AQA A2 COMPUTER SCIENCE NEA REPORT

DYNAMIC LIFEGUARD ROTA AND EMPLOYEE PORTAL PROGRAM

Archie Moon

Worthing College
CENTRE: 65355
CANDIDATE: 9219

Contents

Analysis

Introduction

This project is going to allow lifeguards to have a hub where they can manage their shifts, chat with other users, calculate an estimate for their pay and allow managers to upload weekly shifts that lifeguards can choose to pick up. It will also allow managers to create the daily 'matrix' which holds the rota of positions around poolside that each lifeguard that day has.

Project Background

I'm making this program to solve a real-life issue within the leisure centre company I work for because the system they use to produce the daily rota and communicate with employees is primitive and although is simple enough to use it lacks centralisation with the other applications the company uses with the daily rota being made on excel, companywide communication being done within WhatsApp and timesheets and pay being done on an external website.

This approach is very inefficient and relies on multiple third-party apps for everything to run smoothly therefore I plan on making a centralised application where lifeguards can see and manage everything they need to.

Before I start implementation of the system, I am going to do extensive research into the range of functionality I intend on implementing through looking into other examples and existing systems as well as interacting with potential clients and users through surveys and Q&A's with managers and lifeguards on multiple occasions to develop a bespoke system tailored to the needs of the potential client.

General Approach

I plan on using WinForms and C# to create and manage the GUI on the client-side application with an SQLite database to manage user data, where there passwords will be hashed and stored using my own algorithm, as well as storing user data it will also hold employees shifts, which will be linked through a foreign Employee ID key, this will be used to generate the daily rotas.

I also plan on exploring WebSockets with the use of SignalR to allow for real time communication within a group messaging hub with a server side ASP.Net application providing the open connection that each client will connect to on the client side WinForms program which store and load past messages through another table in the SQLite database.

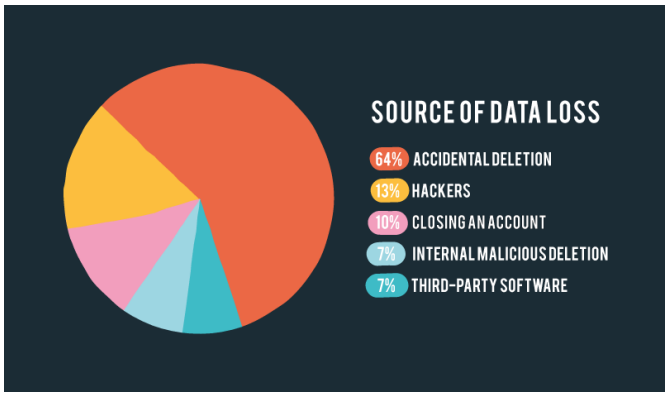
The matrix will be generated by users, with managerial access, using an algorithm to create each employee's shift for that day and can be fully edited by the manager this will all be handled on the client-side WinForms application.

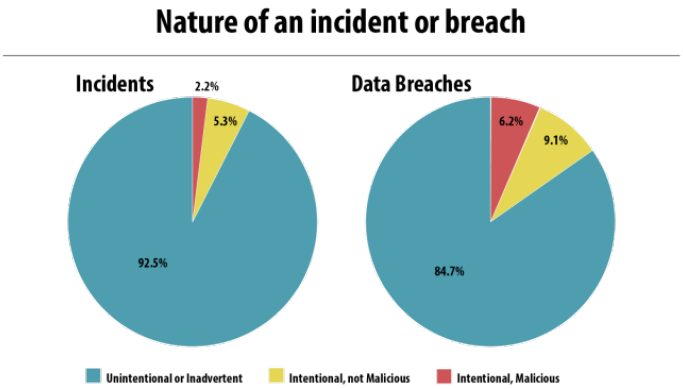
Within the WinForms application users will also be able to submit timesheets that will allow them to calculate an estimate for their pay.

I will also add functionality for managers to add remove and update the database of users.

Alongside this, producing a real time communication application and matrix making software with an automatic dynamic customisable grid will be a nice challenge that appeals to me and being able to create a GUI alongside adds to the challenge and is an interest I've always enjoyed exploring.

Importance of reducing human error





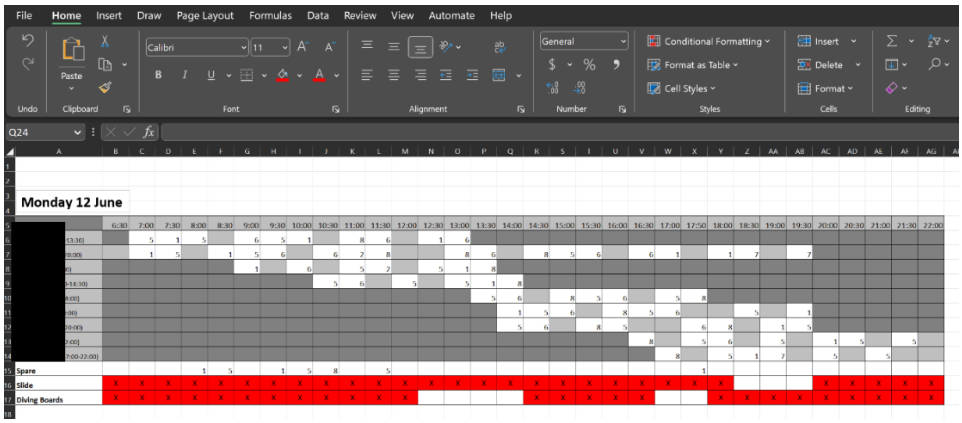
These graphs both highlight the significant factor of human error in data related environments with accidental deletion being the most predominant cause of data loss in office365 applications (left hand graph) and 92.5% of incidents and 84.7% of data breaches being caused by unintentional or inadvertent users (coming from an IAPP, international Association of Privacy Professionals, study in 2018, right hand graph). This highlights the need for an application that removes the human factor from generating the daily rota and from assigning users to shifts and this will be one of the aims of my program.

Research

Examples from my place of work.

Excel

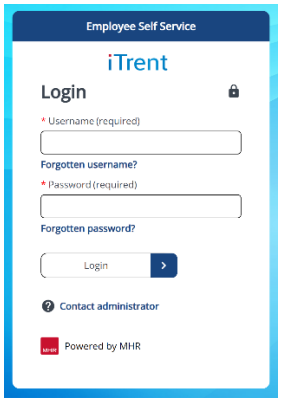
Figure 1a.



This is an example of how the daily rotas are made, this approach is manual and tedious as it requires the manager to fill in the blank template for every day. Excel, although has vast functionality can be quite frightening at first due to its complex and busy interface making it difficult for new users to adapt to. I would like to improve on this by making the interface easier to use and easier for the managers to create the daily rota and implement an algorithm to produce the matrix which will free up the managers to help with the daily running.

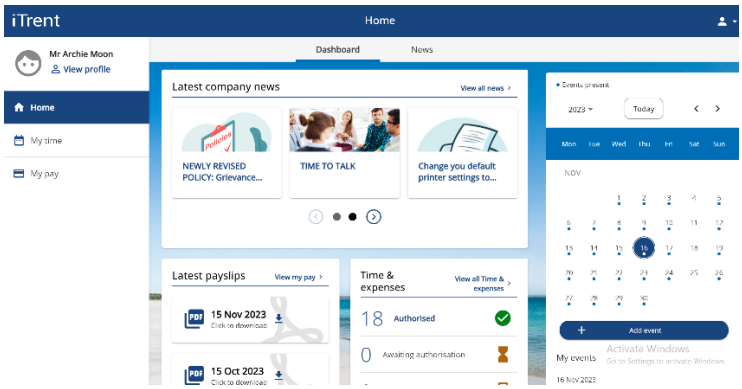
iTrent (Current employee self-service program)

Figure 1b.



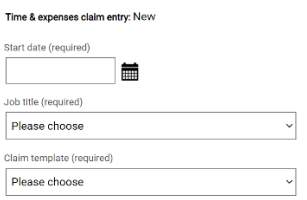
This is the login screen for the current application the lifeguards use to manage their pay and time. This resembles most login pages and mine will look similar however they have missed the functionality of being able to view your password, which I actually found an issue with while getting the screenshots for this figure as when entering a long password with many unique characters it can be hard to remember where you've gotten to forcing you to re-enter it. Therefore implementing this feature would solve that issue instantly without much effort.

Figure 1c.



This is the home page taken from the iTrent site, this example uses something similar to what I intend on implementing which is the widgets. I do however feel like it could be organised a bit clearer so I'm going to look at a minimalistic approach with possibly a menu bar at the top that allows the user to access all the functionality. The idea of adding a calendar showing any events is a good idea though which I might look at implementing to show upcoming events.

Figure 1d.



This screenshot is taken from the payslip functionality of the iTrent system, I will take heavy inspiration from this design as it is simple and intuitive and the addition of more complex functionality will likely just clutter the UI unnecessarily.

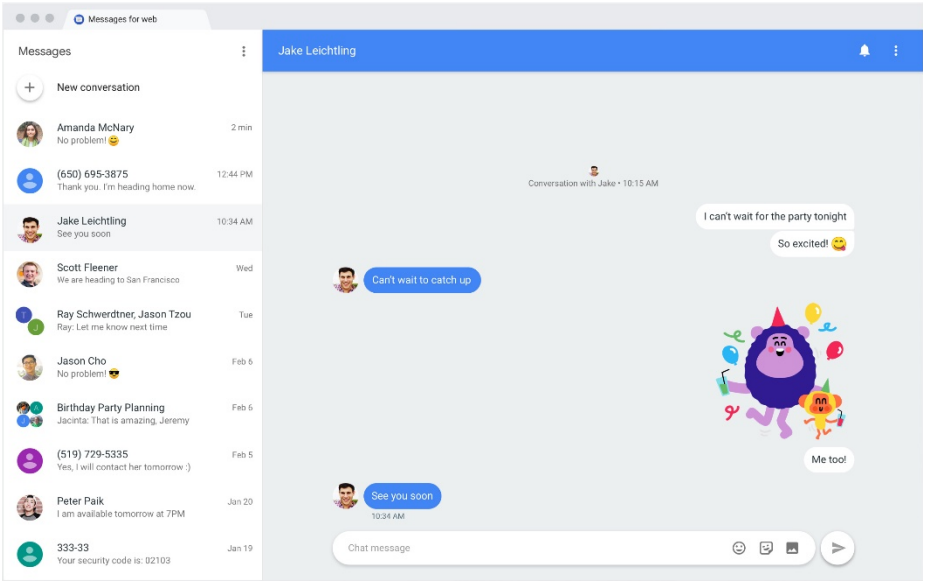
Figure 1e.

This is the screen that follows when you fill the fields from the previous figure (1d). I find the use of two separate pages for this useless as it slows down the user, so I intend on combining the previous two figures into one page this should make is quicker to submit any timesheets. I also find the horizontal layout slightly confusing with the 'Hourly Rate' label looks like it is linked to the text box next to it whereas that text box is actually for the date. To solve this ill implement these fields vertically.

Messages For Web

By Google

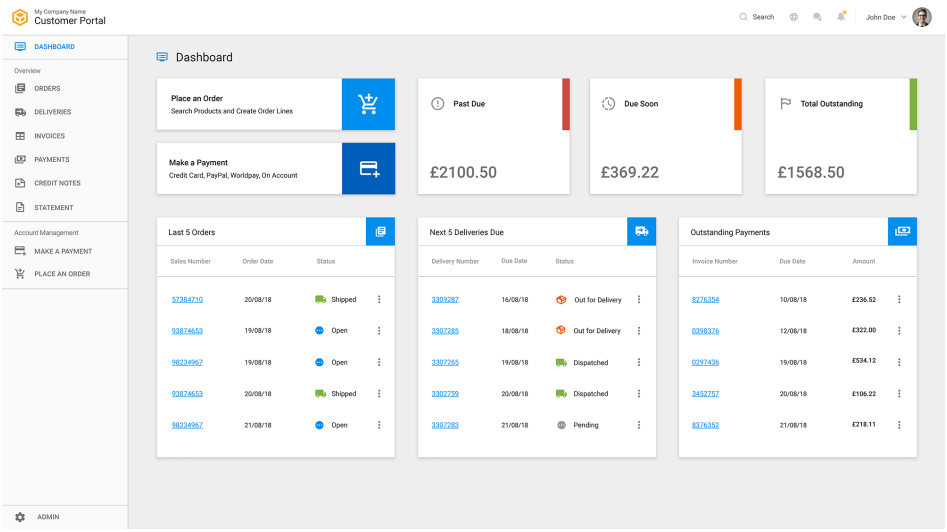
figure 2a.



This is an example of a web messaging service made by google. I like the way this is laid out and intend to produce something similar to implement into my project allowing for lifeguards and managers to communicate through the group messaging system. This project however will not have the need for the messages bar column on the left as group messaging will be the only functionality, however in the future I might look at implementing one – one messaging.

Web Portal Dashboard

Figure 3a.



This is another example of a dashboard layout id be interested in using as the idea of having widgets to display useful information on the main screen is something I am looking into so this provides a good example to take inspiration from. I will implement this on the home screen of my app and use it to display useful bits of data such as the users next shift and maybe even recent messages. For separation of concerns reasons the data shown in the widgets will likely be separate forms to the one the widget links to, this is just to make debugging easier.

Account Settings Page

By Facebook

Figure 4a.

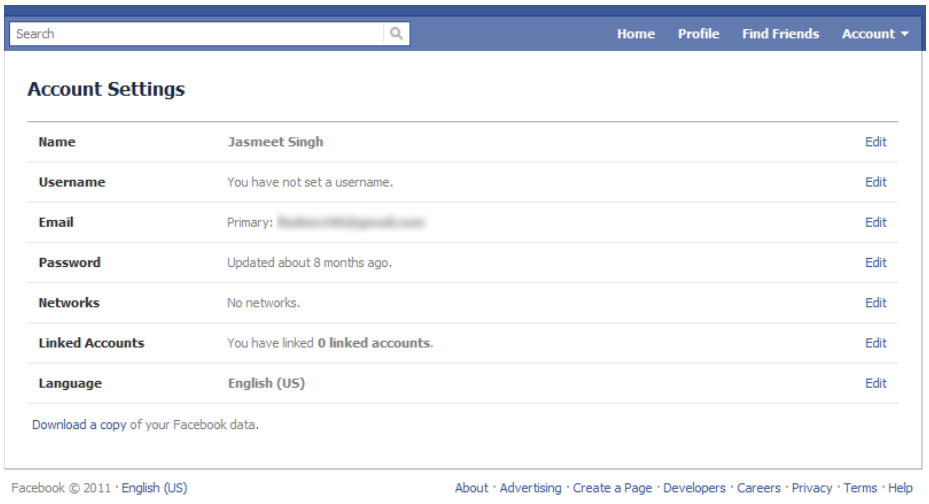


Figure 4a shows Facebooks approach to account management this example sticks out to me as its simple and easy to use as I find some modern settings menus can be confusing and complex so I am to make a simple easy to change account page, similar to this, so users can change their details with all changes updated in the database.

Systems

WinForms

I'm going to use WinForms for the client-side application, I've chosen this because WinForms makes it really easy, using drag and drop elements, to create fully customisable GUI's that can make use of the entire functionality of the C# language.

I plan on making the GUI simple and easy to navigate with tab like buttons at the top of the screen which will bring up the corresponding page in an panel on the screen.

SignalR: WebSockets

I will be relying on SignalR with the use of WebSockets combined to provide the server side of my application that will allow for instant communication within the messaging functionality of my app.

This provides a seamless instantaneous messaging experience because it eliminates the need for the client application to constantly check the server for updates as it uses event listeners to provide updates when an event occurs such as a message is sent.

It does this by opening a persistent connection between the server and the client which means when something happens on the server all the connected clients are updated in real time without the need for the client to request data from the server.

