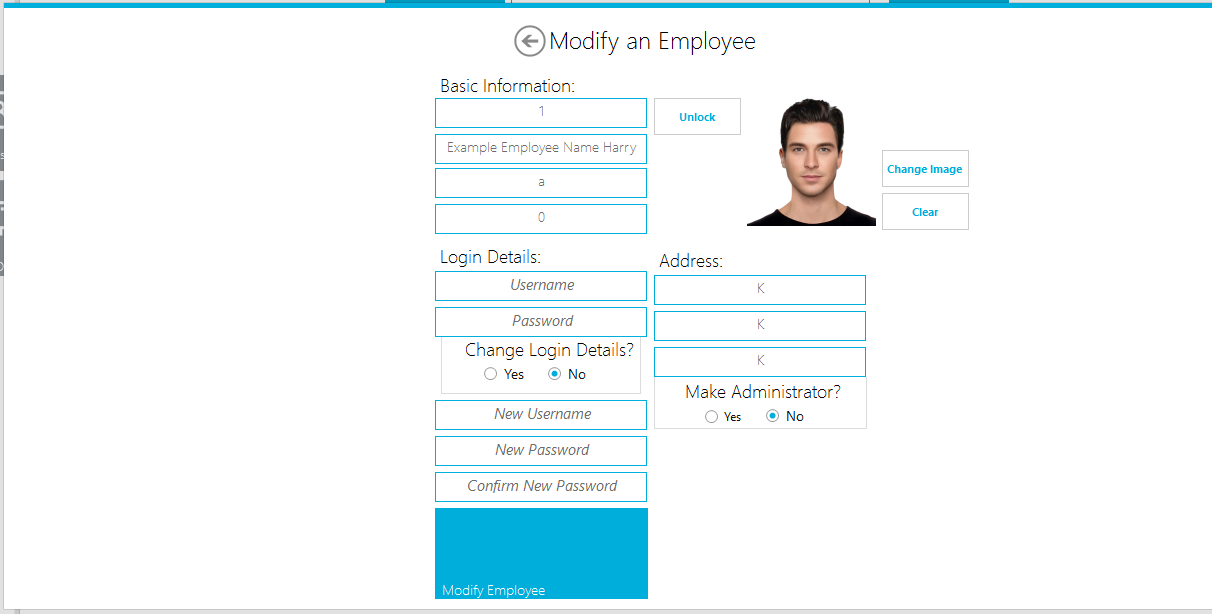
}

}

### Update Employee Form

(accessed when the user click BtnModifyEmployee)

Update Employee Form



using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using MetroFramework.Forms;

using MetroFramework;

using Transitions;

using System.IO;

using TheLaptopsClinic.Classes;

using System.Globalization;

namespace TheLaptopsClinic.Forms

{

public partial class UpdateEmployeeForm : MetroForm

{

/// <summary>

/// Below are the Variables, which values are filled when the User clicked btnModifyEmployee from Employee UC

/// </summary>

string id;

public string GetID

{

get { return id; }

set { id = value; }

}

private string FullName;

public string GetFullName

{

get { return FullName; }

set { FullName = value; }

}

private string Email;

public string GetEmail

{

get { return Email; }

set { Email = value; }

}

private string PhoneNumber;

public string GetPhoneNumber

{

get { return PhoneNumber; }

set { PhoneNumber = value; }

}

private string Address;

public string GetAddress

{

get { return Address; }

set { Address = value; }

}

private string Area;

public string GetArea

{

get { return Area; }

set { Area = value; }

}

private string PostCode;

public string GetPostCode

{

get { return PostCode; }

set { PostCode = value; }

}

private string UserImageLocation;

public string GetImage

{

get { return UserImageLocation; }

set { UserImageLocation = value; }

}

public UpdateEmployeeForm()

{

InitializeComponent();

}

private void UpdateEmployeeForm\_Load(object sender, EventArgs e)

{

this.StyleManager = msmUpdateEmployeeForm;

//Sets the variables that was passed by EmployeeUC to the textboxes.

txtID.Text = id;

txtFullName.Text = FullName;

txtEmail.Text = Email;

txtPhoneNumber.Text = PhoneNumber;

txtAddress.Text = Address;

txtPostCode.Text = PostCode;

txtArea.Text = Area;

GetEmployeeImage(); //gets the user image

}

void GetEmployeeImage() //uses the id that was passed from the Employee UC and locates the image directory and draws the image.

{

try

{

string imagelocation = Classes.DataQueries.ExecuteReaderToGetUserImage(string.Format("SELECT ImageDirectory FROM Employee WHERE EmployeeID = '{0}'", txtID.Text));

if (imagelocation == "")

{

pbEmployeeImage.Image = null;

}

else

{

byte[] imageBt = null;

FileStream fstream = new FileStream(imagelocation, FileMode.Open, FileAccess.Read);

BinaryReader br = new BinaryReader(fstream);

imageBt = br.ReadBytes((int)fstream.Length);

MemoryStream mstream = new MemoryStream(imageBt);

pbEmployeeImage.Image = System.Drawing.Image.FromStream(mstream);

pbEmployeeImage.ImageLocation = UserImageLocation;

}

}

catch (Exception ex)

{

System.Diagnostics.Debug.WriteLine(ex);

}

}

private void btnChangeImage\_Click(object sender, EventArgs e)

{

//User seeks to change their image picture.

OpenFileDialog dlg = new OpenFileDialog();

dlg.Filter = "JPG Files(\*.jpg)|\*.jpg| PNG Files(\*.png)|\*.png|All Files(\*.\*)|\*.\*";

if (dlg.ShowDialog() == DialogResult.OK)

{

string picLocation = dlg.FileName.ToString();

pbEmployeeImage.ImageLocation = picLocation;

UserImageLocation = picLocation;

}

}

private void btnClear\_Click(object sender, EventArgs e)

{

//Clears the Image

pbEmployeeImage.ImageLocation = "";

pbEmployeeImage.Image = null;

}

private void btnLock\_Click(object sender, EventArgs e)

{

//User has to unlock the textboxes to change that

//just a small feature? since user might not seek to change their basic information ( address, email, full name) e.g. but rather their login detials

//also to stop them from accidently editing textboxes

if (btnLock.Text == "Unlock")

{

txtAddress.Enabled = true;

txtArea.Enabled = true;

txtEmail.Enabled = true;

txtFullName.Enabled = true;

txtPhoneNumber.Enabled = true;

txtPostCode.Enabled = true;

btnLock.Text = "Lock";

}

else

{

txtAddress.Enabled = false;

txtArea.Enabled = false;

txtEmail.Enabled = false;

txtFullName.Enabled = false;

txtPhoneNumber.Enabled = false;

txtPostCode.Enabled = false;

btnLock.Text = "Unlock";

}

}

private void rbYesChangeLogin\_CheckedChanged(object sender, EventArgs e)

{

txtNewUsername.Enabled = true;

txtNewPassword.Enabled = true;

txtConfirmNewPassword.Enabled = true;

}

private void rbNoChangeLogin\_CheckedChanged(object sender, EventArgs e)

{

txtNewUsername.Enabled = false;

txtNewPassword.Enabled = false;

txtConfirmNewPassword.Enabled = false;

}

private void rbYesMakeAdmin\_CheckedChanged(object sender, EventArgs e)

{

if (rbYesMakeAdmin.Checked == true) //If checked it will perform the animation that will make the components needed to Make a Emplooyee Admin , come into view.

