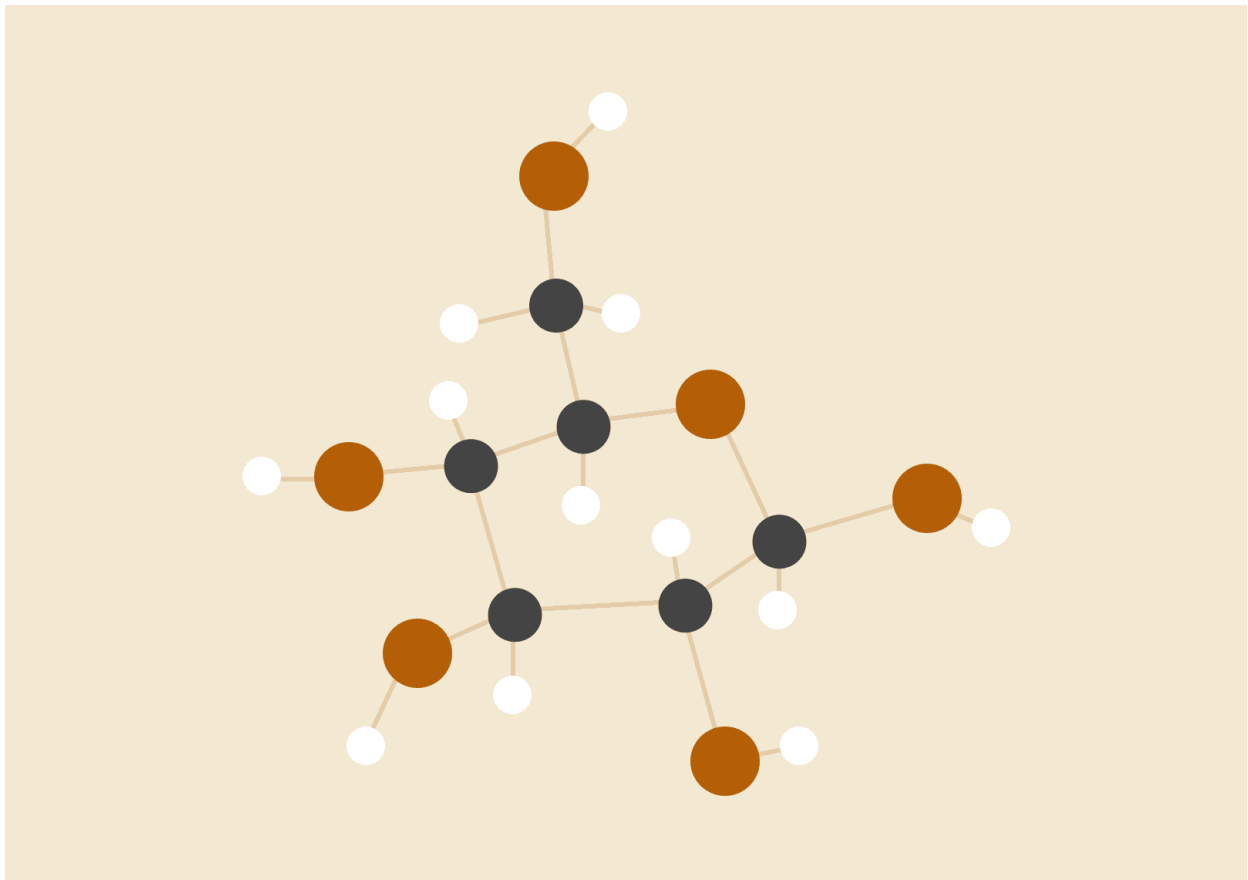


BUSINESS MANAGEMENT SYSTEM



Sonny Pritesh

Candidate Number - 2332

A-Level Computer Science

ANALYSIS	6
Outlining the Problem	6
Target Audience	6
Solution Details	7
User Requirements	7
Research	7
Research into Communication and Organisation	7
Research into User Interface	8
Potential Future Problems	9
Criteria and Objectives	10
DOCUMENTED DESIGN	12
GUI Designs	12
Login Screen	13
Welcome Page	13
Main Screen Layout	13
Inbox Page Layout (Version 1)	14
Inbox Page Layout (Version 2)	15
Transactions Page Layout	16
Reports Page Layout	17
Calendar Page Layout	18
Database Diagram (Entity Relationship Diagram)	19
Flow Charts	20
User Login Flowchart	20
Layout Flowchart	21
Logout Flowchart	22
Inbox Page Flowchart (Version 1)	23
Inbox Page Flowchart (Version 2)	24
Reports Page Flowchart	25
Model Algorithms	26
User Login Algorithm	27
User Logout Algorithm	27
TECHNICAL SOLUTION	27
Database	28
Database Diagram	28
LoginInfo Table	28
UserInfo Table	29
Events Table	30

Income Table	31
Expenditure Table	32
Stock Table	33
Supplier Table	34
System.Data.SQLite Reference	35
DatabaseConnection Class	36
Imports	36
getConnection Function	37
getConnVar Function	37
createDatabase Function	37
LoadDatabase Class	39
Imports	39
Globalised Property	39
storedLoad Function	40
updatedLoad Function	40
FormOpener Class	41
Login Page	42
Imports	42
Globalised Properties	42
Login Form	43
checkAccount Function	43
correctInfo Function	44
LoginButton_Click Event Handler	45
RegisterButton_Click Event Handler	46
Error Message	47
Error Form	47
Code	47
Register Page	48
Imports	48
Globalised Property	48
Register Form	48
AddFullName Function	49
AddLoginInfo Function	49
checkLoginInfo Fucntion	50
RegisterButton_Click Event Handler	51
Welcome Message	52
Welcome Form	52
Imports	52
ContinueButton_Click Event Handler	52

Welcome_Page_Load Event Handler	53
Main Window	54
Form Showing Events Window	54
Form Showing Transactions Window and Income Tab	54
Form Showing Transactions Window and Expenditure Tab	55
Form Showing Transactions Window and Stock Tab	57
Form Showing Reports Window and Income Tab	58
Form Showing Reports Window and Expenditure Tab	59
Imports	59
Income Class	60
Expenditure Class	60
StreamlineWindow Class	60
Globalised Properties	60
TransactionButton_Click Event Handler	60
EventButton_Click Event Handler	61
ReportsButton_Click Event Handler	61
StreamlineWindow_Load Event Handler	61
IncomeAddButton_Click Event Handler	62
RefreshButton_Click Event Handler	63
ExpenditureAdd_Click Event Handler	63
ExpenditureRefresh_Click Event Handler	63
StockRefreshButton_Click Event Handler	63
AddStockButton_Click Event Handler	64
SupplierButton_Click Event Handler	64
LogoutButton_Click Event Handler	64
IncomeChartRefreshButton_Click Event Handler	64
ExpenditureChartRefreshButton_Click Event Handler	65
RefreshEventButton_Click Event Handler	66
AEButton_Click Event Handler	67
Add Income	67
Form	67
Imports	67
Code	68
Add Expenditure	69
Form	69
Imports	69
Code	69
Add Stock	71
Form	71

Imports	71
Globalised Properties	71
Form Initialiser	72
DBValues Function	72
GetNextItemToDisplay Function	73
RestartEnumeration Function	73
NextButton_Click Event Handler	73
AddButton_Click Event Handler	74
Add-Delete Event	75
Form	75
Imports	75
Globalised Property	75
AddMonthCalendar_DateChanged Event Handler	76
DeleteMonthCalendar_DateChanged Event Handler	76
AddButton_Click Event Handler	76
DeleteButton_Click Event Handler	77
Supplier Page	77
Form	78
Imports	78
Globalised Property	78
Suppliers_Load Event Handler	78
AddSupplierButton_Click Event Handler	78
RefreshButton_Click Event Handler	79
TESTING	79
Test Plan	80
Evidence	85
Improvements Made	85
Changes to Code for Tests 9.1, 9.2, 9.3 & 9.7	85
Changes to Code for Test 20.1 & 20.2	87
Changes to Code for Test 26	88
Changes to Code for Test 31	91
Changes to Code for Test 35	92
Changes to Code for Test 48	94
Testing	95
Evidence	97
EVALUATION	98
Objective Analysis	98
Possible Improvements	99

ANALYSIS

Outlining the Problem

I am investigating the uses of business management systems now and how they can be more streamlined so that the program (system) has a much simpler design with the main core features. Therefore I have decided to call the application “Streamline” as it represents a simpler, more streamlined program.

Within the business management discipline, the term “Business Management System” is used to describe the high-level tools for strategic business planning and implementation. This term gives a description of the foundation for initiating business activities, making critical decisions, introducing business solutions, and employing business tactics.

The major idea of a business management system is to provide management staff with tools for planning, monitoring and controlling management activities and measuring business performance, and to implement continuous improvement processes within an organization.

I believe the main core features of a business management system is:

- The ability to stay organised and to communicate within the business and outside the business
 - to be able to set up meetings with other employees
 - to be able to add important dates to the calendar
- The ability keep track of transactions within the business
 - to have the ability to store all the details of the business’ transactions
 - Such as income, profit/loss, stock, sales
 - to have the ability to use the transaction detail and produce a visual representation of it
 - From sales reports, stock reports etc.
 - to allow the user to be able to manipulate and predict future transactions
- to have the ability to store the details of multiple users
- to have the ability to give users roles based on the authority allowing them access to required parts of the system

Target Audience

This investigation is aimed at a small start-up business that needs a simple business management

system due to them unable to cope with the already existing applications due to factors such as bad user interface and complex features causing the user experience to be unsatisfactory for the business' employees.

Solution Details

For the investigation, I will be using a Visual Studio 2017 (VS) as my IDE. In VS, many languages are supported however I will be using the language C#. The reason for this is that I would not have to learn another language in order to produce the business management system. I would, however, have to learn SQL to allow my code to communicate with the database. I will, specifically, be using SQLite as this stores the data in file rather than server and I will use DB Browser for SQLite.

The system will be produced as a Graphical User Interface (GUI) therefore, in VS, I will be using Windows Forms to produce the GUI due to its simplicity and ease of use. The system would not require any extra hardware other than the typical monitor, mouse and keyboard.

User Requirements

The user would require the following hardware and software to be able to use the business management system:

- The user would require a device to allow the system to run on; a computer or laptop
- The device should be running on the intended software; Windows
- The user would require a device the business information or login details; a keyboard or a touchscreen keyboard depending on their device

Research

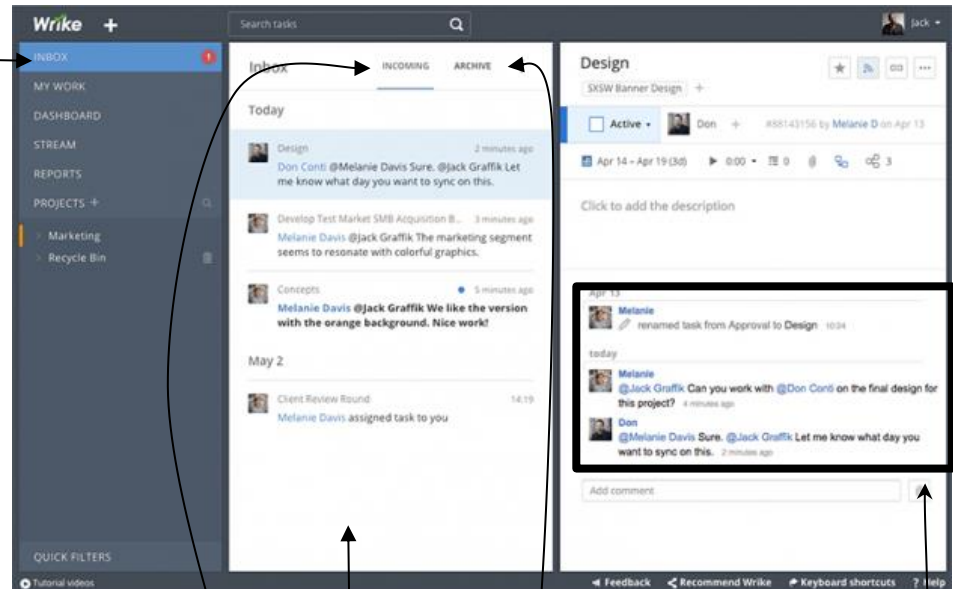
Research into Communication and Organisation

Wrike is a management software that facilitates to project management and team collaboration within

the business.

Below shows the inbox page from the Wrike application

The inbox page from Wrike show

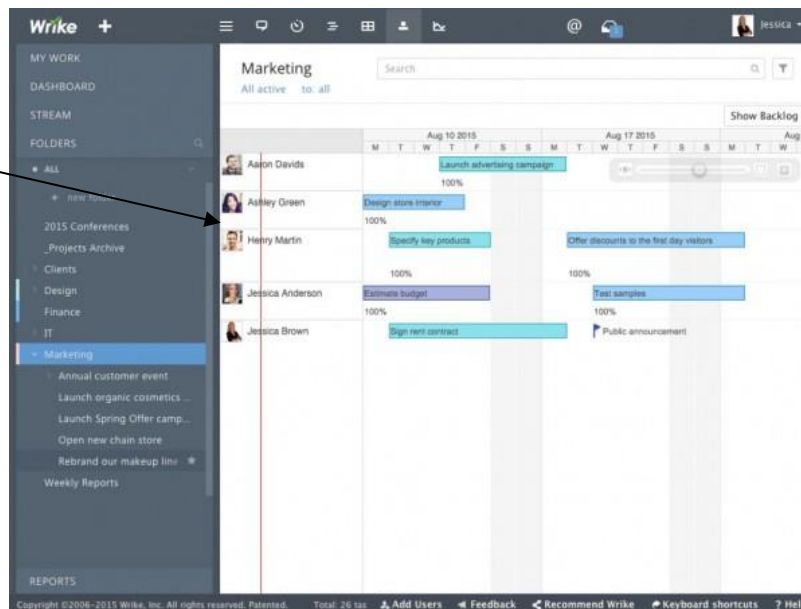


The inbox page itself is

The inbox section, where that is split into

Next to the inbox section is the viewing section, where

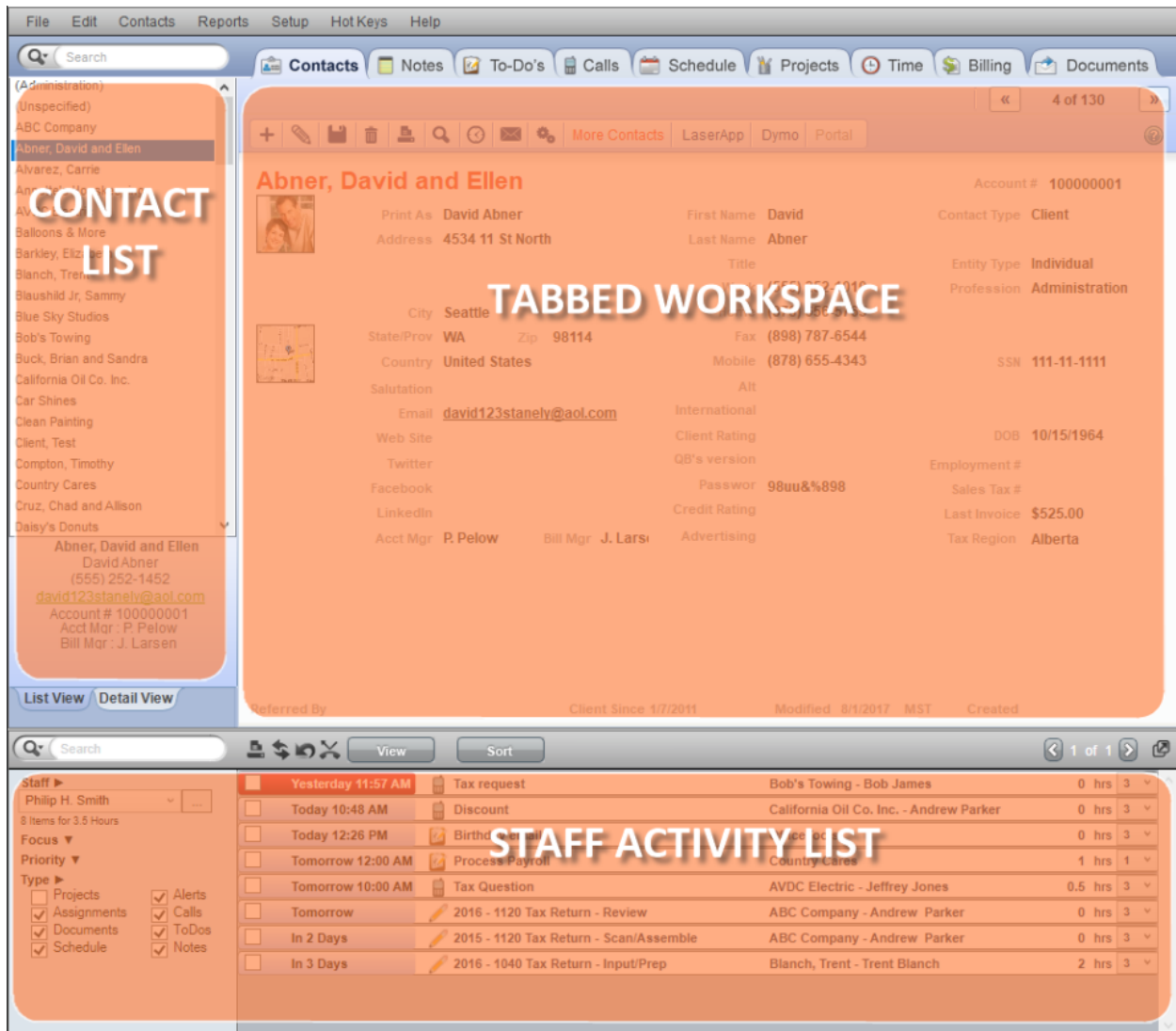
The image shows how an employee is able to put together a chart of



Research into

User Interface

Below shows how Office Tools has laid out their application for their users.



WorkSpace's main screen is split into three parts: 1) the Contact List along the left side; 2) the Tabbed Workspace in the center, which displays contextual information about the selected contact according to the chosen tab; 3) the Staff Activity List along the bottom, showing all tasks assigned to a given staff member like an inbox for work assignments.

Potential Future Problems

There may be many limitations to my project. These may be in the form of back-end code, front-end GUI, database issues or hardware issues.

Limitation/Potential Problem	Description of the problem and how I will potentially overcome the problem.
Time	One of the main limitations is the time restriction. I will overcome this restriction by organising and planning future task so that it can be completed within the time frame and in such a manner that it meets the core objectives and criteria.
GUI	As stated above, the system will be designed as a GUI. The problem from element restrictions, i.e. some of the design elements may not work as required. I will try to overcome this problem by having simple to use features that only have limited functionality. However, if the problem keeps occurring then I would have to remove the feature and produce the system without it.
Complex Code	Another problem would be the complexity of the code when producing the system. The system may have lots of bugs and I would have to find a way to fix those bugs. However, if I am unable to find a fix for the bug, I may have to remove the feature from the system depending on the feature; if it is a minor or major feature.
Linking Code to Database	This is also one of the main limitations for producing the BMS. The system would require a database to store all the details required to use the system, e.g. login details, user roles, calendar dates, transaction details. The language I will use will be SQL, so to overcome the problem is by learning SQL in my spare time.

Criteria and Objectives

The business management system should be able to:

- store all the details of the business' transactions
- use the transaction detail and produce a visual representation of it
- store details of multiple users
- give users roles based on their authority allowing them access to required parts of the system
- Communicate with other employees within the business
- allow the user to add important dates to the calendar

The business management system should overall:

- Be displayed as a GUI
- Startup with a login screen

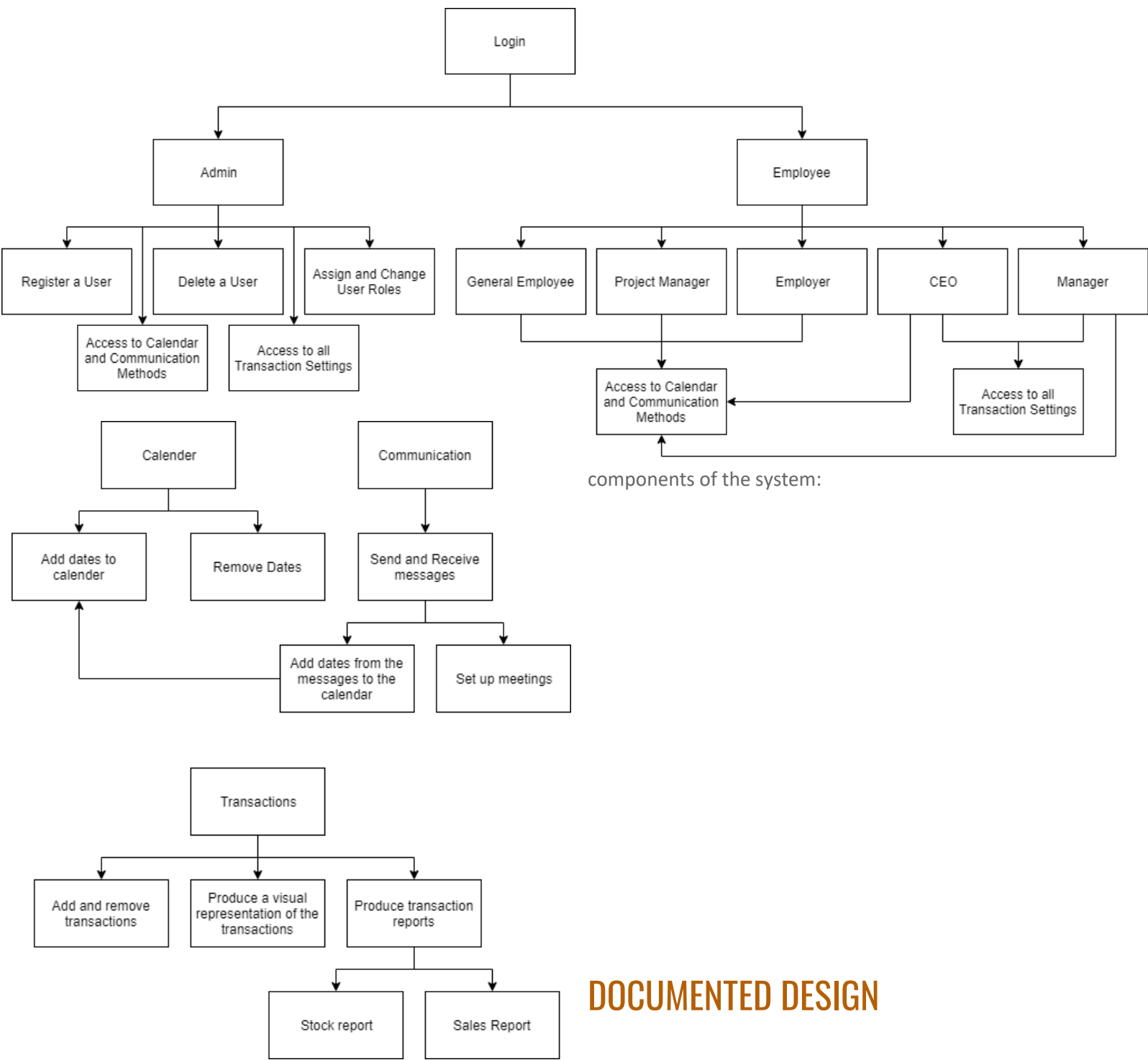
- Stay at a fixed resolution

Specifically, the business management system should do the following:

- Startup with a login screen
 - Login as an employee
 - Each employee is set a certain role by the admin (*covered below*)
 - Login as admin
 - Control over all the setting in the system (*covered below*)
- Admin login
 - Register a new employee
 - Give them access to the required parts of the system
 - Remove an employee's login
 - Change the role of the employee
 - Their access to certain to parts of the system is either granted or denied
- Employee login (*these are just example roles and they can be altered by the admin, if they wish to*)
 - General Employee has access to
 - Communication Methods
 - Calendar
 - Project Manager has access to
 - Communication Methods
 - Calendar
 - The employer has access to
 - Communication Methods
 - Calendar
 - CEO has access to
 - Communication Methods
 - Calendar
 - All transaction setting
 - The manager has access to
 - Communication Methods
 - Calendar
 - All transaction settings

The diagram below shows what the user will be able to gain access to when they login, from the information above:

The diagram to the left shows what the user can do, based on their role, in each of the main

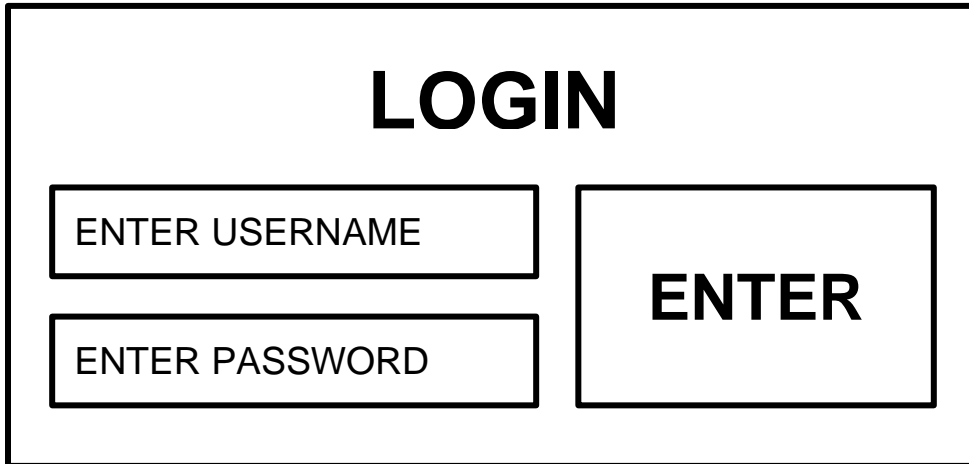


DOCUMENTED DESIGN

GUI Designs

The entire BMS would consist of front-end GUIs and backend code to allow the GUIs to function and to allow communication to the database

Login Screen

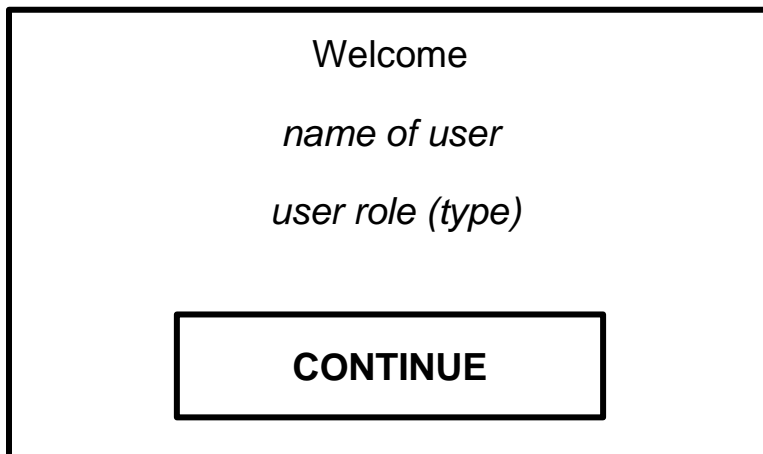


A login screen UI mockup. At the top center is the word "LOGIN" in a large, bold, black sans-serif font. Below it are two input fields: the first is labeled "ENTER USERNAME" and the second is labeled "ENTER PASSWORD", both in a smaller, bold, black sans-serif font. To the right of these fields is a large rectangular button labeled "ENTER" in a bold, black sans-serif font. The entire login screen is enclosed in a black rectangular border.