The WebSocket functionality within SignalR allows for Bi-Directional data transmission thanks to the persistent open connection, this also means that it can be multiuser with many clients connected to the server at one time sending and receiving updates.

WebSockets are also what's called full duplex which means that both the client and the server can send data at the same time without any collision issues.

SQLite Databases

The majority of the program will rely on SQLite databases to provide the backend for data storage from employee details such as name email usernames and passwords, as well as what shifts they work, and the messages sent.

Although SQLite databases don't provide any real time functionality, they are really simple and easy to implement. And the ability to send and receive data in real time, from applications such as 'Fire Store', just isn't necessary for the messaging functionality thanks to WebSockets' capabilities.

SQLite with WebSockets and SignalR

The use of SignalR and WebSockets, within my messaging functionality, means I don't need a real time database such as FireStore, this is because I can update the SQLite database when the send message listener is triggered on the server meaning the database is updated in real time but the sending and displaying of new messages, not stored in the database, is handled by the server and they don't have to be written to and read from a real time database as they are sent.

This means that I can simplify the data storage aspect by keeping it to one main database within a single application instead of multiple databases that use different platforms. And also reduces the total read/write requests sent to and from the database.

ASP.NET core and SignalR server hosting

In order to allow users to connect to the signalR server across multiple devices I plan on using microsoft azure to host my sever. This is because their free plan allows me to test the system on a small scale allowing for up to 20 simultanious connections which is more than enough for testing purposes, it also allows me to monitor activity of data being sent and recivevd by the server which when scaled up to a full deployment scenario will provide the useful ability to monitor access and usage of the server.

Password Hashing

The way hashing works is that instead of storing a password in plain text or encrypted by a key in a database you run their password through a hashing algorithm typically when signing up or creating the user, this is then stored in the database.

Then when it comes to logging in instead of seeing if the two passwords match you, you run them through a verifying function which essentially hashes the password they enter and can verify that against the hash that is stored in the database.

What makes my algorithm more secure is that it uses something called a ‘salt’. This is where a unique string is added to the password before it is hashed meaning that if two users were to have the same password the hashed result would still be different due to the unique salt.

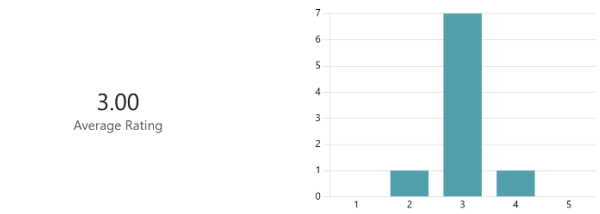
Client Interaction

Lifeguard Forms responses

This is a Microsoft forms sent out to 10 lifeguards at the leisure centre I work at and out of which 9 replied and here are their responses.

1. How would you rate the reliability of the daily matrix when you are on shift.

More Details



The average rating for the reliability of the matrix is 3 out of 5 this is also the most common answer and shows that the reliability is okay but not good this shows that this can be improved which I aim to do with my program and can be done with a few simple tweaks that allow for a more reliable system to be put in place.

2. How many errors in the matrix are there on average per day?

More Details

This question shows an average of 3 errors on each daily matrix this number is too high in my opinion especially if you look at the next question…

3. How long of a delay does the average error in the matrix cause? (in minutes)

More Details

This question shows that each error takes an average of 6-10 minutes to sort out this would mean that on each day about 30 minutes to sort out this has lots of room for improvement which is my intention to fix. Alongside this I know there are often one or two mess ups that take up to 30 minutes alone to fix such as when an employee isn't added to the matrix I intend to prevent this by making it clear who needs to be on the matrix for that day.

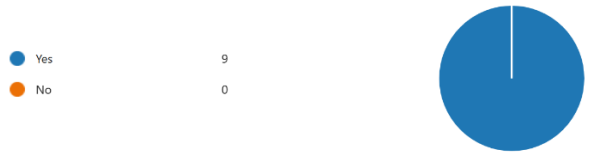
4. Do you find the current use of WhatsApp as an easy way to communicate.

More Details

This question shows a pretty balanced preference for WhatsApp as a method of communication however the results are slightly skewed towards using an alternative method this suggests that my application would have a positive impact on the working experience.

5. Would you appreciate being able to pick up shifts through an easy-to-use application?

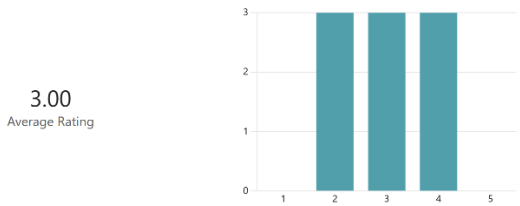
More Details



When asked directly weather they would appreciate a centralised app to pick up shifts and the response was unanimous with all 9 participants saying they would rather an easier method to pick up shifts. This is a clear direct indication that a new method is in dire need and I intend to provide the perfect solution.

6. How would you rate the overall experience of picking up shifts

More Details



7. How do you think this whole process can be improved and do you have any other comments?

9 Responses

ID ↑	Name	Responses
1	anonymous	i guess it could be more efficient maybe something to make the matrix easier to do?
2	anonymous	It would be really useful if it was easier to pick up shifts and like see which ones you've picked up instead of having to scroll through a WhatsApp chat which is annoying.
3	anonymous	no
4	anonymous	matrix seems okay but when there are errors they do cause a delay, and picking up shifts is bad imo as you gotta go through a whole group chat just to pick one up that might have aready been taken by someone else so it would be easier to see like what ones are available to pick up and just do that
5	anonymous	picking up shifts is easy but could be better like its a bit sketch as its not like official
6	anonymous	its annoying when i get let off late because ofan error in the matrix so if there was some way to minimise these errors then that would be great.
7	anonymous	too many errors on the matrix alwasy causes issues on every shift

This was a general comment and how the whole process could be improved question and the responses show a general distaste with the way shift management occurs and also the disruption caused by human error in the matrix.

Using this form has shown me that there is a need for a new easy way to run and manage the way lifeguard's rota and weekly shifts.

Questions with Manager (Client)

Project Proposal & General overview of client needs

Me: How long does each matrix take to create each day?

Manager: Each one probably takes about 45 minutes to over an hour each, so I have to set aside a good 5 hours or so in order to get them ready for the week.

Me: How much does human error in the matrix contribute to delays and complaints in from staff and customers in the day to day running?

Manager: Human error is very frequent as it's hard to organise everyone's shifts without one or two mistakes which does lead to a fair-few complaints like you said from staff when they are being let off to their lunches late but also from customers when lanes change, and pools open later than stated due to missing positions and errors in the timings.

Me: What would your general thoughts on an automated system to do this for you be?

Manager: That would be a great idea it would definitely save a lot of time and stress; it would also allow the place to run smoother which would definitely benefit everyone from customers and staff to managers and instructors.

General Needs: A quick efficient rota generating algorithm.

Me: How would you feel about a central web application that allows you to do the daily matrix as well as communicate and also organise and schedule shifts?

Manager: That would be a godsend, it's always annoying to keep switching between WhatsApp and excel having to manually enter each shift especially when I could be using the time to supervise the staff and pools, so yeah something like that would really be a help.

General Needs: Messaging and shift functionality in one app.

Me: Would a site like this save both time and money for the company?

Manager: Oh yes definitely the amount of time wasted on organisation is not little so yeah, this whole process being streamlined would definitely increase productivity and yeah, I guess that in turn would lead to money being saved as the whole thing will be running smoother.

Me: So, what do you think your needs will be for this project?

Manager: "Well basically I guess it would need to merge our excel and WhatsApp use into one app so it would need to be able to generate the rota automatically or at least to some extent as well as allowing for messaging between lifeguards and the managers. Which would be separate from the shift side of it where we can send out shifts that lifeguards can pick up quickly and easily.

What I've learnt:

- Shows a demand for my solution.
- My project solves a real-life problem.
- Client wants Rota, Messaging, Shift scheduling within the program.

Additional functionality and review of existing project

In this meeting I showed my target client the current state of the project.

At the time the project contained:

- Full Group Messaging capability
- Basic Shift scheduling
- Account management and logging in functionality
- Complete UI for all pages (Login, Account, Messaging, Shifts, Home)

These are the general thoughts and comments made.

Manager: "This is looking really good, that messaging page works perfectly, and I like the blue and white colour scheme". "The home page is looking nice too I like the layout and it should look really nice one you add the widgets and stuff."

Me: "What are your thoughts on the current state of the shift functionality"

Manager: “I mean obviously it’s not complete, but the adding shift feature works nice, but yeah it will much better once complete, but the idea is definitely there. It would be nice to maybe be able to *edit shifts after they’ve been posted* but I’m sure that’s something you could look at.”

Additional needs: Allow for editing of shift posts after they’ve been added.

Me: “Is there any additional functionality that you’ve thought about that would be beneficial since the last time we spoke.”

Manager: “Yeah actually, we were thinking that maybe some sort of *timesheet or pay sort of system* would be useful, even if you don’t add the whole payroll aspect but just the ability for people to create a payslip and then going off of that in the future, we can look at connecting that to payroll and HR.”

Additional needs: Timesheet functionality, allow users to create and submit a timesheet.

Manager: “I also noticed that there wasn’t any way for us managers to like add new people and do all the data stuff so that would be extremely useful to be able to do all that within the program.”

Additional needs: Allow for managers to update, add and remove data from the database.

What I’ve learnt:

- A timesheet functionality would be a great addition.
- Ability for managers manipulate the database and add users etc.
- Allow for shifts to be edited after they’ve been posted.
- Happy with the current state.

Proof Testing

Creating the localhost server

To start I am using a web application builder to set up the server-side interface.

```
var builder = WebApplication.CreateBuilder(args);
```

Then we need to add the SignalR service to the builder.

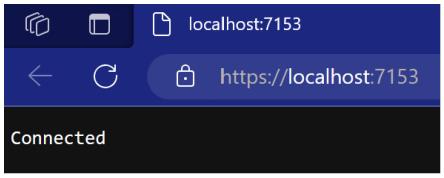
```
builder.Services.AddSignalR();
```

Then we can create the app by building our builder.

```
var app = builder.Build();
```

We can then add a few more details to allow for error handing and an endpoint for the users to connect to called 'chatHub'. We will also display a message on the localhost on connection.

```
// Decides where to output any error messages
if (app.Environment.IsDevelopment())
{
    // displays errors in the development environment
    app.UseDeveloperExceptionPage();
}
else
{
    // otherwise to the /Home/Error extension
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

// This endpoint is what users will connect to for communication
app.MapHub<ChatHub>("/chatHub");

// writes connected upon the initial request to the server proving its connected
app.MapGet("/", async context =>
{
    await context.Response.WriteAsync("Connected");
});

// runs the app
app.Run();
```



Matrix Algorithm

//

Password Hashing

My password hashing algorithm will work by converting their password into the ascii values of each character and then comparing each character's ascii integer against the ascii value of the corresponding character in the key.

```
int XOR_Result = Password[i] ^ KEY[i];
```

The ^ operator performs an XOR on the ascii of the characters at index 'i' in the password and key.

The int value this is stored as represents the ascii of the hashed character.

The output looks something like this:

```
a ^ z
= 27
```

This will return a new value which is then converted into hexadecimal, this is to stop any issues with un-printable (patterns of) characters, which I experienced while developing this algorithm.

XOR_Result.ToString("X2");

```
Hex of 27:
= 1B
```

The hexadecimal of the result of the XOR of each character is then outputted as the hashed password and by converting them from hex into an int and then performing an XOR against the key you get the original character which can be used to verify the password.

HashedAsciiValue = Convert.ToInt32(HashedPassword.Substring(x, 2), 16)

“HashedPassword.Substring(x, 2)” gets every two characters of the hashed password as each, 2 character long, hexadecimal value represents each character.

OriginalAsciiValue = (HashedAsciiValue ^ KEY[x / 2])

Convert.ToChar(OriginalAsciiValue)

```
1B to decimal:
= 27

27 ^ z
= 97

Character of ascii value 97:
= a
```

To make it more secure the algorithm will generate what's called a salt, this is a 32-character long string which is unique to each user. This is appended onto the password before its hashed to ensure that even if two users have the same password the hash will be different.

```
Random rnd = new Random();
const string Characters =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()_+-=[]{};:,.<>?";
int SaltLength = 32;

StringBuilder Salt = new StringBuilder();

for (int i = 0; i < SaltLength; i++)
{
   int index = rnd.Next(Characters.Length);
   Salt.Append(Characters[index]);
}

return Salt.ToString();
```

All this subroutine does is generate a random number between 0 and the total number of possible characters. This value represents the index of a character in the list and this character is the appended to the salt.

The output would look like this:

```
H-XmUe.=Fdj9w<GDXTc=b4WxM?2u^fWC
```

The use of a salt guarantees every hash will be completely unique.

Objectives

Create the algorithm to generate the daily rota.

- The algorithm will generate the rota where lifeguards contracted shifts and any overtime are loaded in from the database.
- Their positions will be randomly selected from the list of allowed numbers.
- The algorithm will ensure that the rota doesn't include duplicate positions where possible.
- As well as ensuring that each lifeguard has a maximum of three positions in a row before an 'off'.

A simple Home Page with 'widgets' that show key and recent information to the user.

- The main page that will allow users to see small widgets of info which can be expanded into a full screen page of the widget.
- It will include widgets such as what shifts they have picked up, their previous pay slips, and more.

Make a secure Login System

- Users will enter their username and password.
- The password will then be hashed using my algorithm.

- The user will have 5 total incorrect attempts before they are locked out, to prevent brute force attacks.

Include a Password Hashing algorithm

- The password will be hashed using my algorithm (View Systems for more info).
- To verify the password the one in the database will be unhashed and compared to the one entered, if the two match the user can login.
- The algorithm will also include a salt to ensure that there can be no duplication issues in the database.

Create a multi-user messaging system using SignalR.

- Create server-side app that opens persistent connection, through WebSockets and SignalR, to a host server for clients to connect to.
- Implement an indicator light indicating whether they are connected.
- Any error messages will also be displayed to the user.
- Users that are logged in will be able to send real time messages to any other logged in user.
- All sent messages will be stored in the databases of the logged in users.
- To reduce load; I will only load the previous 10 messages, when a new user logs in, but allow the user to load another previous 10 messages when and as they need to.

An Account management page where users can edit their details.

- Allows users to manage their profile such as passwords and usernames.
- In order to edit their password, they will be required to enter their current password.
- Any changes they make will be updated in the database.
- All inputs will be checked to ensure they are in the correct format for the database.

Add a page where users can view and accept available shifts.

- Only shifts that haven't been taken will be displayed meaning they can pick up any shift they see, and their name will be added to the database.
- Users can press a button and they will be added to that shifts record in the database
- I will also use this table of the database in the rota generator, so it automatically adds anyone that has picked up a shift for that day.

Create a Third Normal form (3NF) database which acts as the back end for the system.

- A normalised database with multiple tables to track a user's activity across the application.
- Multiple tables for each major functionality

Allow users to create timesheets to track of their shifts.

- A simple data entry form.
- Data entered is formatted and inserted into the database.
- Shows timesheets history to the user.

Allow for users with admin permissions to create edit and remove users and upload shifts.

- Users with admin permissions will be able to access a further range of tools to add and remove users as well as edit and update information.
- They can also add shifts which will be available to be picked up by other users
- The permissions of users will be assigned at creation of the user.