{

Transition t1 = new Transition(new TransitionType\_Linear(700));

t1.add(lblCreateAdminKey, "Top", 427);

t1.add(txtCreateAdminKey, "Top", 455);

t1.add(LblAdminKey, "Top", 488);

t1.add(txtAdminKey, "Top", 516);

t1.run();

}

else if (rbNoMakeAdmin.Checked == true) //else if checked no, they will disapear out of view

{

Transition t2 = new Transition(new TransitionType\_Linear(700));

t2.add(lblCreateAdminKey, "Top", 645);

t2.add(txtCreateAdminKey, "Top", 673);

t2.add(LblAdminKey, "Top", 706);

t2.add(txtAdminKey, "Top", 734);

t2.run();

}

}

private void btnSave\_Click(object sender, EventArgs e)

{

//1.Need to check if the login details are correct first. Check Employee Username exists

//2.Check password is correct

//3.User login details are correct

//4.Rb ..

Int64 CheckIfInteger;

if (string.IsNullOrWhiteSpace(txtFullName.Text.Trim()) || (string.IsNullOrWhiteSpace(txtEmail.Text.Trim()) || !Int64.TryParse(txtPhoneNumber.Text.Trim(), out CheckIfInteger)))

{

MetroMessageBox.Show(this, "You're are required to provide a FullName and Email with correct Phone Number(0 if Blank) before Updating Employee", "Failed to Update", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{

Boolean CheckUsername = DataQueries.ExecuteReaderBolean(string.Format("SELECT \* FROM Employee WHERE Username = '{0}'", txtUsername.Text.Trim()));

if (CheckUsername == true)

{

//Employee Username is correct

//Need to get the stored password for the Employee and compare it to the one entered.

string GetPasswordFromDatabase = DataQueries.ExecuteReader(string.Format("SELECT \* FROM Employee WHERE Username = '{0}'", txtUsername.Text.Trim()), "Password");

MessageBox.Show(GetPasswordFromDatabase);

//Check and Confirm if the input password and database password are both the same.

Boolean CheckPasswordsMatch = Hashing.Confirm(txtPassword.Text, GetPasswordFromDatabase);

if (CheckPasswordsMatch == true)

{

//User has inputted correct employee and password. Login Details are Correct

//Updates the Employee Details

//If user wants to change their login detials

#region

if (rbYesChangeLogin.Checked == true && rbNoMakeAdmin.Checked == true)

{

//Checks that the new password and confirm new password match

if (txtNewPassword.Text.Trim() == txtConfirmNewPassword.Text.Trim())

{

//new and confirm password match

//Need to hash and salt their new password

string hashNewPassword = Hashing.ComputeHash(txtNewPassword.Text.Trim(), null);

//Updates their login details along with all the other details

Boolean UpdateEmployeeDetails = DataQueries.ExecuteNonQuery(string.Format("UPDATE Employee SET eFullName = '{0}', eEmail = '{1}', ePhoneNumber = '{2}', eAddress = '{3}', eArea = '{4}', ePostCode = '{5}', ImageDirectory = '{6}', Username = '{7}' , Password = '{8}' WHERE Username = '{9}' AND Password = '{10}'",

CultureInfo.CurrentCulture.TextInfo.ToTitleCase(txtFullName.Text.Trim()),

CultureInfo.CurrentCulture.TextInfo.ToLower(txtEmail.Text.Trim()),

txtPhoneNumber.Text.Trim(),

CultureInfo.CurrentUICulture.TextInfo.ToTitleCase(txtAddress.Text.Trim()),

CultureInfo.CurrentUICulture.TextInfo.ToTitleCase(txtArea.Text.Trim()),

CultureInfo.CurrentUICulture.TextInfo.ToUpper(txtPostCode.Text.Trim()),

pbEmployeeImage.ImageLocation.ToString(),

txtNewUsername.Text.Trim(),

hashNewPassword,

txtUsername.Text.Trim(),

GetPasswordFromDatabase

));

if (UpdateEmployeeDetails == true)

MetroMessageBox.Show(this, "Sucessfully Update Employee Details and Login Details", "Updated Employee Details", MessageBoxButtons.OK, MessageBoxIcon.Information);

else

MetroMessageBox.Show(this, "Failed to Update Employee Details, try again with correct details", "Failed to Update Employee", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{

//new and confirm password don't match

MetroMessageBox.Show(this, "New Password and Try Password Don't Match, Make Sure They Match!", "Failed to Update Employee", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

else if (rbNoChangeLogin.Checked == true && rbNoMakeAdmin.Checked == true) //rbYesChangeLogin.checked == false, therefore user must not want to see to update their login details

{

MessageBox.Show("here");

//Updates employee information but not login details.

MessageBox.Show(txtUsername.Text);

MessageBox.Show(GetPasswordFromDatabase);

Boolean UpdateEmployeeDetails = DataQueries.ExecuteNonQuery(string.Format("UPDATE Employee SET eFullName = '{0}', eEmail = '{1}', ePhoneNumber = '{2}', eAddress = '{3}', eArea = '{4}', ePostCode = '{5}', ImageDirectory = '{6}' WHERE Username = '{7}' AND Password = '{8}'",

CultureInfo.CurrentCulture.TextInfo.ToTitleCase(txtFullName.Text.Trim()),

CultureInfo.CurrentCulture.TextInfo.ToLower(txtEmail.Text.Trim()),

txtPhoneNumber.Text.Trim(),

CultureInfo.CurrentUICulture.TextInfo.ToTitleCase(txtAddress.Text.Trim()),

CultureInfo.CurrentUICulture.TextInfo.ToTitleCase(txtArea.Text.Trim()),

CultureInfo.CurrentUICulture.TextInfo.ToUpper(txtPostCode.Text.Trim()),

pbEmployeeImage.ImageLocation.ToString(),

txtUsername.Text.Trim(),

GetPasswordFromDatabase));

if (UpdateEmployeeDetails == true)

MetroMessageBox.Show(this, "Sucessfully Update Employee Details", "Updated Employee Details", MessageBoxButtons.OK, MessageBoxIcon.Information);

else

MetroMessageBox.Show(this, "Failed to Update Employee Details, try again with correct details", "Failed to Update Employee", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

#endregion

//User seeks to promote the employee account to Administrator

#region

if (rbNoChangeLogin.Checked == true && (rbYesMakeAdmin.Checked == true))

{

// Gets the Employee ID and stores it along with the AdminKey to Administrator Table

//Check if Employee is Admin Already

Boolean CheckIfAdmin = DataQueries.ExecuteReaderBolean(string.Format("SELECT \* FROM Employee WHERE EmployeeID = '{0}'", txtID.Text));

if (CheckIfAdmin == true)

{

//EmployeeID exists in Administrator Table so therefore Employee is already an Administrator

MetroMessageBox.Show(this, "This Employee is already an Administrator, can't promote", "Failed to Promote", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else if (rbNoChangeLogin.Checked == true || rbYesMakeAdmin.Checked == true)

{

//EmployeeID doesn't exist in Administrator, therefore isn't an Administrator

//Gets the Employee ID and the AdminKey Of the Employee that is to be promoted, requires AdminKey of existing Administrator also.