This is the login screen that the user will see once the application is opened.

All the user has to do is enter their username and password to login

Welcome Page



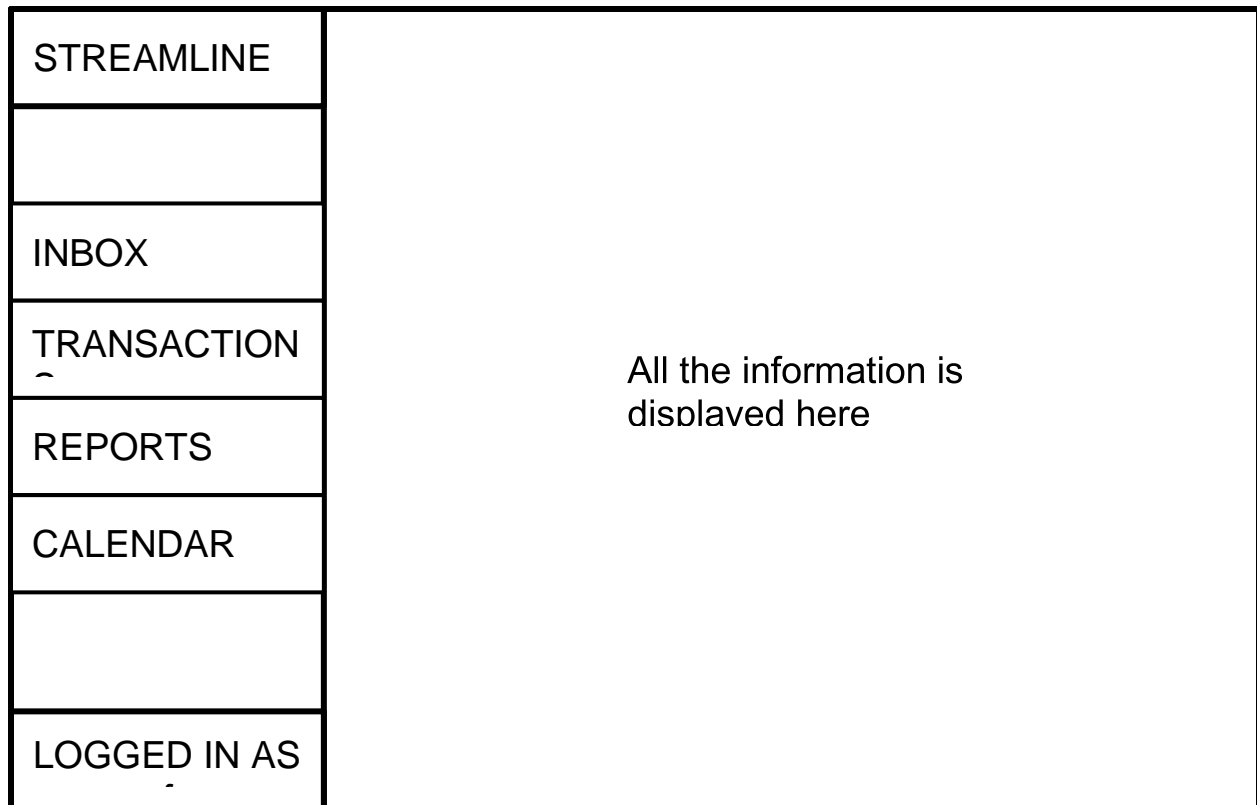
A welcome page UI mockup. At the top center is the word "Welcome" in a bold, black sans-serif font. Below it are two lines of text: "name of user" and "user role (type)", both in an italicized black sans-serif font. At the bottom center is a rectangular button labeled "CONTINUE" in a bold, black sans-serif font. The entire welcome page is enclosed in a black rectangular border.

This is the welcome screen, which is shown to the user once they have logged.

It will show their name and their role within the business.

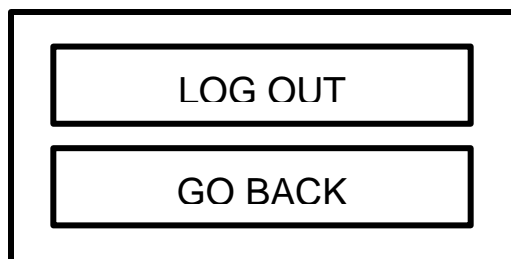
Then they can continue to the main screen.

Main Screen Layout



Above is the base layout of the application. The user is able to navigate through the system via the left-hand side column where they are able to access the inbox, transactions, reports and calendar pages, depending on the role in the business.

Also, the user is able to log out by clicking at the bottom, where it says “LOGGED IN AS ...”, where they will be shown a pop up shown below. If the user clicks on the “LOG OUT” button then they will be



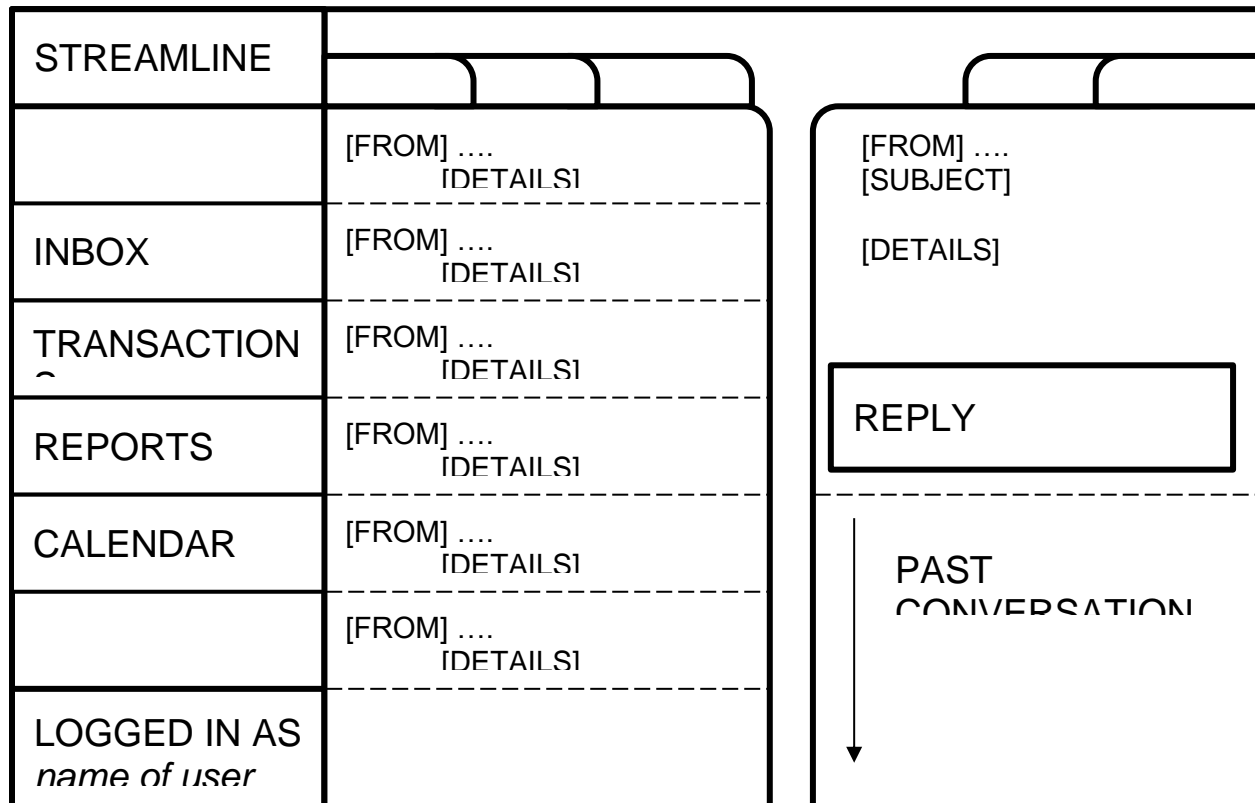
logged out of the system and they will be sent back to the login screen.

However, if they click on the “GO BACK” button then the pop up will disappear and they will stay on the page they were on.

Inbox Page Layout (Version 1)

When the user has logged in, they will always be taken to their inbox. There two main section. The left-

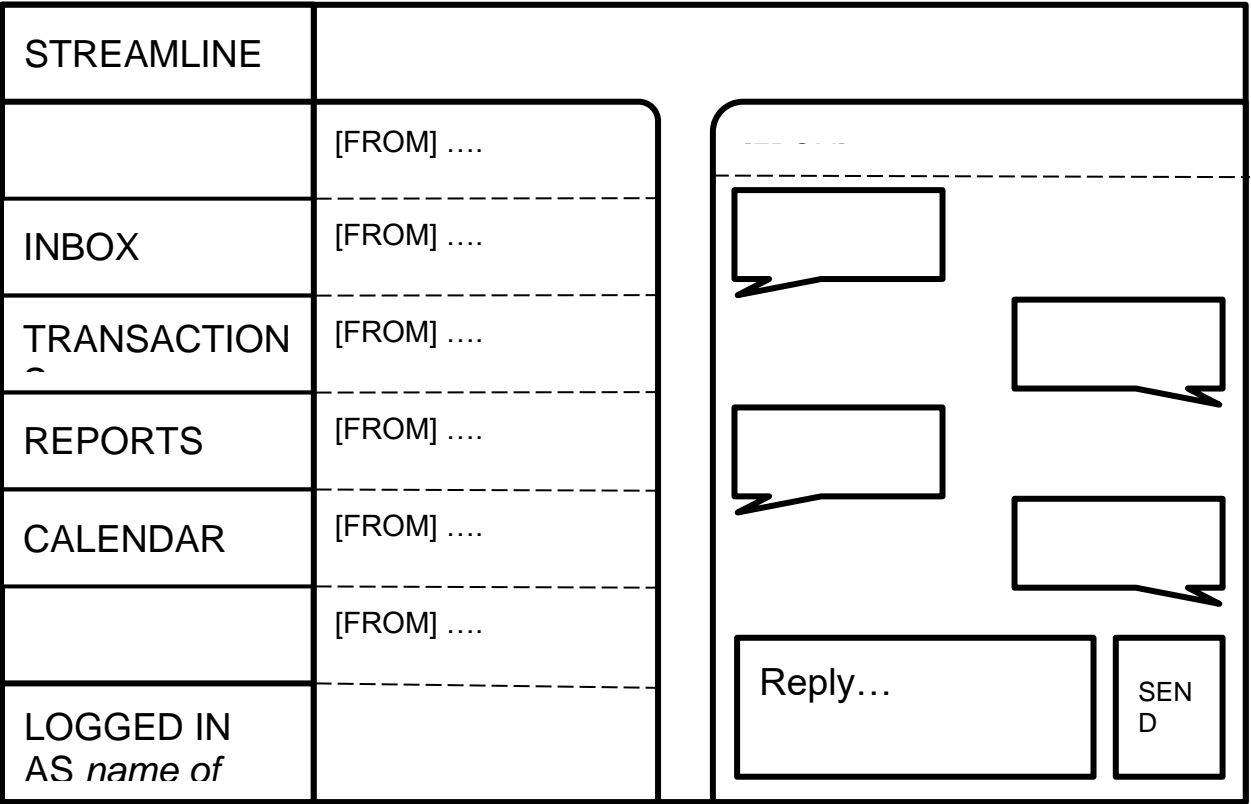
hand side section is for received messages and the right-hand side is for opening those messages. I have taken inspiration from the OfficeTools design and implemented tabs into my design, as seen from the top of the two main sections. I have also taken inspiration from Wrike in creating a two column inbox section.



Inbox Page Layout (Version 2)

However, if I am unable to produce the inbox page (version 1), I may change the email inspired system into a messaging inspired system potentially due to the complexity of the system. As seen below, it

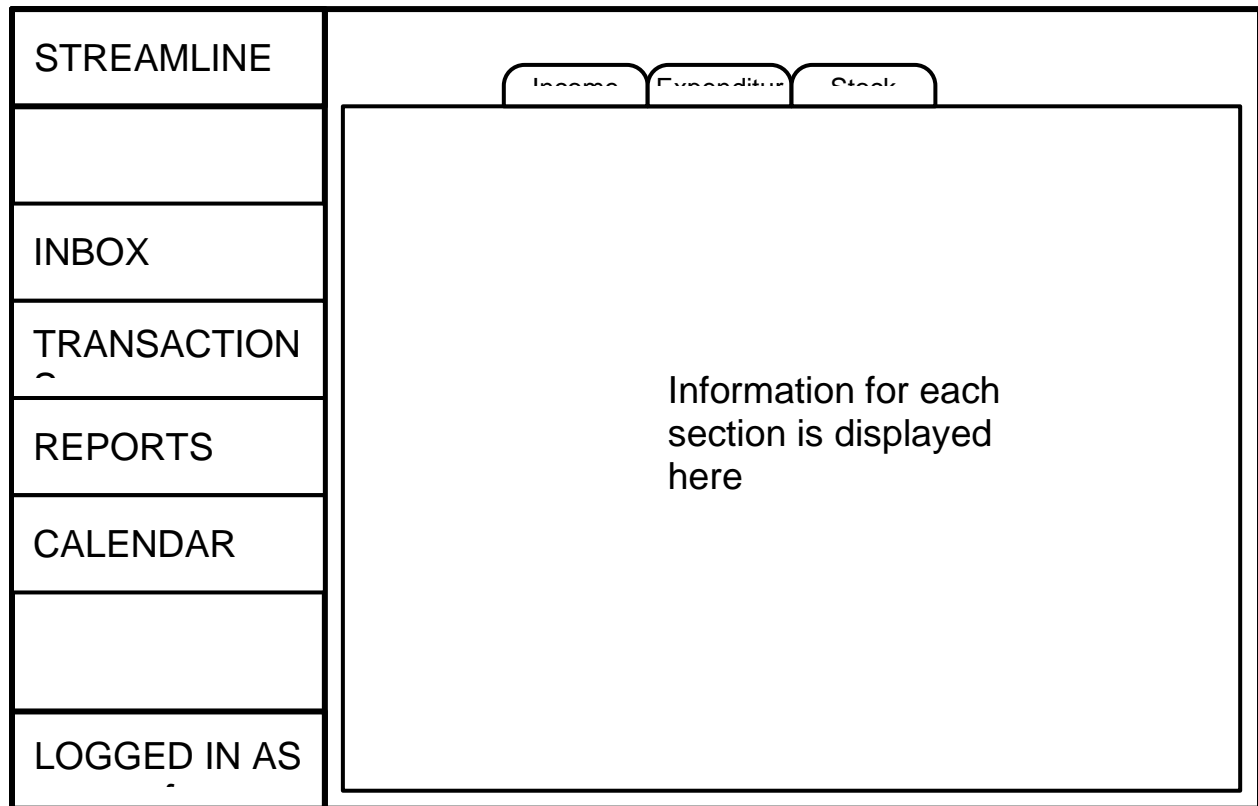
looks a lot simpler than the above idea as it will just show the who the user is messaging on the left and their chat on the right



Transactions Page Layout

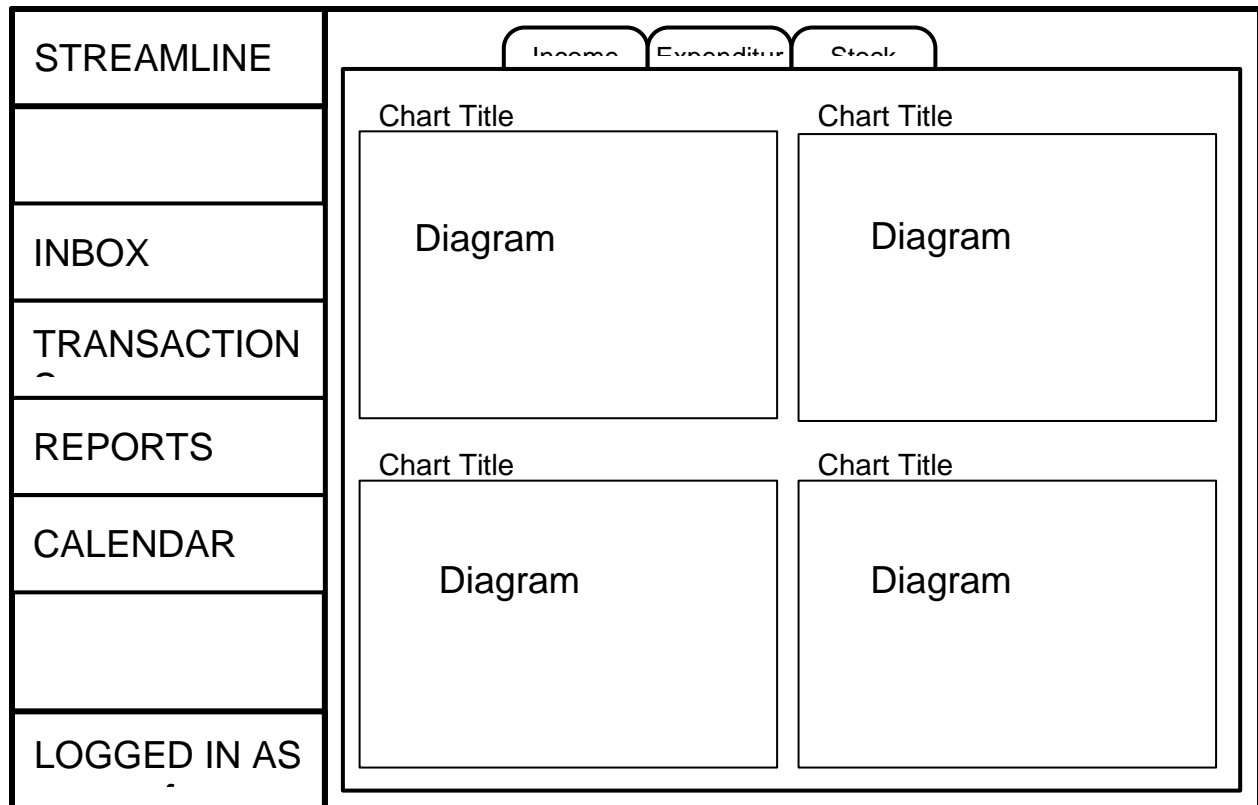
When the user chooses the transactions page, they will show with the page below. Overall there is a main section for all the information and above it is all the tabs that take the user to each of the different

sections, income, expenditure and stock. I have taken inspiration from the OfficeTools design and implemented tabs into my design, as seen from the top of the main section of the page.



Reports Page Layout

The below diagram is the potential layout for the reports page for the BMS. Each section will have its own reports section, e.g. income will have its reports separate to expenditure. I have taken inspiration from the OfficeTools design and implemented tabs into my design, as seen from the top of the page.



Calendar Page Layout

This is the potential layout of the calendar page, as seen below, at the top, it shows the days and bellow are the dates on each day. Below each date shows the events that take place on each day.

STREAMLINE	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY	SATURDAY	SUNDAY
	1	2	3	4	5	6	7
INBOX	8	9	10	11	12	13	
TRANSACTION							
REPORTS	15	16	17	18	19	20	
CALENDAR							
	22	23	24	25	26	27	
LOGGED IN AS	<div> <div></div> <div></div> <div></div> </div>						

If there are multiple events on one day and not all of them are visible then the user can click on the date and the pop up on the left would show to the user with the date written at the top and the event written just below; with the time of each event at the start.

Once the user is done with viewing all the events in detail, they can press the back button to go back to the calendar page.

Monday 01 January 2018

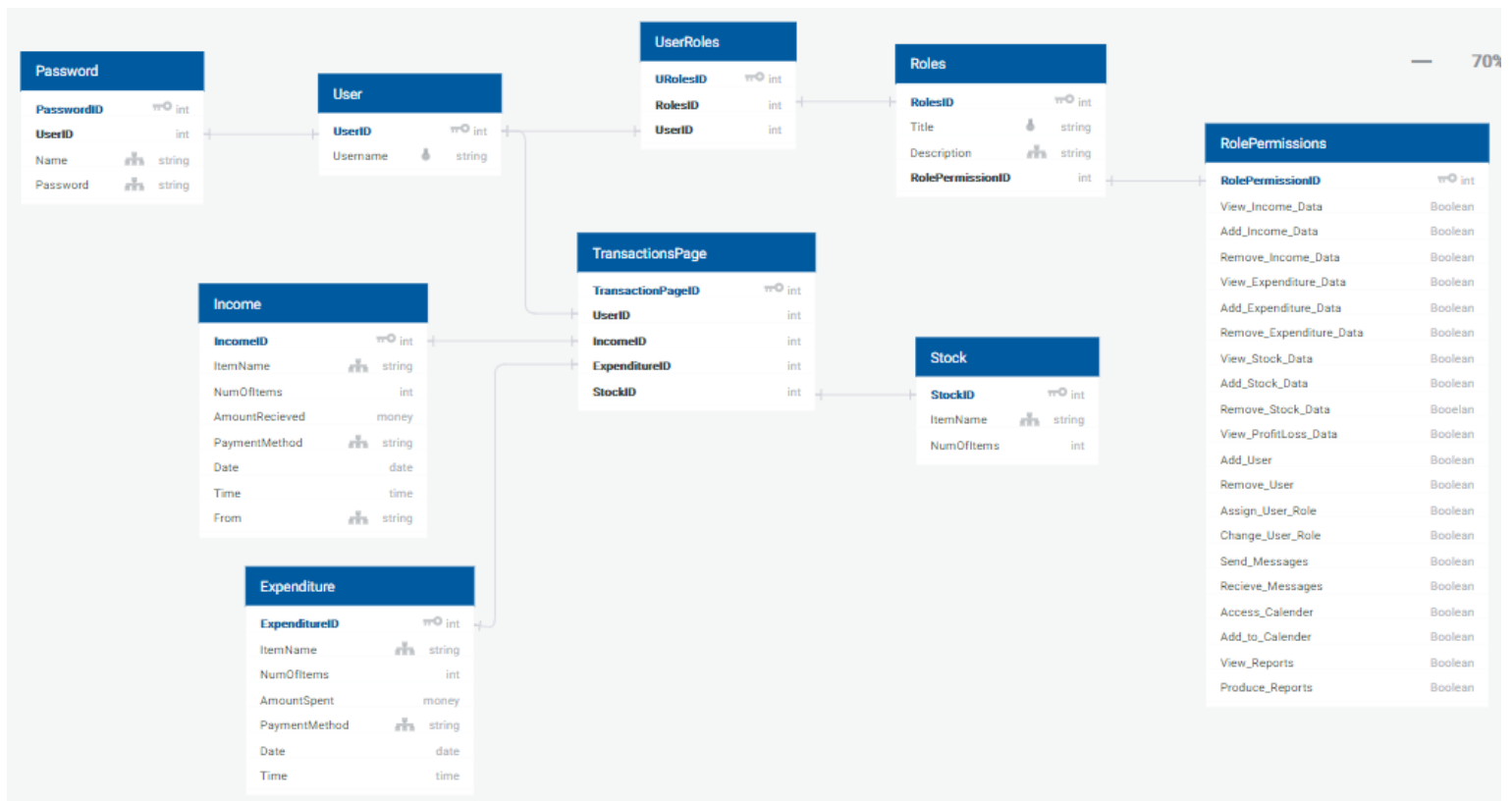
Events:

Time, event details

Time event

Database Diagram (Entity Relationship Diagram)

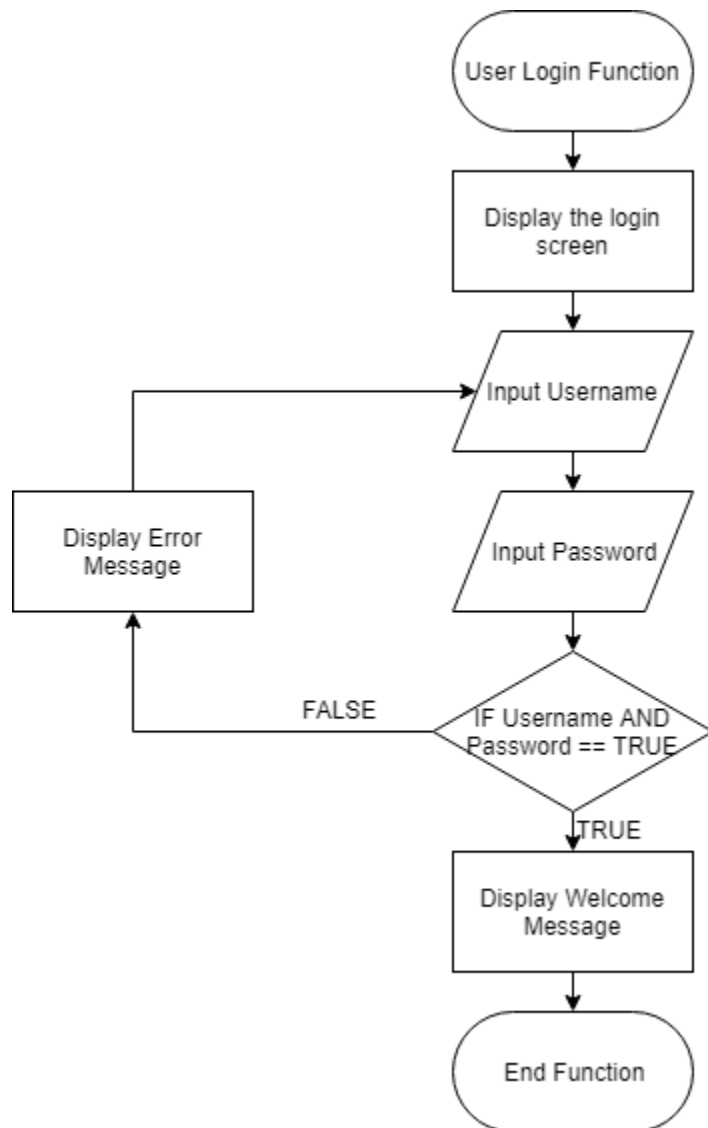
The diagram shows how the database will be laid out and how each table is related to each other.