Objective Risk Evaluation

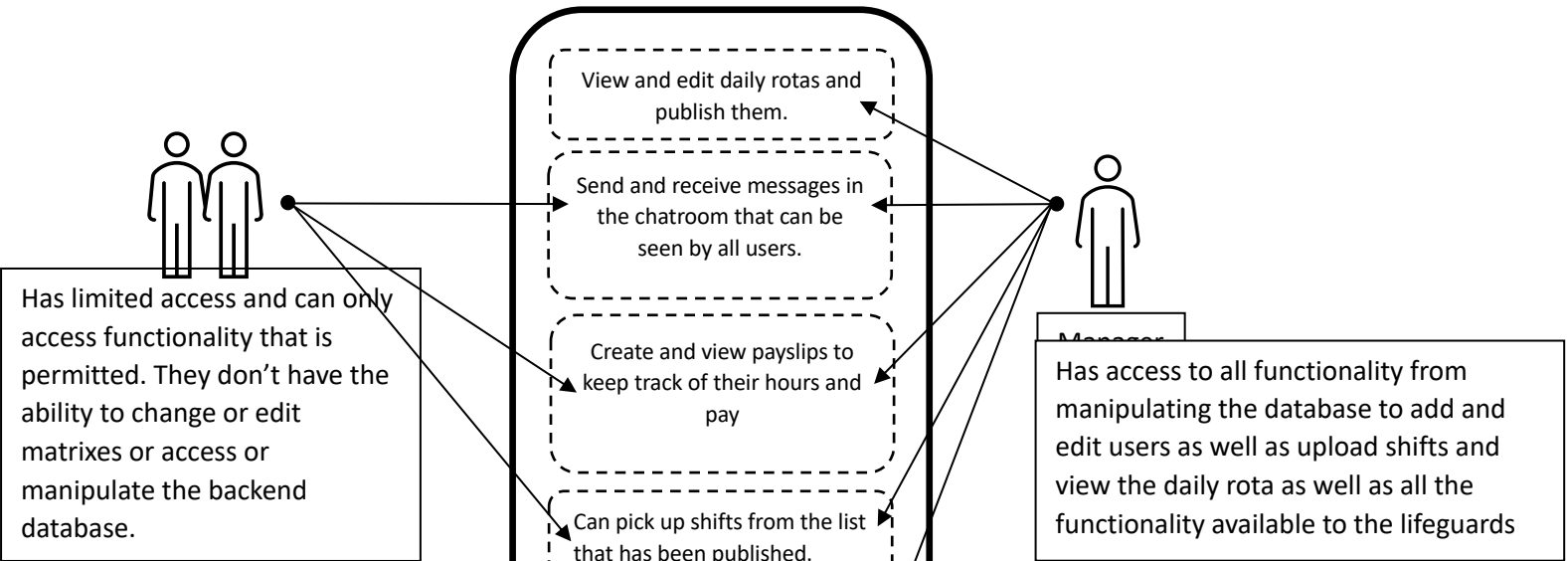
This table highlights the development details for each objective as well as their priority this will help to create a development path with the higher priority objectives having more time allocated to them.

Objective	Implementation Difficulty	Time for sufficient implementation	Priority Level
Daily rota generation	High	High	High
Home Page	Mid	Mid	Mid
Login System	Mid	Mid	Mid
Password Hashing	Low	Low	Mid
Multi-User real time Messaging	High	High	High
Account management	Low	Low	Mid - Low
Accepting shifts	Mid - High	High	High
SQLite Databases	Low	Low	Mid
Timesheet Creation	Mid	Mid	Mid
Admin Control (User / shift creation and modification)	Low	Mid	Mid

Critical Path of Development

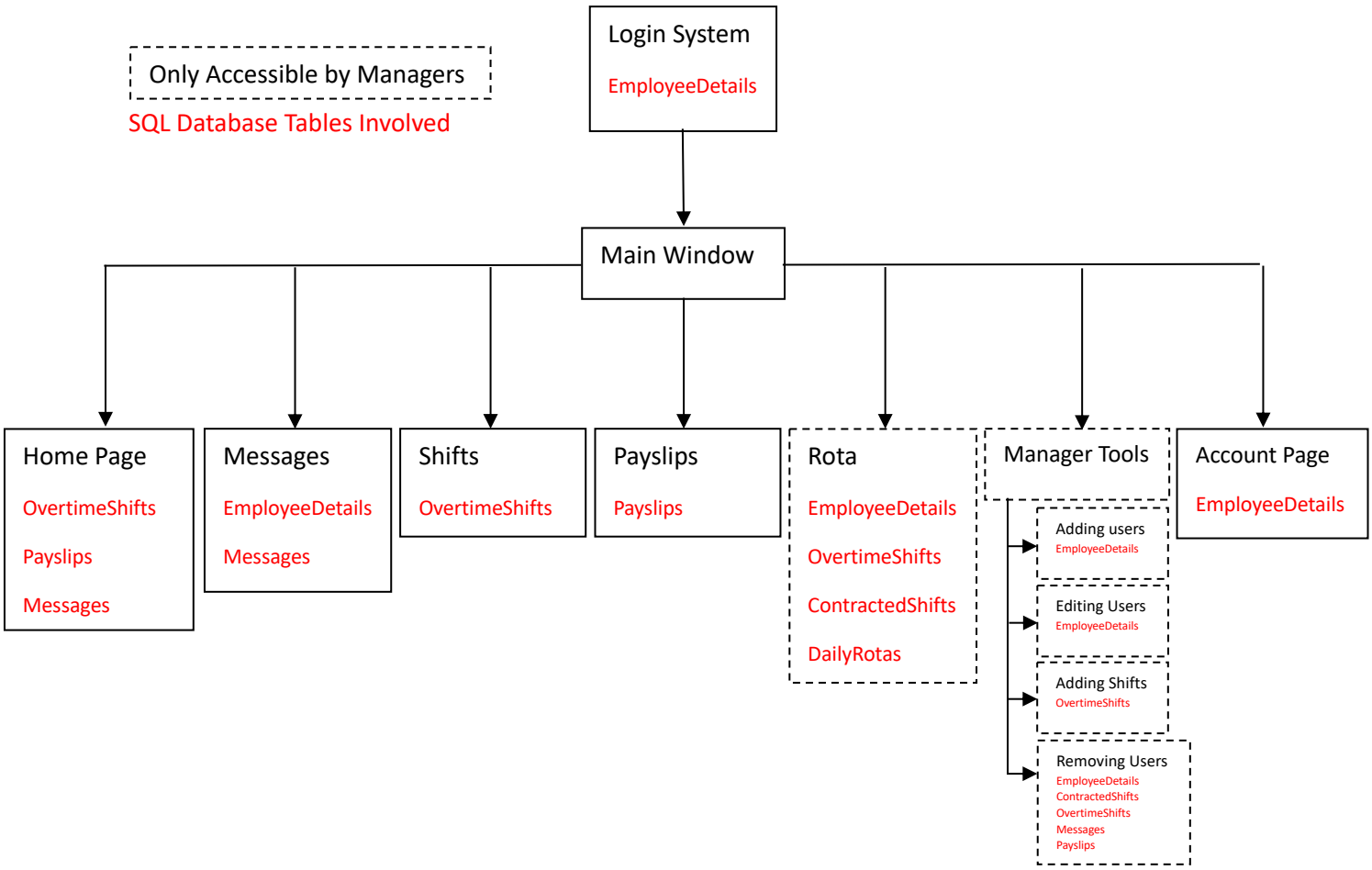
My focus will be on getting the High priority funtionality working first so I will start by learning and working out how to implement the multi-user messaging as this will ivolve me developing my understanding of SignalR and then developing the daily rota and working on how to accept shifts, as these are all high priority objectives that are key to achieving the goal of this project, before I focus on the less necessary objectives such as the login system.

Architectural Diagram



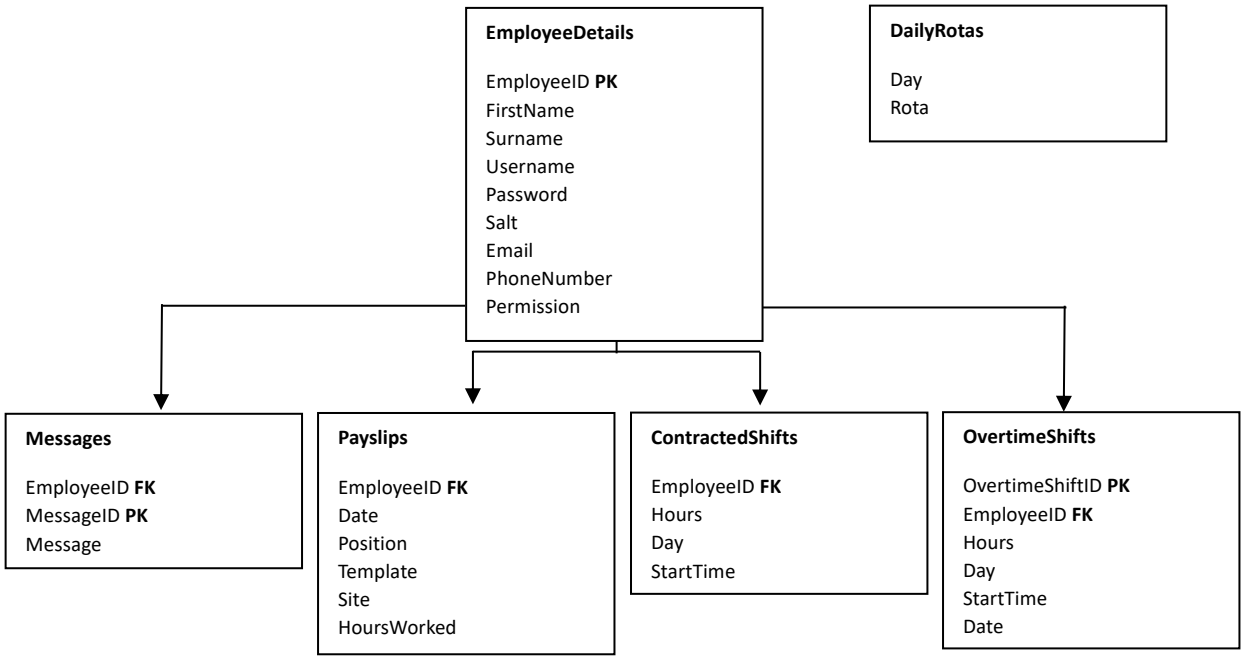
Design

System Structure / Hierarchy chart

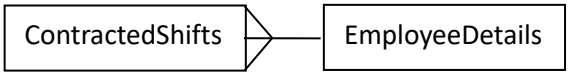


Database Schema

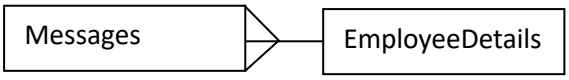
This represents how the tables in the database tables will interact and the connections they make.



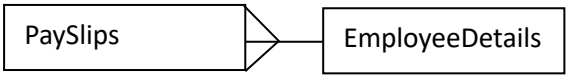
Entity Relationship diagram



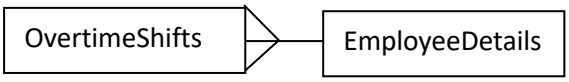
There is a Many to One relationship between the ContractedShifts table and the EmployeeDetails table, they are linked with the foreign key 'EmployeeID' This is because one Employee can have multiple Contracted shifts.



The Messages table also has a Many to One relationship with the EmployeeDetails table again linked with the employeeID foreign key because one employee will have many messages stored in the database, but each message will be linked to the user that sent it with the foreign key.



There is a Many to One relationship between the PaySlips and EmployeeDetails table these table are linked with the foreign key EmployeeID.



There is a Many to One relationship between the OvertimeShifts and EmployeeDetails table these table are linked with the foreign key EmployeeID.

Entity Description diagram

EmployeeDetails (EmployeeID, FirstName, Surname, Username, Password, Salt, Email, PhoneNumber, Permission)

The EmployeeDetails table contains the Primary Key ‘EmployeeID’ which is used to connect and track a user’s actions within the application. It also contains their first name, surname, username, password, and their unique salt, as well as email and phone number which are all the details that the manager might need to know. There is also a permissions attribute, this determines what access the user will have and what forms they can view and use.

Messages (EmployeeID, MessageID, Message)

The messages table contains the messages sent by each user, but each message has its own unique primary key and also the foreign key (EmployeeID) of the user who sent.

ContractedShifts (EmployeeID, Hours, Day, StartTime)

The contracted shifts table contains the information for the shifts each employee is contracted to work, this is linked to the employee through the EmployeeID foreign key, and contains the number of hours they work, what day the shift is on, and what time it starts. This will allow me to use this data to display a daily rota including who is working at what times on each day.

OvertimeShifts (OvertimeShiftID, EmployeeID, Hours, Day, StartTime, Date)

The OvertimeShifts table will contain a unique id for each shift in the table along with the employeeID attached to that shift. It will also contain the hours, day and start time as well as the date, the date is important as these shifts will be changed weekly.

Payslips (EmployeeID, Date, Position, Template, Site, HoursWorked)

Each record in the PaySlips table is uniquely identified by the EmployeeID field and contains the date, position, template (Contracted or Overtime), What site they were working at and the number of hours they worked.

DailyRotas(Day, Rota)

This table stores all 7 days of the week and a nullable column for a rota to be saved into when needed this is the table we will call when checking weather to load the rota from the one saved in this database or to generate a new random rota.

DDL Scripts

```
CREATE TABLE EmployeeDetails (
    EmployeeID  INTEGER PRIMARY KEY AUTOINCREMENT
                        NOT NULL,
    FirstName   TEXT    NOT NULL,
    Surname     TEXT    NOT NULL,
    Username    TEXT    NOT NULL,
    Password    TEXT    NOT NULL,
    Salt        TEXT    NOT NULL,
    Email       TEXT    NOT NULL,
    PhoneNumber TEXT    NOT NULL,
    Permission  TEXT    NOT NULL
);
```

This DDL is used to create the employee details table, it contains the employees First name and surname, their auto generated EmployeeID, their username, password, their unique salt as well as email and phone number and their permission level. EmployeeID is the primary key as it uniquely identifies each individual employee and auto increments for new employees and will be stored as an integer. All of the other entities will be stored as TEXT including the phone number, all entities are defined as NOT NULL meaning they can't be left blank as all of the fields are required to form a record for the user. The primary key will be used as a foreign key in other tables to link the employee to their messages as well as their shifts and track what each employee does throughout the system.

```
CREATE TABLE Messages (
    EmployeeID INTEGER REFERENCES EmployeeDetails (EmployeeID)
                       NOT NULL,
    MessageID  INTEGER PRIMARY KEY AUTOINCREMENT
                       NOT NULL,
    Message    TEXT    NOT NULL
);
```

This DDL is for the Messages table which will store the messages sent in the message form. The EmployeeID is a foreign key referenced from the Employee details table. Each message is identified by the MessageID which autoincrements. The message is just stored as text and all attributes are defined as NOT NULL which means they can't be left blank.

```
CREATE TABLE ContractedShifts (
    EmployeeID INTEGER REFERENCES EmployeeDetails (EmployeeID)
                       NOT NULL,
    Hours      INTEGER NOT NULL,
    Day        TEXT    NOT NULL,
    StartTime  TEXT    NOT NULL
);
```

This DDL shows the ContractedShifts table which contains data about individual employees shifts, using the foreign key (EmployeeID) from Employee Details to link the employee to the shift. It also holds how many hours they work and at what time the shift starts. The Day and StartTime fields are stored as NOT NULL and as TEXT, and the Hours are stored as a NOT NULL INT.

```
CREATE TABLE OvertimeShifts (
    OvertimeShiftID INTEGER PRIMARY KEY AUTOINCREMENT
                            NOT NULL,
    EmployeeID      INTEGER REFERENCES EmployeeDetails (EmployeeID),
    Hours           TEXT    NOT NULL,
    Day             TEXT    NOT NULL,
    StartTime       TEXT    NOT NULL,
    Date            TEXT    NOT NULL
);
```

This is the constructor for the OvertimeShifts table. Each record has a unique autoincremented primary key and is linked to an employee through the EmployeeID foreign key. This attribute is nullable as when the shift record Is created there will be no employee assigned to it so it will be null and then if an employee picks up that shift the record will be updated with their ID. Hours, day, StartTime and date are set as NOT NULL TEXT.

```
CREATE TABLE PaySlips (
    EmployeeID       NOT NULL
                     REFERENCES EmployeeDetails (EmployeeID),
    Date        TEXT NOT NULL,
    Position    TEXT NOT NULL,
    Template    TEXT NOT NULL,
    Site        TEXT NOT NULL,
    HoursWorked TEXT NOT NULL
);
```

This Is the DDL for the PaySlips table, again here each record is linked to an employee with the EmployeeID foreign key and all the remaining fields, Date, Position, Template, Site, HoursWorked are set to NOT NULL TEXT.