Boolean PromoteEmployee = DataQueries.ExecuteNonQuery(string.Format("INSERT INTO Administrator(EmployeeID, AdminKey) Values('{0}', '{1}') SELECT \* Administrator FROM AdminKey = '{2}'", txtID.Text, txtCreateAdminKey.Text.Trim(), txtAdminKey.Text.Trim()));

if (PromoteEmployee == true)

{

MetroMessageBox.Show(this, "Successfully Promoted Employee To Administrator", "Promotion Worked", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{

MetroMessageBox.Show(this, "Failed to Promote Employee to Administrator", "Promotion Failed", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

else

{

//Password entered doesn't match the password stored in the system.

MetroMessageBox.Show(this, "Incorrect Password, try again", "Failed to Update Employee", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

#endregion

else

{

//Employee Username is false

MetroMessageBox.Show(this, "There is no Employee in the Database with that Username, try again!", "Failed to Update Employee", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

private void lnkBack\_Click(object sender, EventArgs e)

{

this.Close();

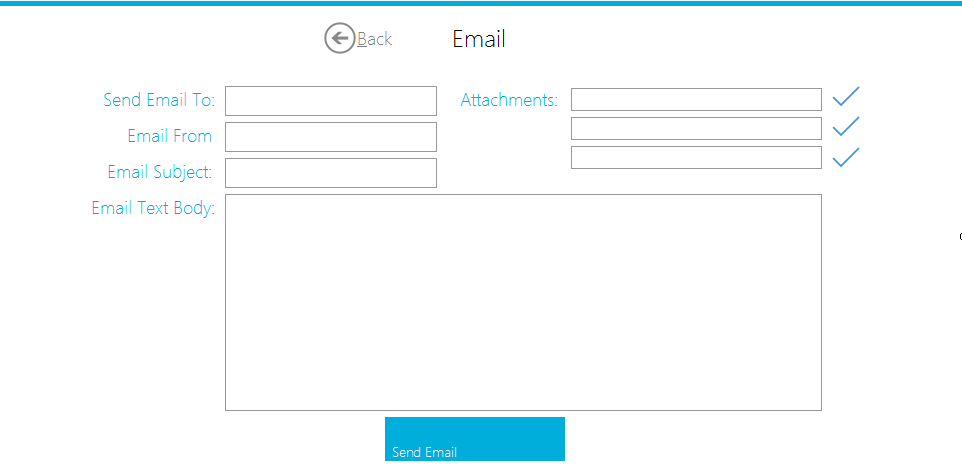
}

}

}

### Email Form

– Accessible via Employee UC, Customer UC and Main Form.



using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using MetroFramework.Forms;

using System.Net.Mail;

using MetroFramework;

namespace TheLaptopsClinic.Forms

{

public partial class EmailForm : MetroForm

{

public EmailForm()

{

InitializeComponent();

}

private void EmailForm\_Load(object sender, EventArgs e)

{

txtEmailFrom.Text = Classes.Variables.GetEmail;

this.StyleManager = msmEmailForm;

}

private void btnSendEmail\_Click(object sender, EventArgs e)

{

//Sends the email

bool bb = System.Net.NetworkInformation.NetworkInterface.GetIsNetworkAvailable(); ///Check for internet Connection

if (bb == true)//There is internet connection

{

MailMessage mail = new MailMessage(txtEmailFrom.Text, txtSendToEmail.Text, txtEmailSubject.Text, txtEmailText.Text);

if (txtAttachmentOne.Text != "")//If their empty, they won't be added to the Mail Attachment if not, gets addded

{

mail.Attachments.Add(new Attachment(txtAttachmentOne.Text));

}

if (txtAttachmentTwo.Text != "")

{

mail.Attachments.Add(new Attachment(txtAttachmentTwo.Text));

}

if (txtAttachmentThree.Text != "")

{

mail.Attachments.Add(new Attachment(txtAttachmentThree.Text));

}

SmtpClient client = new SmtpClient(Classes.Variables.GetSmtpClient);

client.Port = 587;

client.Credentials = new System.Net.NetworkCredential(Classes.Variables.GetEmailUsername, Classes.Variables.GetEmailPassword);

client.EnableSsl = true;

client.Send(mail);

MetroMessageBox.Show(this, "Message has succesfully been sent!", "Sent", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MetroMessageBox.Show(this, "Internet Is Not Available", "Connection Failed", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

private void btnAttachOne\_Click(object sender, EventArgs e)

{

OpenFileDialog dlg = new OpenFileDialog();

dlg.Filter = "JPG Files(\*.jpg)|\*.jpg| PNG Files(\*.png)|\*.png|All Files(\*.\*)|\*.\*";

if (dlg.ShowDialog() == DialogResult.OK)

{

string fileLocation = dlg.FileName.ToString();

txtAttachmentOne.Text = fileLocation;

}

}

private void btnAttachTwo\_Click(object sender, EventArgs e)

{

OpenFileDialog dlg = new OpenFileDialog();

dlg.Filter = "JPG Files(\*.jpg)|\*.jpg| PNG Files(\*.png)|\*.png|All Files(\*.\*)|\*.\*";

if (dlg.ShowDialog() == DialogResult.OK)

{

string fileLocation = dlg.FileName.ToString();

txtAttachmentTwo.Text = fileLocation;

}

}

private void btnAttachThree\_Click(object sender, EventArgs e)

{

OpenFileDialog dlg = new OpenFileDialog();

dlg.Filter = "JPG Files(\*.jpg)|\*.jpg| PNG Files(\*.png)|\*.png|All Files(\*.\*)|\*.\*";

if (dlg.ShowDialog() == DialogResult.OK)

{

string fileLocation = dlg.FileName.ToString();

txtAttachmentThree.Text = fileLocation;

}

}

private void lnkBack\_Click(object sender, EventArgs e)

{

this.Close();