Flow Charts

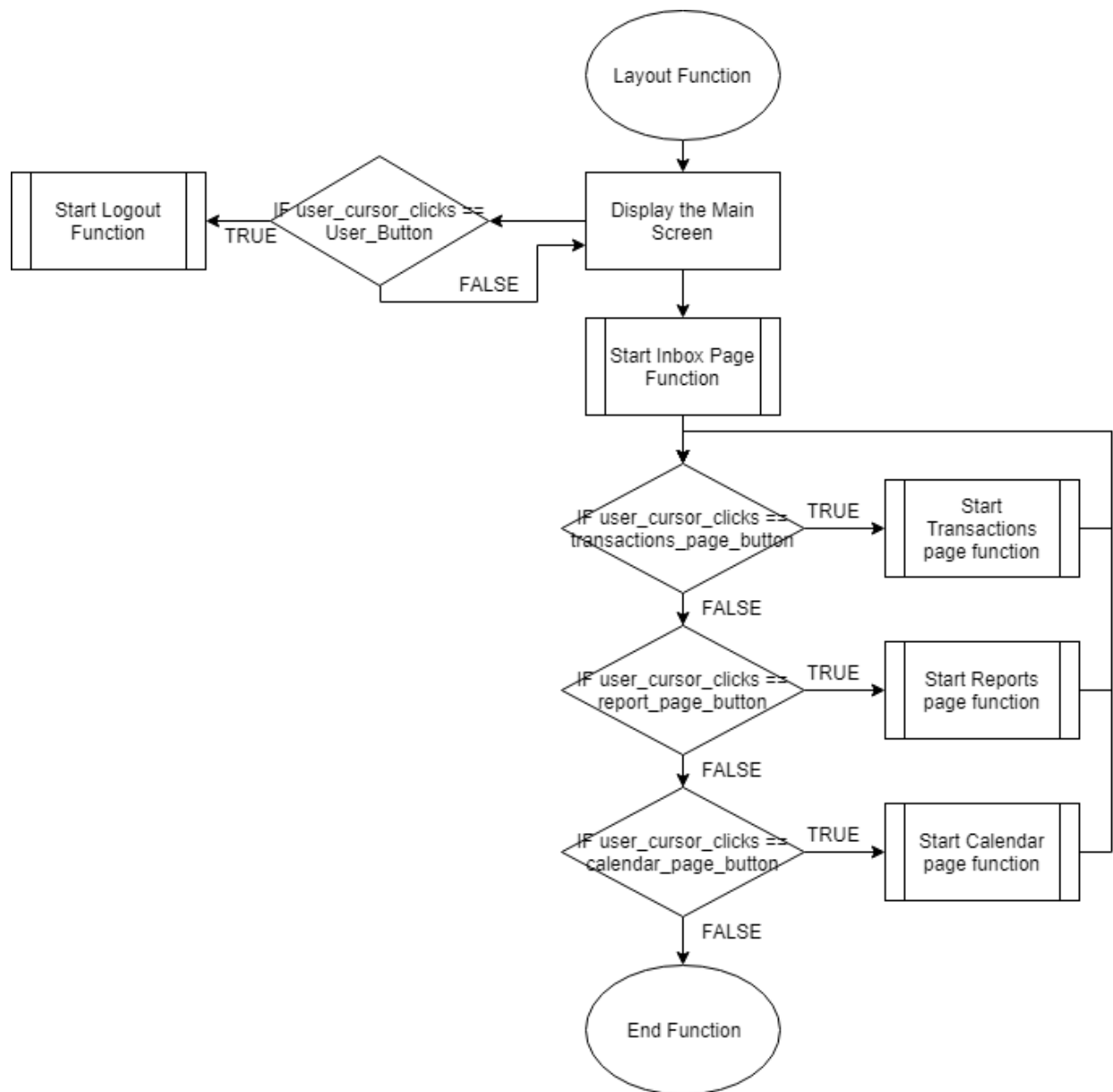
User Login Flowchart

The flowchart below shows the user login function and how the function should work

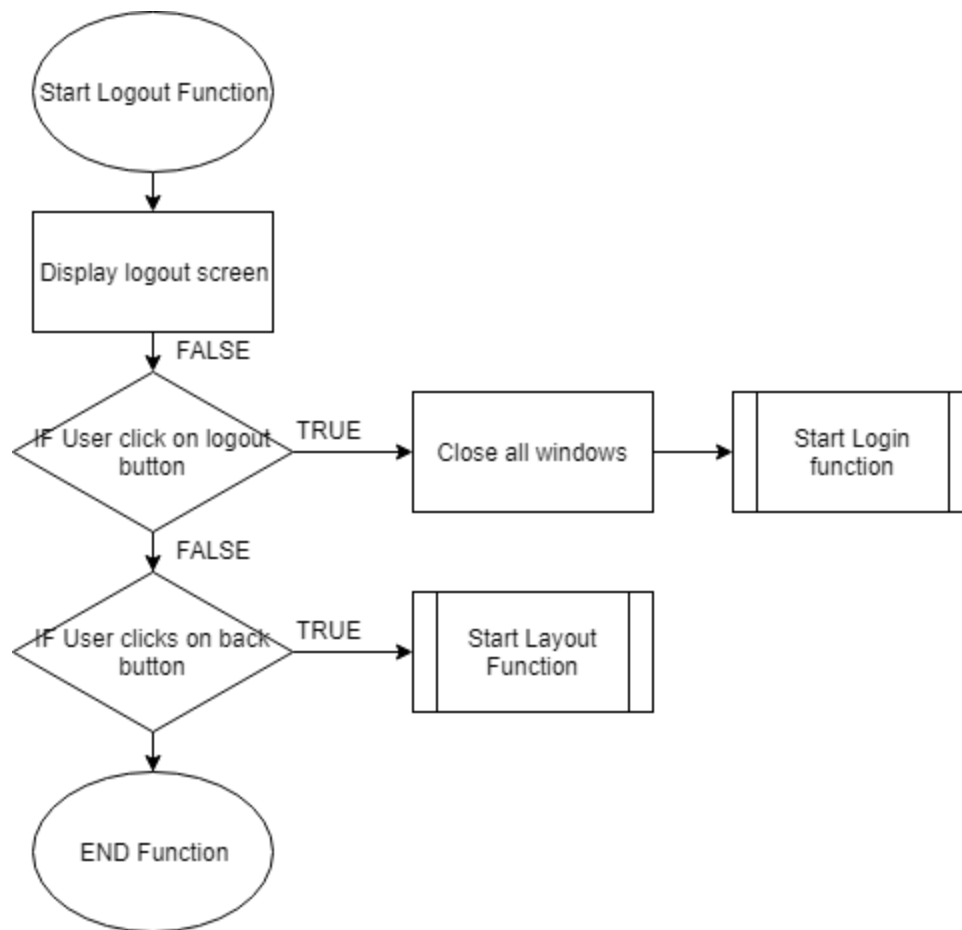


Layout Flowchart

The flowchart below shows how the main section of the system will work and how the different function connect together

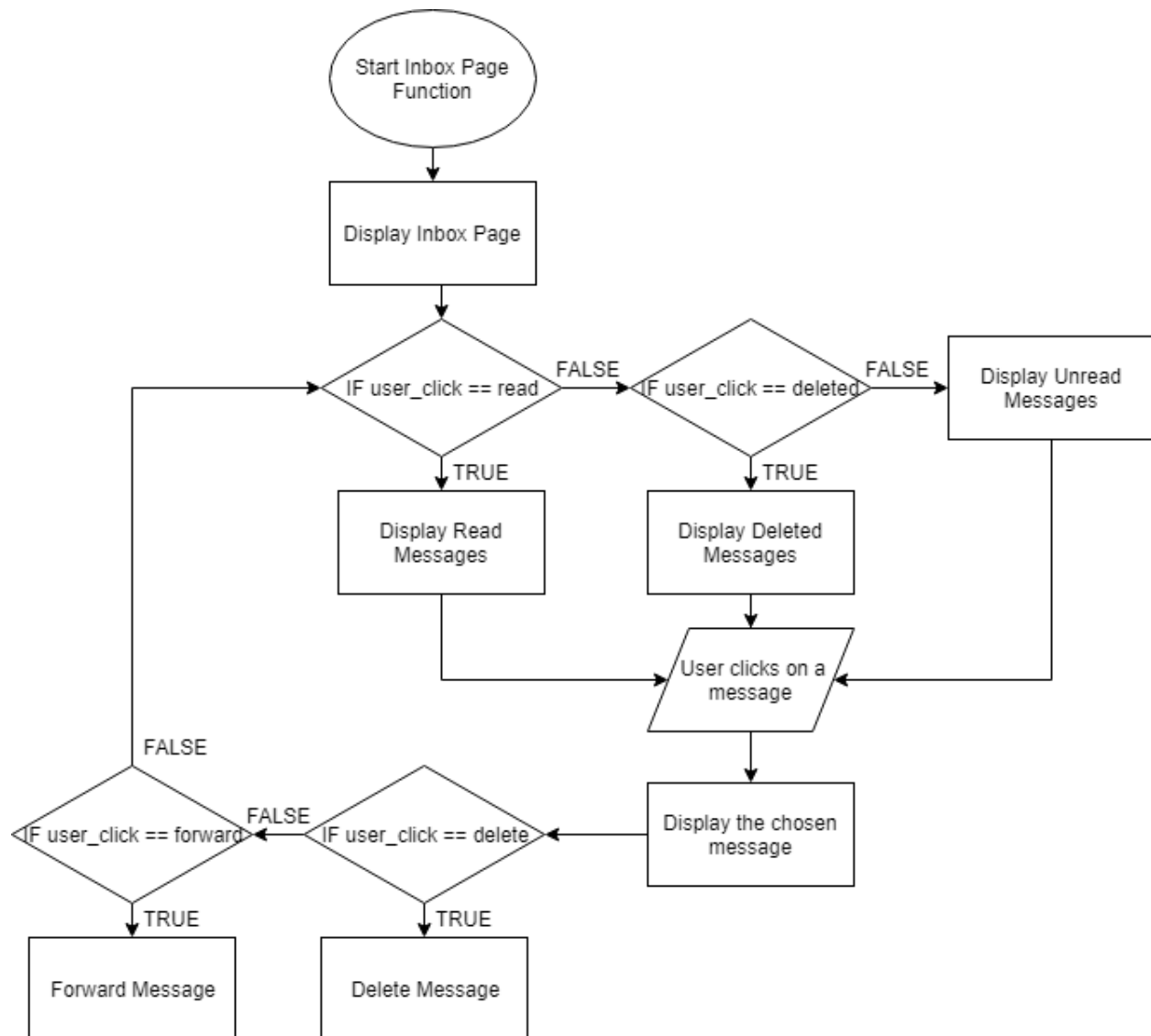


Logout Flowchart

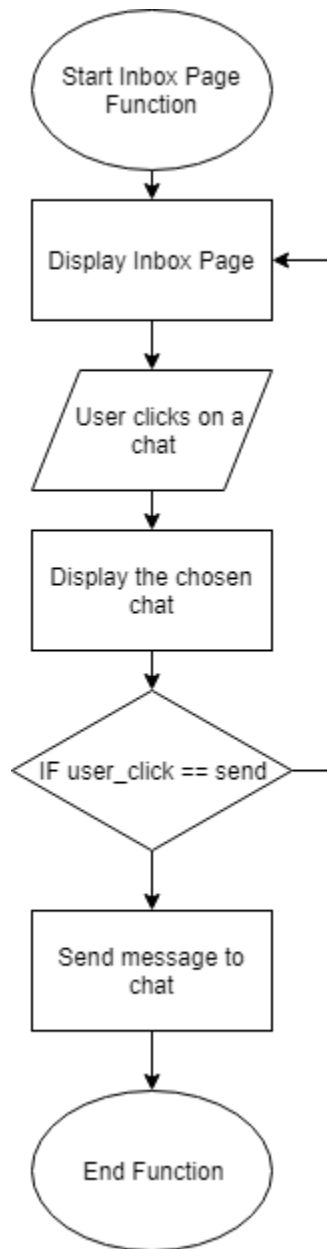


Inbox Page Flowchart (Version 1)

This flowchart corresponds to the Inbox Page Layout (Version 1).



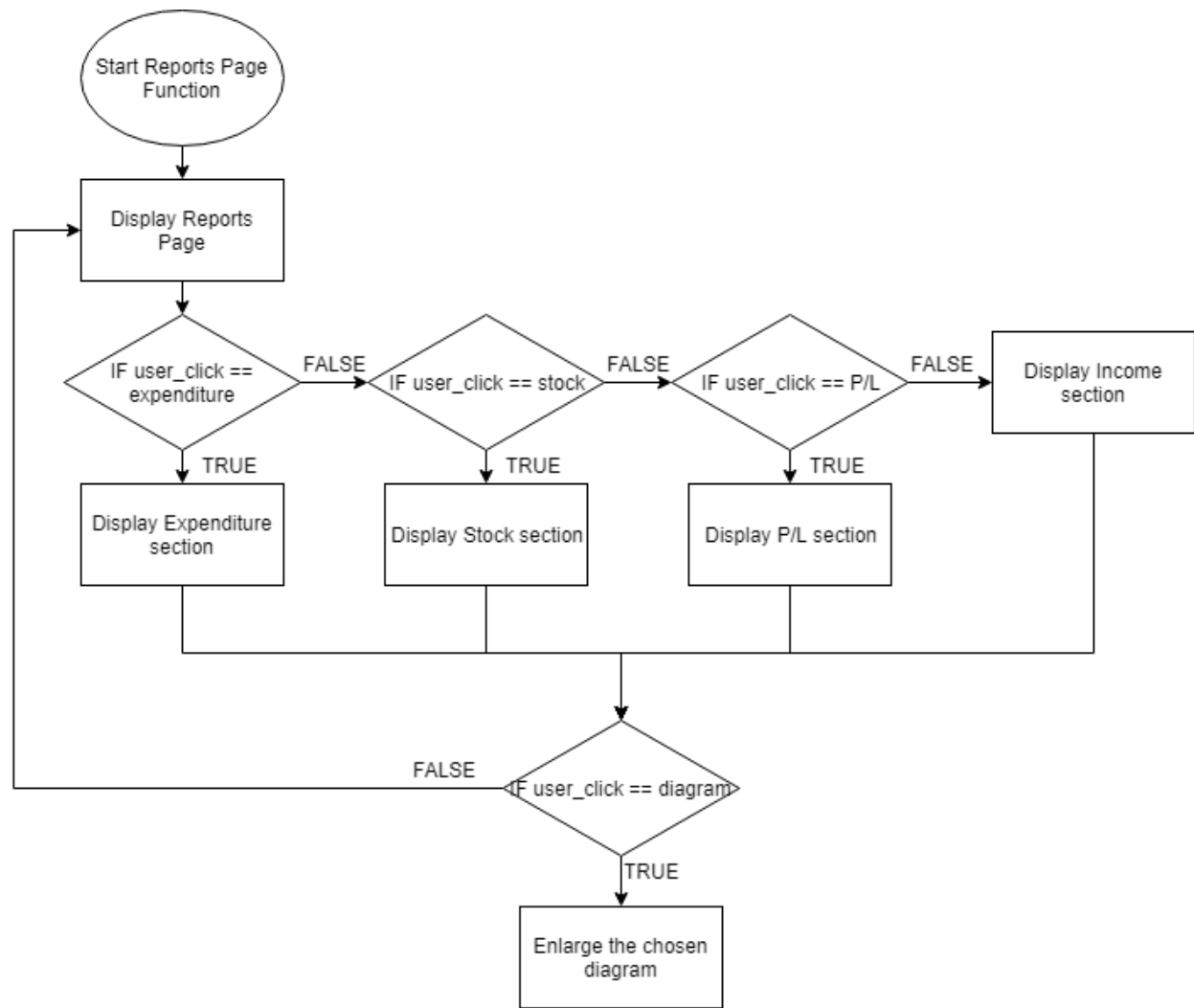
Inbox Page Flowchart (Version 2)



The flowchart (on the left) corresponds to the Inbox Page Layout (Version 2).

As you can see, it is a much simpler flowchart than version 1, as it is based on the Inbox Page Layout (Version 2).

Reports Page Flowchart



Model Algorithms

User Login Algorithm

I will be needing an algorithm to allow the users to login to their accounts within the system, allowing them to fully utilize the BMS.

Pseudocode:

```
Function UserLogin()
    username ← USERINPUT
    password ← USERINPUT
    REPEAT
        IF (username AND password are TRUE) THEN
            display welcome page
        ELSE
            display error message
        ENDIF
    UNTIL username AND password are TRUE
END FUNCTION
```

User Logout Algorithm

I will be needing an algorithm to allow the users to logout of their accounts.

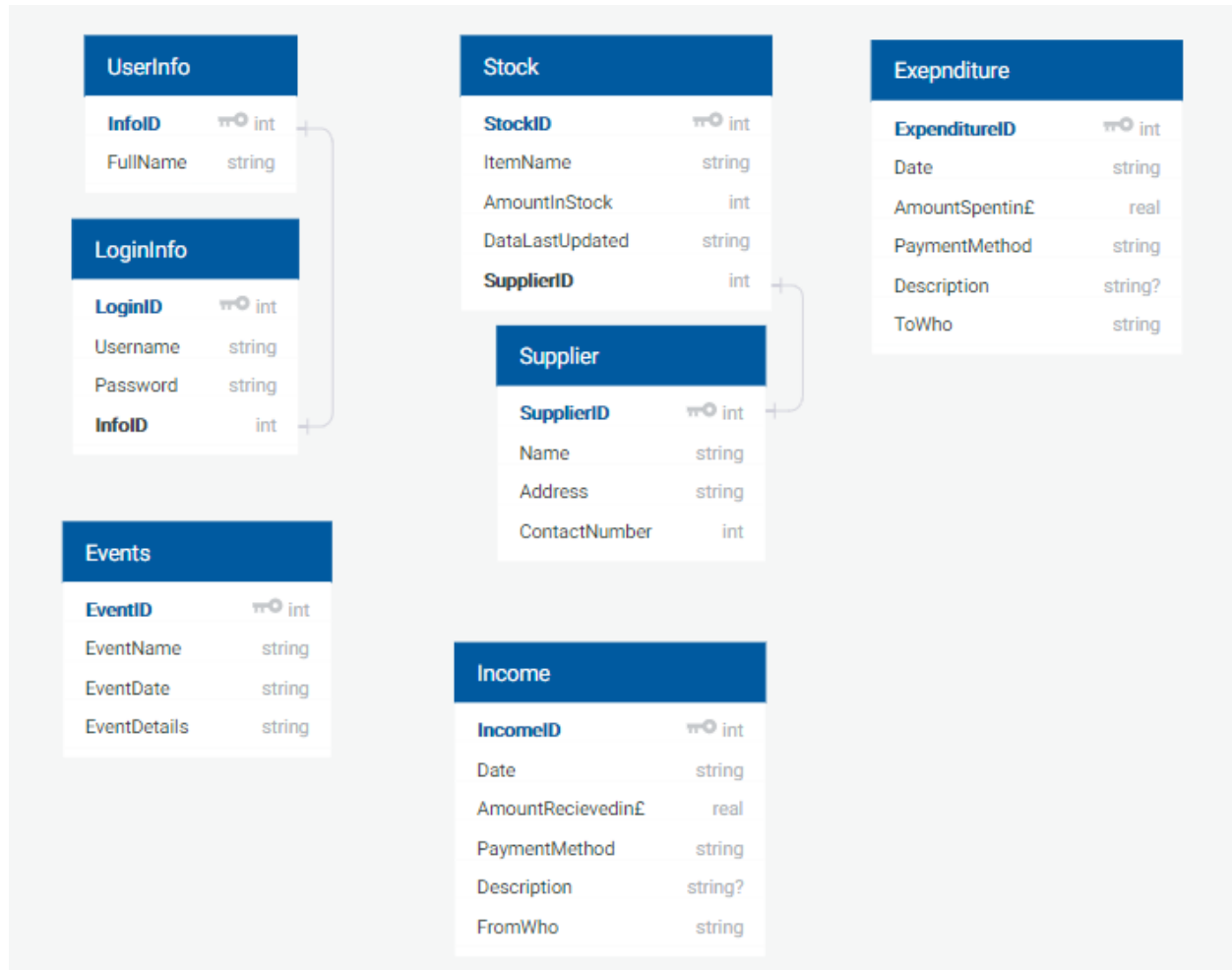
Pseudocode:

```
Function UserLogout()
    IF (User clicks on options button) THEN
        DISPLAY Logout Form
        IF (User clicks on logout button) THEN
            close all forms AND display Login Form
        ELSE IF (User clicks on back button) THEN
            Close current Form
        ENDIF
    ENDIF
END FUNCTION
```

TECHNICAL SOLUTION

Database

Database Diagram



LoginInfo Table

This is the LoginInfo table within the database. This table contains 4 columns. The first one is for the primary key. Then the next two is for the username and password. The last is for the foreign key which is

the primary key in the UserInfo table.

Table

LoginInfo

▼ Advanced

Fields

Add field

Remove field

Move field up

Move field down

Name	Type	Not	PK	AI	U	Default	Check	Foreign Key
LoginID	INTEGER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
Username	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			
Password	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			
InfoID	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			'UserInfo'('InfoID')

```
1 CREATE TABLE `LoginInfo` (  
2     `LoginID`    INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,  
3     `Username`   TEXT NOT NULL,  
4     `Password`   TEXT NOT NULL,  
5     `InfoID`     INTEGER UNIQUE,  
6     FOREIGN KEY(`InfoID`) REFERENCES `UserInfo`(`InfoID`)  
7 );
```

UserInfo Table

This is the UserInfo table within the database. This table has two columns. One is for the primary key which is the foreign key in the LoginInfo table. The other is for the full name.

Table

UserInfo

▼

Advanced

Fields

Add field

Remove field

Move field up

Move field down

Name	Type	Not	PK	AI	U	Default	Check
InfoID	INTEGER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
FullName	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		

<

>

```

1 CREATE TABLE `UserInfo` (
2     `InfoID`    INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
3     `FullName`  TEXT NOT NULL
4 );

```

<

>

Events Table

This is the Events table within the database. The table has four columns. The first one is for the primary key. The next three are for the name, date and details of the event.

Table

Events

▼ Advanced

Fields

Add field

Remove field

Move field up

Move field down

Name	Type	Not	PK	AI	U	Default	Check
EventID	INTEGER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
EventName	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
EventDate	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
Details	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		

<

>

```

1 CREATE TABLE `Events` (
2     `EventID`    INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
3     `EventName`  TEXT NOT NULL,
4     `EventDate`  TEXT NOT NULL,
5     `Details`    TEXT NOT NULL
6 );

```

<

>

Income Table

This is the Income table within the database. This table has six columns. The first one is for the primary key. Then the rest are for the date, the amount received, the payment method, description, and who it is from. I have kept the description null as I have allowed this to be an optional input for the user.