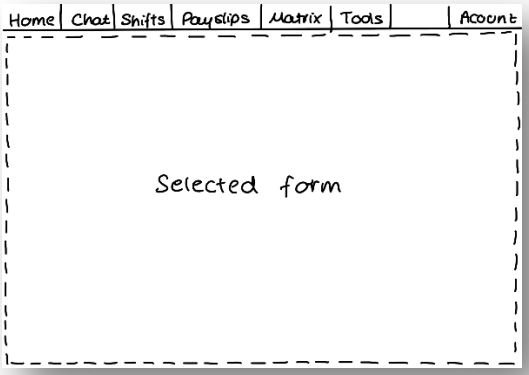
```
CREATE TABLE DailyRotas (
    Day   NOT NULL,
    Rota
);
```

This DDL script defines the Day attribute as NOT NULL this is because these are the days of the week which wont change and therefore cant be null. The rota can be null however as there wont always be a rota saved in the database.

Main Window

This will be the form that will contain the form selected by the user, There will be a toolbar at the top with each form the user can display and when the button for a form is pressed that form will be displayed in the panel within this form.

Concept Art



Pseudocode

```
FormLoad()
        If 'Permission' == 'Employee'
                MatrixButton != visible
                ToolsButton != Visible

LoadSelectedForm('FormName')
        MainPanel.Add(FormName)
        FormName.Show()

ButtonPress()
        LoadSelectedForm('FormName')
```

Test Plan

Tab Buttons

- Ensure that each button loads the correct form.
- The right tabs are displayed to the user based in their permissions (managers can see and access manager only buttons)

Rota Generation and output

This will be the form that contains the daily rota, The design will be similar to the current solution used where I work this will allow it to be implemented easier as it will be familiar to everyone. It will display a grid of rows and columns with a row of times at the top and a column of names down the side with a unique rota for each lifeguard. There will also be a secondary toolbar on the bottom that allows them to select what days rota they want to see and the corresponding day will be generated.

Generation Algorithms

Creating the grid

The algorithm will work by

(Given number of rows)

(Number of columns is hard coded)

- Looping through each row and each column
- Calculating the size and location of that current text box
- Creating a text box at that location and with that size and location

Generating the numbers

This will work by

List of numbers [0, 1, 5, 6]

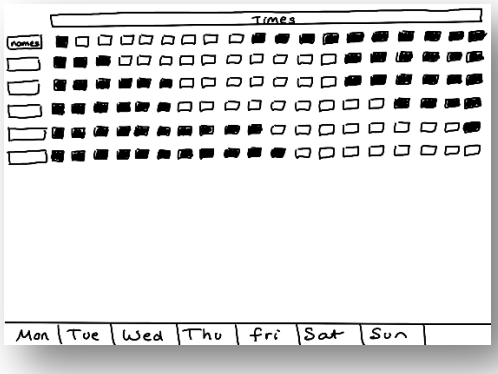
- Chosing a random number from the list of allowed numbers
- Once theres more than three numbers we check the last 3 for any 0's
- If there are no 0's we make the next number a 0
- If there is at least a 0 we can choose another random number
- We also check to see if the last 2 numbers are the same
- If they are we replace the current one with a newly chosen one

Blocking out text boxes

To do this we will

- Work out the starting index of each shift
- Work out the ending index of each shift
- Then we will change the back colour of each text box from the 0^{th} index to the start index
- And from the ending index to the last index

Concept Art



Pseudocode

```
CreateGrid(Num. Rows)

        For (Num. Rows)

                For (Num. Columns)
```

```
                X-Coord = Col x ColWidth
                Y-Coord = Row x Multiplier
                CreateTextBox (
                        Name = "[X, Y]"
                        Size = (ColWidth, RowHeight)
                        Location = (X-Coord, Y-Coord)
                )

ChangeText(X, Y, Text)
        TextBox[X, Y].Text = 'Text'
LoadNames(Day)
        Results = SELECT EmployeeDetails.Name
                FROM EmployeeDetails
                JOIN ContractedShifts ON EmployeeDetails.EmployeeID =
ContractedShifts.EmployeeID
                WHERE ContractedShift.Day = Day
                UNION ALL
                SELECT EmployeeDetails.Name
                FROM EmployeeDetails
                JOIN OvertimeShifts ON EmployeeDetails.EmployeeID =
OvertimeShifts.EmployeeID
                WHERE OvertimeShifts.Day = Day

        CreateGrid(Results.Length)
        For (Results.Length)
                ChangeText(0 , Row, Results.Text)

GenerateNumbers()
        Loop Through rows
                Loop Through columns
                        Place a random number assuring it follows the rules:
                        -The number doesn't make a sequence longer then 3
```

-The number isnt the same as the previous number

-That number is the only instance of that number in that column index

ButtonPress(Day)

LoadNames(Day)

GenerateNumbers()

Queries

```
SELECT EmployeeDetails.Name

FROM EmployeeDetails

JOIN ContractedShifts ON EmployeeDetails.EmployeeID = ContractedShifts.EmployeeID

WHERE ContractedShift.Day = Day

UNION ALL

SELECT EmployeeDetails.Name

FROM EmployeeDetails

JOIN OvertimeShifts ON EmployeeDetails.EmployeeID = OvertimeShifts.EmployeeID

WHERE OvertimeShifts.Day = Day
```

This query Selects the names linked to the employeeID of every shift on a given day, the UNION ALL statements basically combines the results of the two statements allowing me to select the shifts from both the overtime and contracted tables.

Test Plan

Day Select Buttons

- Make sure that each day loads the correct shifts by comparing the output to the database.

Rota Generation

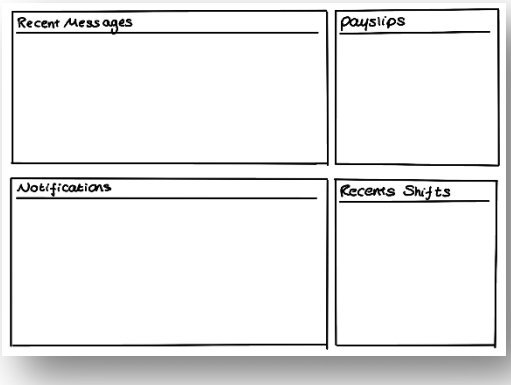
- Check through each row and column of the rota it generates to ensure it follows the rules enforced.

Home Page

This will a page that displays a few key bits of information in the form of widgets to the user, some of this will be unique to the logged in user such as the payslips but some of it will be general

information displayed to all users such as recent shifts and recent messages. All the information will be loaded when the form is loaded.

Concept Art



Pseudocode

```
PayslipsLoader()
        Payslips = SELECT *
                   FROM PayslipTable
                   WHERE EmployeeID = *CurrentUsersID*
        For (Results.Length)
                TextBox.Text += Record


RecentMessageLoader()
        MessageCount = SELECT Count(*)
                       FROM Messages
        If (MessageCount < 10)
                StartIndex = 0
                EndIndex = MessageCount
        Else
                StartIndex = MessageCount – 10
                EndIndex = MessageCount
        Messages = SELECT EmployeeDetails.FirstName, Messages.Message
                   FROM MessageTable
                   JOIN EmployeeDetails ON Messages.EmployeeId = EmployeeDetails.EmployeeID
```

```
        From (StartIndex to EndIndex)
                Message TextBox.Add(Name[index], Message[index]

LoadRecentShifts()
        Shifts = SELECT *
                FROM OvertimeShiftTable
                WHERE Name IS NULL
        From ( (Shifts.Length – 4) to (Shifts.Length) )
                ShiftTextBox.Text += Shift (Format the data into a string)

LoadNotifications()
        Notifications = SELECT Notifications
                        FROM NotificationsTable
        For (Notifications.Length)
                NotificationTextBox.Text += Notification[Index]
```

Queries

```
SELECT *
FROM PayslipTable
WHERE EmpolyeeID == *CurrentUsersID*
```

This query uses the wildcard * to select all fields from every record the payslips table where the employeeID of the record matched the ID of the user logged in.

```
SELECT Count(*)
FROM Messages
```

This query returns the number of records in the messages table and is used to load the last 10 messages

```
SELECT EmployeeDetails.FirstName, Messages.Message
FROM MessageTable
JOIN EmployeeDetails ON Messages.EmployeeId = EmployeeDetails.EmployeeID
```

This query selects the Name linked to the employeeID attatched to the message

```
SELECT *
FROM OvertimeShift
WHERE EmployeeID IS NULL
```

This uses the wildcard function to select all the shift records that don't have an employeeID assigned to them.

Test Plan

Shifts

- Make sure the shifts loaded are the last 5 added.
- Test what happens when a shift is accepted through the home page.

Payslips

- Check that the correct payslips are loaded for the correct user and are in the correct format.

Recent Messages

- Make sure the messages are loaded in the correct format and that the most recent 10 are loaded.

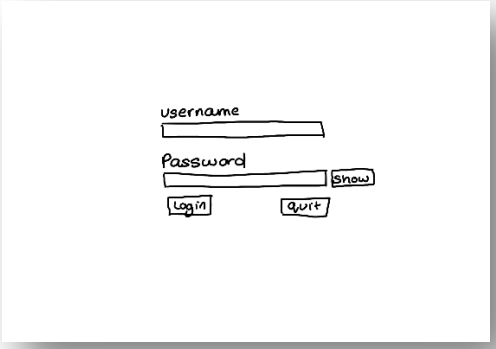
Notifications

- Test if notifications are loaded in the correct order.

Login System

This will be the first form the users are presented with and will be simple and intuitive. The user will enter their username and password, which can be made visible for eases sake, and then can either attempt to log in, where they will get 5 attempts, and the entered password will be hashed and compared to the password of the user with the matching username, or they can quit and the application will exit.

Concept Art



Pseudocode

```
LoginButtonPress()

        Count = SELECT Count(*)

                FROM EmployeeInfoTable

                WHERE Username = 'UsernameTextBox.Text'

        UserDetails = SELECT EmployeeID, FirstName, Surname, Username, Password, Salt, Email,
                              PhoneNumber, Permission
                      FROM EmployeeInfoTable
                      WHERE Username = 'UsernameTextBox.Text'

        If (Count > 0 && Username != "" && Password != "")
                If (VerifyPasswordFuntion(PasswordTextBox.Text, Password, Salt)
                        LoginInfoClass.EmployeeID = UserDetails[0]
                        LoginInfoClass.FirstName = UserDetails[1]
                        LoginInfoClass.Surname = UserDetails[2]
                        LoginInfoClass.Username = UserDetails[3]
                        LoginInfoClass.Password = UserDetails[4]
                        LoginInfoClass.Salt = UserDetails[5]
                        LoginInfoClass.Email = UserDetails[6]
                        LoginInfoClass.PhoneNumber = UserDetails[7]
                        LoginInfoClass.Permission = UserDetails[8]

                        Load(MainWindow.Frm)
                )
                Else (MessageBox ("Login Failed, Please try again"))
        )
        Else (MessageBox ("Login Failed, Please try again"))

        ShowPasswordButton()
                PasswordTextBox.PasswordChar = false

        QuitButton()
                this.Close()
```

Queries

```
SELECT Count(*)

FROM EmployeeDetails

WHERE Username = 'UsernameTextBox.Text'
```

This returns the number of records in the database where the username is the same as the one entered.

```
SELECT EmployeeID, FirstName, Surname, Username, Password, Salt, Email, PhoneNumber,
Permission
FROM EmployeeDetails
WHERE Username = 'UsernameTextBox.Text'
```

This selects all the details from the record with the same username as the one entered so we can store them locally once the user has logged in.

Test Plan

Login Button

- The password Is hashed correctly.
- The correct user is logged into.
- Maybe also text for people with the same password.

Show Button

- Make sure the password is shown in text.

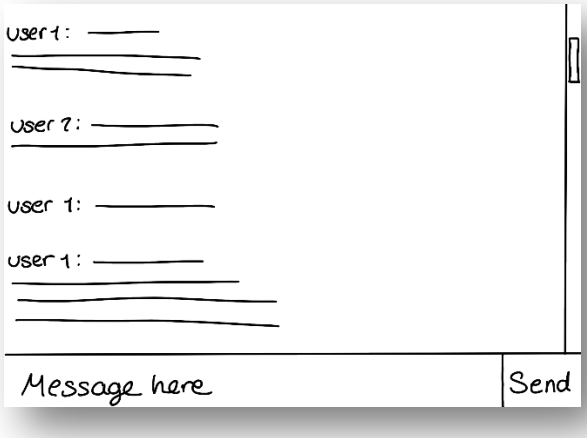
Close Button

- Make sure the program exits.

Multi-User real time messaging

The messaging function will use a simple basic but also familiar design with a large panel to show the messages which is scrollable, a large text box to enter their message and a send button to send it

Concept Art



Pseudocode

FormLoad()

    MessageLoader()

ConnectToServer()

    Start SignalR connection

    ConnectButton.Colour = Green

    If (error)

```
                MessagesTextBox.Text += ErrorMessage
                ConnectButton.Colour = Red

SendMessageEvent(Message)
        Call SendMessage method from server
        DisplayMessage(Message)

DisplayMessage(Message)
        MessagesTextBox.Text += Message

MessageSendButtonPress()
        SendMessageEvent(Message)
        SQLquery( INSERT INTO MessageTable (Message)
                VALUES ('Message') )
        MessageTextBox.Text = ""

ConnectButtonPress()
        ConnectToServer()

MessageLoader()
        Messages = SELECT FirstName, Message
                FROM MessageTable
        For (Messages.Length)
                MessagesTextBox.Text += Messages[x]
```

Queries

```
INSERT INTO Message (EmployeeID, Message)
VALUES ('MessageTextbox.text', *UsersID*)
```

This command adds a new record to the messages table conatining the ID of the sender and the content of the message.

```
SELECT EmployeeDetails.FirstName, Messages.Message
```

```
FROM Messages
JOIN EmployeeDetails ON Messages.EmployeeID = EmployeeDetails.EmployeeID
```

This query returns the name linked to the message, by the employeeID foreign key, and the content of the message to be loaded.

Test Plan

Form Load

- Check that the application connects to the server
- See how it handles errors eg. When the serve isn't running
- Make sure the messages load correctly and the message loader works

Connect button

- See how it handles trying to connect when already connected
- Make sure it correctly reconnects

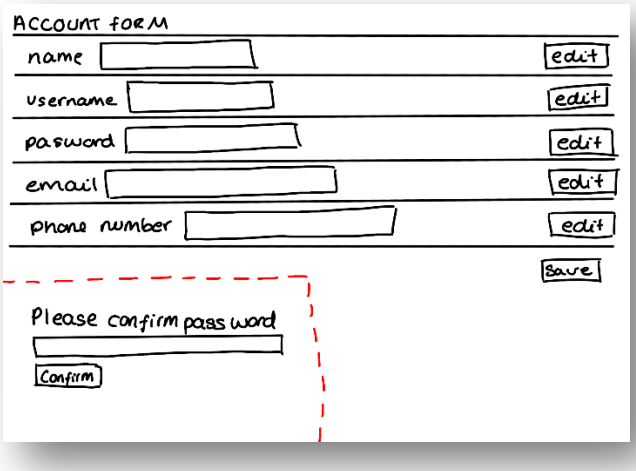
Send Button

- The send message function works
- The message is added to the database and then loaded correctly

Account Management Page

The account page is where users will edit their details that are linked to their account, it has a simple and intuitive design which allows the user to edit a single detail or multiple details at the same time. As well as this it also protects against an unauthorised person gaining access to someone's machine while their logged in by forcing them to confirm their password if they want to be able to edit it.