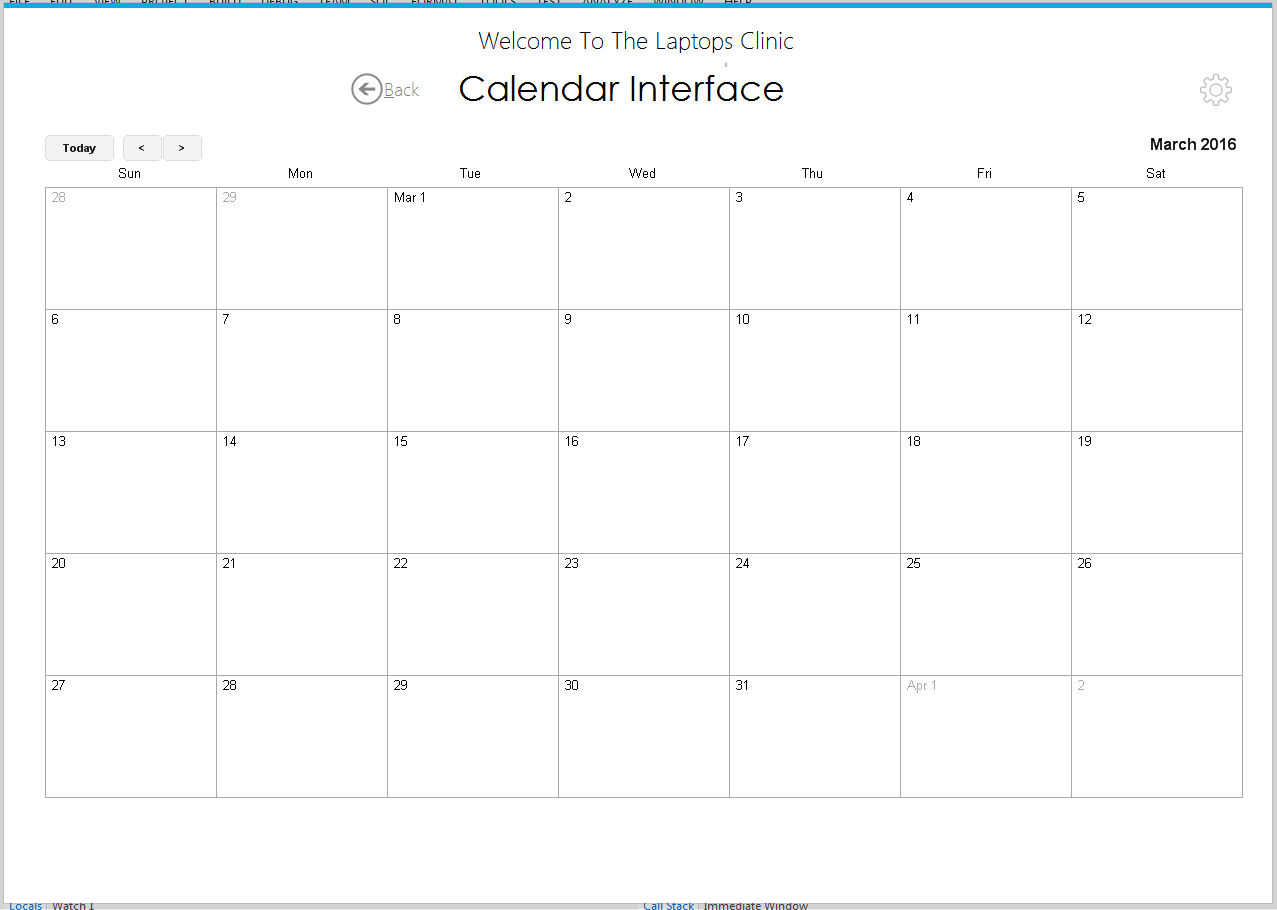
}

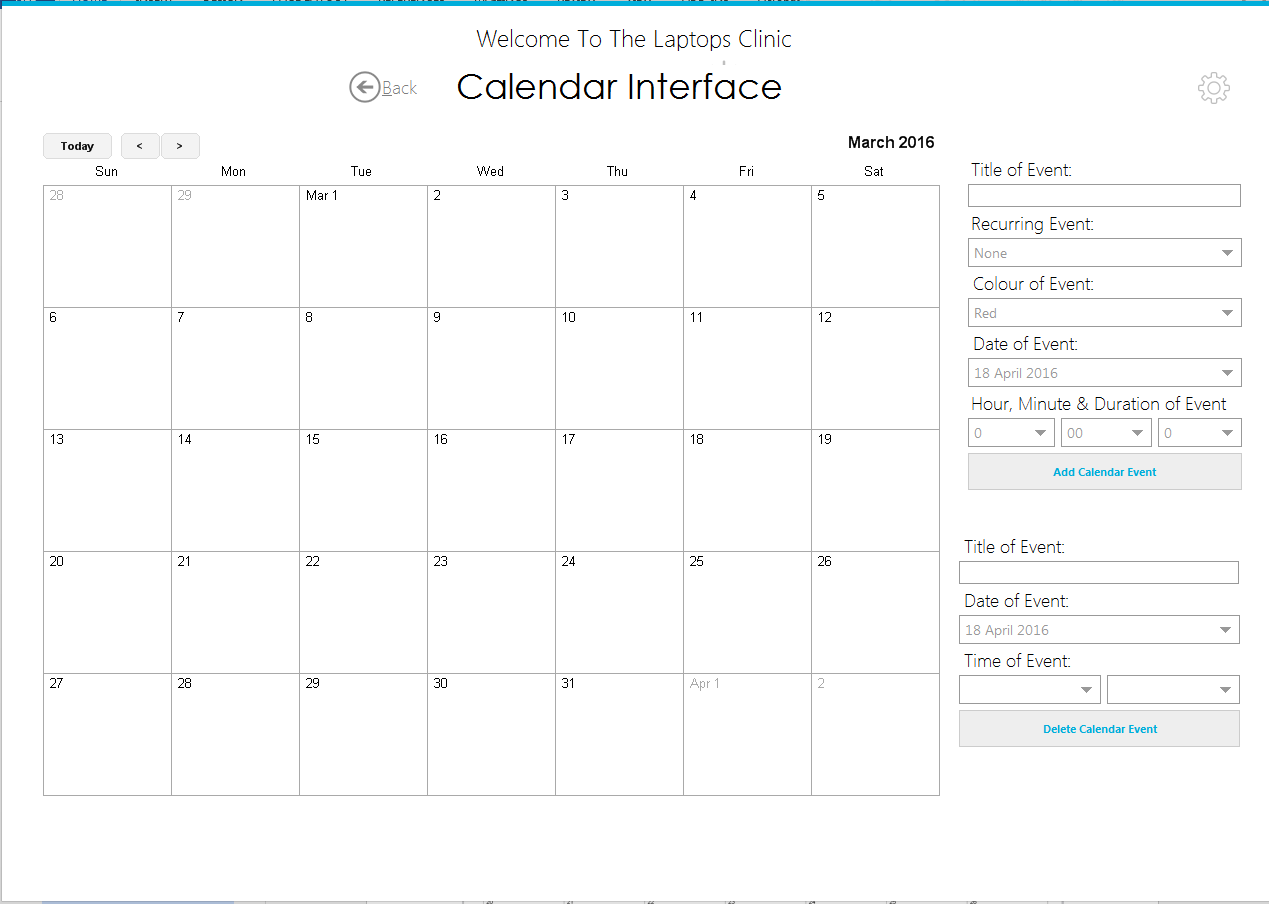
}

}

### Calendar UC

is accessible via Main Form, this represents all the calendar appointment/remainders and presents them to the User.





using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Calendar.NET;

using MetroFramework;

using Transitions;

namespace TheLaptopsClinic.UCs

{

public partial class CalendarUC : TemplateUC

{

public CalendarUC(Form owner)

: base(owner)

{

InitializeComponent();

}

private void CalendarUC\_Load(object sender, EventArgs e)

{

lnkCalendarOptions.Image.Tag = "0";