Table

Income

▼ Advanced

Fields

Add field

Remove field

Move field up

Move field down

Name	Type	Not	PK	AI	U	Default	Check
IncomeID	INTEGER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
Date	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
AmountRecievedin£	REAL	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
PaymentMethod	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
Description	TEXT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
FromWho	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		

<

>

```

1 CREATE TABLE `Income` (
2     `IncomeID` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
3     `Date` TEXT NOT NULL,
4     `AmountRecievedin£` REAL NOT NULL,
5     `PaymentMethod` TEXT NOT NULL,
6     `Description` TEXT,
7     `FromWho` TEXT NOT NULL
8 );

```

<

>

Expenditure Table

This is the expenditure table within the database. This table has six columns. One for the primary key and the rest for the date, amount spent, payment method, description and who it is to. I have kept the description null as I have allowed this to be an optional input for the user.

Table

Expenditure

▼

Advanced

Fields

Add field

Remove field

Move field up

Move field down

Name	Type	Not	PK	AI	U	Default	Check
ExpenditureID	INTEGER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
Date	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
AmountSpentinf	REAL	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
PaymentMethod	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
Description	TEXT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		
ToWho	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		

<

>

```

1 CREATE TABLE `Expenditure` (
2     `ExpenditureID` INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
3     `Date` TEXT NOT NULL,
4     `AmountSpentinf` REAL NOT NULL,
5     `PaymentMethod` TEXT NOT NULL,
6     `Description` TEXT,
7     `ToWho` TEXT NOT NULL
8 );

```

<

>

Stock Table

This is Stock table within the database. This table has 5 columns; the first one is for the primary key, the next 3 are for the name of the item, the amount in stock and the date of when the data was last updated and last column is for the supplier id which is foreign key for the primary key in the Supplier table.

Table

Stock

▼

Advanced

Fields

Add field

Remove field

Move field up

Move field down

Name	Type	Not	PK	AI	U	Default	Check	Foreign Key
StockID	INTEGER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>			
ItemName	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			
AmountInStock	INTEGER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			
DataLastUpdated	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			
SupplierID	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			<code>`Supplier`(`SupplierID`)</code>

<

>

```

1 CREATE TABLE `Stock` (
2     `StockID`    INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
3     `ItemName`   TEXT NOT NULL,
4     `AmountInStock` INTEGER NOT NULL,
5     `DataLastUpdated` TEXT NOT NULL,
6     `SupplierID` INTEGER,
7     FOREIGN KEY(`SupplierID`) REFERENCES `Supplier`(`SupplierID`)
8 );

```

<

>

Supplier Table





This is the Supplier table within the database. The table has 4 columns; the first is for the primary key which also a foreign key in the Stock table and the next three are for the name, address and contact number of the supplier.

Table

Supplier

▼ Advanced

Fields

 Add field
  Remove field
  Move field up
  Move field down

Name	Type	Not	PK	AI	U	Default	Check
SupplierID	INTEGER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
Name	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
Address	TEXT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
ContactNumber	INTEGER	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		

```

1 CREATE TABLE `Supplier` (
2     `SupplierID`    INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
3     `Name`    TEXT NOT NULL UNIQUE,
4     `Address`    TEXT NOT NULL UNIQUE,
5     `ContactNumber` INTEGER NOT NULL UNIQUE
6 );

```

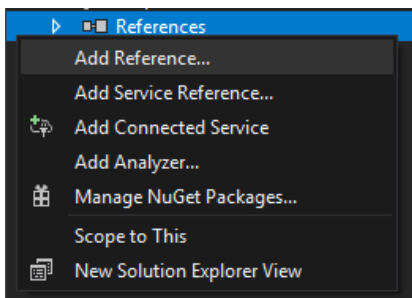
System.Data.SQLite Reference

To allow any of the SQL queries to work. I required an application extension. This was a

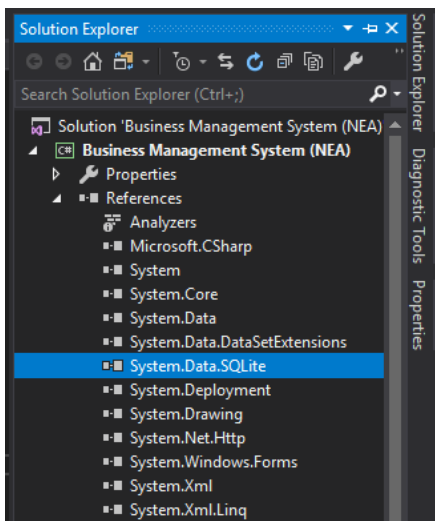
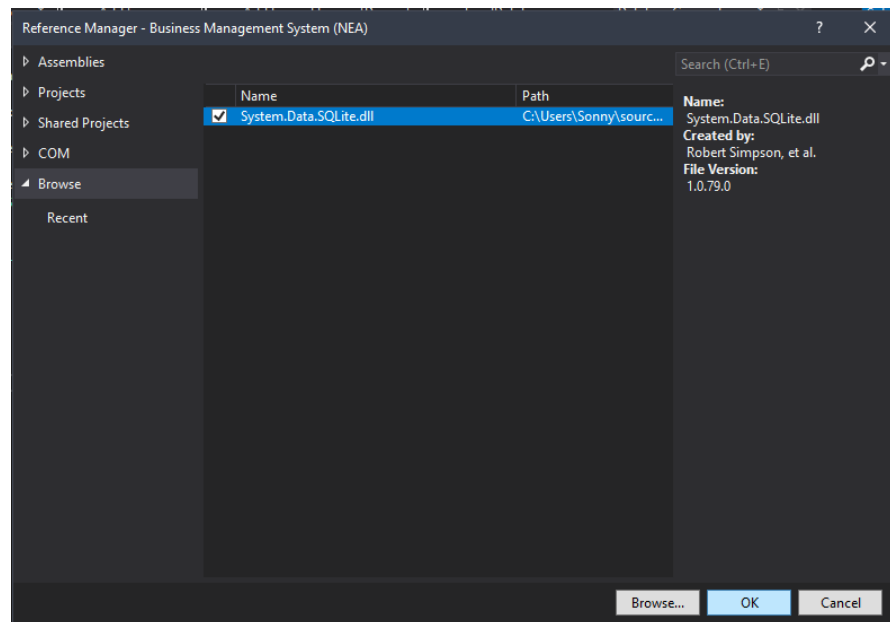
System.Data.SQLite.dll file.  **System.Data.SQLite.dll**

Application extension

I then added it to References in my solution in Visual Studio.



Which is then shown in
Solution Explorer



DatabaseConnection Class

Imports

```

using System.IO;
using System.Data.SQLite;
using System.Reflection;

```

getConnection Function

This function allows for the form to establish a connection with the database file (.db file).

```

public string connectionString { get; set; }

public virtual void getConnection()
{
    string connection = @"DataSource=NEAdb.db; Version=3"; // connection to .db file
    connectionString = connection;
}

```

getConnVar Function

This is essentially the same function as before however, it can allow for the modification of the data source - what database file it can use and its path to it. This will later allow me to update tables and charts. This is done by declaring a string (executableLocation) which uses the System.IO.Path class which calls the function GetDirectoryName which returns the directory information from a specified path. The class System.Reflection.Assembly is used and the GetExecutingAssembly function is called which gets the assembly that contains the code that is currently executing and .Location is used to get the full path of the loaded file. This is all parsed through the GetDirectoryName function and stored in the variable executableLocation. Then a new string variable is declared which uses the System.IO.Path class and call the Combine function to combine the file path and the parameter *source* to create a file path. This is then used within the query.

```

public void getConnVar(string source)
{
    string executableLocation = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
    string dbLocation = Path.Combine(executableLocation, source);
    string connection = @"DataSource='" + dbLocation + "'; Version=3";
    connectionString = connection;
}

```

createDatabase Function

This function checks for the database file. If it doesn't exist. It would then create the necessary file and create the necessary table (LoginInfo table).

The IF statement checks if the file doesn't exist. It would then create a new file and then the getConnection function is called allowing it to establish a connection with the database. Then it would initiate a new SQLiteConnection and it would use the public connectionString. Within this, it would open the database and initiate multiple new SQLiteCommand objects which is used to create the required tables for the database. Once the SQL query has been executed the connection is closed.

```
public void createDatabase()
{
    if (!File.Exists("NEA.db")) // checks if .db file doesn't exists
    {
        File.Create("NEA.db"); // create .db file

        getConnection();
        using (SQLiteConnection con = new SQLiteConnection(connectionString))
        {
            con.Open(); // opens .db file
            SQLiteCommand cmd1 = new SQLiteCommand // create UserInfo table
            {
                CommandText = @"CREATE TABLE UserInfo (InfoID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, FullName TEXT NOT NULL)",
                Connection = con
            };
            cmd1.ExecuteNonQuery();

            SQLiteCommand cmd2 = new SQLiteCommand // create LoginInfo table
            {
                CommandText = @"CREATE TABLE LoginInfo (LoginID INTERGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
                    Username TEXT NOT NULL,
                    Password TEXT NOT NULL
                    InfoID INTEGER UNIQUE
                    FOREIGN KEY(InfoID) REFERENCES UserInfo(InfoID))",
                Connection = con
            };
            cmd2.ExecuteNonQuery();

            SQLiteCommand cmd3 = new SQLiteCommand // create Income table
            {
                CommandText = @"CREATE TABLE Income (IncomeID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
                    Date TEXT NOT NULL,
                    AmountRecievedin£ REAL NOT NULL,
                    PaymentMethod TEXT NOT NULL,
                    Description TEXT,
                    FromWho TEXT NOT NULL)",
                Connection = con
            };
            cmd3.ExecuteNonQuery();

            SQLiteCommand cmd4 = new SQLiteCommand // create Expenditure table
            {
                CommandText = @"CREATE TABLE Expenditure (ExpenditureID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
                    Date TEXT NOT NULL,
                    AmountSpentin£ REAL NOT NULL,
                    PaymentMethod TEXT NOT NULL,
                    Description TEXT,
                    ToWho TEXT NOT NULL)",
                Connection = con
            };
            cmd4.ExecuteNonQuery();

            SQLiteCommand cmd5 = new SQLiteCommand // create Supplier table
            {
                CommandText = @"CREATE TABLE Supplier (SupplierID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
                    Name TEXT NOT NULL UNIQUE,
                    Address TEXT NOT NULL UNIQUE,
                    ContactNumber INTEGER NOT NULL UNIQUE)",
                Connection = con
            };
            cmd5.ExecuteNonQuery();
        }
    }
}
```

```

        SQLiteCommand cmd6 = new SQLiteCommand // create Stock table
        {
            CommandText = @"CREATE TABLE Stock (StockID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
                                ItemName TEXT NOT NULL,
                                AmountInStock INTEGER NOT NULL,
                                DataLastUpdated TEXT NOT NULL,
                                SupplierID INTEGER,
                                FOREIGN KEY(SupplierID) REFERENCES Supplier(SupplierID))",
            Connection = con
        };
        cmd6.ExecuteNonQuery();

        SQLiteCommand cmd7 = new SQLiteCommand // create Events table
        {
            CommandText = @"CREATE TABLE Events (EventID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
                                EventName TEXT NOT NULL,
                                EventDate TEXT NOT NULL,
                                Details TEXT NOT NULL)",
            Connection = con
        };
        cmd7.ExecuteNonQuery();

        con.Close(); // close database
    }
}
else // if file exist then nothing would happen
{
    return;
}
}

```

LoadDatabase Class

Imports

```

using System;
using System.Data.SQLite;
using System.Windows.Forms;

```

Globalised Property

This class allows me to, later on, load the database into a DataGridView. I have had to call upon my DatabaseConnection class and initialise a new global object of it to allow for the correct function to be called which will allow it to connect to the database.


```
class LoadDatabase
{
    DatabaseConnection DBC = new DatabaseConnection();
```

storedLoad Function

This function takes in two parameters. A string (tableName) and DataGridView(DGVname). The function initialises a new object using the DatabaseConnection class and then uses the getConnection method to allow a connection to the database.

Then using a new SQLiteConnection which uses the connectionString from the DatabaseConnection class it would open the database and, within a try and catch block, initialises a new SQLiteDataAdapter (da) which constructs a data adapter with the SQL query and a specified connection, which was already created earlier.

I then initialise a new instance of the System.Data.DataSet class (ds). I then use da.Fill and ds to add/refresh the rows in ds. Then underneath that the code adds each record within the table into the DataGridView which will allow the user to view the table.

```
public void storedLoad(string tableName, DataGridView DGVname)
{
    DBC.getConnection();

    using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
    {
        con.Open(); // open database

        try
        {
            SQLiteDataAdapter da = new SQLiteDataAdapter($"SELECT * FROM {tableName}", con);
            System.Data.DataSet ds = new System.Data.DataSet();

            da.Fill(ds);
            DGVname.DataSource = ds.Tables[0];
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
        con.Close(); // close database
    }
}
```

updatedLoad Function

This function, functions exactly like the previous one except one thing is changed. That is the connection to the database. I have used the getConnVar method from the DatabaseConnection class rather than

the getConnection method. This method allows the data source. This has been done to change the connection to the database located in the ...\\bin\\debug\\... location as the data is added to the database within that location so this function essentially allows the user to refresh the database with the new data that has been inserted in the table.

```
public void updatedLoad(string tableName, DataGridView DGVname)
{
    // connects to the .db that is updated with new data
    DBC.getConnVar("NEAdb.db");

    using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
    {
        con.Open(); // open database

        try
        {
            SQLiteDataAdapter da = new SQLiteDataAdapter($"SELECT * FROM {tableName}", con);
            System.Data.DataSet ds = new System.Data.DataSet();

            da.Fill(ds);
            DGVname.DataSource = ds.Tables[0];
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
        con.Close(); // close database
    }
}
```

FormOpener Class

This entire contain multiple functions which will, later, allow the program to open and close form simultaneously.

```

using System.Windows.Forms;

namespace Business_Management_System__NEA_
{
    class FormOpener
    {
        public void OpenLoginForm()
        {
            Application.Run(new LoginForm());
        }

        public void OpenStreamlineWindow()
        {
            Application.Run(new StreamlineWindow());
        }

        public void OpenWelcomeForm()
        {
            Application.Run(new Welcome_Page());
        }

        public void OpenError()
        {
            Application.Run(new ErrorForm());
        }
    }
}

```

Login Page

Imports

```

using System;
using System.Text;
using System.Windows.Forms;
using System.Data.SQLite;
using System.Security.Cryptography;

```

Globalised Properties

In the beginning I have two global properties. One is a variable which will be used later on to initialise a new object and the other initialise a new object from the FormOpener class

```

DatabaseConnection DBC = new DatabaseConnection();
FormOpener FO = new FormOpener();

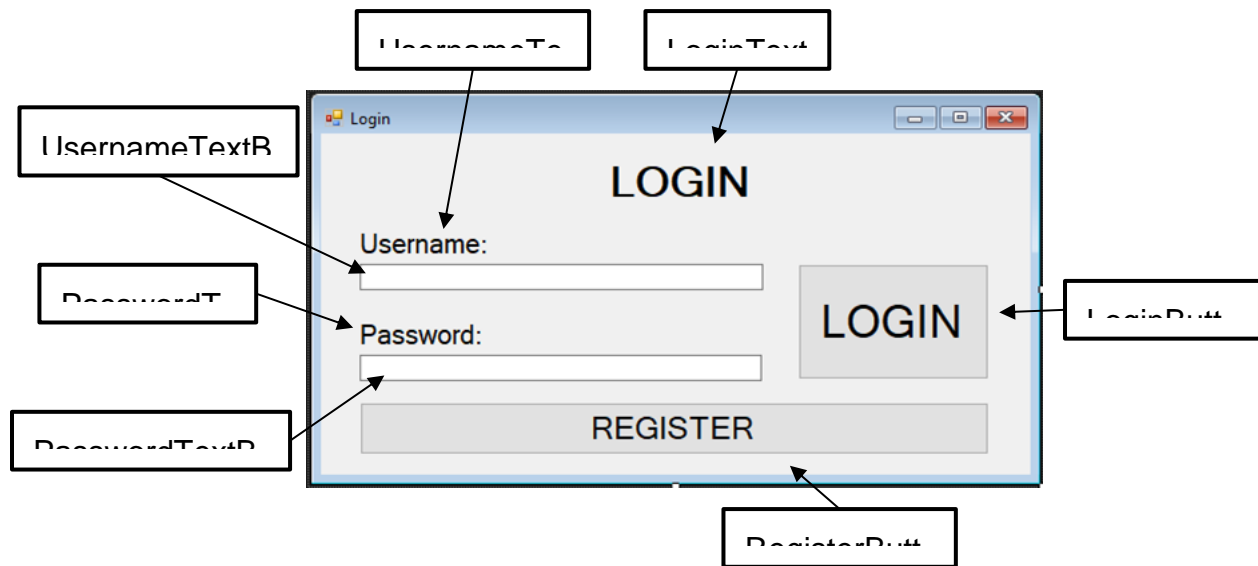
```

I have also declared a public static string variable passingText which will be used to pass the entered username text and retrieve their full name which will be used in the welcome page.

```
public static string passingText;
```

Login Form

This form allows the user to login by entering their detail into the textboxes. The user is also able to register a new account by clicking on the register button which will bring up the register form.



checkAccount Function

This function takes in a parameter, username, which is later used in the SQL query. At the start, it would initialise a new object from the DatabaseConnection class and it uses the two methods, getConnection and createDatabase. Then using a new SQLiteConnection object. It opens the database, initialises a new SQLiteCommand object. cmd.CommandText stores the SQL query and cmd.Connection allows for a connection to the database.

Then SQLiteDataReader store cmd.ExecuteReader which reads the data and if the account is in the database then the count will increment return true else it will show the error form.

```

private bool checkAccount(string username)
{
    DBC.getConnection(); // connects to .db file
    DBC.createDatabase(); // if file is not there then creates new database

    using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
    {
        con.Open(); // open .db
        SQLiteCommand cmd = new SQLiteCommand
        {
            CommandText = @"SELECT * FROM LoginInfo WHERE Username='" + username + "'", // SQL Query
            Connection = con
        };

        SQLiteDataReader rd = cmd.ExecuteReader(); // reads data
        int count = 0;
        while (rd.Read())
        {
            count++; // if data is there then increments
        }

        if (count == 1) // account is in database
        {
            rd.Close(); // close data reader
            con.Close(); // close database
            return true;
        }
        else
        {
            // Opens Error Form
            System.Threading.Thread thread = new System.Threading.Thread(new System.Threading.ThreadStart(FO.OpenError));
            this.Close();
            thread.Start();
            return false;
        }
    }
}

```

correctInfo Function

This function takes in two parameter, username and password which will be used later on in the SQL query. This function work similarly to the checkAccount function but this checks if the username and password are correct and if it is correct then it would open up the welcome form by initialising a new instance of the System.Threading.Thread class and within it, new System.Threading.ThreadStart, which represents the method that executes on a System.Threading.Thread, and within that calls the OpenWelcomeForm method from the FormOpener class by using the new initialised object FO. Else it would open the error form by executing similar code with the only difference being that the OpenErroForm method is called from the FormOpener class.

```

private void correctInfo(string username, string password)
{
    DBC.getConnection(); // calls getConnection function from DatabaseConnection class

    using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
    {
        con.Open(); // open database
        // select login information using the inputted parameters
        SQLiteCommand cmd = new SQLiteCommand
        {
            CommandText = string.Format(@"SELECT * FROM LoginInfo WHERE Username='{0}' AND Password='{1}'", username, password),
            Connection = con
        };

        int count = 0; // counter
        SQLiteDataReader read = cmd.ExecuteReader();
        while (read.Read())
        {
            count++; // increment counter
        }
        if (count == 1)
        {
            read.Close(); // close data reader
            con.Close(); // close database
            // Opens Welcome Form
            System.Threading.Thread t = new System.Threading.Thread(new System.Threading.ThreadStart(FO.OpenWelcomeForm));
            this.Close();
            t.Start();
        }
        else
        {
            read.Close(); // close data reader
            con.Close(); // close database
            // Opens Error Form
            System.Threading.Thread thread = new System.Threading.Thread(new System.Threading.ThreadStart(FO.OpenError));
            this.Close();
            thread.Start();
        }
    }
}

```

LoginButton_Click Event Handler

This function executes when the login button is clicked. If both textboxes are not empty then the checkAccount method is called and if that returns true then the correctInfo method is called. If the details entered are admin and password then those are used as the parameters other if different account details are used then the password would be encrypted with an MD5 hash.

The hash works as follows, the local variable encodePassword is a byte array which initialises a new instance of the UTF8Encoding class which calls the GetBytes method. This encodes all the characters in a specified string into a sequence of bytes, in this case that would be the password that is entered by the user.

The local variable hashp, which is also a byte array, uses the System.Security.Cryptography.HashAlgorithm class, which represents the base class from which all implementations of cryptographic hash algorithms must derive, and the System.Security.Cryptography.CryptoConfig class, which accesses the cryptography configuration information, to create a new instance of a specified cryptographic object, in this case that would be MD5. It then calls the ComputerHash method which computes the hash value for a specified byte array,

in this case that would be the variable `encodedPassword`.

Then two local variables are declared. The first one, `encodedp`, converts the byte array stored in `hashp` into a string. The second one, `encodep`, removes all dashes from the string.

The encrypted password and entered username are then used as the parameters for the `correctInfo` method.

If the password text box is empty then a message box would appear and ask the user to enter a password. The same goes for the username textbox.

If both text boxes are empty then a message box would appear to ask the user to enter a username and password.

```
private void LoginButton_Click(object sender, EventArgs e)
{
    passingText = UsernameTextBox.Text;

    if (UsernameTextBox.Text != string.Empty && PasswordTextBox.Text != string.Empty) // both textboxes are not empty
    {
        if (checkAccount(UsernameTextBox.Text) == true) // account is in database then...
        {
            if (UsernameTextBox.Text == "admin" && PasswordTextBox.Text == "password") // default data used
            {
                correctInfo("admin", "password"); // call correctInfo function
            }
            else
            {
                // Password Hash
                byte[] encodedPassword = new UTF8Encoding().GetBytes(PasswordTextBox.Text);

                byte[] hashp = ((HashAlgorithm)CryptoConfig.CreateFromName("MD5")).ComputeHash(encodedPassword);

                string encodedp = BitConverter.ToString(hashp);
                string encodep = encodedp.Replace("-", string.Empty);

                correctInfo(UsernameTextBox.Text, encodep); // call correctInfo function
            }
        }
    }

    else if (UsernameTextBox.Text != string.Empty && PasswordTextBox.Text == string.Empty) // password textbox is empty
    {
        MessageBox.Show("Enter Password");
    }
    else if (UsernameTextBox.Text == string.Empty && PasswordTextBox.Text != string.Empty) // username textbox is empty
    {
        MessageBox.Show("Enter Username");
    }
    else if (UsernameTextBox.Text == string.Empty && PasswordTextBox.Text == string.Empty) // both texboxes are empty
    {
        MessageBox.Show("Enter Username and Password");
    }
}
```

RegisterButton_Click Event Handler

This just allows the user to open up the register form when clicked on the register button. This is done

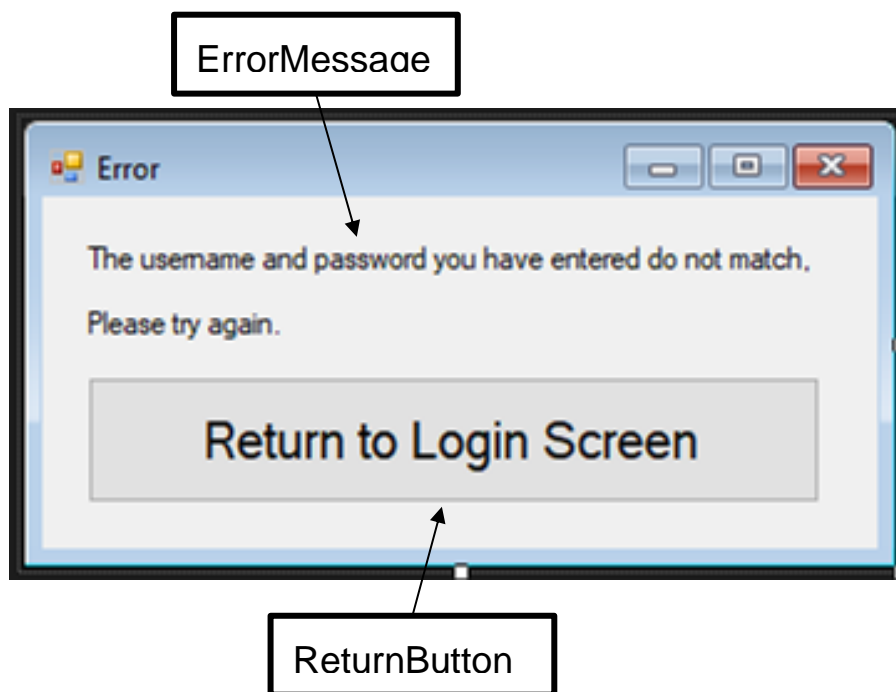
by creating a new register form object and by using the showDialog method to show the form.

```
private void RegisterButton_Click(object sender, EventArgs e)
{
    RegisterForm rf = new RegisterForm(); // new form object
    rf.ShowDialog(); // show form
}
```

Error Message

Error Form

This form is pretty simple, showing the user an error and allowing them to go back to the login form by clicking on the button. This was created using a label and a button.