Concept Art



Pseudocode

```
FormLoad()

    NameTextBox = LoginInfoClass.Firstname + LoginInfoClass.Surname

    EmailTextBox = LoginInfoClass.Email
```

```
        PhoneNumberTextBox = LoginInfoClass.PhoneNumber

        UsernameTextBox = LoginInfoClass.Username

        PasswordTextBox = LoginInfoClass.Password


SaveButton()

    If(NameIsValid && EmailIsValid && PhoneNumIsValid && UsernameIsValid && PasswordIsValid)

            FirstName = Name[0]

            Surname = Name[1]

            Password = PasswordHasher(Password)

            UpdateDatabase ( UPDATE EmployeeDetailsTable

                                SET Firstname = 'Firstname', Surname = 'Surname', Username =
                                    'Username.Text', Password = 'Password', PhoneNumber =
                                    'PhoneNumber.Text'

                                WHERE Username = 'LoginInfo.Username', AND Password =
                                                LoginInfo.Password' )

                    LoginInfoClass.FirstName = FirstName
                    LoginInfoClass.Surname = Surname
                    LoginInfoClass.Username = Username
                    LoginInfoClass.Password = Password
                    LoginInfoClass.Email = Email
                    LoginInfoClass.PhoneNumber = PhoneNumber
    Else ( MessageBox("One or more fields is in the incorrect format please try again')


EditButtonPressed()

        TextBox.Enabled = true
```

Queries

```
UPDATE EmployeeDetailsTable

SET Firstname = 'Firstname', Surname = 'Surname', Username = 'Username.Text', Password =
'Password', PhoneNumber =  'PhoneNumber.Text'

WHERE Username = 'LoginInfo.Username', AND Password = LoginInfo.Password'
```

This query updates all the fields of the logged in employee to the values in the unput fields, it updates them weather they have been changed or not.

Test Plan

Edit Buttons

- The Correct text box is enabled

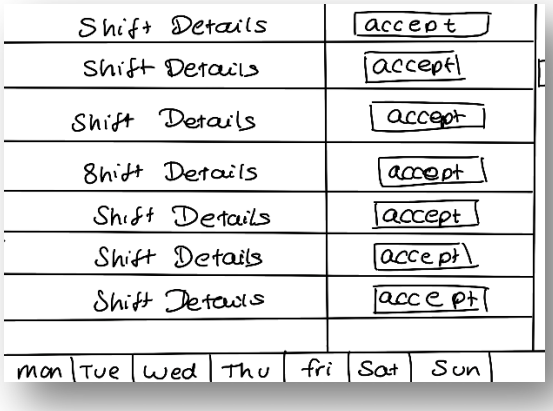
Save Button

- All field are updated correctly
- Test erroneous data inputs

Accepting and picking up shifts

This will be how users pick up any overtime shifts so I will include another week toolbar at the bottom allowing them to clearly pick a day that they want to work and a large grid showing all of the shifts that are available. This will be scrollable in case there are too many shifts to fit on the screen.

Concept Art



Pseudocode

```
ShiftLoader(Day)

    Shifts = SELECT OvertimeShiftID, Date, Day, StartTime, Hours

        FROM OvertimeShitsTable

        WHERE EmployeeID IS NULL AND Day = 'Day'

    ShiftsList = LIST

    For (X → Shifts.length)

        ShiftID = Shifts[0, X]

        ShiftsList.Add(Shifts[0, X], (Shifts[1, X], (Shifts[2, X], (Shifts[3, X], (Shifts[4, X])

        Row (0, X) = ShiftList[X]

        Row (1, X) = AcceptButton

            AcceptButton (

                DoesShiftHaveID = SELECT EmployeeID

                                  FROM OvertimeShifts

                                  WHERE OvertimeShiftID = ShiftID

                If (DoesShiftHaveID[0] == 0)
```

MessageBox(“Accepted”)

UPDATE DATABASE (UPDATE OvertimeShiftsTable

SET EmployeeID = LoginInfoClass.EmployeeID

WHERE OvertimeShiftID = ShiftID)

)

Queries

SELECT OvertimeShiftID, Date, Day, StartTime, Hours

FROM OvertimeShitsTable

WHERE EmployeeID IS NULL AND Day = ‘Day’

This query returns all the fields for each record in the overtime shifts table where there is no assigned employeeID and the day is the same as the given day this is so we can load the shifts that don’t have anyone assigned to them yet.

SELECT EmployeeID

FROM OvertimeShifts

WHERE OvertimeShiftID = ShiftID

This selects the employeeID field of the selected shift, used to check If the shift hasn’t been accepted by another user in the time that the view hasn’t been updated.

UPDATE OvertimeShiftsTable

SET EmployeeID = LoginInfoClass.EmployeeID

WHERE OvertimeShiftID = ShiftID

This adds the employeeID of the logged in user to the shift that they have chosen to pick up.

Test Plan

Accept Buttons

- Make sure the database updates correctly
- Check the shifts disappears when its accepted

Shift Loading

- Make sure the shifts are loaded int correct order for the right day

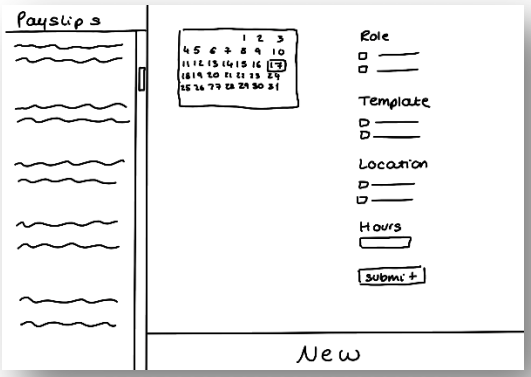
Day Selecting

- Make sure the new shifts are loaded replacing the old ones

Payslip Creation

For a user to create a new payslip they will press the new button which will prompt them to enter in the details of the shift this will include the day, the job role, the template (Contracted or overtime), the location and the hours worked this will then be allowed to be saved. It will also display any past timesheets on the left hand side in a separate column.

Concept Art



Pseudocode

```
FormLoad()

        Payslips = SELECT *

                FROM PayslipTable

                WHERE EmployeeID = LoginInfoID


        For (x: 0 – Payslips.Count)

                PaySlipTextBox = Payslips [x, 1] +  Payslips [x, 2] +  Payslips [x, 3] +  Payslips [x, 4] +
                Payslips [x, 5] +  PayslipTextBox


NewButtonPressed()

        Clear All Fields

SaveButtonPressed()

        Role

                SelectedIndex = 0: Position → Role 1
```

SelectedIndex = 1: Role → Role 2

Template

SelectedIndex = 0: Template → Template1

SelectedIndex = 1: Template → Template2

SelectedIndex = 0: Location → Location1

SelectedIndex = 1: Location → Location2

UpdateDatabase(INSERT INTO PayslipsTable (EmployeeID, Date, Position, Template, Site, Hours

VALUES (EmployeeID, DateValue, PositionChosen, TemplateChosen, SiteChosen, HoursValue))

Queries

SELECT *

FROM PayslipTable

WHERE EmployeeID = LoginInfoID

This selects all the payslip records from the databse linked to the currently logged in user so we can display the right payslips for the right user.

INSERT INTO PayslipsTable (EmployeeID, Date, Position, Template, Site, Hours)

VALUES (EmployeeID, DateValue, PositionChosen, TemplateChosen, SiteChosen, HoursValue)

This updates and adds a new payslip record to the database with the selected fields and hours.

Test Plan

Payslip loading

- Make sure all the payslips are loaded in the correct format

Saving payslips

- Ensure the database is updated correctly
- The new shifts are then loaded in the payslips column on the left

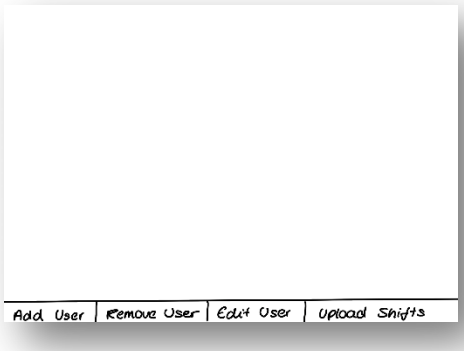
Admin Control (Adding Shifts / Adding Editing and Deleting Users)

All of these following forms will only be visible to users with manager permissions meaning any sensitive data will be kept away from people who don't need access to it. It will also mean that any general users cant go adding and removing users or manipulating any data in the database.

Toolbar Form

Much like the main window toolbar form this will have a new toolbar at the bottom of the page to allow the user to chose what feature they want to access which will then be displayed in the panel docked within this form.

Concept Art



Pseudocode

```
LoadSelectedForm('FormName')

        MainPanel.Add(FormName)

        FormName.Show()


DayButtonPress()

        LoadSelectedForm('FormName')
```

Test Plan

- Make sure the right form is loaded based on what button is pressed

Add User Form

Rows of labels and text boxes will prompt the manager to fill in the account details for the user they want to add they will choose their password which can then be changed later on by the user once they have logged in. To create the new user the manager will have to confirm his password ensuring that everything is correct and its what they want to do.

Rules for inputs

Name: Has to be two words

Email: Must contain an @ and a .

Phone Number: Must start with +44 and be 13 Characters long

Username: Must be between 5 and 20 characters

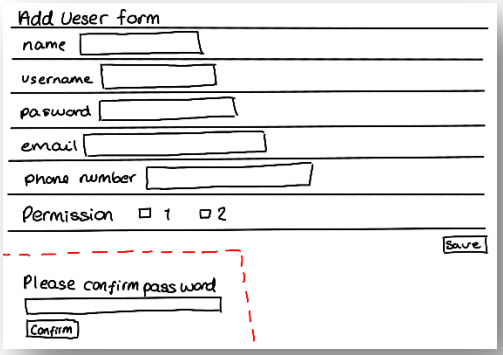
Password: Must have a special character

Must have a Capital letter

Must contain a number

Must be between 5 and 20 characters long

Concept Art



Pseudocode

```
SaveButtonClick()
    Check all inputs valid
    If (valid)
        Show password confirm panel
ConfirmButtonClick()
    VerifyPassword(TextBoxInput, PasswordFromDatabase)
    If (TextBoxInput == PasswordFromDatabase)
        Get all values from input fields
        ExecuteSQL( INSERT INTO EmployeeDetails (FirstName, Surname, Username,
                Password, Salt, Email, PhoneNumber, Permission
                Values (*stored values*)
                WHERE EmployeeID = *UsersEmployeeID*)
    Else
        Messagebox.Show("Error message")
```

Queries

```
INSERT INTO EmployeeDetails (FirstName, Surname, Username, Password , Email, PhoneNumber)
Values (FirstName.TxtBox, Surname.TxtBox, Username.TxtBox, Password.TxtBox, Email.TxtBox,
PhoneNumber.TxtBox)
WHERE EmployeeID = *UsersEmployeeID*
```

This updates all the fields in the employee details table with the new values weather they've been changed or not.

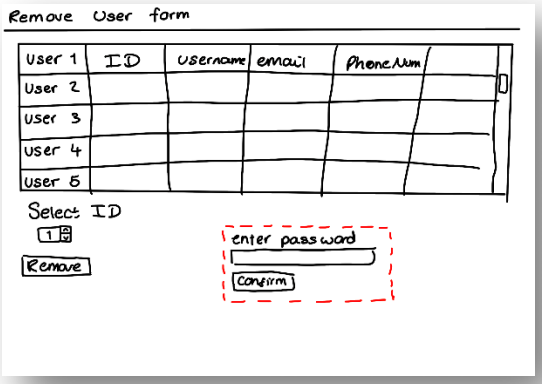
Test Plan

- Make sure the field validation all works
- Password confirmation works
- Database is updated with the new user
- The new user can log in

Remove User Form

A grid of all current users will be displayed with their ID and a few other details, this can all only be accessed by a managers account, the user can then select an ID from the grid and press remove to remove the user with that ID. To do this they will have to confirm their password.

Concept Art



Pseudocode

```
FormLoad()

        String[,] Employees = Execute( SELECT EmployeeID, Firstname, Surname, Email,
                                       PhoneNumber FROM EmployeeDetails)
        New List = ListOfEmployees
        For (Employees.Length)
                ListOfEmployees.Add(New EmployeeDetailsClass(Employees[0], Employees[1],
                                    Employees[2], Employees[3], Employees[4]))
                ValidEmployeeIDs += Employees[0]
        GridView.Datasource = ListOfEmployees

RemoveButtonPress()
        Show password confirm panel


ConfirmButtonPress()
        If (VerifyPassword == true and ValidID == true)
                DELETE all records with EmployeeID selected
```

Queries

```
SELECT EmployeeID, Firstname, Surname, Email, PhoneNumber

FROM EmployeeDetails
```

This returns all the atribute fields we need to display in the datagridview.

```
DELETE FROM EmployeeDetails
WHERE EmployeeID = 'SelectedID'
DELETE FROM Messages
WHERE EmployeeID = ' SelectedID
DELETE FROM ContractedShifts
WHERE EmployeeID = ' SelectedID
DELETE FROM OvertimeShifts
WHERE EmployeeID = ' SelectedID
DELETE FROM PaySlips
WHERE EmployeeID = ' SelectedID
```

Delets any record form all tables with the selected users ID

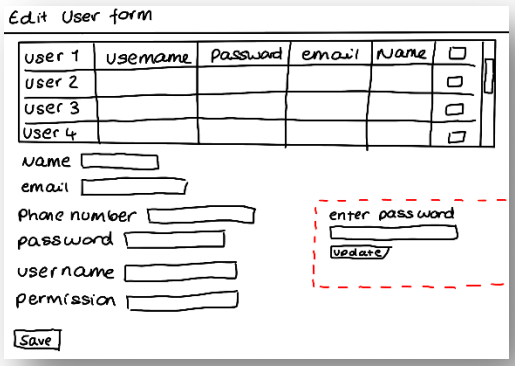
Test Plan

- The selected id is removed
- The database is loaded and is scrollable
- Test what happens when an ID that doesn't have a user attached is selected

Edit User Form

Much like the remove user form a grid will all users will be displayed and from their the manager can select a user and the details will be loaded into the text boxes which the manager can then edit and update.

Concept Art



Pseudocode

UpdateDatabase()

```
String[,] Employees = Execute( SELECT EmployeeID, Firstname, Surname, Email,
                                PhoneNumber FROM EmployeeDetails)
New List = ListOfEmployees
For (Employees.Length)
    ListOfEmployees.Add(New EmployeeDetailsClass(Employees[0], Employees[1],
                        Employees[2], Employees[3], Employees[4]))
    ValidEmployeeIDs += Employees[0]
GridView.Datasource = ListOfEmployees
```

```
SelectButtonPress()

    If (ValidIDSelected == true)

        Enable all text fields

        Details = ExecuteSQL( SELECT Firstname, surname, username, email, password,
                              phonenumber, permssion FROM EmployeeDetails WHERE
                              EmployeeID = SelectedID )

        NameTextBox = Details[0] + Details[1]

        UsernameTextBox = Details[2]

        EmailTextBox = Details[3]

        PasswordTextBox = Details[4]

        PermissionTextBox = Details[5]
```

```
SaveButtonPress()

    Format all input fields and store in local variables

    Update database where EmployeeID == EmployeeID_Selected

    Set Values to input fields.
```

Queries

```
SELECT EmployeeID, Firstname, Surname, Email, PhoneNumber

FROM EmployeeDetails
```

This returns all the atribute fields we need to display in the datagridview.

```
SELECT Firstname, surname, username, email, password, phonenumber, permssion FROM
EmployeeDetails

WHERE EmployeeID = SelectedID
```

This Query selects all the editable fields from the database so we can load them into the text boxes for them to changed and saved.