ShowCalendarDate();

if (msmCalendarUC.Theme == MetroThemeStyle.Dark)

lblCalendarTitle.ForeColor = InvertMeAColour(lblCalendarTitle.ForeColor);

lblAddCalendarEvent.ForeColor = InvertMeAColour(lblAddCalendarEvent.ForeColor);

lblDeleteCalendarEvent.ForeColor = InvertMeAColour(lblDeleteCalendarEvent.ForeColor);

cbAddEventHour.SelectedIndex = 0;

cbAddEventMinute.SelectedIndex = 0;

cbAddEventDuration.SelectedIndex = 0;

cbColours.SelectedIndex = 0;

cbRecurring.SelectedIndex = 0;

}

Color InvertMeAColour(Color ColourToInvert)

{

return Color.FromArgb((byte)~ColourToInvert.R, (byte)~ColourToInvert.G, (byte)~ColourToInvert.B);

}

private void lnkBack\_Click(object sender, EventArgs e)

{

swipe(false);

}

private void lnkCalendarOptions\_Click(object sender, EventArgs e)

{

if (lnkCalendarOptions.Image.Tag == "0")

{

Transition t1 = new Transition(new TransitionType\_Linear(500));

t1.add(pnlCalendarOptions, "Left", 950);

t1.run();

lnkCalendarOptions.Image.Tag = "1";

CalendarView.Size = new Size(937, 702);

CalendarView.Refresh();

}

else if (lnkCalendarOptions.Image.Tag == "1") //Gets the setting out of view and returns the Calendar to regular size then refreshes it.

//Needs to be refreshed for the changes of size to occur.

{

Transition t2 = new Transition(new TransitionType\_Linear(200));

t2.add(pnlCalendarOptions, "Left", 1300);

t2.run();

lnkCalendarOptions.Image.Tag = "0";

CalendarView.Size = new Size(1237, 702);

CalendarView.Refresh();

}

}

private void btnAddCalendarEvent\_Click(object sender, EventArgs e)

{

//When Adding a Calendar Event, we add the event first then Insert it to the Calendar Table

MessageBox.Show(cbAddEventHour.SelectedIndex.ToString() + cbDeleteEventMinute.SelectedIndex.ToString());

MessageBox.Show(cbAddEventHour.SelectedItem + Environment.NewLine + cbAddEventMinute.SelectedItem);

MessageBox.Show(dtAddCalendarEvent.Value.Year.ToString() + Environment.NewLine + dtAddCalendarEvent.Value.Month.ToString() + Environment.NewLine + dtAddCalendarEvent.Value.Day.ToString() + Environment.NewLine + cbAddEventHour.SelectedItem + Environment.NewLine + cbAddEventMinute.SelectedItem + Environment.NewLine + '0');

var addnewdaywithtime = new HolidayEvent

{

IgnoreTimeComponent = false,

Date = new DateTime(dtAddCalendarEvent.Value.Year, dtAddCalendarEvent.Value.Month, dtAddCalendarEvent.Value.Day, cbAddEventHour.SelectedIndex, cbAddEventMinute.SelectedIndex, '0'),

EventText = txtAddEventTitle.Text,

RecurringFrequency = (RecurringFrequencies)Enum.Parse(typeof(RecurringFrequencies), cbRecurring.SelectedItem.ToString()),

EventColor = Color.FromName(cbColours.SelectedItem.ToString()),

EventLengthInHours = int.Parse(cbAddEventDuration.SelectedItem.ToString()),

ThisDayForwardOnly = true,

};

CalendarView.AddEvent(addnewdaywithtime);

try

{

MessageBox.Show(txtAddEventTitle.Text.Trim() + Environment.NewLine + cbAddEventHour.SelectedItem + ":" + cbAddEventMinute.SelectedItem + Environment.NewLine + dtAddCalendarEvent.Value + Environment.NewLine + cbRecurring.SelectedItem.ToString() + Environment.NewLine + cbAddEventDuration.SelectedIndex.ToString() + Environment.NewLine + cbColours.SelectedItem.ToString());

Boolean AddCalendarEvent = Classes.DataQueries.ExecuteNonQuery(string.Format("INSERT INTO Calendar (Title, Time, Date, Frequency, EventLength, Colour) VALUES ( '{0}', '{1}', '{2:d}', '{3}', '{4}', '{5}')",

txtAddEventTitle.Text.Trim(),

cbAddEventHour.SelectedItem + ":" + cbAddEventMinute.SelectedItem,

dtAddCalendarEvent.Value,

cbRecurring.SelectedItem.ToString(),

cbAddEventDuration.SelectedIndex.ToString(),

cbColours.SelectedItem.ToString()));

if (AddCalendarEvent == true) //Checks that the event has been added.

{

MetroMessageBox.Show(this, "Calendar Event Added", "Calendar Successful ", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MetroMessageBox.Show(this, "Calendar Event Failed to Add", "Calendar Failed", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

System.Diagnostics.Debug.WriteLine(ex);

}

}

private void btnDeleteCalendarEvent\_Click(object sender, EventArgs e)

{

MessageBox.Show( txtDeleteEventTitle.Text + cbDeleteEventHour.SelectedItem + ":" + cbDeleteEventMinute.SelectedItem + dtDeleteCalendarEvent.Value );

//Used to a Calendar Event

Boolean DeleteCalendarEvent = Classes.DataQueries.ExecuteNonQuery(string.Format("DELETE FROM Calendar WHERE Title = '{0}' AND Time = '{1}' AND Date = '{2:d}'",

txtDeleteEventTitle.Text.Trim(),

cbDeleteEventHour.SelectedItem + ":" + cbDeleteEventMinute.SelectedItem,

dtDeleteCalendarEvent.Value));

MessageBox.Show(DeleteCalendarEvent.ToString());

CalendarView.Refresh();

ShowCalendarDate();//Refreshes the Calendar Event and shows the Calendar Event again.

}

void ShowCalendarDate()

{

CalendarView.Refresh(); //Removes all the events that we added before. As we are able to refresh the database.

//If it is not refreshed, events from the previous SELECT \* From Calendar would still be there.

DataTable CalendarDT = Classes.DataQueries.ShowDataResults("SELECT \* FROM Calendar");

foreach (DataRow myRow in CalendarDT.Rows) //Returns Calendar DataTable , For each row in the Calendar Table, it will get the title of the event, time, data, how frequent it is, the length of the event, and colour of the event and then show it in the Calendar

{

string Title = (myRow[1].ToString());//MyRow[n] represents the position of the string we seek. e.g MyRow[0] will returns CalendarID, the position is the same as the position in the data. It just represents the column number

string Time = (myRow[2].ToString());

string Date = myRow[3].ToString();

string Frequency = myRow[4].ToString();

string EventLength = myRow[5].ToString();

string Colour = myRow[6].ToString();

string[] separatedates = Date.ToString().Split('/'); /// The date is stored as 00/00/00 , we split the components by "/" to find the day, month and year. While time is stored as 00:00 so we split by ":" to get the Hour and Minute, we use this to add the event data and time of it.

string[] separatetime = Time.Split(':');

Int32 DayofTheMonth = int.Parse(separatedates[0]);

Int32 MonthOfTheYear = int.Parse(separatedates[1]);

Int32 YearOfDecade = int.Parse(separatedates[2]);

Int32 HourOfTheDay = int.Parse(separatetime[0]);

Int32 MinuteOfTheHour = int.Parse(separatetime[1]);

try

{

var addCalendarDate = new HolidayEvent //Adds the event

{

IgnoreTimeComponent = false, //using Time

Date = new DateTime(YearOfDecade, MonthOfTheYear, DayofTheMonth, HourOfTheDay, MinuteOfTheHour, '0'), //Adds the Date of the event, Year/Month/Day/Minute /Second. Second is set to "0" as it isn't important. None will add an event at a time of 3:20 past 20 seconds..