Code

The code below is pretty simple. A global object is initialised from the FormOpener class. Below that is the method which execute when the button is clicked by the user.

The code initialises a new instance of the System.Threading.Thread class which then creates a new instance of System.Threading.ThreadStart which uses the OpenLoginForm method from the FormOpener class. The code then closes the currently visible form and starts the thread.


```

using System;
using System.Windows.Forms;

namespace Business_Management_System__NEA_
{
    public partial class ErrorForm : Form
    {
        public ErrorForm()
        {
            InitializeComponent();

            FormOpener FO = new FormOpener();

            private void ReturnButton_Click(object sender, EventArgs e)
            {
                System.Threading.Thread t = new System.Threading.Thread(new System.Threading.ThreadStart(FO.OpenLoginForm)); // create a new thread
                this.Close(); // close current form
                t.Start(); // start the thread
            }
        }
    }
}

```

Register Page

Imports

```

using System;
using System.Text;
using System.Windows.Forms;
using System.Data.SQLite;
using System.Security.Cryptography;

```

Globalised Property

In the beginning I have initialised a global properties. That property is a new DatabaseConnection object.

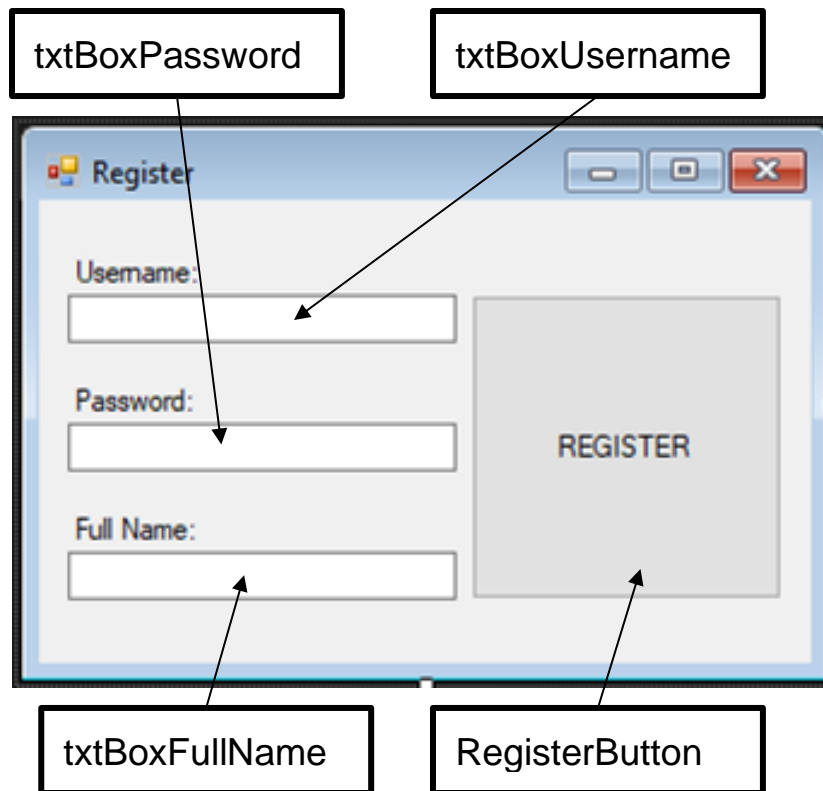
```

public partial class RegisterForm : Form
{
    DatabaseConnection DBC = new DatabaseConnection();
}

```

Register Form

This form allows the user to register a new account by entering their details into the textboxes. The user would then click the button and an message would be displayed depending on a few factors which will be covered below.



AddFullName Function

This function has one parameter, FN. To begin with, the getConnection class is called from the DatabaseConnection class using the new object, this allows for a connection to the database. Using the new SQLiteConnection object that uses the connectionString from the DatabaseConnection class, the database is opened. A new SQLiteCommand is created which inserts the parameter, FN, value into the FullName column within the UserInfo table. The database is then closed.

```
private void AddFullName(string FN)
{
    DBC.getConnection(); // connects to database file

    using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
    {
        con.Open(); // open database
        // insert full name into table UserInfo
        SQLiteCommand cmd = new SQLiteCommand("INSERT INTO UserInfo(FullName) VALUES (@Full)", con);
        cmd.Parameters.AddWithValue("@Full", FN);
        cmd.ExecuteNonQuery();
        con.Close(); // close database
    }
}
```

AddLoginInfo Function

This function uses three parameters. One for the username, the other for the password and the last for

the full name of the user. The function would first call the getConnection method to connect to the database. Then the AddFullName function would be called with the parameter being the text that is entered in the textbox for the full name. After that a local variable with data type integer is declared.

```
private void AddLoginInfo(string username, string password, string fullname)
{
    DBC.getConnection(); // connects to database file
    // calls AddFullName function
    AddFullName(txtBoxFullName.Text);

    int ID = 0;

    using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
    {
        con.Open(); // open database
        // select Primary Key value from UserInfo table
        SQLiteCommand cmdS = new SQLiteCommand(string.Format(@"SELECT InfoID FROM UserInfo WHERE FullName='{0}'", fullname), con);
        SQLiteDataReader SQLread = cmdS.ExecuteReader();

        if (SQLread.HasRows)
        {
            SQLread.Read();
            ID = SQLread.GetInt32(0); // store PK value from UserInfo table
            // insert new username and password into LoginInfo table
            SQLiteCommand cmd = new SQLiteCommand(@"INSERT INTO LoginInfo(Username, Password, InfoID) VALUES (@username, @password, @info)", con);
            cmd.Parameters.AddWithValue("@username", username);
            cmd.Parameters.AddWithValue("@password", password);
            cmd.Parameters.AddWithValue("@info", ID);
            cmd.ExecuteNonQuery();
            MessageBox.Show("Account Created!"); // exception handling
            // close current form
            this.Close();
        }

        con.Close(); // close database
    }
}
```

checkLoginInfo Fuction

This function allows the inputted information to be checked against the database to see if the account already exist. If not a new account would be created.

The function will start by calling the createDatabase function from the DatabaseConnection class using the global object DBC. Then the getConnection function would be called to establish a connection with the database.

Using the new SQLiteConnection which uses the connectionString from the DatabaseConnection class, the database is opened and within a try and catch block, a new SQLiteCommand is created with an SQL query which will select the row from the LoginInfo table where the Username column contain the parameter that has been parsed through the function. Then a new counter is declared and while the .Read() function return true, increment the counter, the .Read function will only return true if the record is present within the table. If the counter doesn't increment then the password is encrypted with an MD5 hash and then AddLoginInfo function called with username (the parameter), the hashed password and the inputted full name is parse through the function. If the counter has incremented then a message is shown to the user saying that the account already exists. Then the database is closed.

```

private void checkLoginInfo(string username, string password)
{
    DBC.createDatabase();
    DBC.getConnection();

    using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
    {
        con.Open(); // open database

        try
        {
            // select username from LoginInfo
            SQLiteCommand cmd = new SQLiteCommand(@"SELECT * FROM LoginInfo WHERE Username ='" + username + "'", con);

            int count = 0; // counter
            SQLiteDataReader reader = cmd.ExecuteReader();
            while (reader.Read())
            {
                count++; // increment counter
            }

            if (count == 0) // account does not exist
            {
                // Password Hash
                byte[] encodedPassword = new UTF8Encoding().GetBytes(password);

                byte[] hashp = ((HashAlgorithm)CryptoConfig.CreateFromName("MD5")).ComputeHash(encodedPassword);

                string encodedp = BitConverter.ToString(hashp);
                string encodep = encodedp.Replace("-", string.Empty);
                // call AddLoginInfo Function
                AddLoginInfo(username, encodep, txtBoxFullName.Text);
            }
            else if (count == 1) // account does exist
            {
                MessageBox.Show("Account already exists", "Try Again"); // exception handling
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }

        con.Close(); // close database
    }
}

```

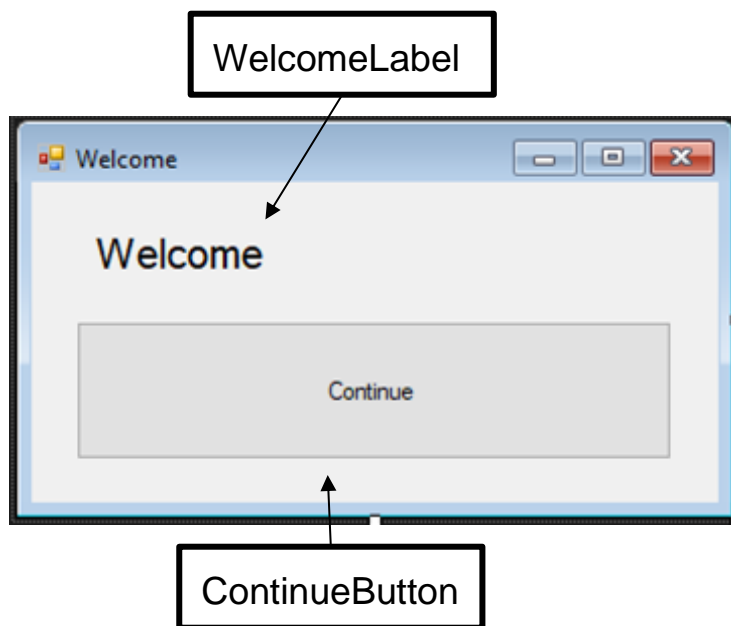
RegisterButton_Click Event Handler

This execute when the register button is clicked. This will check if each textbox has been filled in by the user. If it hasn't been filled in then the relevant message will be shown to the user.

```
private void RegisterButton_Click(object sender, EventArgs e)
{
    if (txtBoxUsername.Text != string.Empty && txtBoxPassword.Text != string.Empty && txtBoxFullName.Text != string.Empty)
    {
        checkLoginInfo(txtBoxUsername.Text, txtBoxPassword.Text); // calls checkLoginInfo function
    }
    else if (txtBoxUsername.Text != string.Empty && txtBoxPassword.Text == string.Empty && txtBoxFullName.Text != string.Empty)
    {
        MessageBox.Show("Enter Password"); // exception handling
    }
    else if (txtBoxUsername.Text == string.Empty && txtBoxPassword.Text != string.Empty && txtBoxFullName.Text != string.Empty)
    {
        MessageBox.Show("Enter Username"); // exception handling
    }
    else if (txtBoxUsername.Text != string.Empty && txtBoxPassword.Text != string.Empty && txtBoxFullName.Text == string.Empty)
    {
        MessageBox.Show("Enter Your Full Name"); // exception handling
    }
}
```

Welcome Message

Welcome Form



Imports

```
using System;
using System.Windows.Forms;
using System.Data.SQLite;
```

ContinueButton_Click Event Handler

This allows the form to close and start the main window (StreamLineWindow) to open by starting a new thread.

```
private void ContinueButton_Click(object sender, EventArgs e)
{
    FormOpener FO = new FormOpener();
    System.Threading.Thread t = new System.Threading.Thread(new System.Threading.ThreadStart(FO.OpenStreamlineWindow));
    this.Close();
    t.Start();
}
```

Welcome_Page_Load Event Handler

This allows the form to load with the correct name of the user by selecting the foreign key of the relevant record in the LoginInfo table and then store that key and use it to find the relevant record in the UserInfo table. Then store the name and use it to edit the label in the form to say “Welcome, *name of user*”.

```
private void Welcome_Page_Load(object sender, EventArgs e)
{
    // gets text from username textbox in the LoginForm
    string name = LoginForm.passingText;
    int id; // declaration
    string fullName; // declaration

    DatabaseConnection DBC = new DatabaseConnection();
    // get connection to updated database
    DBC.getConnVar("NEAdb.db");

    using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
    {
        con.Open(); // open database
        // select PK from LoginInfo where username = userInput
        SQLiteCommand cmd = new SQLiteCommand(string.Format(@"SELECT InfoID FROM LoginInfo WHERE Username='{0}'", name), con);
        SQLiteDataReader SQLread = cmd.ExecuteReader();

        if (SQLread.HasRows) // true if there are rows in table
        {
            SQLread.Read(); // reads the row
            id = SQLread.GetInt32(0); // stores PK of fullname
            // close data reader
            SQLread.Close();
            // selects full name using stored PK value from earlier
            SQLiteCommand cmdFN = new SQLiteCommand(string.Format(@"SELECT FullName FROM UserInfo WHERE InfoID={0}", id), con);
            SQLiteDataReader sqlRead = cmd.ExecuteReader();

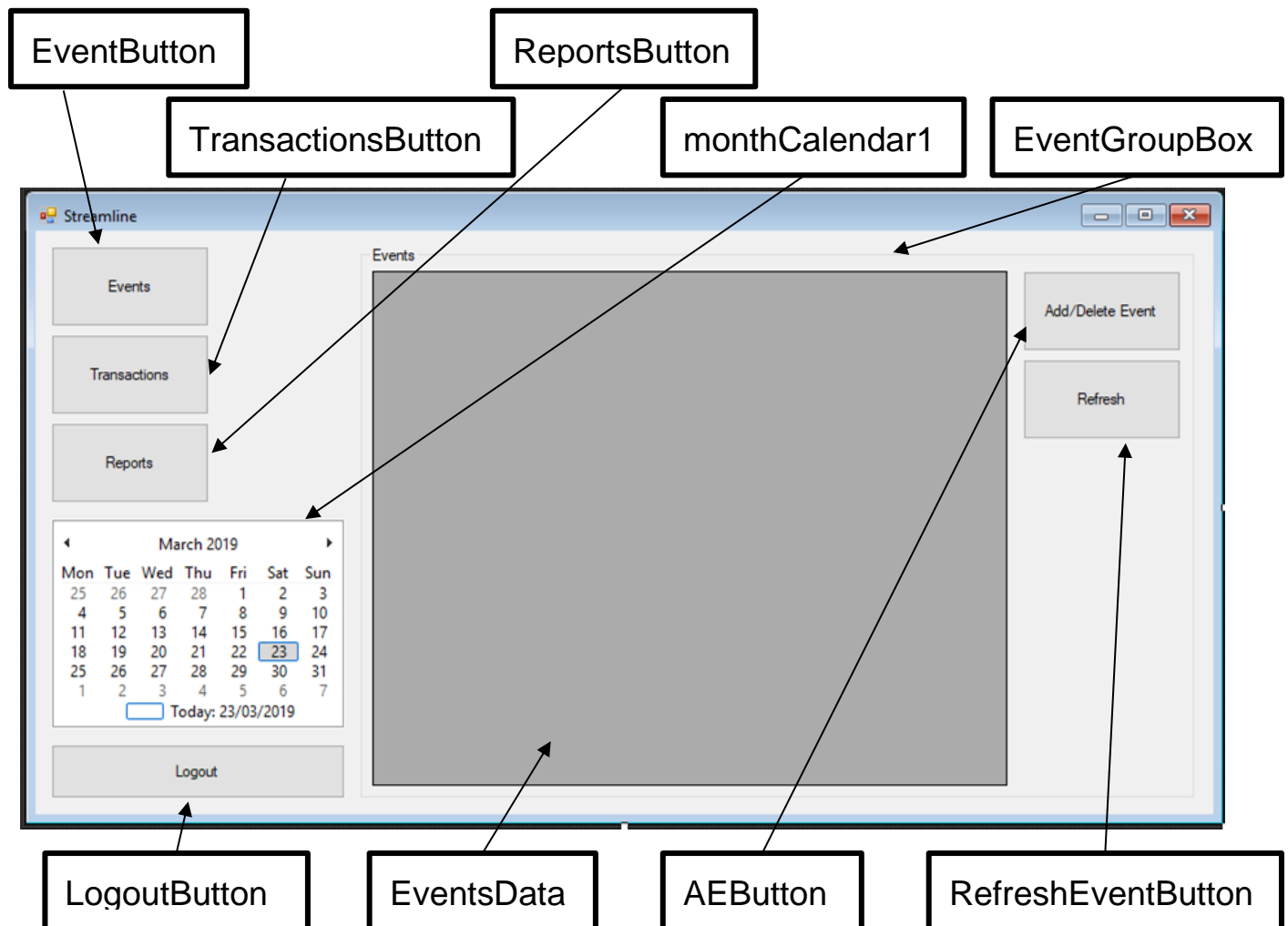
            if (sqlRead.HasRows) // true is there are rows in table
            {
                sqlRead.Read(); // reads the rows
                fullName = Convert.ToString(cmdFN.ExecuteScalar()); // stores full name
                // edits the WelcomeLabel
                WelcomeLabel.Text = string.Format("Welcome, {0}", fullName);
            }

            sqlRead.Close(); // close data reader
            con.Close(); // close database
        }
    }
}
```

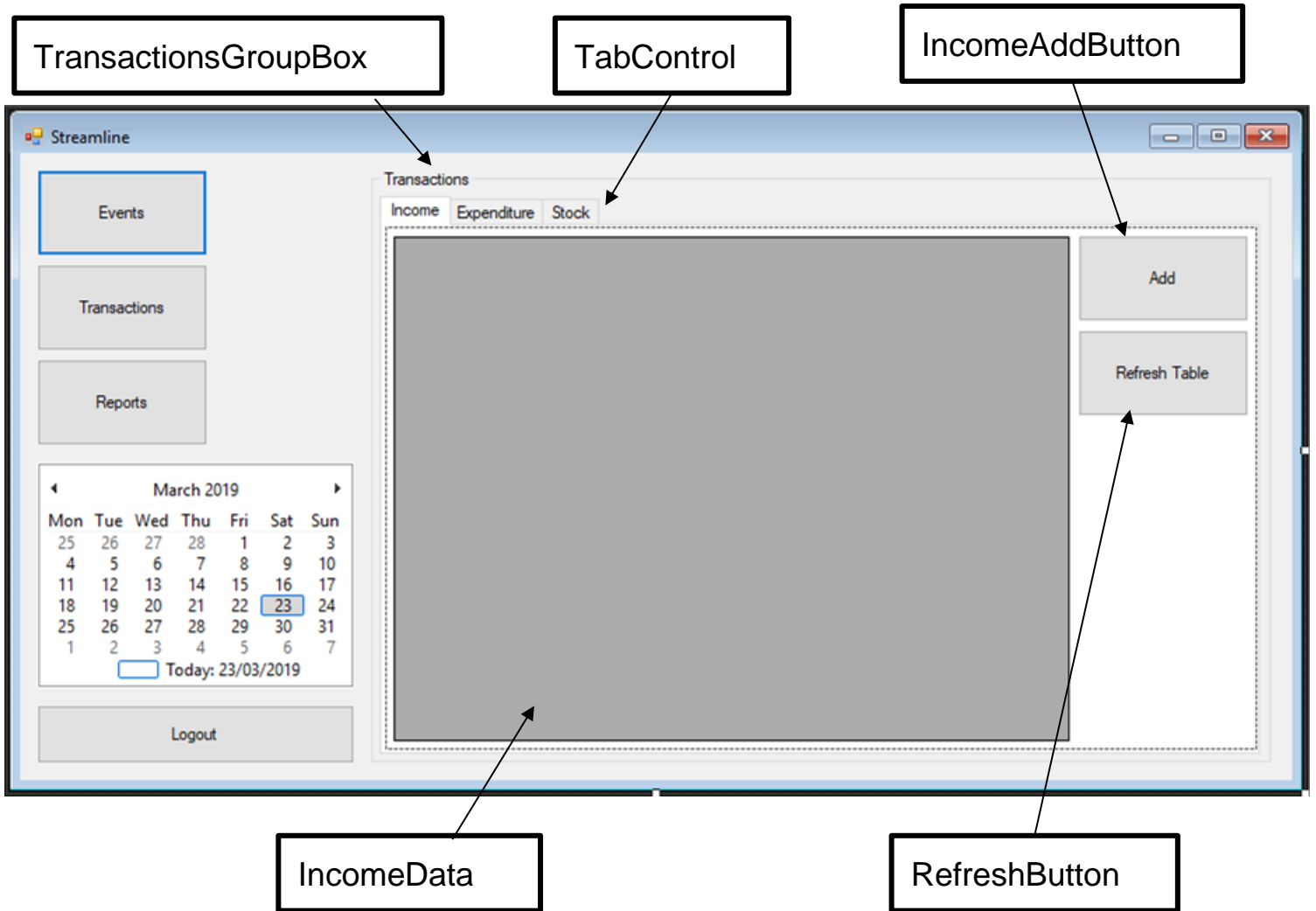
Main Window

The main window was created using a number of tools including textboxes, labels, groupboxes, a month calendar, data grid views, charts and tab controls.