INSERT INTO EmployeeDetails (FirstName, Surname, Username, Password , Email, PhoneNumber)

Values (FirstName.TxtBox, Surname.TxtBox, Username.TxtBox, Password.TxtBox, Email.TxtBox, PhoneNumber.TxtBox)

WHERE EmployeeID = *UsersEmployeeID*

This updates all the fields in the employee details table with the new values weather they've been changed or not.

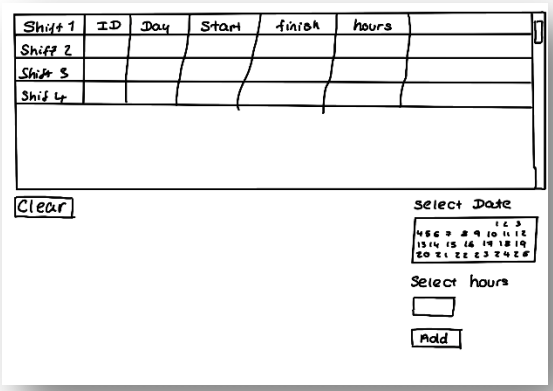
Test Plan

- Any updated fields are updated in the database too
- The password confirmation works

Add Shift Form

The upload shift form won't have a too dissimilar design with a grid showing any current shifts. There will be a clear button which will delete all uploaded shifts this will be used at the start of a new week for example. A manager can then select a date start time and number of hours for the shift and press the upload button to add it to the database.

Concept Art



Pseudocode

```
UpdateGridView()

        String[,] Shifts = Execute( SELECT EmployeeID, Day, StartTime, Hours,
                                    PhoneNumber FROM OvertimeShifts WHERE EmployeeID IS
                                    NULL)
        New List = ShiftList
        For (Shifts.Length)
                ShiftList.Add(New ShiftDetailsClass(ShiftList [0], ShiftList [1],
                                        ShiftList [2], ShiftList [3]]))
        GridView.Datasource = ShiftList
```

```
FormLoad()
        UpdateGridView()

ClearButtonPress()
        ExecuteSQL( DELETE FROM OvertimeShifts )
        UpdateGridView()

AddButtonPress()
        Hours = Hours.TextBox.Text
        Time = DateTimePicker.Time
        Day = DateTimePicker.Day
        ExexuteSQL( INSERT INTO OvertimeShifts (Hours, Day, StartTime, Date)
                VALUES (Hours, Day, Time, DateTimePicker.Date)
        UpdateGridView()
```

Queries

```
SELECT EmployeeID, Day, StartTime, Hours, PhoneNumber FROM OvertimeShifts
WHERE EmployeeID IS NULL
```

This is used to get all of the details from the shifts with no employeeID so we can display them into the data grid view

```
DELETE FROM OvertimeShifts
```

This delets everything from the overtime shifts table and is used to clear all the previous weeks shifts

```
INSERT INTO OvertimeShifts (Hours, Day, StartTime, Date)
VALUES (Hours, Day, Time, DateTimePicker.Date)
```

This is used to add a new shift record to the database with the given details

Test Plan

- Shifts are added to the db and the view is updated
- The added shifts are displayed to any other users
- The clear feature works deleting all shifts in the database

Technical Solution

Classes and Function libraries

Database Connection and SQL execution

This class was NOT made by me, it was provided by a past teacher of mine and I have been using it in all of my projects that require database manipulation this has allowed me to full understand and utilize this class and the comments were written by me. But essentially it will connect to the database executing an SQL query from the call statement which will then either return a 2d array with the selected records or a boolean value to indicate weather the execution was succcessful.

```
using System.Data.SQLite;

namespace DBSQLite_Class
{

    internal class DBSQLite_Class
    {
        private string connectionString = "";

        public DBSQLite_Class(string connectionString)
        {
            //Assign the passed in string to the private attribute of the class connString
            this.connectionString = connectionString;
        }
        //This function takes an sql statement and returns a 2d arrray
        public string[,] GetRecords(string SQLString)
        {
                // Creates a new connection to the database
            SQLiteConnection con = new System.Data.SQLite.SQLiteConnection(connectionString);

            con.Open();

            SQLiteCommand command = new System.Data.SQLite.SQLiteCommand(SQLString, con);

                // Executes the SQLcommand
            SQLiteDataReader reader = command.ExecuteReader();

            //Calling the FieldCount attribute to get the number of columns returned from the sql query
            int col_count = reader.FieldCount;

            int row_count = 0;

            while (reader.Read())
            {
                row_count = row_count + 1;
            }
            reader.Close();

            //2D array initialised with the number of rows and columns needed to store the results of the query
            string[,] records = new string[row_count, col_count];
```

```
            //Execute the sql again
            reader = com.ExecuteReader();

            int count = 0;

            //Loop through the reader and pupulate 2d array with the data
            while (reader.Read())
            {
                for (int x = 0; x < col_count; x++)
                {
                    records[count, x] = reader.GetValue(x).ToString();
                }
                        // Incriment the loop count
                count++;
            }

            reader.Close();
            con.Close();

            //return the the 2d array to the procedure
            return records;

        }

        //This method takes an sql (insert, update, delete) statement
        public bool UpdateInsertDeleteRecords(string sqlString)
        {
            bool success = false;

                // Opens a new connection to the database
            System.Data.SQLite.SQLiteConnection con = new System.Data.SQLite.SQLiteConnection(connString);
            System.Data.SQLite.SQLiteCommand com = new System.Data.SQLite.SQLiteCommand(con);
            con.Open();

                // Executes the command sql query
            com.CommandText = sqlString;
            com.ExecuteNonQuery();

            con.Close();

                // Indicates that the procedure was executed successfully
            success = true;
            return success;
        }
    }

}
```

This next class is one I made to just simplify the calling of the DBSQLite class buy combining the connection string with the choice of execution so you don't have to create a new connection every time as its created in the class.

```
using System;

namespace Moon_Archie_Winforms_NEA
{
    public class Database_Opperations_Class
```

```
    {
            // Takes the query and the choice of operation
        public static string[,] Connect_Read_Write(string Query, string Opperation)
        {
                // Createas a new connection to the database wich is hardcoded into the program as only one
database is ever used
            DBSQLite_Class.DBSQLite_Class db = new DBSQLite_Class.DBSQLite_Class(@"data source =
MainDatabase.db");

            if (Opperation.ToLower() == "write")
            {
                        // Calls the dbsqlite class to execute the query but returns a null array as there are no
records to return
                db.UpdateInsertDeleteRecords(Query);
                string[,] Null_Return = { };
                return Null_Return;
            }
            else if (Opperation.ToLower() == "read")
            {
                        // Returns the records retrieved in a 2d array
                string[,] results = db.GetRecords(Query);
                return results;
            }
            else { throw new Exception(); }
        }
    }
}
```

This makes the call function to manipulate the database go from this:

```
DBSQLite_Class.DBSQLite_Class db = new DBSQLite_Class.DBSQLite_Class(@"data source =
MainDatabase.db");
string query = "...";
string[,] results = db.GetRecords(query);
```

To this:

```
string Query = "...";
string[,] results = Database_Opperations_Class.Connect_Read_Write(Query, "read");
```

Input validation class

This class allows me to input any string field and test for a variety of reqyirements such as the length weather its an email a phone number and more.

```
using System;
using System.Linq;

namespace Moon_Archie_Winforms_NEA
{
    public class Data_Validation_Functions_Class
    {
```

```
    // Tests weather the string is empty
public static bool IsEmpty(string String)
{
    return String.IsNullOrWhiteSpace(String);
}

    // Tests weather the string is bellow a max specified length
public static bool IsBellowMaxLength(string String, int Length)
{
    if (String.IsNullOrEmpty(String)) return false;

    return String.Length <= Length;
}

    // Tests if the string is abobve a max length
public static bool IsAboveMinLength(string String, int Length)
{
    if (String.IsNullOrEmpty(String)) return false;

    return String.Length >= Length;
}

    // Tests weather a string is between a max and min specified length
public static bool IsBetweenLength(string String, int Min, int Max)
{
    if (String.IsNullOrEmpty(String)) return false;

    return String.Length >= Min && String.Length <= Max;
}

    // Tests weather a string contains an @ and a . representing an email
public static bool IsValidEmial(string String)
{
    if (String.IsNullOrEmpty(String)) return false;

    return String.Contains("@") && String.Contains(".");
}

    // Test weather a string is a phone number by it starting with +44
public static bool IsValidPhoneNumber(string String)
{
    if (String.IsNullOrEmpty(String)) return false;

    return String.Substring(0, 3) == "+44" && IsBetweenLength(String, 13, 13);
}

    // Tests weather a string is a valid password by it meeting specific requirements
public static bool IsValidPassword(string String, int Min, int Max)
{
    if (String.IsNullOrEmpty(String)) return false;

        // Sets a series of bools weather the string matches each condition
    bool ContainsSpecialCharacter = String.Any(char.IsPunctuation);
    bool ContainsNumber = String.Any(char.IsDigit);
    bool ContainsUppercase = String.Any(char.IsUpper);
```

```
            bool BetweenLength = IsBetweenLength(String, Min, Max);

                // Returns the output of all the conditions
            return ContainsSpecialCharacter && ContainsUppercase && ContainsNumber && BetweenLength;
        }

            // Tests weather a string is a specified number of words
        public static bool IsNumberOfWords(string String, int NumberOfWords)
        {
            if (String.IsNullOrEmpty(String)) return false;

            string[] NameParts = String.Split(' ');
            return NameParts.Length == NumberOfWords; ;
        }
    }
}
```

Passowrd hashing

Hashing function

This is my hashing algorithm used to make users passwords secure in my database, it converts each character in the users password to its ascii value which is then XOR'd agains the ascii value of the character at the same index of the key, which is hardcoded into the program, which is then converted into hexadecimal and makes up the hashed password.

```
using System;
using System.Text;

public class Password_Hasher_Class
{
    // PRIVATE Key embedded within the project
    private static string KEY = "cYl9jS}9b#Xdm=CtHy(sDN:qTNW6U4b9";

    public static string HashPassword(string Password, string salt)
    {
        // appends the salt to the password
        Password = salt + Password;

        StringBuilder hashedPassword = new StringBuilder();

        //If the lenght of the password > the length of the key, key will be doubled
        while (KEY.Length < Password.Length)
        {
            KEY += KEY;
        }

        for (int i = 0; i < Password.Length; i++)
        {
            //Converts the XOR result of the password character and the character of the key at the same index
to ascii
            int hashedChar = Password[i] ^ KEY[i];

            //converts the ascii into Hexadecimal and appends it to the hashedPassword string
            hashedPassword.Append(hashedChar.ToString("X2"));
```

```
        }

        return hashedPassword.ToString();
    }
```

Password verification

The verify password function takes the entered password along with a salt and the hashed password stored in the database. It then converts each 2 characters (each hexadecimal value) back into ascii and then XOR's this value with the ascii of the character at the same index of the key to get the original value. It then compares the complete unhashed password to the entered password.

```
    public static bool VerifyPassword(string Password, string Salt, string HashedPassword)
    {
        // appends the users salt to the password the user entered
        Password = Salt + Password;

        StringBuilder UnhashedPassword = new StringBuilder();

        //If the lenght of the password is more than the length of the key the key will be doubled
        while (KEY.Length < Password.Length)
        {
            KEY += KEY;
        }

        // loops for the number of characters in the hashed password
        for (int x = 0; x < HashedPassword.Length; x += 2)
        {
            // gets the current hexadecimal value to be 'un-hashed'
            string hexChar = HashedPassword.Substring(x, 2);

            //converts that hex value into the ascii value
            int charValue = Convert.ToInt32(hexChar, 16);

            // Performs an XOR on the password and key ascii values to find the original character
            UnhashedPassword.Append(Convert.ToChar(charValue ^ KEY[x / 2]));
        }

        //returns the boolean result of the unhashed password compared to the one they entered
        return UnhashedPassword.ToString() == Password;
    }
```

Salt Generation

The generate salt funtion is whats used when a new user is made to create their unique salt. All it does is selects 32 random characters from a list of all possible characters to generate a unique string.

```
    public static string GenerateRandomSalt()
    {
        Random rnd = new Random();

        const string Characters =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()_+-=[]{};:,.<>?";
```

```
        // length of the salt
        int SaltLength = 32;

        StringBuilder Salt = new StringBuilder();

        for (int i = 0; i < SaltLength; i++)
        {
            // gets a random index from the array of characters
            int index = rnd.Next(Characters.Length);

            // appends the corrosponding character to the salt
            Salt.Append(Characters[index]);
        }

        // returns the salt
        return Salt.ToString();
    }
}
```

Login details class

Although I use multiple classes the login info class is the most inportant and most referenced as it stores all the current logged in users details retireved from the database and is used in a majority of the forms.

```
namespace Moon_Archie_Winforms_NEA
{
    internal class LoginInfo
    {
        //database info stored at login
        public static string EmployeeID { get; set; }
        public static string Username { get; set; }
        public static string Password { get; set; }
        public static string Salt { get; set; }
        public static string First_Name { get; set; }
        public static string Surname { get; set; }
        public static string Email { get; set; }
        public static string PhoneNumber { get; set; }
        public static string Permissions { get; set; }
    }
}
```

Algorithm to generate and load the daily rota

Code Listings

Constants and string initialization

First we need to initialize a few constants, these define the number of columns as well as the width of the first, names, column and the columns that hold the numbers. We also initialize a placeholder for the number of rows, the selected day and the string (rota) to save.

```
private const int NumColumns = 33;
private const int ColumnWidth = 20;
private const int FirstColumnWidth = 55;
```

```
private int NumRows = 0;

private string SelectedDay = "";
private string StringToSave = "";
```

Creating the grid dynamically

In order to create the grid that will display the details of the rota we need to first loop through each row of the grid checking if there is a saved rota stored in the database, this will decide weather we load that in and use the saved rota or begin to generate our own.