EventText = Title,

RecurringFrequency = (RecurringFrequencies)Enum.Parse(typeof(RecurringFrequencies), Frequency), //Enum.Parse is used to Converts the string representation of the RecurryingFrequency to one or more enumerated constants to an equivalent enumerated object. Previously Setting RecurringFrequency = "Weekly"; creates an error as an Object can't be set to a string. However Enum allows us to find the object which represents Weekly and use it as our object.

EventColor = Color.FromName(Colour), //Event color is set the colour we colour we got from the Calendar Table ,

EventLengthInHours = int.Parse(EventLength),

ThisDayForwardOnly = true, //Means that the event can only occur ahead of this time , not previous time.

};

CalendarView.AddEvent(addCalendarDate); //Adds the event.

//This will do it for every row until all the Events are added.

}

catch (Exception ex)

{

System.Diagnostics.Debug.WriteLine(ex);

}

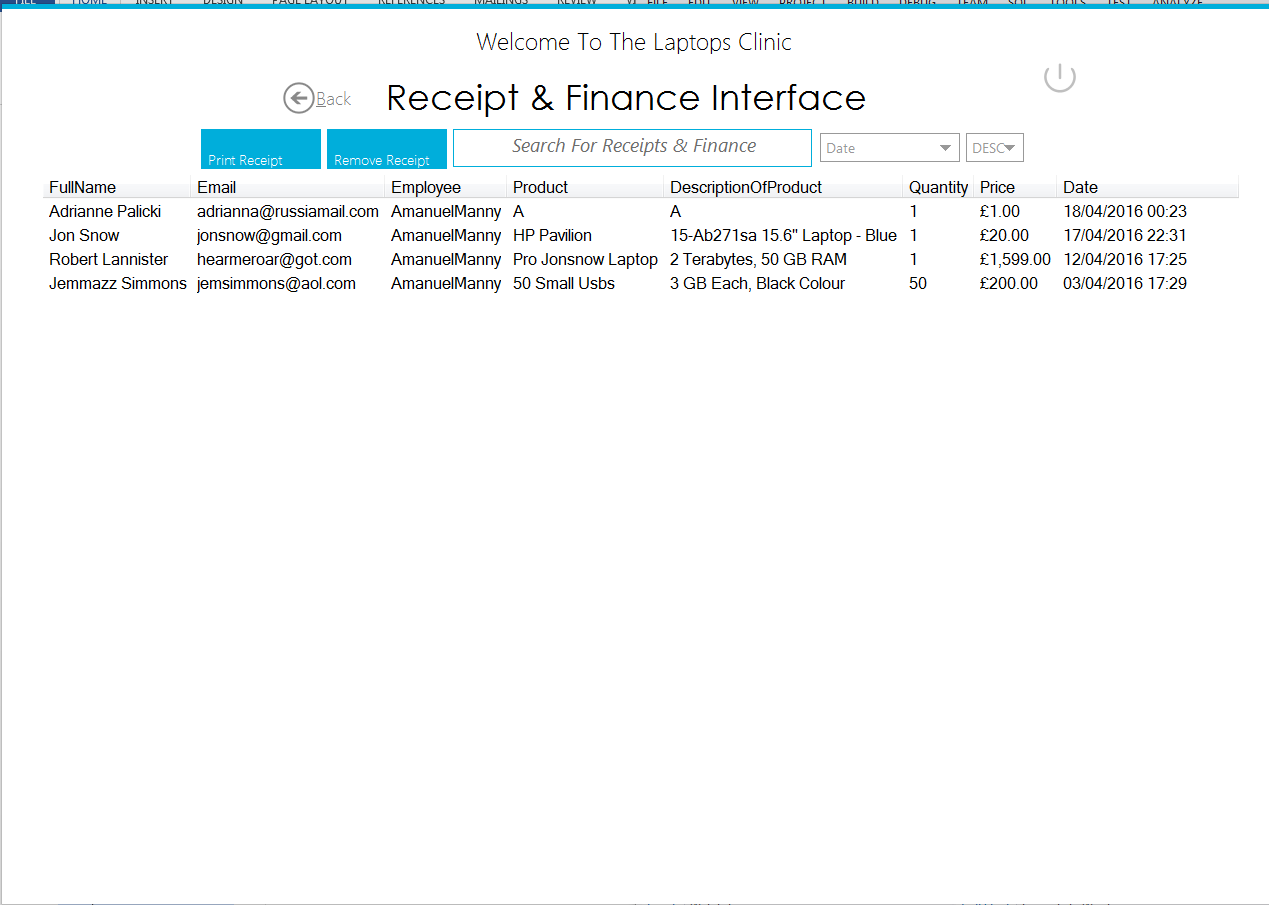
}

}

}

}

### ReceiptAndFinance UC



using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Drawing;

using System.Data;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using MetroFramework;

using TheLaptopsClinic.Classes;

using System.Drawing.Printing;

namespace TheLaptopsClinic.UCs

{

public partial class ReceiptAndFinanceUC : TemplateUC

{

public ReceiptAndFinanceUC(Form owner) : base (owner)

{

InitializeComponent();

}

private void lnkBack\_Click(object sender, EventArgs e)

{

swipe(false);

}

Color InvertMeAColour(Color ColourToInvert)

{

return Color.FromArgb((byte)~ColourToInvert.R, (byte)~ColourToInvert.G, (byte)~ColourToInvert.B);

}

private void ReceiptAndFinance\_Load(object sender, EventArgs e)

{

if (msmReceiptAndFinanceUC.Theme == MetroThemeStyle.Dark)

lblReceipeAndFinanceTitle.ForeColor = InvertMeAColour(lblReceipeAndFinanceTitle.ForeColor);

cbASCDesc.Text = "DESC";

cbOrderBy.Text = "Date";

ShowReceiptAndFinanceTable(cbOrderBy.Text, cbASCDesc.Text);

}

private void cbOrderBy\_SelectedIndexChanged(object sender, EventArgs e)

{

}

private void cbOrderBy\_SelectedValueChanged(object sender, EventArgs e)

{

ShowReceiptAndFinanceTable(cbOrderBy.Text, cbASCDesc.Text);

}

private void cbASCDesc\_SelectedIndexChanged(object sender, EventArgs e)

{

ShowReceiptAndFinanceTable(cbOrderBy.Text, cbASCDesc.Text);

}

void ShowReceiptAndFinanceTable(string OrderBy, string ASCDesc)