Form Showing Events Window



Form Showing Transactions Window and Income Tab

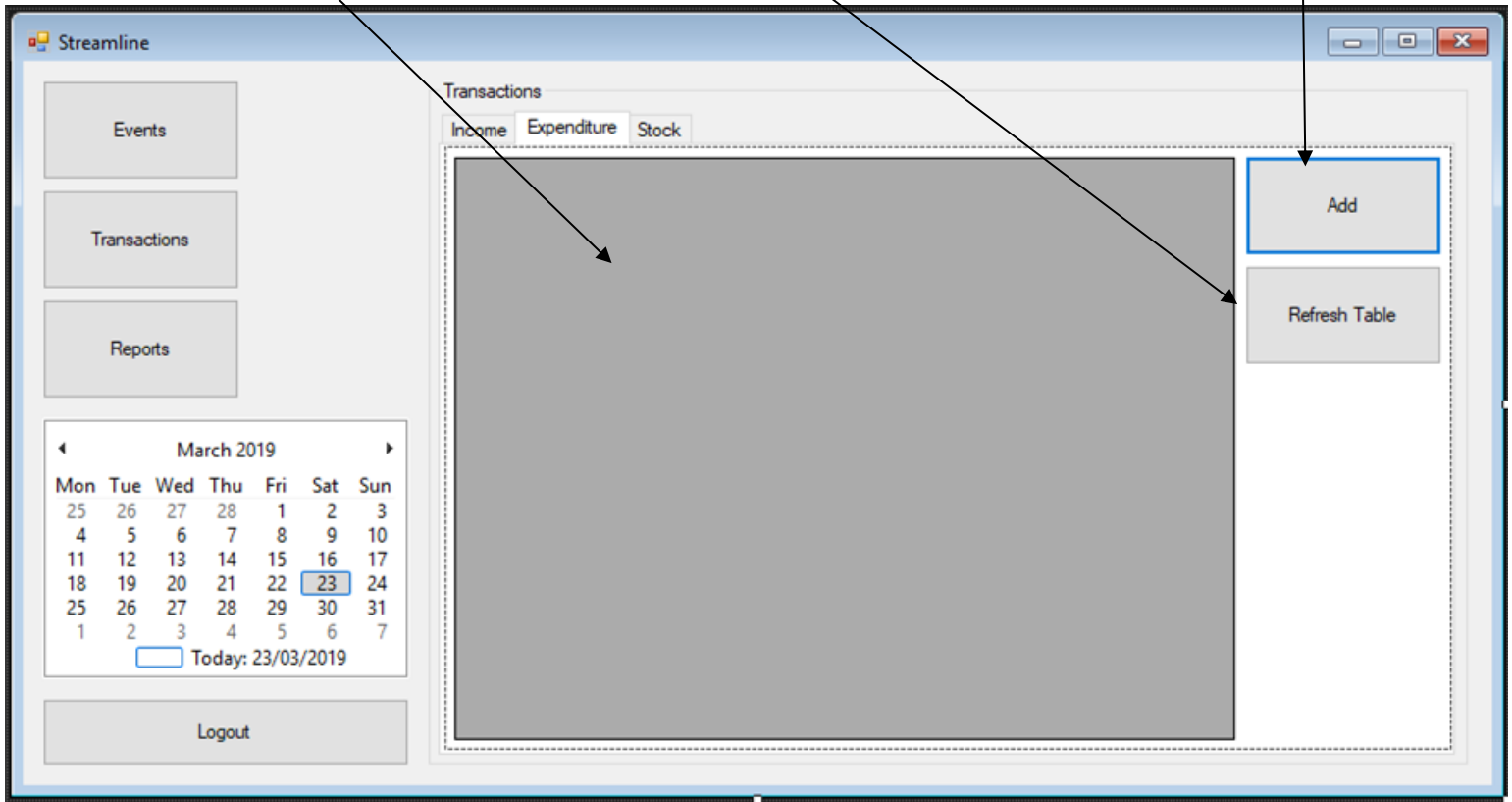


Form Showing Transactions Window and Expenditure Tab

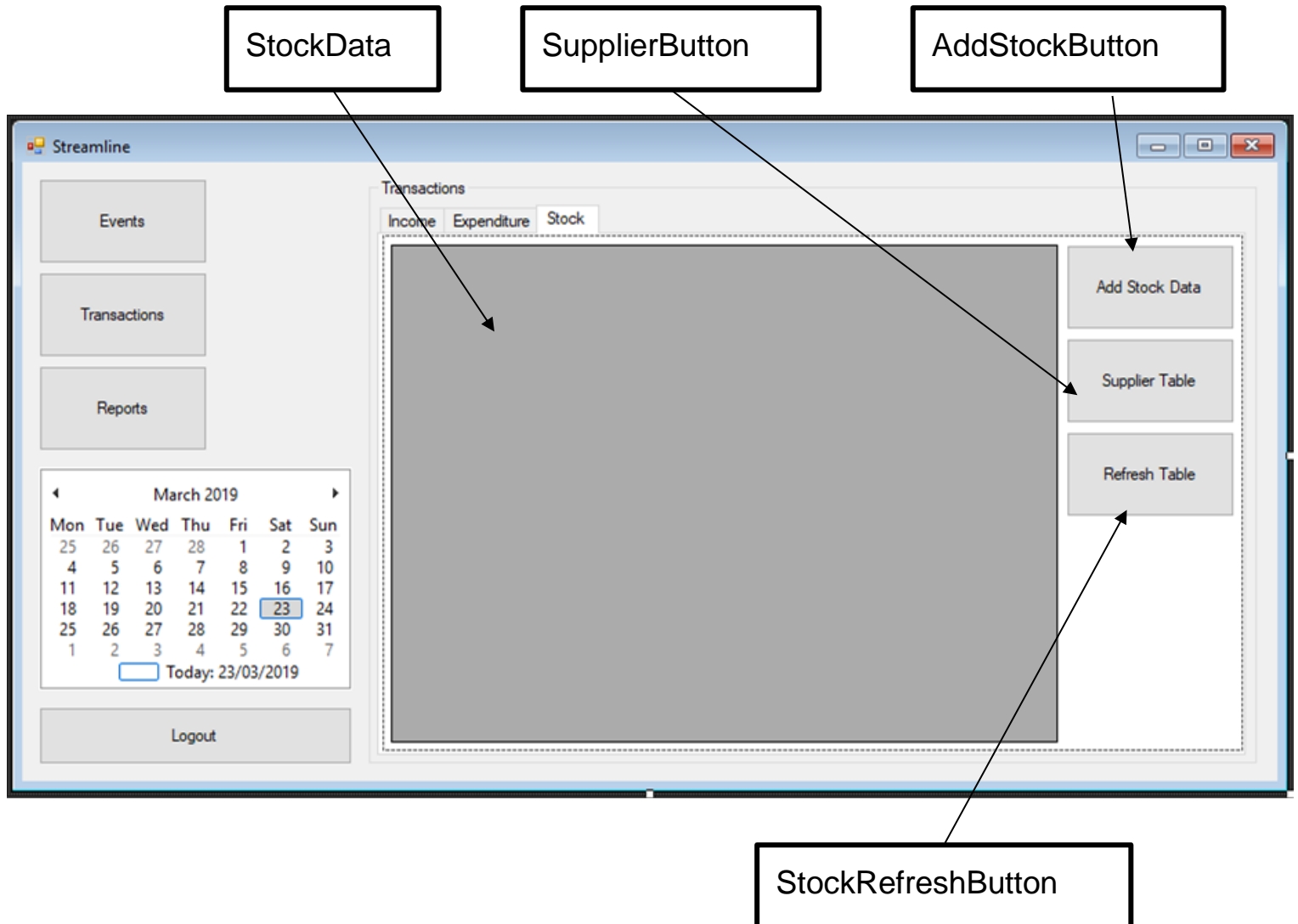
ExpenditureData

ExpenditureRefresh

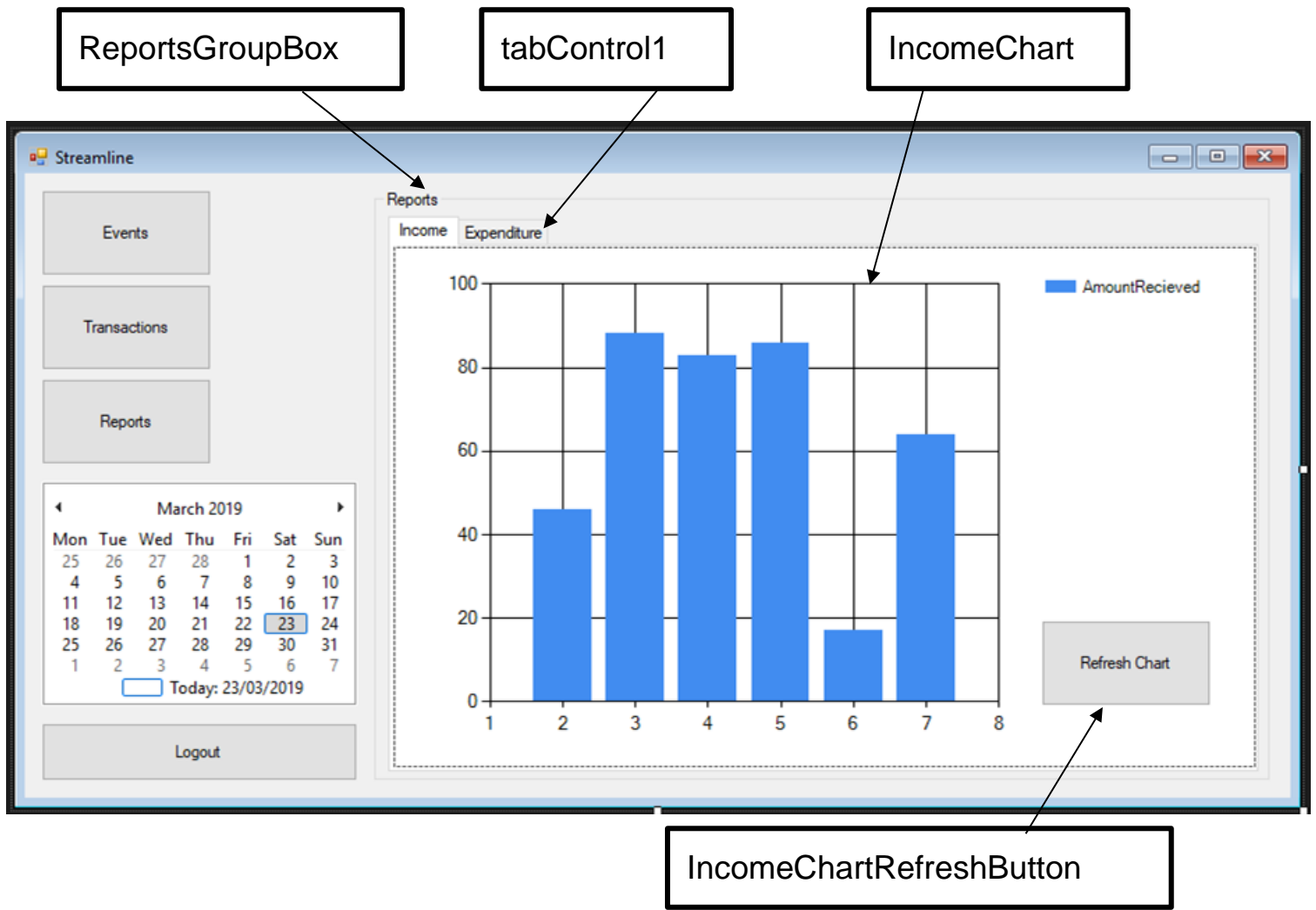
ExpenditureAdd



Form Showing Transactions Window and Stock Tab



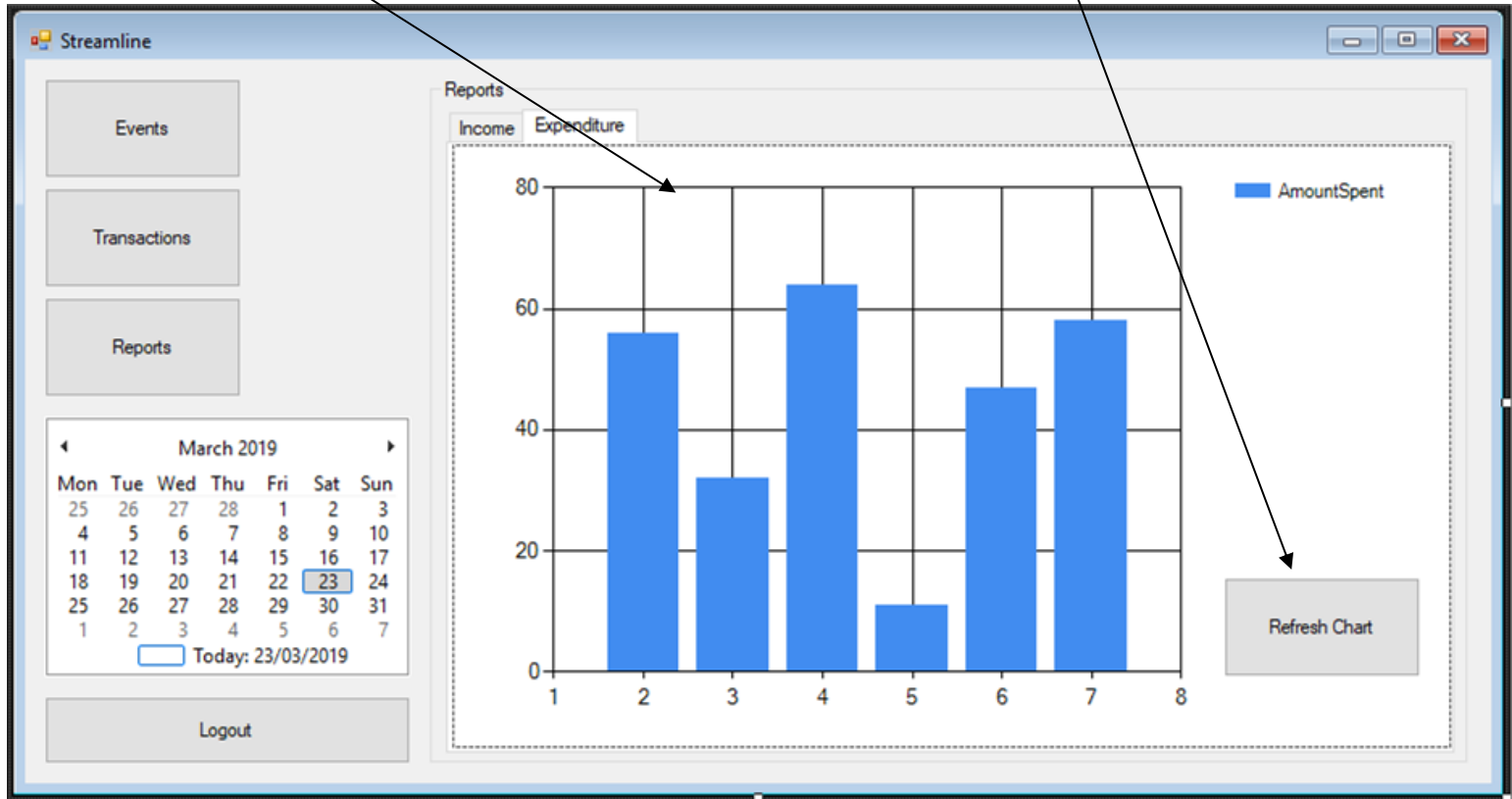
Form Showing Reports Window and Income Tab



Form Showing Reports Window and Expenditure Tab

ExpenditureChart

ExpenditureChartRefreshButton



Imports

```
using System;  
using System.Windows.Forms;  
using System.Data.SQLite;
```

Income Class

This class will be used later on the store data which will be used to add to a dictionary

```
public class Income
{
    public string Date { get; set; }
    public float AmountRecieved { get; set; }
}
```

Expenditure Class

This class will be used later on the store data which will be used to add to a dictionary

```
public class Expenditure
{
    public string Date { get; set; }
    public float AmountSpent { get; set; }
}
```

StreamlineWindow Class

Globalised Properties

I have initialised three global objects. One is to load the database into the data grid view. The other is to allow forms to open by creating a new thread. The third is to allow for a connection to the database.

```
LoadDatabase LDB = new LoadDatabase();
FormOpener FO = new FormOpener();
DatabaseConnection DBC = new DatabaseConnection();
```

I have initialised two dictionaries which will be used to input points into a chart. One is for the income chart and the other is for the expenditure chart.

```
Dictionary<string, float> incomeData = new Dictionary<string, float>();
Dictionary<string, float> expendData = new Dictionary<string, float>();
```

TransactionButton_Click Event Handler

This allows for the TransactionsGroupBox to show when the TransactionsButton is clicked by hiding the other group boxes.

```
private void TransactionsButton_Click(object sender, EventArgs e)
{
    TransactionsGroupBox.Visible = true;
    EventGroupBox.Visible = false;
    ReportsGroupBox.Visible = false;
}
```

EventButton_Click Event Handler

This allows for the EventGroupBox to show when the EventButton is clicked by hiding the other group boxes.

```
private void EventButton_Click(object sender, EventArgs e)
{
    EventGroupBox.Visible = true;
    TransactionsGroupBox.Visible = false;
    ReportsGroupBox.Visible = false;
}
```

ReportsButton_Click Event Handler

This allows for the ReportsGroupBox to show when the ReportsButton is clicked by hiding the other group boxes.

```
private void ReportsButton_Click(object sender, EventArgs e)
{
    TransactionsGroupBox.Visible = false;
    EventGroupBox.Visible = false;
    ReportsGroupBox.Visible = true;
}
```

StreamlineWindow_Load Event Handler

```

private void StreamlineWindow_Load(object sender, EventArgs e)
{
    // call storedLoad method from LoadDatabase class
    LDB.storedLoad("Income", IncomeData);
    LDB.storedLoad("Expenditure", ExpenditureData);
    LDB.storedLoad("Stock", StockData);
    LDB.storedLoad("Events", EventsData);

    DBC.getConnection();
    using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
    {
        con.Open(); // open database

        // INCOME CHART
        // select values from Date and Amount Recieved columns in Income table
        SQLiteCommand cmdI = new SQLiteCommand(@"SELECT Date, AmountRecievedin£ FROM Income", con);
        SQLiteDataReader SQLreadI = cmdI.ExecuteReader();
        Income Idata = new Income();

        while (SQLreadI.Read())
        {
            Idata.Date = Convert.ToString(SQLreadI["Date"]); // store Date values in variable
            Idata.AmountRecieved = Convert.ToInt32(SQLreadI["AmountRecievedin£"]); // store Amount Recieved values in variable
            incomeData.Add(Idata.Date, Idata.AmountRecieved);
        };

        foreach (KeyValuePair<string, float> ID in incomeData)
        {
            // ...add them as point in the chart
            IncomeChart.Series["AmountRecieved"].Points.AddXY(ID.Key, ID.Value);
        }

        // EXPENDITURE CHART
        // select values from Date and Amount Recieved columns in Income table
        SQLiteCommand cmdE = new SQLiteCommand(@"SELECT Date, AmountSpentIn£ FROM Expenditure", con);
        SQLiteDataReader SQLreadE = cmdE.ExecuteReader();
        Expenditure Edata = new Expenditure();

        while (SQLreadE.Read())
        {
            Edata.Date = Convert.ToString(SQLreadE["Date"]); // store Date values in variable
            Edata.AmountSpent = Convert.ToInt32(SQLreadE["AmountSpentIn£"]); // store Amount Recieved values in variable
            expendData.Add(Edata.Date, Edata.AmountSpent);
        };

        foreach (KeyValuePair<string, float> ED in expendData)
        {
            // ...add them as point in the chart
            ExpenditureChart.Series["AmountSpent"].Points.AddXY(ED.Key, ED.Value);
        }

        con.Close();
    }
}

```

IncomeAddButton_Click Event Handler

When the IncomeAddButton is clicked by the user the Add_Income form is shown.

```
private void IncomeAddButton_Click(object sender, EventArgs e)
{
    Add_Income AI = new Add_Income();
    AI.Show();
}
```

RefreshButton_Click Event Handler

When the RefreshButton is clicked then DataGridView is reloaded with the new and updated table.

```
private void RefreshButton_Click(object sender, EventArgs e)
{
    LDB.updatedLoad("Income", IncomeData); // calls updatedLoad function from LoadDatabase class
}
```

ExpenditureAdd_Click Event Handler

When the ExpenditureAdd button is clicked by the user the Add_Expenditure form is shown.

```
private void ExpenditureAdd_Click(object sender, EventArgs e)
{
    Add_Expenditure AE = new Add_Expenditure();
    AE.Show();
}
```

ExpenditureRefresh_Click Event Handler

When the ExpenditureRefresh is clicked then DataGridView is reloaded with the new and updated table.

```
private void ExpenditureRefresh_Click(object sender, EventArgs e)
{
    LDB.updatedLoad("Expenditure", ExpenditureData); // calls updatedLoad function from LoadDatabase class
}
```

StockRefreshButton_Click Event Handler

When the StockRefreshButton is clicked then DataGridView is reloaded with the new and updated table.


```
private void StockRefreshButton_Click(object sender, EventArgs e)
{
    LDB.updatedLoad("Stock", StockData); // calls updatedLoad function from LoadDatabase class
}
```

AddStockButton_Click Event Handler

When the AddStockButton is clicked by the user the Add_Stock form is shown.

```
private void AddStockButton_Click(object sender, EventArgs e)
{
    Add_Stock AS = new Add_Stock();
    AS.Show();
}
```

SupplierButton_Click Event Handler

When the SupplierButton is clicked by the user the Suppliers form is shown.

```
private void SupplierButton_Click(object sender, EventArgs e)
{
    Suppliers S = new Suppliers();
    S.Show();
}
```

LogoutButton_Click Event Handler

When the LogoutButton is clicked by the user then the entire window is closed a new thread starts which leads the user back to the login page.

```
private void LogoutButton_Click(object sender, EventArgs e)
{
    System.Threading.Thread t = new System.Threading.Thread(new System.Threading.ThreadStart(FO.OpenLoginForm)); // create a new thread
    this.Close(); // close current form
    t.Start(); // start the thread
}
```

IncomeChartRefreshButton_Click Event Handler

To sum up what this does, when the button is clicked the entire dictionary is cleared and all the points within the series are removed and all the data within the income table including the newly added data is

added into the dictionary and then each point is added to series, so entire chart is refreshed with the new data and the old data

```
private void IncomeChartRefreshButton_Click(object sender, EventArgs e)
{
    incomeData.Clear(); // clears the dictionary

    foreach (var series in IncomeChart.Series)
    {
        series.Points.Clear(); // removes each point in series
    }

    DBC.getConnVar("NEAdb.db");
    using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
    {
        con.Open(); // open database
        // select values from Date and Amount Recieved columns in Income table
        SQLiteCommand cmdI = new SQLiteCommand(@"SELECT Date, AmountRecievedinf FROM Income", con);
        SQLiteDataReader SQLreadI = cmdI.ExecuteReader();
        Income Idata = new Income(); // new object using income class

        while (SQLreadI.Read())
        {
            Idata.Date = Convert.ToString(SQLreadI["Date"]); // store Date values in variable
            Idata.AmountRecieved = Convert.ToInt32(SQLreadI["AmountRecievedinf"]); // store Amount Recieved values in variable
            incomeData.Add(Idata.Date, Idata.AmountRecieved); // add values to dictionary
        };

        foreach (KeyValuePair<string, float> ID in incomeData)
        {
            // ...add them as point in the chart
            IncomeChart.Series["AmountRecieved"].Points.AddXY(ID.Key, ID.Value);
        }
        con.Close(); // close database
    }
}
```

ExpenditureChartRefreshButton_Click Event Handler

To sum up what this does, when the button is clicked the entire dictionary is cleared and all the points within the series are removed and all the data within the expenditure table including the newly added data is added into the dictionary and then each point is added to series, so entire chart is refreshed with the new data and the old data

```

private void ExpenditureChartRefreshButton_Click(object sender, EventArgs e)
{
    expendData.Clear(); // clears the dictionary

    foreach (var series in ExpenditureChart.Series)
    {
        series.Points.Clear(); // removes each point in series
    }

    DBC.getConnVar("NEAdb.db");
    using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
    {
        con.Open(); // open database
        // select values from Date and Amount Spent columns in Expenditure table
        SQLiteCommand cmdE = new SQLiteCommand(@"SELECT Date, AmountSpentInE FROM Expenditure", con);
        SQLiteDataReader SQLreadE = cmdE.ExecuteReader();
        Expenditure Edata = new Expenditure(); // new object using Expenditure class

        while (SQLreadE.Read())
        {
            Edata.Date = Convert.ToString(SQLreadE["Date"]); // store Date values in variable
            Edata.AmountSpent = Convert.ToInt32(SQLreadE["AmountSpentInE"]); // store Amount Spent values in variable
            expendData.Add(Edata.Date, Edata.AmountSpent); // adds values to dictionary
        };

        foreach (KeyValuePair<string, float> ED in expendData)
        {
            // ...add them as point in the chart
            ExpenditureChart.Series["AmountSpent"].Points.AddXY(ED.Key, ED.Value);
        }

        con.Close(); // close database
    }
}

```

RefreshEventButton_Click Event Handler

When the RefreshEventButton is clicked by the user, the updatedLoad function called which parses the parameters "Events" which is used in the SQL query for the table name and EventsData which is the name of the DataGridView name.

```

private void RefreshEventButton_Click(object sender, EventArgs e)
{
    LDB.updatedLoad("Events", EventsData); // calls updatedLoad function from LoadDatabase class
}

```

AButton_Click Event Handler

When the AButton is clicked by the user the Add_Delete_Event form is shown.

```
private void AButton_Click(object sender, EventArgs e)
{
    Add_Delete_Event ADE = new Add_Delete_Event();
    ADE.Show();
}
```

Add Income

Form

The diagram shows a Windows form titled "Add Income Data". It contains several text boxes and a button, each with a label pointing to it. On the left side, labels include "DateLabel", "AmountReceivedLabel", "PaymentMethodLabel", "DescriptionLabel", and "FromLabel". On the right side, labels include "DateTextR", "AmountReceivedText", "PaymentMethodText", "DescriptionTextB", "FromTextR", and "AddButton". The form itself has the following fields: "Add Date (dd/mm/yyyy):", "Add the Amount Recieved (£00.00):", "Add the Payment Method:", "Add a Description (optional):", "Who is it from? :", and an "Add Details" button at the bottom.