Within each row loop we also loop through each column calculating the x coordinate of the column and creating a text box with the given size location and name. We then load this into a 2D array so we can manipulate them later if we need. We also load the corresponding value from the generated random string into the text box.

```
private void CreateDynamicGrid(int rows)
{
    Random random = new Random();

    NumRows = rows;

    // The 2D array of TextBoxes that holds the grid
    TextBoxes = new TextBox[NumColumns, NumRows];

    // Loops through each row
    for (int row = 0; row < NumRows; row++)
    {
        string generatedString = "";
        if (CheckForSavedRota(SelectedDay) == true)
        {
            // if no rota is saved in the databse we generate a new string for each row
            generatedString = GenerateString(random);
            StringToSave = StringToSave + ":" + generatedString;
        }
        else
        {
            // if there is a saved rota we get each rows string from the database
            string Query = "SELECT Rota " +
                           "FROM DailyRotas " +
                           $"WHERE Day = '{SelectedDay}'";
            string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");
            string Rota = Results[0, 0];

            string[] substrings = Rota.Split(new char[] { ':' }, StringSplitOptions.RemoveEmptyEntries);

            generatedString = substrings[row];
        }
        // Loops through each column
        for (int col = 0; col < NumColumns; col++)
        {
            // Calculates the x-coordinate based on the index of the column
            // This is as the first column will have larger text boxes meaning the second column will need to be
adjusted acordingly
```

```
            int xCoordinate = col == 0 ? col * FirstColumnWidth : FirstColumnWidth + col * ColumnWidth;

            // Creates a new text box
            TextBox textBox = new TextBox
            {
                // Sets the size based on weather its the first column or not
                Size = new Size(col == 0 ? FirstColumnWidth : ColumnWidth, 0),

                // Sets the location of each text box
                Location = new Point(xCoordinate, (row * 21) + 20),

                // Sets the name for each text box with the x and y coordinates
                Name = $"TextBox_{col}_{row}",
            };
            Controls.Add(textBox);

            // Stores it in the 2D array so it can be manipulated if needed
            TextBoxes[col, row] = textBox;
            ChangeTextOfTextBox(col, row, generatedString[col].ToString());
        }
    }
}
```

Generating the random string

To do this we get all the details of every shift on a give day from the databse we then generate a new grid with the length being the number of records in the database. To fill the first index with the names we then loop for the number of records changing the text, of the first column with the given row index, to the name of the record at that same index in the database.

```
private void Load_Employee_Names_and_Generate(string Day)
{
    // Selects the firstname from the given employeeID that is ascociated with all shifts, overtime and
contracted, on a given day
    // The union all is used to combine the results of two select statements
    string Query = "SELECT EmployeeDetails.FirstName " +
                   "FROM EmployeeDetails " +
                   "JOIN ContractedShifts ON EmployeeDetails.EmployeeID = ContractedShifts.EmployeeID " +
                   $"WHERE ContractedShifts.Day = '{Day}' " +
                   "UNION ALL " +
                   "SELECT EmployeeDetails.FirstName " +
                   "FROM EmployeeDetails " +
                   "JOIN OvertimeShifts ON EmployeeDetails.EmployeeID = OvertimeShifts.EmployeeID " +
                   $"WHERE OvertimeShifts.Day = '{Day}'";
    string[,] results = Database_Opperations_Class.Connect_Read_Write(Query, "Read");

    CreateDynamicGrid(results.Length);

    for (int i = 0; i < results.Length; i++)
    {
        // Calls the change text function for the text boxes in the first column and sets them to the name of
the employee
        ChangeTextOfTextBox(0, i, results[i, 0]);
    }
```

```
}
```

Blocking out the text boxes that arent needed

This invloves changing the colour of the text boxes that don't cover the time of the shift. To do this we need to select the start time and the number of hours of every shift in both contracted and overtime databases.

To do this we need to loop through every retrieved shift record and work out the starting index for the first box of the shift, we then work out the end index of the shift and then we loop through it again chaning the colours of all boxes, to indicate theyre blocked out, from the 0^{th} index to the calculated start index and then from the end index to the final, 32^{nd} index.

```
private void BlockOut_TextBoxes(string Day)
{
   // The query that selects the start time and ammount of hours for every shift on a given day
   string Query = "SELECT StartTime, Hours " +
                  "FROM ContractedShifts " +
                  $"WHERE Day = '{Day}' " +
                  "UNION ALL " +
                  "SELECT StartTime, Hours " +
                  "FROM OvertimeShifts " +
                  $"WHERE Day = '{Day}'";
   string[,] results = Database_Opperations_Class.Connect_Read_Write(Query, "Read");

   // Loops through each row in the results array
   for (int i = 0; i < results.GetLength(0); i++)
   {
      // Calculates the value for the starting index for each shift based off the start time
      // Performs: ( Start time - Earliest start time ) * 2 to get the apropriate index
      double StartIndex = (double.Parse(results[i, 0]) - 6.5) * 2;

      results[i, 0] = StartIndex.ToString();

      // Calculates the end index for the shift
      // Each index (box) is 30 mins worth so the end is start index (calculated above) + (number of hours *
2)
      StartIndex = (double.Parse(results[i, 0]) + (double.Parse(results[i, 1])) * 2) + 1;

      results[i, 1] = StartIndex.ToString();

   }
   // Loops through each row in the results array
   for (int i = 0; i < results.GetLength(0); i++)
   {
      // Sets the boxes from index 0 to the start index to DarkGrey
      for (int j = 1; j <= int.Parse(results[i, 0]); j++)
      {
         ChangeBackColorOfTextBox(j, i, Color.DarkGray);
         ChangeTextOfTextBox(j, i, "");
      }
      // Sets the boxes from the end index to the final index to DarkGrey
      for (int k = int.Parse(results[i, 1]); k <= 32; k++)
      {
```

```
            ChangeBackColorOfTextBox(k, i, Color.DarkGray);
            ChangeTextOfTextBox(k, i, "");
        }
    }
}
```

Secure login system

Code Listings

Login Handling

This form takes the username the user has entered and checks for any records in the database with that username, then it gets that password unhashes it and compares it to the one the user entered. If they match the user can then login but if they arent the user is denied access and the attempt tally is incrimented.

```
private void Login_Btn_Click(object sender, EventArgs e)
{
    // Checks if the user has reached the max number of attempts
    if (attempts == 5)
    {
        MessageBox.Show("You have reached the maximun number of login attempts.");
        Username_TxtBox.Text = "";
        Password_TxtBox.Text = "";
    }
    else
    {
        string username = Username_TxtBox.Text;
        string password = Password_TxtBox.Text;

        // Creates a new connection to the MainDatabase
            DBSQLite_Class.DBSQLite_Class db = new DBSQLite_Class.DBSQLite_Class(@"data source =
            MainDatabase.db");

            //selects count of how many entries in the database match username entered
        string query = "SELECT Count(*) FROM EmployeeDetails " +
                "WHERE Username = '" + username + "'";

        string[,] results = db.GetRecords(query);

            // Selects all details of the user with the entered username from   employeedetails
            query = "SELECT EmployeeID, FirstName, Surname, Username, Password, Salt,                Email,
            PhoneNumber, Permission " +
             "FROM EmployeeDetails " +
             "WHERE Username = '" + username + "'";
        string[,] UserInfo = db.GetRecords(query);

            // checks if there is more than one account retrieved from the db
            // and if the password and username aren't null
        if (int.Parse(results[0, 0]) > 0 && password != "" && username != "")
        {
                    // Checks If the password entered is verified agains the one stored in   the database
                if (Password_Hasher_Class.VerifyPassword(password, UserInfo[0, 5],          UserInfo[0, 4]) == true)
```

```
        {
                        // All the details for that user are loaded into the userinfo class
            LoginInfo.EmployeeID = UserInfo[0, 0];
            LoginInfo.First_Name = UserInfo[0, 1];
            LoginInfo.Surname = UserInfo[0, 2];
            LoginInfo.Username = UserInfo[0, 3];
            LoginInfo.Password = UserInfo[0, 4];
            LoginInfo.Salt = UserInfo[0, 5];
            LoginInfo.Email = UserInfo[0, 6];
            LoginInfo.PhoneNumber = UserInfo[0, 7];
            LoginInfo.Permissions = UserInfo[0, 8];

                        // Toolbar form is loaded and the login form is hidden then closed
            Main_WindowToolbar Main_WindowToolbar = new Main_WindowToolbar();
            this.Hide();
            Main_WindowToolbar.ShowDialog();
            this.Close();
        }
        else
        {
            MessageBox.Show($"Login Failed - Please try again.");
            Username_TxtBox.Text = "";
            Password_TxtBox.Text = "";
            attempts++;
            return;
        }
      }
      else
      {
          MessageBox.Show($"Login Failed - Please try again.");
          Username_TxtBox.Text = "";
          Password_TxtBox.Text = "";
          attempts++;
          return;
      }
   }
}
```

Multi-User real time messaging

Code Listings

Server-Side code

This java script is used to set up event listners one for the message received event and one for the send button being pressed. The message received listner just takes the message and adds it to a list which can be used to display it to the user. The send button click event just stores the message and calls the send message task which sends it to all connected clients.

```
<!DOCTYPE html>
<html>
<body>
```

```
    <script>
        //when the listner is triggered by the ReceiveMessage function on the websocket server
        connection.on("ReceiveMessage", (message) => {
            const li = document.createElement("li");
            li.textContent = message;
            // this adds the new list element to the message list allowing it to be displayed to the user
            document.getElementById("messagesList").appendChild(li);
        });

        //starts the signalR connecton
        connection.start().catch((error) => console.error(error.toString()));

        // creates a listner for when the send message button is pressed
        document.getElementById("sendButton").addEventListener("click", () => {
            //stores the message inputed
            const message = document.getElementById("messageInput").value;
            document.getElementById("messageInput").value = "";
            //invokes the send message function and catches and outputs any errors
            connection.invoke("SendMessage", message).catch((error) => console.error(error));
        });
    </script>
</body>
</html>
```

The server side class contains a SendMessage task this takes in a string message and sends the message to all connected clients and sends it to the receiveMessage method. The reason this class inherits from 'Hub' is to allow it use funtionality offered by signalR in this case it uses the SendMessage method to send messages to the server and then back out to every client.

```
public class ChatHub : Hub
{
    public async Task SendMessage(string message)
    {
        // sends out the message to all connected clients Using the recieveMessage event
        await Clients.All.SendAsync("ReceiveMessage", message);
    }
}
```

Finally we have the code to create the ASP.NET application and define the endpoints for the signalR communication as well as handle basic error handling and to indicate weather the server is responding.

```
// Creates a web app builder which will be used as our server interface
var builder = WebApplication.CreateBuilder(args);

// adds the signalR library to the web application
builder.Services.AddSignalR();

var app = builder.Build();


app.UseExceptionHandler("/Home/Error");
```

```
// Forces communication to use HTTPS
app.UseHsts();


// This endpoint is what users will connect to for communication
app.MapHub<ChatHub>("/chatHub");

// writes connected upon the initial request to the server proving its connected
app.MapGet("/", async context =>
{
    await context.Response.WriteAsync("Connected");
});

app.Run();
```

Client-Side messaging code

Connection set up and Message handling

Initializes a signalR connection hosted on a specified URL

```
// Creates the signalR connection and tells it what host to connect to
hubConnection = new HubConnectionBuilder()
    .WithUrl("https://moonarchieserversidenea.azurewebsites.net/chatHub")
    .Build();
```

This is the listner for when a message is received using the ReceiveMessage event from the server. It displays the message as well as the name of the user who sent it in the message field.

```
// Event handler for when a message is recieved
hubConnection.On<string>("ReceiveMessage", (Message) =>
{
    // Splits and stores the name and the ID into seperate strings
    string[] Message_And_ID_Strings = Message.Split(':');
    string Id_From_Message = Message_And_ID_Strings[1];

    // Connects to the database and gets the name if the employee with that id
    string Query = "SELECT FirstName FROM EmployeeDetails " +
                $"WHERE EmployeeID = '{Id_From_Message}'";
    string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");

    // If the ID of the user isnt the same as the ID of the sender
    // The message is displayed with the name of the sender
    if (LoginInfo.EmployeeID != Id_From_Message)
    {
        Message = Message_And_ID_Strings[0];

        // Displays the Message (Name + : + Message Text) in the text box
        DisplayMessage(Results[0, 0] + $": {Message}\n", Messages_TxtBox);
    }
```

```
    // If there is a connection it adds the new message to the database
    // Any messages sent when there is no connection will not be stored
    if (connected == true)
    {
        Query = "INSERT INTO Messages (EmployeeID, Message) " +
                $"VALUES ('{LoginInfo.EmployeeID}',                    '{MessageInput_TxtBox.Text}');";
        Database_Opperations_Class.Connect_Read_Write(Query, "Write");
    }
});
```

Connecting to server

Here we try and connect to the server using the StartAsync() method, if the connection is successful we display an indication to the user, if there is an error in connecting the error message is displayed to the user and a negative indication is displayed.

```
private async Task ConnectAsync()
{
    try
    {
        await hubConnection.StartAsync();

        // Sets the connection button to green if connection is made
        Connect_Btn.BackColor = Color.LimeGreen;
        connected = true;
    }
    catch (Exception error)
    {
        if (error.Message == "The HubConnection cannot be started if it is not in the Disconnected state.")
        {
            // If the error is because the connection is already made connection the button stays green
            Connect_Btn.BackColor = Color.LimeGreen;
        }
        else
        {
            // If there is a connection error for another reason the button turns red and the error is displayed
            DisplayMessage($"Connection error: {error.Message}\n", Messages_TxtBox);
            Connect_Btn.BackColor = Color.Firebrick;
        }
    }
}
```

Message Sending

We now need to handle what happens when the user sends a message, this simply calls the send message event on the server, which will send the message to all connected clients, and also displays the message to the user who sent it too. This is an Asyncronous function to allow for multiple messages to be sent at the same time avoiding any concurence or collision issues.

```
private async Task SendMessageAsync(string Message)
```

```
{
    try
    {
        // Calls the send message method on the server
        await hubConnection.InvokeAsync("SendMessage", Message);

        // Splits the name and the ID into seperate strings
        string[] Message_And_ID_Strings = Message.Split(':');
        Message = Message_And_ID_Strings[0];

        // Displays the Message (You + : + Message Text) in the text box
        DisplayMessage($"You: {Message}\n", Messages_TxtBox);
    }
    catch (Exception ex)
    {
        DisplayMessage($"Error sending message: {ex.Message}\n", Messages_TxtBox);
    }
}
```

This is the funtion that is called when the user presses the send message button.

Account management

Code Listings

Account field population

This code retrieves the users data stored in the loginInfo class and loades it into the appropriate text boxes.

```
private void Account_Frm_Load(object sender, EventArgs e)
{
    // Data is loaded into the text box fields from the static logininfo class
    Name_TxtBox.Text = LoginInfo.First_Name + " " + LoginInfo.Surname;
    Email_TxtBox.Text = LoginInfo.Email;
    PhoneNumber_TxtBox.Text = "+" + LoginInfo.PhoneNumber;
    Username_TxtBox.Text = LoginInfo.Username;
    // A placeholder string otherwise the field will show the hashed string which is far too long
    Password_TxtBox.Text = "Password";
}
```

Saving Changes

On the save button click event it starts off by using the data validation class to verify every field, given the specific requirements given, regardless of weather its been changed or not.

```
// Uses the data validation class to check each field is appropriate and can be added to the database
bool ValidName = Data_Validation_Functions_Class.IsNumberOfWords(Name_TxtBox.Text, 2);

bool ValidEmail = Data_Validation_Functions_Class.IsValidEmial(Email_TxtBox.Text);

bool ValidPhoneNumber =
Data_Validation_Functions_Class.IsValidPhoneNumber(PhoneNumber_TxtBox.Text);
```

```
bool ValidUsername = Data_Validation_Functions_Class.IsBetweenLength(Username_TxtBox.Text, 5, 20);
```

If all of the fields are correct we then need to update the database but we need to check if the password has been chnaged first (this is because if it hasn't been changed and is updated to the database then the placeholder string will be inserted instead)

If the password has been changed we check if its valid then hash the new password and update the database and all the fields in the loginInfo class.

```
if (Password_Changed == true)
{
   bool ValidPassword = Data_Validation_Functions_Class.IsValidPassword(Password_TxtBox.Text, 5, 20);
   if (ValidPassword)
   {
      // The new password is hashed and stored locally
      string Password = Password_Hasher_Class.HashPassword(Password_TxtBox.Text, LoginInfo.Salt);

      // All changes are updated in the SQLite database and within the loginInfo static class
      Query = $"UPDATE EmployeeDetails " +
         $"SET FirstName = '{FirstName}', Surname = '{Surname}', Username = '{Username_TxtBox.Text}',
Password = '{Password}', PhoneNumber = '{PhoneNumber_TxtBox.Text.Substring(1)}' " +
         $"WHERE Username = '{LoginInfo.Username}' AND Password = '{LoginInfo.Password}'";

      Database_Opperations_Class.Connect_Read_Write(Query, "write");

      // The loginInfo class is updated
      LoginInfo.Email = Email_TxtBox.Text;
      LoginInfo.PhoneNumber = PhoneNumber_TxtBox.Text.Substring(1);
      LoginInfo.Username = Username_TxtBox.Text;
      LoginInfo.First_Name = FirstName;
      LoginInfo.Surname = Surname;
           LoginInfo.Password = Password;

      Password_TxtBox.Text = "Password";
   }
   else
   {
      MessageBox.Show("One or more data fields is in incorrect format.\nPlease double check all fields are
correct.\nPlease make sure your password contains:\nUppercase (1)\nSpecial Characters (1)\nNumbers
(1)\nCharacters (5 - 20)");

   }
}
```