{

DataTable dt = DataQueries.ShowDataResults("SELECT cFullName, cEmail, Username, Product, DescriptionOfProduct, Quantity, Price, Date From Customer AS C, Receipt AS R, Employee AS E, Finance AS F WHERE R.CustomerID = C.CustomerID AND R.FinanceID = F.FinanceID AND R.EmployeeID = E.EmployeeID ORDER BY " + OrderBy + " " + ASCDesc + "");

ReceiptFinanceListView.Items.Clear();

foreach (DataRow myRow in dt.Rows)

{

ReceiptFinanceListView.Items.Add(myRow[0].ToString());

ReceiptFinanceListView.Items[ReceiptFinanceListView.Items.Count - 1].SubItems.Add(myRow[1].ToString());

ReceiptFinanceListView.Items[ReceiptFinanceListView.Items.Count - 1].SubItems.Add(myRow[2].ToString());

ReceiptFinanceListView.Items[ReceiptFinanceListView.Items.Count - 1].SubItems.Add(myRow[3].ToString());

ReceiptFinanceListView.Items[ReceiptFinanceListView.Items.Count - 1].SubItems.Add(myRow[4].ToString());

ReceiptFinanceListView.Items[ReceiptFinanceListView.Items.Count - 1].SubItems.Add(myRow[5].ToString());

ReceiptFinanceListView.Items[ReceiptFinanceListView.Items.Count - 1].SubItems.Add(myRow[6].ToString());

ReceiptFinanceListView.Items[ReceiptFinanceListView.Items.Count - 1].SubItems.Add(myRow[7].ToString());

ReceiptFinanceListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.ColumnContent);

ReceiptFinanceListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.HeaderSize);

}

}

private void txtSearchBar\_TextChanged(object sender, EventArgs e)

{

DataTable sdt = DataQueries.ShowDataResults("SELECT cFullName, cEmail, eFullName, Product, DescriptionOfProduct, Quantity, Price, Date From Customer AS C, Receipt AS R, Employee AS E, Finance AS F WHERE R.CustomerID = C.CustomerID AND R.FinanceID = F.FinanceID AND R.EmployeeID = E.EmployeeID AND (eFullName LIKE '%" + txtSearchBar.Text + "%') OR (eEmail LIKE '%" + txtSearchBar.Text + "%') OR (eFullName LIKE '%" + txtSearchBar.Text + "%') OR (Product LIKE '%" + txtSearchBar.Text + "%') OR (DescriptionOfProduct LIKE '%" + txtSearchBar.Text + "%') OR (Quantity LIKE '%" + txtSearchBar.Text + "%') OR (Price LIKE '%" + txtSearchBar.Text + "%') OR (Date LIKE '%" + txtSearchBar.Text + "%') ORDER BY " + cbOrderBy.Text + " " + cbASCDesc.Text + ""); //WildCard " & " is used to search to find a specific customer

ReceiptFinanceListView.Items.Clear();

foreach (DataRow myRow in sdt.Rows)

{

ReceiptFinanceListView.Items.Add(myRow[0].ToString());

ReceiptFinanceListView.Items[ReceiptFinanceListView.Items.Count - 1].SubItems.Add(myRow[1].ToString());

ReceiptFinanceListView.Items[ReceiptFinanceListView.Items.Count - 1].SubItems.Add(myRow[2].ToString());

ReceiptFinanceListView.Items[ReceiptFinanceListView.Items.Count - 1].SubItems.Add(myRow[3].ToString());

ReceiptFinanceListView.Items[ReceiptFinanceListView.Items.Count - 1].SubItems.Add(myRow[4].ToString());

ReceiptFinanceListView.Items[ReceiptFinanceListView.Items.Count - 1].SubItems.Add(myRow[5].ToString());

ReceiptFinanceListView.Items[ReceiptFinanceListView.Items.Count - 1].SubItems.Add(myRow[6].ToString());

ReceiptFinanceListView.Items[ReceiptFinanceListView.Items.Count - 1].SubItems.Add(myRow[7].ToString());

ReceiptFinanceListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.ColumnContent);

ReceiptFinanceListView.AutoResizeColumns(ColumnHeaderAutoResizeStyle.HeaderSize);

}

}

private void txtSearchBar\_Click(object sender, EventArgs e)

{

}

private void btnPrintReceipt\_Click(object sender, EventArgs e)

{

PrintReceipt();

}

private void PrintReceipt()

{

PrintDialog printDialog = new PrintDialog();

PrintDocument printDocument = new PrintDocument();

printDialog.Document = printDocument;

printDocument.PrintPage += PrintDocument\_PrintPage;

DialogResult result = printDialog.ShowDialog();

if (result == DialogResult.OK)

{

printDocument.Print();

}

}

private void PrintDocument\_PrintPage(object sender, PrintPageEventArgs e)

{

Graphics graphic = e.Graphics;

Font font = new Font("Calibri (Body)", 15);

float fontHeight = font.GetHeight();

int startX = 20;

int startY = 20;

int offset = 40;

graphic.DrawString("Welcome To The Laptop's CLinic", new Font("Courier New", 18), new SolidBrush(Color.Black), startX, startY);

graphic.DrawString("20 Selsdon Road, CR2 6PA Croydon, United Kingdom", new Font("Cambria", 15), new SolidBrush(Color.Black), startX, startY + offset);

offset = offset + 20;

graphic.DrawString("Phone us at : 0208 688 4167 - 0800 01 41677 ", new Font("Cambria", 15), new SolidBrush(Color.Black), startX, startY + offset);

offset = offset + 20;

graphic.DrawString("Email us at info@thelaptopsclinic.co.uk", new Font("Cambria", 15), new SolidBrush(Color.Black), startX, startY + offset);

offset = offset + 35;

ListViewItem item = ReceiptFinanceListView.SelectedItems[0];

string CustomerFullName = item.SubItems[0].Text;

string CustomerEmail = item.SubItems[1].Text;

string Employee = item.SubItems[2].Text;

string product = item.SubItems[3].Text;

string productDescription = item.SubItems[4].Text;

string Quantity = item.SubItems[5].Text;

string price = item.SubItems[6].Text;

string date = item.SubItems[7].Text;

graphic.DrawString("Customer's Full Name : " + CustomerFullName, font, new SolidBrush(Color.Black), startX, startY + offset);

offset = offset + 20;

graphic.DrawString("Customer's Email :" + CustomerEmail, font, new SolidBrush(Color.Black), startX, startY + offset);

offset = offset + 20;

graphic.DrawString("Employee Dealt With :" + Employee, font, new SolidBrush(Color.Black), startX, startY + offset);

offset = offset + 20;

graphic.DrawString("Prouct Bought :" + product, font, new SolidBrush(Color.Black), startX, startY + offset);

offset = offset + 20;

graphic.DrawString("Product Description :" + productDescription, font, new SolidBrush(Color.Black), startX, startY + offset);

offset = offset + 20;

graphic.DrawString("Quantity :" + Quantity, font, new SolidBrush(Color.Black), startX, startY + offset);

offset = offset + 20;

graphic.DrawString("Date of Purchase :" + date, font, new SolidBrush(Color.Black), startX, startY + offset);

offset = offset + 20;

offset = offset + (int)fontHeight + 5;

graphic.DrawString("Total Price :" + price, font, new SolidBrush(Color.Black), startX, startY + offset);

offset = offset + 20;