Imports

```
using System;
using System.Windows.Forms;
using System.Data.SQLite;
```

Code

When the add button is clicked a new object is created using the DatabaseConnection class and then the getConnection function is called and then using a new SQLite connection using the connectionString from the DatabaseConnection class using the new object created, the connection is opened and new SQLite command is created. Within the command is an SQL query which inserts all the values that were inputted into the textboxes into the Income table. Then the connections is closed and a message should show to user saying that the data has been added.

```
namespace Business_Management_System__NEA_
{
    public partial class Add_Income : Form
    {
        public Add_Income()
        {
            InitializeComponent();
        }

        private void AddButton_Click(object sender, EventArgs e)
        {
            DatabaseConnection DBC = new DatabaseConnection(); // new object using DatabaseConnection class
            DBC.getConnection(); // connect to database

            using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
            {
                con.Open(); // open database
                // insert data in Income table
                SQLiteCommand cmd = new SQLiteCommand
                {
                    CommandText = @"INSERT INTO Income([Date], [AmountRecievedin£], PaymentMethod, Description, FromWho)
                                   VALUES (@Date, @AmountRecievedin£, @PaymentMethod, @Description, @FromWho)",
                    Connection = con
                };
                cmd.Parameters.AddWithValue("@Date", DateTextBox.Text);
                cmd.Parameters.AddWithValue("@AmountRecievedin£", AmountRecievedTextBox.Text);
                cmd.Parameters.AddWithValue("@PaymentMethod", PaymentMethodTextBox.Text);
                cmd.Parameters.AddWithValue("@Description", DescriptionTextBox.Text);
                cmd.Parameters.AddWithValue("@FromWho", FromTextBox.Text);
                cmd.ExecuteNonQuery();

                con.Close(); // close database
                MessageBox.Show("Data has been added"); // exception handling
                this.Close(); // close form
            }
        }
    }
}
```

Add Expenditure

Form

DateLabel

AmountSpentLabel

PaymentMethodLabel

DescriptionLabel

ToLabel

DateTextBox

AmountSpentTextBox

PaymentMethodTextBox

DescriptionTextBox

ToTextBox

AddButton

Add Date (dd/mm/yyyy):

Add the Amount Spent (£00.00):

Add the Payment Method:

Add a Description (optional):

Who did you Pay? :

Add Details

Imports

```
using System;  
using System.Windows.Forms;  
using System.Data.SQLite;
```

Code

When the add button is clicked a new object is created using the DatabaseConnection class and then the getConnection function is called and the using a new SQLite connection using the connectionString from the DatabaseConnection class using the new object created, the connection is opened and new SQLite command is created. Within the command is an SQL query which inserts all the values that were inputted into the textboxes into the Expenditure table. Then the connections is closed and a message should show to user saying that the data has been added.

```

namespace Business_Management_System__NEA_
{
    public partial class Add_Expenditure : Form
    {
        public Add_Expenditure()
        {
            InitializeComponent();
        }

        private void AddButton_Click(object sender, EventArgs e)
        {
            DatabaseConnection DBC = new DatabaseConnection(); // new object using DatabaseConnection class
            DBC.getConnection(); // connect to database

            using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
            {
                con.Open(); // open database
                // add data into Expenditure table
                SQLiteCommand cmd = new SQLiteCommand
                {
                    CommandText = @"INSERT INTO Expenditure([Date], [AmountSpentIn£], PaymentMethod, Description, ToWho)
                                   VALUES (@Date, @AmountSpentIn£, @PaymentMethod, @Description, @ToWho)",
                    Connection = con
                };
                cmd.Parameters.AddWithValue("@Date", DateTextBox.Text);
                cmd.Parameters.AddWithValue("@AmountSpentIn£", AmountSpentTextBox.Text);
                cmd.Parameters.AddWithValue("@PaymentMethod", PaymentMethodTextBox.Text);
                cmd.Parameters.AddWithValue("@Description", DescriptionTextBox.Text);
                cmd.Parameters.AddWithValue("@ToWho", ToTextBox.Text);
                cmd.ExecuteNonQuery();

                con.Close(); // close database
                MessageBox.Show("Data has been added"); // exception handling
                this.Close(); // close current form
            }
        }
    }
}

```

Add Stock

Form

ItemNameLabel → Add the Name of the Item: → ItemNameText

AmountInStockLabel → Add the Amount in Stock Currently: → AmountInStockText

SupplierNameLabel → Who is the Supplier? : → SupplierNameText

SupplierNameText → > → NextButton

DateLabel → The Date Today: → DateTextRo

Add Details → AddButton

Imports

```
using System;  
using System.Windows.Forms;  
using System.Data.SQLite;  
using System.Collections.Generic;
```

Globalised Properties

I have created three global properties. The first one is a new object created using the DatabaseConnection class. The second is a new list which will be used to store the name of each supplier. The third, and last, is a new enumerator which will allow the user to iterate through the list to find the correct supplier.


```

DatabaseConnection DBC = new DatabaseConnection();
public List<string> List = new List<string>(); // new public string list

IEnumerator<string> myEnumerator = null;

```

Form Initialiser

This is automatically generated which initialises the form. Within it I have a method, DBValues, which is being called. This method allows the name of the suppliers to be loaded

```

public Add_Stock()
{
    InitializeComponent();
    DBValues();
}

```

DBValues Function

This function essentially allow all the names of the suppliers to be loaded into the list in alphabetical order. This is done by getting the number of records within the Supplier table, storing that in a variable and then using that in a for loop to get each name and then store it in the list.

```

public void DBValues()
{
    DBC.getConnection(); // connect to database

    using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
    {
        con.Open(); // open database

        SQLiteCommand cmdCount = new SQLiteCommand(@"SELECT COUNT(*) FROM Supplier", con); // get number of rows in Supplier table
        int RecordCount = Convert.ToInt32(cmdCount.ExecuteScalar()); // store the number of rows

        for (int i = 1; i <= RecordCount; i++) // iterates through each record, stores value and add the value
        {
            SQLiteCommand cmd = new SQLiteCommand(string.Format(@"SELECT Name FROM Supplier WHERE SupplierID='{0}'", i), con);
            List.Add((string)cmd.ExecuteScalar()); // adds each value to list
        }

        List.Sort(); // sorts list alphabetically

        con.Close(); // close database
    }
}

```

GetNextItemToDisplay Function

This function allows for the iteration of the list on command. The function takes in a string list parameter. If the enumerator is null then enumerator will get an enumerator from the string list. The while loop then gets the next element or the first element depending on the situation

```
public string GetNextItemToDisplay(List<string> strList)
{
    if (myEnumerator == null) // not started
    {
        myEnumerator = strList.GetEnumerator(); // get enumerator
    }
    // get the next element (or if we haven't fetched anything yet: get the first element)
    while (myEnumerator.MoveNext())
    {
        // There is a next string. It is in Current:
        return myEnumerator.Current;
    }

    return null; // no strings left, return null
}
```

RestartEnumeration Function

This function will allow the list to restart from the beginning.

```
public void RestartEnumeration()
{
    myEnumerator = null;
}
```

NextButton_Click Event Handler

When the next button is clicked then a new string is declared which call the GetNextItemToDisplay function which parses the list. If the variable doesn't equal null then string stored in the variables is then visible in the textbox else if the variable does equal null then the RestartEnumeration() function is called.

```
private void NextButton_Click(object sender, EventArgs e)
{
    string nextItemToDisplay = GetNextItemToDisplay(List);
    if (nextItemToDisplay != null)
    {
        SupplierNameTextBox.Text = nextItemToDisplay;
    }
    else
    {
        RestartEnumeration();
    }
}
```

AddButton_Click Event Handler

When the add button is clicked, the system will check if the supplier name inputted into the textbox exists in the Supplier database. If the supplier name does exist within the table then all the values inputted into the textbox are added to the supplier table except for the supplier name. The system will identify the supplier name and get the supplier id and then that will be added into the table.

```
private void AddButton_Click(object sender, EventArgs e)
{
    DBC.getConnection();

    using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
    {
        con.Open(); // open database
        // select supplierID using inputted supplier name
        SQLiteCommand cmdSup = new SQLiteCommand(string.Format(@"SELECT SupplierID FROM Supplier WHERE Name='{0}'", SupplierNameTextBox.Text), con);
        SQLiteDataReader reader = cmdSup.ExecuteReader();

        if (reader.HasRows)
        {
            reader.Read();
            int sID = reader.GetInt32(0); // store PK value
            reader.Close(); // close data reader
            // insert data into Stock table
            SQLiteCommand cmd = new SQLiteCommand(@"INSERT INTO Stock(ItemName, AmountInStock, DataLastUpdated, SupplierID) VALUES (@Name, @AIS, @DLU, @S)", con);
            cmd.Parameters.AddWithValue("@Name", ItemNameTextBox.Text);
            cmd.Parameters.AddWithValue("@AIS", AmountInStkTextBox.Text);
            cmd.Parameters.AddWithValue("@DLU", DateTextBox.Text);
            cmd.Parameters.AddWithValue("@S", sID);
            cmd.ExecuteNonQuery();

            MessageBox.Show("Data has been added"); // exception handling
        }
        else // if supplier name inputted is not in Supplier table then shows following message and sends user to add upplier info
        {
            MessageBox.Show("Supplier doesn't exist. Please input the supplier details before adding stock data.");
            Suppliers sup = new Suppliers();
            this.Close(); // close form
            sup.Show(); // show supplier form
        }

        con.Close(); // close database
    }
}
```

Add-Delete Event

Form

The diagram illustrates the 'Add/Delete Event' form, which is divided into two main sections: 'Add Event' and 'Delete Event'.

Add Event Section:

- AddMonthCalendar:** Points to the calendar widget at the top of the 'Add Event' panel.
- AddDateTextBox:** Points to the text box labeled 'Select Date using Calendar Above'.
- AddEventTextBox:** Points to the text box labeled 'Add Event Name'.
- AddDetailsTextBox:** Points to the text box labeled 'Add the Event Details'.
- AddButton:** Points to the 'Add Event' button at the bottom of the 'Add Event' panel.

Delete Event Section:

- DeleteMonthCalendar:** Points to the calendar widget at the top of the 'Delete Event' panel.
- DeleteDateTextBox:** Points to the text box labeled 'Select Date using Calendar Above'.
- DeleteEventTextBox:** Points to the text box labeled 'Enter Event Name'.
- DeleteButton:** Points to the 'Delete Event' button at the bottom of the 'Delete Event' panel.

Imports

```
using System;  
using System.Windows.Forms;  
using System.Data.SQLite;
```

Globalised Property

```
DatabaseConnection DBC = new DatabaseConnection();
```

AddMonthCalendar_DateChanged Event Handler

```
private void AddMonthCalendar_DateChanged(object sender, DateRangeEventArgs e)
{
    AddDateTextBox.Text = AddMonthCalendar.SelectionStart.ToString(); // changes text to selected date
}
```

DeleteMonthCalendar_DateChanged Event Handler

```
private void DeleteMonthCalendar_DateChanged(object sender, DateRangeEventArgs e)
{
    DeleteDateTextBox.Text = DeleteMonthCalendar.SelectionStart.ToString(); // changes text to selected date
}
```

AddButton_Click Event Handler

When the add button is clicked then all the values that have been inputted into the textboxes will be add into the Events table.

```
private void AddButton_Click(object sender, EventArgs e)
{
    DBC.getConnection(); // call getConnection function from DatabaseConnection class

    using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
    {
        con.Open(); // open database
        // insert data into Events table
        SQLiteCommand cmd = new SQLiteCommand(@"INSERT INTO Events(EventName, EventDate, Details) VALUES (@EN, @ED, @EDetails)", con);
        cmd.Parameters.AddWithValue("@EN", AddEventTextBox.Text);
        cmd.Parameters.AddWithValue("@ED", AddDateTextBox.Text);
        cmd.Parameters.AddWithValue("@EDetails", AddDetailsTextBox.Text);
        cmd.ExecuteNonQuery();

        con.Close(); // close database
        MessageBox.Show("Event has been added");
    }
}
```

DeleteButton_Click Event Handler

When the delete button is clicked, the system will check use the inputted date is in the Events table. If it is not in the table then an error message will appear. If it is in the table then the event will be deleted.

```
private void DeleteButton_Click(object sender, EventArgs e)
{
    DBC.getConnection(); // call getConnection function from DatabaseConnection class

    using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
    {
        con.Open(); // open database

        SQLiteCommand cmdCheck = new SQLiteCommand(string.Format(@"SELECT EventDate FROM Events WHERE EventDate='{0}'", DeleteDateTextBox.Text), con);
        SQLiteDataReader rd = cmdCheck.ExecuteReader();

        int i = 0;
        while (rd.Read())
        {
            i++; // date is in table, i will increment
        }

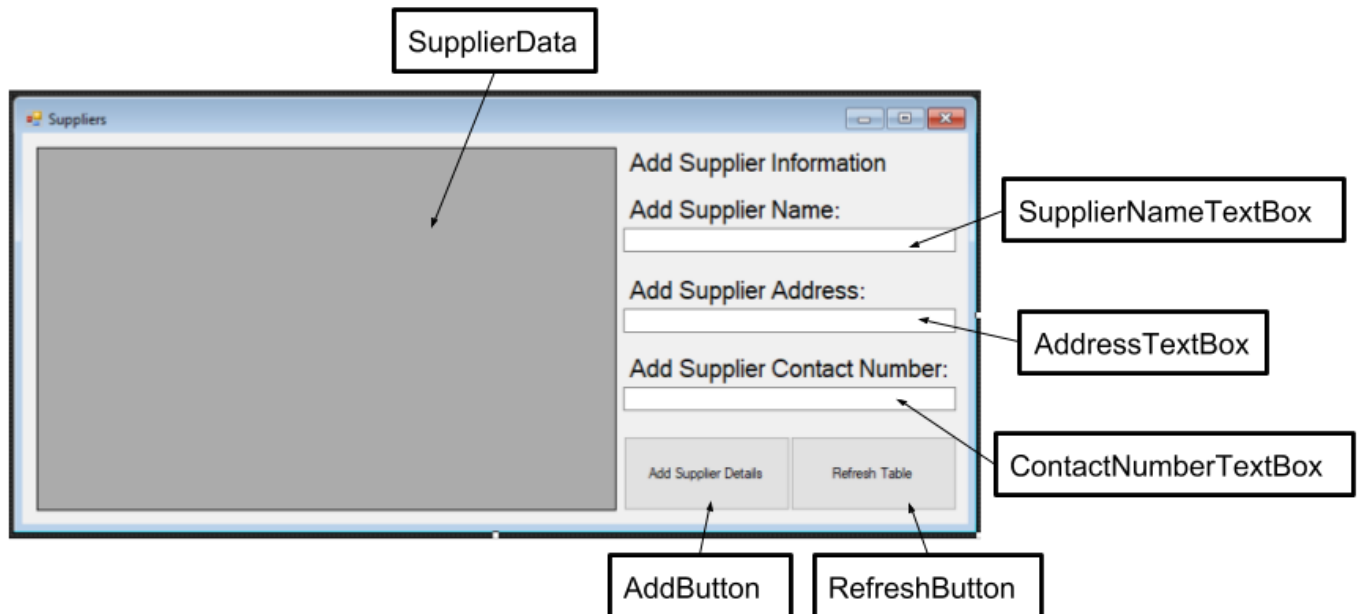
        if (i == 0) // date is not stored in database
        {
            MessageBox.Show("That date is not in the table. Please choose another date");
        }
        else // date is stored in database
        {
            // delete data from Events table
            SQLiteCommand cmd = new SQLiteCommand(@"DELETE FROM Events WHERE EventDate=@dte AND EventName=@EN", con);
            cmd.Parameters.AddWithValue("@dte", DeleteDateTextBox.Text);
            cmd.Parameters.AddWithValue("@EN", DeleteEventTextBox.Text);
            cmd.ExecuteNonQuery();

            MessageBox.Show("Event has been deleted");
        }

        con.Close(); // close database
    }
}
```

Supplier Page

Form



Imports

```
using System;  
using System.Windows.Forms;  
using System.Data.SQLite;
```

Globalised Property

```
LoadDatabase LDB = new LoadDatabase();
```

Suppliers_Load Event Handler

```
private void Suppliers_Load(object sender, EventArgs e)  
{  
    ...  
    LDB.storedLoad("Supplier", SupplierData); // load storedLoad function from LoadDatabase class  
}
```

AddSupplierButton_Click Event Handler

When the add supplier button is clicked, all the inputted values within the textboxes are added to the

database.

```
private void AddSupplierButton_Click(object sender, EventArgs e)
{
    DatabaseConnection DBC = new DatabaseConnection();
    DBC.getConnection();

    using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
    {
        con.Open(); // open database
        // insert data into Supplier table
        SQLiteCommand cmd = new SQLiteCommand(@"INSERT INTO Supplier(Name, Address, ContactNumber) VALUES (@Name, @Address, @CN)", con);
        cmd.Parameters.AddWithValue("@Name", SupplierNameTextBox.Text);
        cmd.Parameters.AddWithValue("@Address", AddressTextBox.Text);
        cmd.Parameters.AddWithValue("@CN", ContactNumberTextBox.Text);
        cmd.ExecuteNonQuery();

        con.Close(); // close database
        MessageBox.Show("Data has been added"); // exception handling
    }
}
```

RefreshButton_Click Event Handler

When the RefreshButton is clicked the updatedLoad function is called using the LDB object to refresh the supplier information

```
private void RefreshButton_Click(object sender, EventArgs e)
{
    LDB.updatedLoad("Supplier", SupplierData); // call updatedLoad function from LoadDatabase class
}
```

TESTING

Test Plan

* Test Video 1

** Test Video 2

***Test Video 3

Test No.	Description	Inputs/Actions	Expected Outcome	Timestamps	Pass/Fail
1	Attempt to login using default information	Username: 'admin' Password: 'password'	Should close the login page and open the welcome message	0:10**	Pass
2	Attempt to login using incorrect information	Username: 'admin' Password: 'pasw'	Should show the error message to the user	0:31**	Pass
3	Show error message	Incorrect username and password	Error message should show due to incorrect login information	0:32**	Pass
4.1	Attempt to login without a password	Username: 'admin' Password: <i>empty</i>	Should show a message asking the user to enter a password	1:05**	Pass
4.2	Attempt to login without a username	Username: <i>empty</i> Password: 'password'	Should show a message asking the user to enter a username	1:17**	Pass
5	Welcome page should show the name of the user	Enter correct details and welcome page will show	The welcome page would have the message: 'Welcome, <i>name of user</i> '	0:11**	Pass
6	Opening the register page	Clicking on the register button on the login screen	The register form will appear	1:34**	Pass
7	Adding a new user to the system	Username: 'test' Password: 'testing' Full Name: 'Test Test' Click the add button	A message will be shown to the user saying that the account was created	2:01**	Pass
8	Login with the new user account	Username: 'test' Password: 'testing'	The welcome page should appear.	2:22**	Pass
9.1	Entering only username into text box in the register form	Username: 'testing' Password: <i>empty</i> Full Name: <i>empty</i>	A message will show asking the user to enter a password and their full name	2:41**	Fail

		Click the add button			
9.2	Entering only password into textbox in the register form	Username: <i>empty</i> Password: 'testing' Fullname: <i>empty</i> Click the add button	A message will show asking the user to enter a username and their full name	2:49**	Fail
9.3	Entering only full name into textbox in register form	Username: <i>empty</i> Password: <i>empty</i> Fullname: 'Test Name' Click the add button	A message will show asking the user to enter a username and password	2:57**	Fail
9.4	Entering only username and password into text box in register form	Username: 'test' Password: 'testing' Fullname: <i>empty</i> Click the add button	A message will show asking the user to enter their full name	3:09**	Pass
9.5	Entering only username and full name into text box in register form	Username: 'test' Password: <i>empty</i> Fullname: 'Test Name' Click the add button	A message will show asking the user to add a password	3:22**	Pass
9.6	Entering only password and full name into text box in register form	Username: <i>empty</i> Password: 'testing' Fullname: 'Test Name' Click the add button	A message will show asking the user to enter a username	3:35**	Pass
9.7	Entering nothing in the register form	All text boxes are empty Click the add button	A message will show asking the user to enter a username, password and their full name	3:46**	Fail
10.1	Entering the username as a space character in the register form	Username: <i>space</i> Password: 'testing' Full Name: 'Space Test' Click the add button	The account will be created using the username inputted by the user	4:04**	Pass
10.2	Attempting to login using the username: <i>space</i>	Username: <i>space</i> Password: 'testing'	The welcome page would appear	4:32**	Pass
11.1	Entering the password as multiple space characters	Username: 'spaceTest' Password: <i>spacespacespace</i> Full Name: 'T. Space'	The account will be created using the password inputted by the user	5:06**	Pass
11.2	Attempting to login using the account with the	UserName: 'spaceTest' Password:	The welcome page would appear	5:28**	Pass

	password with <i>spacespacespace</i>	<i>spacespacespace</i>			
12	Opening the StreamlineWindow (Main Window)	Clicking continue on the Welcome Page	Open the main window to the user and close the welcome page	0:17**	Pass
13	Viewing the Events table	<i>Should show up when the main window opens</i>	The table should appear within the data grid view	1:29*	Pass
14	Opening the Add/Delete Event Form	Clicking the Add/Delete Event button	Open the form that will allow the user to add or delete an event	2:26*	Pass
15	Adding a date into the text box in the add event group box	Clicking on a date using the Month Calendar in the Add Event Section	Whatever date is clicked by the user, that date will be added into the textbox	2:00*	Pass
16	Adding a new event	Date: <i>any date</i> Event Name: 'Test Event' Event Details: 'This is a Test Event' Click on add button	A message will be shown to the user that the event has been added	1:55*	Pass
17	Refreshing the Data Grid View with the added data within the table	Clicking on the refresh button in the events group box	The table should appear with the new data added	2:24*	Pass
18	Deleting an event that exist in the table Events	Date and event name that exists within the table Click the delete button	A message will be shown to the user that the event had been deleted	2:49*	Pass
19	Refreshing the Data Grid View with the deleted data within the table	Clicking on the refresh button in the events group box	The table should appear with the data deleted	2:52*	Pass
20.1	Attempting to delete an event that doesn't exist within the table Events	Date: <i>a date that doesn't exist within the table</i> Event Name: <i>a name that doesn't exist within the table.</i> Click on the delete button	A message will be shown to the user saying that the event doesn't exist.	2:38*	Fail

20.2	Attempting to delete an event using a correct date but incorrect name	Date: <i>a date that exists within the table</i> Event Name: <i>a name that doesn't exist within the table.</i> Click on the delete button	A message will be shown to the user stating that the event name is wrong for the date entered.	2:38*	Fail
21	Show the Transactions section	Clicking on the transaction button	The transaction section should open up	2:54*	Pass
22	Show the Events section	Clicking on the events button	The events section should open up	0:27***	Pass
23	View the Income section	Clicking on the income tab	The income section should show	0:22***	Pass
24	Open the Add Income page	Clicking on the add income button	The add income form should open up	2:56*	Pass
25	Add valid income data to the Income table	<i>Check Test Video 1</i>	A message should show saying that the data has been added	3:07*	Pass
26	Attempting to the add income data with at least one empty textbox that is not the description textbox	<i>Check Test Video 1</i>	A message should show to the user asking them to input data into the corresponding text boxes	3:07*	Fail
27	Refresh the Data Grid View in the Income tab with the new income data	Click on the refresh button in the income tab	The data grid view should show the Expenditure table with the newly added data	3:09	Pass
28	Show the Expenditure section	Click on the expenditure tab	The expenditure section show show	3:42*	Pass
29	Open the Add Expenditure Form	Click on the add expenditure button	The add expenditure form should be shown to the user	3:42*	Pass
30	Add valid expenditure data to the Expenditure table	<i>Check Test Video 1</i>	A message should appear to the user saying that the data has been added	3:58*	Pass

31	Attempting to the add expenditure data with at least one empty textbox that is not the description textbox	<i>Check Test Video 1</i>	A message should appear to the user asking them to enter information into the empty textboxes	3:58*	Fail
32	Refresh data grid view with table containing new expenditure data	Click the refresh button in the expenditure tab	The data grid view should show the Expenditure table with the newly added data	4:01*	Pass
33	Show the Stocks section	Click on the stocks tab	The stock section should show	4:12*	Pass
34	Open the add stock page	Click on the add stock button	The add stock form should show to the user	4:17*	Pass
35	Attempting to iterate through the supplier names	Clicking on the next button (has this symbol '>')	The supplier name should appear in the textbox and if clicked again the next supplier name should appear, so on and so forth.	4:18*	Fail
36	Add stock data to the Stock table	<i>Check Test Video 1</i>	A message will be shown saying that the data has been added	5:12*	Pass
37	Attempting to add stock data with a supplier that doesn't exist within the Supplier table	<i>Check Test Video 1</i>	A message should show to the user saying that the supplier doesn't exist and to add the supplier to the table. The user will, then, be directed to the supplier page	5:29*	Pass
38	Adding new stock data with the new supplier name	<i>Check Test Video 1</i>	A message should be shown to the user saying that the data has been added	6:48*	Pass
39	Refresh data grid view with table containing new stock data	Click on the refresh button in the stock tab	The data grid view should show the Stock table with the newly added data	5:15*	Pass
40	Open the Supplier Page	Click on the supplier button in the stock tab	The supplier page should be shown to the user	4:43*	Pass
41	Add supplier data to the Supplier table	<i>Check Test Video 1</i>	A message should show to the user stating that the data has been added	6:25*	Pass

42	Refresh data grid view with table containing new supplier data	Click the refresh button in the supplier form	The data grid view should show the Supplier table with the newly added data	6:28*	Pass
43	Show the Reports section	Click on the reports button	The Reports group box show should appear and the income tab should be open with the income chart loaded with the data.	3:35*	Pass
44	Refresh the income chart using new income data	Click the refresh button after adding income data	The chart should refresh with the new data	3:37*	Pass
45	Show the expenditure section	Click the expenditure tab	The section should appear with the expenditure chart.	4:06*	Pass
46	Refresh the expenditure chart using new expenditure data	Click the refresh button after adding expenditure data	The chart should refresh with the new data	4:07*	Pass
47	Logout	Click the logout button	The current window should close and the login page should appear	7:04*	Pass
48	Attempting to add stock data with at least one empty textbox	<i>Check Test Video 3</i>	A message should be shown to the user asking the to input data into the textboxes	0.52***	Fail

Evidence

All evidence is in the videos that will be supplied to you.

Improvements Made

Changes to Code for Tests 9.1, 9.2, 9.3 & 9.7

I have changed the code from this:

```

private void RegisterButton_Click(object sender, EventArgs e)
{
    if (txtBoxUsername.Text != string.Empty && txtBoxPassword.Text != string.Empty && txtBoxFullName.Text != string.Empty)
    {
        checkLoginInfo(txtBoxUsername.Text, txtBoxPassword.Text); // calls checkLoginInfo function
    }
    else if (txtBoxUsername.Text != string.Empty && txtBoxPassword.Text == string.Empty && txtBoxFullName.Text != string.Empty)
    {
        MessageBox.Show("Enter Password"); // exception handling
    }
    else if (txtBoxUsername.Text == string.Empty && txtBoxPassword.Text != string.Empty && txtBoxFullName.Text != string.Empty)
    {
        MessageBox.Show("Enter Username"); // exception handling
    }
    else if (txtBoxUsername.Text != string.Empty && txtBoxPassword.Text != string.Empty && txtBoxFullName.Text == string.Empty)
    {
        MessageBox.Show("Enter Your Full Name"); // exception handling
    }
}

```

Which tests for specific empty text box variations to this:

```

private void RegisterButton_Click(object sender, EventArgs e)
{
    if (txtBoxUsername.Text != string.Empty && txtBoxPassword.Text != string.Empty && txtBoxFullName.Text != string.Empty)
    {
        checkLoginInfo(txtBoxUsername.Text, txtBoxPassword.Text); // calls checkLoginInfo function
    }
    else
    {
        foreach (Control c in this.Controls)
        {
            if (c is TextBox)
            {
                TextBox textBox = c as TextBox;
                if (textBox.Text == string.Empty)
                {
                    if (textBox == txtBoxUsername)
                    {
                        MessageBox.Show("Enter Username");
                    }
                    else if (textBox == txtBoxPassword)
                    {
                        MessageBox.Show("Enter Password");
                    }
                    else if (textBox == txtBoxFullName)
                    {
                        MessageBox.Show("Enter Full Name");
                    }
                }
            }
        }
    }
}

```

The original IF statement has stayed the same however, I have removed the multiple ELSE IF statements for an ELSE statement. So if all the textboxes are not empty then the checkLoginInfo function will be called, same as before, but if even one of those textboxes are empty then a FOREACH loop will start for each control in the form and if the control is a textbox then a new textbox object is created which is the textbox that is being scanned. Then if that textbox object is empty, then it will check which one is empty and display the correct message(s).