}

private void BtnRemoveReceipt\_Click(object sender, EventArgs e)

{

if (ReceiptFinanceListView.SelectedItems.Count == 0 ) //No Receipt is selected from the ListView

{

MetroMessageBox.Show(this, "Select an Receipt From The Table First before Removing them!", "Selection Error", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

else

{

// Creates a ListViewItem which represents values in the Customer List View

ListViewItem item = ReceiptFinanceListView.SelectedItems[0];

string FullName = item.SubItems[0].Text;

string Email = item.SubItems[1].Text;

string Employee = item.SubItems[2].Text;

string Product = item.SubItems[3].Text;

string DescriptionOfProduct = item.SubItems[4].Text;

string Quantity = item.SubItems[5].Text;

string Price = item.SubItems[6].Text;

string Date = item.SubItems[7].Text;

MessageBox.Show(FullName + " " + Email);

//Shows a (Metro) Messagebox altering that the User will Delete this Customer with his details. If User Clicks YES = deletes while No Closes The Message box and does't nothing which is equilvant to returning null.

DialogResult DeleteReceipt = MetroMessageBox.Show(this, "Are you sure you want to remove this Receipt? " + Environment.NewLine + " + Customer FullName: " + FullName + Environment.NewLine + "Employee Username: " + Employee + Environment.NewLine + "Product: " + Product + Environment.NewLine + "Description Of Product: " + DescriptionOfProduct + Environment.NewLine + "Quantity: " + Quantity + Environment.NewLine + "Price: " + Price + Environment.NewLine + "Date of Receipt: " + Date , "Delete Customer", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);

if (DeleteReceipt == DialogResult.Yes)

{

string CustomerID = DataQueries.ExecuteReaderToGetUserID(string.Format("SELECT \* FROM Customer WHERE cFullName = '{0}' AND cEmail = '{1}'", FullName, Email));

string EmployeeID = DataQueries.ExecuteReaderToGetUserID(string.Format("SELECT \* FROM Employee WHERE Username = '{0}'", Employee));

string FinanceID = DataQueries.ExecuteReaderToGetUserID(string.Format("SELECT \* FROM Finance WHERE Product = '{0}' AND DescriptionOfProduct = '{1}' AND Quantity = '{2}' AND Price = '{3}' AND Date = '{4}'", Product, DescriptionOfProduct, Quantity, Price, Date));

MessageBox.Show(CustomerID + Environment.NewLine + EmployeeID + Environment.NewLine + FinanceID);

DataQueries.ExecuteNonQuery(string.Format("DELETE FROM Receipt WHERE FinanceID = '{0}' AND EmployeeID = '{1}' AND CustomerID", FinanceID, EmployeeID, CustomerID));

ShowReceiptAndFinanceTable(cbOrderBy.Text, cbASCDesc.Text);

}

}

}

}

}

# Appraisal

## Comparison of Project Performance against Objectives

The following table compares the performance of the completed system to the objectives originally laid out on page 28, to assess whether all the user’s needs have been met.

|  |  |  |  |
| --- | --- | --- | --- |
| Key: **Objective not met** | **Objective partially met** | **Objective met** | **Objective exceeded** |

|  |  |
| --- | --- |
| Original Objective | Completed System |
| 1. The new system will allow the TLC to save customer’s details e.g. Name, Address, Phone & employee‘s details e.g. Username, Password, to be able to edit the database, remove and search, also allowing a way of contacting them by email e.g. | The software does store customer and employee details, the employee are able to edit, remove and search and a form of contacting is possible as I have provided an Email Form to email |
| 1. The system will use steganography to encrypt and decrypt user passwords for security purposes, also allowing to be able to receive forgotten password. | The system now doesn’t use steganography, since steganography is easily hackables but now rather uses salting and hashing making it computable time consuming and difficult for the hacker to decrypt the encrypted values |
| 1. The system will have a clear and understandable user interface | Objectve met, the program uses , the selected user colour scheme and provides a simple interface which is clear and understandable. |
| 1. It will allow different administration access with different administration access, they should be able to exit anytime and data edited should be saved correctly and data is not lost. | The system uses AdminKey to validate Administrator Rights, employee have their own rights but can’t perfrom administrator actions. Data is saved correctly in the right format or an messagebox appears or null is entered if the format is incorrect.  Data is not lost |
| 1. It will have a task planner and a calendar to remind the TLC of meetings, deadlines and alarm system. | I wasn’t able to add an Alarm system but I provided the calendar which shows calendar events, deadlines and can be used as a task planner. |
| 1. It will Show receipts and finance and also allow receipts to be printed out and emailed also. | This is all possible, receipts can be printed and saved also. |
| 1. The system will include shortcuts for other parts of the system or the internet. E.g. accessing their website and social media through the software. | Have added shortcut to Email Form and any other forms that relative to themselves e.g. Customer Form has a shortcut to AddCustomer Form. However, I haven’t added shortcut for the social media of the company. Even though it easy to do so, as you just need to provide a hyperlink once linked to open the internet. I wasn’t able |

## Analysis of User Feedback

I gave the installation files, user guide and a feedback questionnaire to each of the three Employees in the Laptop’s Clinic. In order to get an objective assessment of how the users rated the system after the intial use, I asked them to rate the interface, adding/removing customer process, printing receipts, table display process, and asked administrators about the Administration access. They rated the system out of five (five being the highest.) I then asked for additional feedback on any aspect of the system, and have summarised their thoughts below.

|  |
| --- |
| Desirable Features/Changes |
| Moe | Lucy | James |
| * Provide more features such as being able to add and edit text in the Receipt print * Provide more shortcuts and use pictures that relate to the Laptops Clinic * Provide tables that show monthly salary, most product sold e.g. | * Make it possible to change theme and email details after login in the system * More design in the logo instead of plain * Make it possible to also add pictures of customers | * Make it possible to save the receipt in different format such a doc, so it can be edited. |

## Extensions

Even though I just started learning C# at the beginning of the course, I was able to gather tons of information and learn the language. I’m not fairly statified with my program, I love its design and such however, if I had an extra few months. I would have added an Online SQL server so that the database can be linked between all the devices in the Laptops Clinic. I would have added an alarm system, a system timeout (when the user has been idle for a certain time), designed the interface to make it more perfect and make the receipt more user friendly instead of a map and the receipt. It would have even been possible to create a browser in the program so the user can check their social updates on their facebook websites, twitter and such. Maybe even notify them if there is a notifcation. But however what I’m proud of is that I was able to make this program by myself, through searching and watching tons of youtube videos. I was able to uses classes and write less lines of codes. I used to write around 10 lines of code to make a simple query however I turned into one line through learning.

I even believe if I knew c# at the start of the project and how to do the things I have done in my system. I would have been able to finish the program in half the month and had more time to add more features, like the ones suggested in the feedback. If I were to do this again, I would have used more classes and object oreinted programming to represent the Customer, Employee, Product e.g. and even dynamically programmed my project.