Changes to Code for Test 20.1 & 20.2

I have changed the code from this:

```
private void DeleteButton_Click(object sender, EventArgs e)
{
    DBC.getConnection(); // call getConnection function from DatabaseConnection class

    using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
    {
        con.Open(); // open database

        SQLiteCommand cmdCheck = new SQLiteCommand(string.Format(@"SELECT EventDate FROM Events WHERE EventDate='{0}'", DeleteDateTextBox.Text), con);
        SQLiteDataReader rd = cmdCheck.ExecuteReader();

        int i = 0;
        while (rd.Read())
        {
            i++; // date is in table, i will increment
        }

        if (i == 0) // date is not stored in database
        {
            MessageBox.Show("That date is not in the table. Please choose another date");
        }
        else // date is stored in database
        {
            // delete data from Events table
            SQLiteCommand cmd = new SQLiteCommand(@"DELETE FROM Events WHERE EventDate=@dte AND EventName=@EN", con);
            cmd.Parameters.AddWithValue("@dte", DeleteDateTextBox.Text);
            cmd.Parameters.AddWithValue("@EN", DeleteEventTextBox.Text);
            cmd.ExecuteNonQuery();

            MessageBox.Show("Event has been deleted");
        }
    }

    con.Close(); // close database
}
```

Which checks if the date is within the table to this:

The original SQL query (cmdCheck) only checked if the date was in the table, but now I have changed it such that the SQL query now checks to see if the date and event are within the table and within the same record. The only other piece of code that is changed is the error message that is given to the user


```

private void DeleteButton_Click(object sender, EventArgs e)
{
    DBC.getConnection(); // call getConnection function from DatabaseConnection class

    using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
    {
        con.Open(); // open database

        SQLiteCommand cmdCheck = new SQLiteCommand(@"SELECT EventDate, EventName FROM Events WHERE EventDate=@d AND EventName=@e", con);
        cmdCheck.Parameters.AddWithValue("@d", DeleteDateTextBox.Text);
        cmdCheck.Parameters.AddWithValue("@e", DeleteEventTextBox.Text);

        SQLiteDataReader rd = cmdCheck.ExecuteReader();

        int i = 0;
        while (rd.Read())
        {
            i++; // date & name is in table, i will increment
        }

        if (i == 0) // date & name is not stored in database
        {
            MessageBox.Show("That event is not in the table. Please select a valid date and/or event name.");
        }
        else // date & name is stored in database
        {
            // delete data from Events table
            SQLiteCommand cmd = new SQLiteCommand(@"DELETE FROM Events WHERE EventDate=@dte AND EventName=@EN", con);
            cmd.Parameters.AddWithValue("@dte", DeleteDateTextBox.Text);
            cmd.Parameters.AddWithValue("@EN", DeleteEventTextBox.Text);
            cmd.ExecuteNonQuery();

            MessageBox.Show("Event has been deleted");
        }

        con.Close(); // close database
    }
}

```

when the event is invalid.

Changes to Code for Test 26

I have changed the code from this:

```

namespace Business_Management_System_NEA_
{
    public partial class Add_Income : Form
    {
        public Add_Income()
        {
            InitializeComponent();
        }

        private void AddButton_Click(object sender, EventArgs e)
        {
            DatabaseConnection DBC = new DatabaseConnection(); // new object using DatabaseConnection class
            DBC.getConnection(); // connect to database

            using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
            {
                con.Open(); // open database
                // insert data in Income table
                SQLiteCommand cmd = new SQLiteCommand
                {
                    CommandText = @"INSERT INTO Income([Date], [AmountRecievedin£], PaymentMethod, Description, FromWho)
                                   VALUES (@Date, @AmountRecievedin£, @PaymentMethod, @Description, @FromWho)",
                    Connection = con
                };
                cmd.Parameters.AddWithValue("@Date", DateTextBox.Text);
                cmd.Parameters.AddWithValue("@AmountRecievedin£", AmountRecievedTextBox.Text);
                cmd.Parameters.AddWithValue("@PaymentMethod", PaymentMethodTextBox.Text);
                cmd.Parameters.AddWithValue("@Description", DescriptionTextBox.Text);
                cmd.Parameters.AddWithValue("@FromWho", FromTextBox.Text);
                cmd.ExecuteNonQuery();

                con.Close(); // close database
                MessageBox.Show("Data has been added"); // exception handling
                this.Close(); // close form
            }
        }
    }
}

```

Which when the add button is clicked, even if the textboxes are empty, the values are added to the table. To this piece of code:

```

private void AddButton_Click(object sender, EventArgs e)
{
    if (DateTextBox.Text != string.Empty && AmountRecievedTextBox.Text != string.Empty && PaymentMethodTextBox.Text != string.Empty &&
        FromTextBox.Text != string.Empty)
    {
        DatabaseConnection DBC = new DatabaseConnection(); // new object using DatabaseConnection class
        DBC.getConnection(); // connect to database

        using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
        {
            con.Open(); // open database
            // insert data in Income table
            SQLiteCommand cmd = new SQLiteCommand
            {
                CommandText = @"INSERT INTO Income([Date], [AmountRecievedin£], PaymentMethod, Description, FromWho)
                               VALUES (@Date, @AmountRecievedin£, @PaymentMethod, @Description, @FromWho)",
                Connection = con
            };
            cmd.Parameters.AddWithValue("@Date", DateTextBox.Text);
            cmd.Parameters.AddWithValue("@AmountRecievedin£", AmountRecievedTextBox.Text);
            cmd.Parameters.AddWithValue("@PaymentMethod", PaymentMethodTextBox.Text);
            cmd.Parameters.AddWithValue("@Description", DescriptionTextBox.Text);
            cmd.Parameters.AddWithValue("@FromWho", FromTextBox.Text);
            cmd.ExecuteNonQuery();

            con.Close(); // close database
            MessageBox.Show("Data has been added"); // exception handling
            this.Close(); // close form
        }
    }
    else
    {
        foreach (Control c in this.Controls)
        {
            if (c is TextBox)
            {
                TextBox textBox = c as TextBox;
                if (textBox.Text == string.Empty)
                {
                    if (textBox == DateTextBox)
                    {
                        MessageBox.Show("Enter Date");
                    }
                    else if (textBox == AmountRecievedTextBox)
                    {
                        MessageBox.Show("Enter the Amount Recieved");
                    }
                    else if (textBox == PaymentMethodTextBox)
                    {
                        MessageBox.Show("Enter the Payment Method");
                    }
                    else if (textBox == FromTextBox)
                    {
                        MessageBox.Show("Enter who the payment was from");
                    }
                }
            }
        }
    }
}

```

I have added in an IF statement. If the textboxes are not empty then the inputted values are added to the table but if even one of those textboxes are empty then a FOREACH loop will start for each control in the form and if the control is a textbox then a new textbox object is created which is the textbox that is being scanned. Then if that textbox object is empty, then it will check which one is empty and display the correct message(s).

Changes to Code for Test 31

I have changes the code from this:

```
namespace Business_Management_System__NEA_
{
    public partial class Add_Expenditure : Form
    {
        public Add_Expenditure()
        {
            InitializeComponent();
        }

        private void AddButton_Click(object sender, EventArgs e)
        {
            DatabaseConnection DBC = new DatabaseConnection(); // new object using DatabaseConnection class
            DBC.getConnection(); // connect to database

            using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
            {
                con.Open(); // open database
                // add data into Expenditure table
                SQLiteCommand cmd = new SQLiteCommand
                {
                    CommandText = @"INSERT INTO Expenditure([Date], [AmountSpentIn£], PaymentMethod, Description, ToWho)
                                   VALUES (@Date, @AmountSpentIn£, @PaymentMethod, @Description, @ToWho)",
                    Connection = con
                };
                cmd.Parameters.AddWithValue("@Date", DateTextBox.Text);
                cmd.Parameters.AddWithValue("@AmountSpentIn£", AmountSpentTextBox.Text);
                cmd.Parameters.AddWithValue("@PaymentMethod", PaymentMethodTextBox.Text);
                cmd.Parameters.AddWithValue("@Description", DescriptionTextBox.Text);
                cmd.Parameters.AddWithValue("@ToWho", ToTextBox.Text);
                cmd.ExecuteNonQuery();

                con.Close(); // close database
                MessageBox.Show("Data has been added"); // exception handling
                this.Close(); // close current form
            }
        }
    }
}
```

Which when the add button is clicked, even if the textboxes are empty, the values are added to the table. To this piece of code:

I have added in an IF...ELSE statement. So if the textboxes are not empty then the inputted values are added to the table, using the SQL query but if even one of those textboxes are empty then a FOREACH loop will start for each control in the form and if the control is a textbox then a new textbox object is created which is the textbox that is being scanned. Then if that textbox object is empty, then it will

```

private void AddButton_Click(object sender, EventArgs e)
{
    if (DateTextBox.Text != string.Empty && AmountSpentTextBox.Text != string.Empty && PaymentMethodTextBox.Text != string.Empty && ToTextBox.Text != string.Empty)
    {
        DatabaseConnection DBC = new DatabaseConnection(); // new object using DatabaseConnection class
        DBC.getConnection(); // connect to database

        using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
        {
            con.Open(); // open database
            // add data into Expenditure table
            SQLiteCommand cmd = new SQLiteCommand
            {
                CommandText = @"INSERT INTO Expenditure([Date], [AmountSpentIn£], PaymentMethod, Description, ToWho)
                               VALUES (@Date, @AmountSpentIn£, @PaymentMethod, @Description, @ToWho)",
                Connection = con
            };
            cmd.Parameters.AddWithValue("@Date", DateTextBox.Text);
            cmd.Parameters.AddWithValue("@AmountSpentIn£", AmountSpentTextBox.Text);
            cmd.Parameters.AddWithValue("@PaymentMethod", PaymentMethodTextBox.Text);
            cmd.Parameters.AddWithValue("@Description", DescriptionTextBox.Text);
            cmd.Parameters.AddWithValue("@ToWho", ToTextBox.Text);
            cmd.ExecuteNonQuery();

            con.Close(); // close database
            MessageBox.Show("Data has been added"); // exception handling
            this.Close(); // close current form
        }
    }
}

```

```

else
{
    foreach (Control c in this.Controls)
    {
        if (c is TextBox)
        {
            TextBox textBox = c as TextBox;
            if (textBox.Text == string.Empty)
            {
                if (textBox == DateTextBox)
                {
                    MessageBox.Show("Enter Date");
                }
                else if (textBox == AmountSpentTextBox)
                {
                    MessageBox.Show("Enter the Amount Spent");
                }
                else if (textBox == PaymentMethodTextBox)
                {
                    MessageBox.Show("Enter the Payment Method");
                }
                else if (textBox == ToTextBox)
                {
                    MessageBox.Show("Enter who the payment was for");
                }
            }
        }
    }
}

```

check which one is empty and display the correct message(s).

Changes to Code for Test 35

Originally I had this piece of code:

```
private void NextButton_Click(object sender, EventArgs e)
{
    string nextItemToDisplay = GetNextItemToDisplay(List);
    if (nextItemToDisplay != null)
    {
        SupplierNameTextBox.Text = nextItemToDisplay;
    }
    else
    {
        RestartEnumeration();
    }
}
```

Which would allow the user to iterate through the list of name. However the problem was that if the first value within the list was null, then the RestartEnumeration function would be called over and over again, so I added this piece of code:

```
private void NextButton_Click(object sender, EventArgs e)
{
    // this checks to see if a null value is in the list, if it is then it will be removed.
    var temp = new List<string>();
    foreach (var s in List)
    {
        if (!string.IsNullOrEmpty(s))
        {
            temp.Add(s);
        }
    }
    List = temp.ToList();

    string nextItemToDisplay = GetNextItemToDisplay(List);

    if (nextItemToDisplay != null)
    {
        SupplierNameTextBox.Text = nextItemToDisplay;
    }
    else
    {
        RestartEnumeration();
    }
}
```

The piece of code I added checks if the list has any null values and if it does then it would remove it, so given that the value was null at the start, that would be removed. This then allows the user to iterate through the list with no problem.

Changes to Code for Test 48

Originally I had this piece of code:

```
private void AddButton_Click(object sender, EventArgs e)
{
    DBC.getConnection();

    using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
    {
        con.Open(); // open database
        // select supplierID using inputted supplier name
        SQLiteCommand cmdSup = new SQLiteCommand(string.Format(@"SELECT SupplierID FROM Supplier WHERE Name='{0}'", SupplierNameTextBox.Text), con);
        SQLiteDataReader reader = cmdSup.ExecuteReader();

        if (reader.HasRows)
        {
            reader.Read();
            int sID = reader.GetInt32(0); // store PK value
            reader.Close(); // close data reader
            // insert data into Stock table
            SQLiteCommand cmd = new SQLiteCommand(@"INSERT INTO Stock(ItemName, AmountInStock, DataLastUpdated, SupplierID) VALUES (@Name, @AIS, @DLU, @S)", con);
            cmd.Parameters.AddWithValue("@Name", ItemNameTextBox.Text);
            cmd.Parameters.AddWithValue("@AIS", AmountInStkTextBox.Text);
            cmd.Parameters.AddWithValue("@DLU", DateTextBox.Text);
            cmd.Parameters.AddWithValue("@S", sID);
            cmd.ExecuteNonQuery();

            MessageBox.Show("Data has been added"); // exception handling
        }
        else // if supplier name inputted is not in Supplier table then shows following message and sends user to add upplier info
        {
            MessageBox.Show("Supplier doesn't exist. Please input the supplier details before adding stock data.");
            Suppliers sup = new Suppliers();
            this.Close(); // close form
            sup.Show(); // show supplier form
        }

        con.Close(); // close database
    }
}
```

Which would check if the supplier name existed within the table, if it didn't the user would be told to add the supplier, via a message box, and then they would be redirected to the supplier page where they can add the new supplier. However this code doesn't check to see if any of the textboxes are empty. I then modified the code to do exactly that below:

In the above code, I have added the functionality of being able to check whether any of the textboxes

```

private void AddButton_Click(object sender, EventArgs e)
{
    if (ItemNameTextBox.Text != string.Empty && AmountInStkTextBox.Text != string.Empty && SupplierNameTextBox.Text != string.Empty && DateTextBox.Text != string.Empty)
    {
        DBC.getConnection();

        using (SQLiteConnection con = new SQLiteConnection(DBC.connectionString))
        {
            con.Open(); // open database
            // select supplierID using inputted supplier name
            SQLiteCommand cmdSup = new SQLiteCommand(string.Format(@"SELECT SupplierID FROM Supplier WHERE Name='{0}'", SupplierNameTextBox.Text), con);
            SQLiteDataReader reader = cmdSup.ExecuteReader();

            if (reader.HasRows)
            {
                reader.Read();
                int sID = reader.GetInt32(0); // store PK value
                reader.Close(); // close data reader
                // insert data into Stock table
                SQLiteCommand cmd = new SQLiteCommand(@"INSERT INTO Stock(ItemName, AmountInStock, DataLastUpdated, SupplierID) VALUES (@Name, @AIS, @DLU, @S)", con);
                cmd.Parameters.AddWithValue("@Name", ItemNameTextBox.Text);
                cmd.Parameters.AddWithValue("@AIS", AmountInStkTextBox.Text);
                cmd.Parameters.AddWithValue("@DLU", DateTextBox.Text);
                cmd.Parameters.AddWithValue("@S", sID);
                cmd.ExecuteNonQuery();

                MessageBox.Show("Data has been added"); // exception handling
            }
            else // if supplier name inputted is not in Supplier table then shows following message and sends user to add upplier info
            {
                MessageBox.Show("Supplier doesn't exist. Please input the supplier details before adding stock data.");
                Suppliers sup = new Suppliers();
                this.Close(); // close form
                sup.Show(); // show supplier form
            }

            con.Close(); // close database
        }
    }
    else
    {
        foreach (Control c in this.Controls)
        {
            if (c is TextBox)
            {
                TextBox textBox = c as TextBox;
                if (textBox.Text == string.Empty)
                {
                    if (textBox == ItemNameTextBox)
                    {
                        MessageBox.Show("Enter Item Name");
                    }
                    else if (textBox == AmountInStkTextBox)
                    {
                        MessageBox.Show("Enter the Amount in Stock");
                    }
                    else if (textBox == SupplierNameTextBox)
                    {
                        MessageBox.Show("Enter the name of the Supplier");
                    }
                    else if (textBox == DateTextBox)
                    {
                        MessageBox.Show("Enter the date");
                    }
                }
            }
        }
    }
}

```

are empty, if they are empty then the corresponding message will be shown to the user.

Testing

All the evidence is within the 'Improvement Test Video 1' video and all the timestamps are from that video

Test No.	Description	Expected Outcome	Timestamps	Pass/Fail
1.1	Attempting to create an account by only inputting a username	The user will get a message asking them to input a password and their full name	0:09	Pass
1.2	Attempting to create an account by only inputting a username and password	The user will get a message asking them to input their full name	0:15	Pass
1.3	Attempting to create an account by only inputting a username and a full name	The user will get a message asking them to enter a password	0:25	Pass
1.4	Attempting to register an account by inputting a full name	The user will get a message asking them to enter a username and password	0:28	Pass
1.5	Attempting to register an account by leaving all the textboxes blank	The user will get a message asking them to enter a username, password and their full name	0:33	Pass
2.1	Attempting to delete an event by leaving all the textboxes blank	The user will get a message saying that the event doesn't exist	0:48	Pass
2.2	Attempting to delete an event by inputting just a date		0:58	Pass
2.3	Attempting to delete an event by inputting an incorrect date and name		1:04	Pass
2.4	Attempting to delete an event by inputting an incorrect name		1:08	Pass
3.1	Attempting to add income data by leaving all the textboxes blank	The user will receive a message asking them to enter the date, amount received, payment method and who it is from	1:20	Pass
3.2	Attempting to add income data by inputting a date and the amount received	The user will receive a message asking them to enter the payment method and who it is from	1:30	Pass
3.3	Attempting to add income data by inputting a date and who it is from	The user will receive a message asking them to enter the amount received and payment	1:38	Pass

		method		
3.4	Attempting to add income data by inputting who it is from	The user will receive a message asking them to enter the date, amount received and payment method	1:44	Pass
4.1	Iterating through the list of supplier names	When the next button is clicked then the name of the supplier should appear	2:00	Pass
4.2	Add a new supplier to the table	A message should show to the user saying that the data has been added	2:24	Pass
4.3	Iterate through the list of supplier names with the newly added supplier within that list	When the next button is clicked then the name of the new supplier should appear at the relative position due to the list being sorted alphabetically	2:41	Pass
5.1	Attempting to add stock data by leaving all the text boxes empty	The user will receive a message asking them to input the name of the item, amount in stock, the name of the supplier and the current date	3:15	Pass
5.2	Attempting to add stock data by inputting only the name of the item	The user will receive a message asking them to input the amount in stock, the name of the supplier and the current date	3:23	Pass
5.3	Attempting to add stock data by inputting the name of the item and the amount in stock currently	The user will receive a message asking them to input the name of the supplier and the current date	3:29	Pass
5.4	Attempting to add stock data by inputting the name of the item, the amount in stock currently and the current date.	The user will receive a message asking them to input the name of the supplier	3:33	Pass
6	Attempting to the add stock data with the incorrect supplier name	The user will receive a message stating that the supplier doesn't exist within the table. The user will then be redirected to the supplier page where they can add the new supplier.	3:37	Pass

Evidence

All evidence is in the video that will be supplied to you.

EVALUATION

Objective Analysis

Objective	Met?	Comment
The business management system should be able to:		
Store all details of the business' transactions	Yes	When the main screen is loaded, the user is able to view transactions through the use of multiple different filters to view them in a table format
Use the transaction details and produce a visual representation of it	Yes	When the main screen is loaded the user is able to view a reports section and view the graphs created and data stored within the table
Store details of multiple user	Yes	When the user logs in, the verification of the user details and queries which allow communication to the database. If the user tries to register an account then the code is generated. If the account is already registered and if not then a new account is created
Give user roles based on their authority allowing them access to required parts of the system	No	I have not been able to create a system where users have different roles based on their employee roles. This is due to the complexity that comes with creating a system like this.
Communicate with other user within the business	No	I have not been able to create a system where users can communicate with each other within the business. This is due to the complexity that comes with creating a system like this.
Allow the user to add important dates to the calendar	Yes	When the main screen is loaded, a calendar is displayed on the left hand side, a specific events section is viewable on the right hand side. The user can view the events that are stored in the table and add new events to add and delete their own events
The business management system should overall:		
Be displayed as GUI	Yes	I have been able to create the entire system and display it as a GUI Form

Startup with login screen	Yes	When I started the system, the first form created the login page and the automatically the IDE set that as the first form.
Stay at a fixed resolution	Yes & No	I have created the entire system with custom dimensions and when the each form appears in the custom dimension, however the forms can be resized but the dimensions of the forms but the components (textboxes, labels, buttons etc) will not change their positions.

Possible Improvements

After going through the objectives and identifying what has been met. I found that I was unable to add the ability for users to have a user role which later allow them to access different parts of the system. I also found that I was unable to create the system with the ability to communicate with other employees within the business.

So if I was to continue this project, I would like to add the ability for user to have roles within the system. This would most likely require a new table, within the database, for the different types roles the business has. I would then also like to add the ability for users to be able to communicate with other users within the system.

After giving my system to a third-party, they pointed out a few features in the system that would need to be improved or features that should be added into the system. One feature, they pointed out, that could be improved is when viewing the chart produced by the system, the data seems to be ordered by when it was added in the table, e.g. the first piece of data added to the table is at the beginning of the chart and the last piece of data is at the end, rather than in date order. Another feature that could be improved is the ability to update the stock data. Currently the only way to update stock data is by adding it brand new. So one way of potentially fixing this problem is by using an UPDATE statement in the SQL query to update the specific rows. A similar issue came up with trying to update supplier information as currently there is no way to update the specific row, the user would have to add the supplier again with the new information. So to potentially fix this problem, just like before, I could use an UPDATE statement within the SQL query to update the specific row.

One feature they pointed out that is missing from the system is the ability to remove suppliers from the Supplier table. One of the problems for adding a feature like this is that when, for example, a supplier is deleted from the table, the primary key is no longer in order, i.e. 1, 2, 3... , and this can cause a problem when trying to load the list into the add stock form as supplier names can be missing and won't show up when iterating through the list. So potentially I could fix this problem by copying all the details into a new Supplier table and deleting the old table from the database so that primary key are in order.

Another feature they pointed out that could be added into the system, is the ability to calculate the profit and loss of the business in a specific period of time. I could potentially implement this by having another table which would take the values in the income table and the expenditure table and then calculate a value from that.

REFERENCES

<https://www.youtube.com/watch?v=ayp3tHEkRc0&t=441s>

https://www.youtube.com/watch?v=S5bdjxb_LDk&t=866s

<https://www.youtube.com/watch?v=83XnLXRwmbU&t=526s>

https://www.youtube.com/watch?v=Hm_0ucMjRZY&list=PLAS5k1bY27h7ekvGKjsVRhkcM4YEImwiW

<https://www.youtube.com/watch?v=Sm5mxkytfWk>

<https://www.youtube.com/watch?v=NORaHTeW9Xk&t=276s>

<https://stackoverflow.com/questions/8814811/remove-blank-values-in-the-array-using-c-sharp>

<https://stackoverflow.com/questions/54931520/data-is-not-showing-updating-in-the-sqlite-table-when-the-data-is-inserted-throu>

<https://stackoverflow.com/questions/18532691/how-do-i-write-a-backslash-in-a-string>

<https://stackoverflow.com/questions/53988351/how-to-fix-sqlite-exception-insufficient-parameters-supplied-to-the-command-w>

<https://stackoverflow.com/questions/54988572/sqlite-c-sharp-error-system-data-sqlite-sqliteexception-sql-logic-error-near>

<https://stackoverflow.com/questions/55068155/sql-error-regarding-foreign-key-restraint>

<https://stackoverflow.com/questions/55029183/sqlite-c-sharp-error-system-data-sqlite-sqliteexception-sql-logic-error-unrec>

<https://stackoverflow.com/questions/7558651/logout-the-main-form-and-show-the-login-form>

<https://stackoverflow.com/questions/962032/how-do-i-count-the-number-of-rows-retuned-in-my-sqlite-reader-in-c>

<https://stackoverflow.com/questions/31881061/adding-x-axis-string-value-in-a-chart-instead-of-number>

<https://stackoverflow.com/questions/22471648/how-to-access-textbox-text-in-another-form>

<https://stackoverflow.com/questions/28246079/why-do-i-get-invalidoperationexception-no-current-row-with-this-code/29806935>

<https://stackoverflow.com/questions/47597430/assign-a-database-value-to-variable>

<https://stackoverflow.com/questions/4089628/converting-resultset-from-oledbdatareader-into-list>

<https://stackoverflow.com/questions/55229361/be-able-to-show-the-next-value-in-a-linked-list-by-clicking-a-button>

<https://stackoverflow.com/questions/7105230/how-to-access-the-files-in-bin-debug-within-the-project-folder-in-visual-studio>

<https://stackoverflow.com/questions/8750290/how-can-i-check-multiple-textboxes-if-null-or-empty-without-a-unique-test-for-ea>