If the password hasn't been changed then we can do all the same things but don't touch the password stored in the database keeping it the same.

```
else
{
   // The password isnt included in this query as it hasnt been changed
   Query = $"UPDATE EmployeeDetails " +
```

```
        $"SET FirstName = '{FirstName}', Surname = '{Surname}', Username = '{Username_TxtBox.Text}',
PhoneNumber = '{PhoneNumber_TxtBox.Text.Substring(1)}' " +
        $"WHERE Username = '{LoginInfo.Username}' AND Password = '{LoginInfo.Password}'";

    Database_Opperations_Class.Connect_Read_Write(Query, "write");

    // Logininfo class Is updated
    LoginInfo.Email = Email_TxtBox.Text;
    LoginInfo.PhoneNumber = PhoneNumber_TxtBox.Text.Substring(1);
    LoginInfo.Username = Username_TxtBox.Text;
    LoginInfo.First_Name = FirstName;
    LoginInfo.Surname = Surname;

    // Password text box is set back to placeholder string
    Password_TxtBox.Text = "Password";
}
```

Viewing and Accepting shifts

Code Listings

Initialization

The initialize shift loader subroutine is used to calculate the number of shifts to load, if there is more than 10 then the final index for the first page will be 9 but if there is less than 10 the final index will just be the number of records -1.

```
private void Initialize_Shift_Loader(string Day)
{
    string Query = "SELECT OvertimeShiftID, Date, Day, StartTime, Hours " +
                   "FROM OvertimeShifts " +
                   $"WHERE EmployeeID IS NULL AND Day = '{Day}'";
    string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");
    // Gets the number of records with no EmployeeID
    Shift_Loader_Shift_Count = Results.GetLength(0) - 1;

    // If there is less than (or) 10, load them all
    if (Shift_Loader_Shift_Count <= 9)
    {
        Shift_Loader_Start_Index = 0;
        Shift_Loader_End_Index = Shift_Loader_Shift_Count;
        Page_Down_Btn.Enabled = false;
        Page_Up_Btn.Enabled = false;
    }
    // else we only load the first 10
    else
    {
        Shift_Loader_Start_Index = 0;
        Shift_Loader_End_Index = 9;
        Page_Down_Btn.Enabled = true;
    }
}
```

Loading Shifts from database

The loading function retrieves each record from the database on the selected day, formats the data into a more readable form for the user this is then added to a list which is then looped through updating the 'i'th (count of the loop) row of the table layout panel which is filled in with the data and the button is enabled. It also handles the event for when the accep button is pressed which inserts the users employeeID into the selected shift in the database.

```
private void Shift_Loader(string Day)
{
   // Selects the Day Date start time and hours from the overtimeshifts table where no employeeID has
been specified
   string Query = "SELECT OvertimeShiftID, Date, Day, StartTime, Hours " +
                  "FROM OvertimeShifts " +
                  $"WHERE EmployeeID IS NULL AND Day = '{Day}'";
   string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");

   List<string> Shift_Details = new List<string>();

   int i = 0;

   for (int x = Shift_Loader_Start_Index; x <= Shift_Loader_End_Index;  x++)
   {
      // Gets the integer value of the time
      int Hours = (int)double.Parse(Results[x, 3]);

      // Gets the decimal value of the time
      double DecimalPart = double.Parse(Results[x, 3]) - Hours;

      // Calculate minutes from the decimal value
      int Minutes = (int)(DecimalPart * 60);

      // Formats the hours and minutes into HH:mm format
      string FormattedTime = string.Format("{0:00}:{1:00}", Hours, Minutes);

      // Adds the database fields and the formatted time into the list
      Shift_Details.Add($"{Results[x, 1]} | {Results[x, 2]} | Start Time: {FormattedTime} | {Results[x, 4]} Hours");

      // Access' the label and button in the current row i
      Label Label = (Label)Shift_View_Tbl.GetControlFromPosition(0, i);
      Button Button = (Button)Shift_View_Tbl.GetControlFromPosition(1, i);

      // Updates the label with the string from the list
      Label.Text = Shift_Details[i];

      // Set the Tag property of the button to the shift ID
      Button.Tag = Results[x, 0];
      Button.Enabled = true;

      // Update button Text and handle the click event
      Button.Text = "Accept";
      Button.Click += (sender, e) =>
      {
         // Get the shift ID from the Tag property of the button
         string shiftID = Button.Tag.ToString();
```

```
        // Selects the EmployeeID from the OvertimeShifts table Where the shiftID is equal to the one
chosen
        Query = "SELECT EmployeeID " +
                "FROM OvertimeShifts " +
                $"WHERE OvertimeShiftID = {shiftID}";
        Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");
        if (Results[0, 0] == "")
        {
            MessageBox.Show("Accepted");

            // Update the shift in the database using the shiftID
            Query = "UPDATE OvertimeShifts " +
                    $"SET EmployeeID = {LoginInfo.EmployeeID} " +
                    $"WHERE OvertimeShiftID = {shiftID}";
            Database_Opperations_Class.Connect_Read_Write(Query, "write");

            Main_Control_Pnl.Visible = false;
        }
    };
    i++;
  }
}
```

Creating and storing payslips

Code Listings

Input Validation

Before we can update the database adding the new payslip we need to check if the data inputted by the user is in the correct format; this code does that.

```
bool Hours_Is_Double;
try
{
  double.Parse(Hours_Worked_TxtBox.Text);
  Hours_Is_Double = true;
}
catch
{
  Hours_Is_Double = false;
}
if (Position_CheckBox.SelectedIndex.ToString() == "-1" || Template_CheckBox.SelectedIndex.ToString() == "-
1" || Site_CheckBox.SelectedIndex.ToString() == "-1" || Hours_Is_Double != true)
{
  MessageBox.Show("This doesnt look right,\nPlease fill in all fields correctly.");
}
else
{
  // Continue to updating Database
}
```

Updating the Database

Knowing the data is all correct we can then retireve the selected radio buttons and the hours storing them all in the database.

```
//Gets the values of the checkboxes

string Position;
if (Position_CheckBox.SelectedIndex.ToString() == "0") { Position = "Lifeguard"; }
else { Position = "Swim School Assistant Instructor"; }

string Template;
if (Template_CheckBox.SelectedIndex.ToString() == "0") { Template = "Casual"; }
else { Template = "Contracted"; }

string Site;
if (Site_CheckBox.SelectedIndex.ToString() == "0") { Site = "Splashpoint"; }
else { Site = "Wadurs"; }

// Inserts the new record with all the inputs into the database

DBSQLite_Class.DBSQLite_Class db = new DBSQLite_Class.DBSQLite_Class(@"data source =
MainDatabase.db");
string query = "INSERT INTO PaySlips (EmployeeID, Date, Position, Template, Site, HoursWorked) " +
            "VALUES ('" + LoginInfo.EmployeeID + "', '" + Date_Picker.Text + "', '" + Position +"', '" + Template
+"', '" + Site +"', '" + Hours_Worked_TxtBox.Text + "');";
db.UpdateInsertDeleteRecords(query);
```

Home page with widgets to show useful information

Code Listings

The first thing we need to do is to check to see if any of the databases are null in which case we don't want to load the data as it will cause errors, if they arent null then we can go ahead and load from them.

```
private void Home_Frm_Load(object sender, EventArgs e)
 {
    // Checks if any of the databases are empty
    if (DatabaseIsEmpty("PaySlips")) {}
    else { PaySlip_Window_Loader(); }

    if (DatabaseIsEmpty("Messages")) {}
    else { Message_Loader(); }

    if (DatabaseIsEmpty("OvertimeShifts")) {}
    else { Shift_Loader(); }
 }
```

The databaseIsEmpty subroutine contains the code to actually check if the database is empty.

```
private bool DatabaseIsEmpty(string DatabaseName)
{
   // Selects the count from the database entered when the function is called
```

```
    string Query = "SELECT COUNT(*) " +
                   $"FROM {DatabaseName}";
    string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "Read");

    if (int.Parse(Results[0, 0]) == 0)
    {
        // Returns true if count is 0
        return true;
    }
    else {  return false; }
}
```

Payslip Loading

To do this we need to select the count from the database, to know how many times we need to loop through, then we can select all the data from each record; adding each record to the text box one by one.

```
private void PaySlip_Window_Loader()
{
    // Emptys any text from the text box
    Payslip_TxtBox.Text = "";

    // Selects the count from the payslips table where the employeeID is the same as the logged in user
    string query = "SELECT Count(*) " +
                   "FROM PaySlips " +
                   $"WHERE EmployeeID = '{LoginInfo.EmployeeID}'";
    string[,] Results = Database_Opperations_Class.Connect_Read_Write(query, "Read");
    int Count = int.Parse(Results[0, 0]);

    // Selects every field from the payslips table where the employeeID is the same as the logged in user
    query = "SELECT * " +
            "FROM PaySlips " +
            $"WHERE EmployeeID = '{ LoginInfo.EmployeeID}'";
    Results = Database_Opperations_Class.Connect_Read_Write(query, "Read");

    for (int x = 0; x < Count; x++)
    {
        // Formats all the data and adds it to the text box
        Payslip_TxtBox.Text = $"{Results[x, 1]} . {Results[x, 2]} . {Results[x, 3]} . {Results[x, 4]} . {Results[x, 5]}
Hours \n\n{Payslip_TxtBox.Text}";
    }
}
```

Message loading

Firstly we need to work out what our starting index and number of messages we need to load is.

```
// Selects the number of messages in the database
string Query = "SELECT Count(*) " +
               "FROM Messages";
string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");

// Creates an int to store how many times the loader will need to be looped and the index to start at
```

```
int Loop_Count = 0;
int Start_Index = 0;

string Messages_To_add = "";

// If the count minus 9 >= 0, this means that there are more than 10 messages to load
if (int.Parse(Results[0, 0]) - 9 >= 0)
{
    // The loop count is then set to 10
    Loop_Count = 10;

    // The start index is set to the count - 10
    Start_Index = int.Parse(Results[0, 0]) - 10;
}
// If there are less than 10 messages
// We need to work out how many times to loop to avoid an index error
else
{
    // Loop count is set to the number of rows in the database
    Loop_Count = int.Parse(Results[0, 0]);

    // The start index is set to 0
    Start_Index = 0;
}
```

Once we have done this we can then begin to add the messages to the text box using the indexes and loop counts we worked out before, we then get the messages from the database and attatch the apropriate name (the name of the sender or ‘you’ for messages sent by the user) before adding it to the message field

```
// Selects the message and employee name linked to the ID
Query = "SELECT EmployeeDetails.FirstName, Messages.Message " +
            "FROM Messages " +
            "JOIN EmployeeDetails ON Messages.EmployeeId = EmployeeDetails.EmployeeId;";
Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");

for (int x = 0; x < Loop_Count; x++)
{
    // Checks if the Name is the same as the name of the user
    if (Results[Start_Index, 0] == LoginInfo.First_Name)
    {
        // The message is added to string of all messages to add (You: 'Message')
        Messages_To_add = $"{Messages_To_add} You: {Results[Start_Index, 1]} \n\n";
    }
    else
    {
        // The message is added to string of all messages to add (Employee Name: 'Message')
        Messages_To_add = $"{Messages_To_add} {Results[Start_Index, 0]}: {Results[Start_Index, 1]}\n\n";
    }
    Start_Index++;
}
Messages_RichTxtBox.Text = Messages_To_add + Messages_RichTxtBox.Text;
```

Shift Loading

The shift loading function for the home page is very similar to the one used for the shift form. It starts off by selecting the data from each record then it loops for the last 5 index's of the database leading the data of that shift into each column of the table layout panel while enabling the accept button in that row. Again this also handles the accept button click event which uploads the users employeeID to that shift in the database and removes the selected shift from the view.

```
private void Shift_Loader()
{
    List<string> Shift_Details = new List<string>();

    // Selects the Day Date start time and hours from the overtimeshifts table where no employeeID has
been specified
    string Query = "SELECT OvertimeShiftID, Date, Day, StartTime, Hours " +
                   "FROM OvertimeShifts " +
                   "WHERE EmployeeID IS NULL";
    string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");

    int i = 0;

    for (int x = Results.GetLength(0) - 1; x > (Results.GetLength(0) - 6) && x >= 0; x--)
    {
        // Gets the integer value of the time
        int Hours = (int)double.Parse(Results[x, 3]);

        // Gets the decimal value of the time
        double DecimalPart = double.Parse(Results[x, 3]) - Hours;

        // Calculate minutes from the decimal value
        int Minutes = (int)(DecimalPart * 60);

        // Formats the hours and minutes into HH:mm format
        string FormattedTime = string.Format("{0:00}:{1:00}", Hours, Minutes);

        // Adds the database fields and the formatted time into the list
        Shift_Details.Add($"{Results[x, 1]} | {Results[x, 2]} | Start Time: {FormattedTime} | {Results[x, 4]} Hours");

        // Access' the label and button in the current row i
        Label Label = (Label)Shift_View_Tbl.GetControlFromPosition(0, i);
        Button Button = (Button)Shift_View_Tbl.GetControlFromPosition(1, i);

        // Updates the label with the string from the list
        Label.Text = Shift_Details[i];

        // Set the Tag property of the button to the shift ID
        Button.Tag = Results[x, 0];

        // Update button Text and handle the click event
        Button.Text = "Accept";
        Button.Click += (sender, e) =>
        {
            // Get the shift ID from the Tag property of the button
            string shiftID = Button.Tag.ToString();
```

```
        // Selects the EmployeeID from the OvertimeShifts table Where the shiftID is equal to the one
chosen
        Query = "SELECT EmployeeID " +
                "FROM OvertimeShifts " +
                $"WHERE OvertimeShiftID = {shiftID}";
        Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");
        if (Results[0, 0] == "")
        {
            MessageBox.Show("Accepted");

            Query = "UPDATE OvertimeShifts " +
                    $"SET EmployeeID = {LoginInfo.EmployeeID} " +
                    $"WHERE OvertimeShiftID = {shiftID}";
            Database_Opperations_Class.Connect_Read_Write(Query, "write");

            Button.Text = "";
            Label.Text = "Empty Shift";
        }
        else
        {
            MessageBox.Show("This shift has already been taken.\nPlease refresh this page to see updated
view.");
        }
    };
    i++;
  }
}
```

Admin Permissions and functionality

Code Listings

Enabling managerial functions

In order to block access to managerial functions to the non manager users we need to checl the permission of the user when the main window is loaded so we can decide weather to enable the buttons that allow for the user to add remove and edit users.

```
private void Main_Window_Toolbar_Frm_Load(object sender, EventArgs e)
{
  if (LoginInfo.Permissions == "E")
  {
    Tools_Btn.Visible = false;
    Matrix_Btn.Visible = false;
  }
}
```

Adding a user to the system

Adding a user uses a lot of similar funtionality as the user editing their details on the account form, before a new user is added we need to check if all the fields are in the correct format and then assuming they are we can store all the inputed information, generate a salt and hash the passowrd then store it in the database.

Checking the inputed data

```
// Uses the data validation class to check each field is appropriate and the user can proceed to confirming their password
bool ValidName = Data_Validation_Functions_Class.IsNumberOfWords(Name_TxtBox.Text, 2);

bool ValidEmail = Data_Validation_Functions_Class.IsValidEmial(Email_TxtBox.Text);

bool ValidPhoneNumber = Data_Validation_Functions_Class.IsValidPhoneNumber(PhoneNumber_TxtBox.Text);

bool ValidUsername = Data_Validation_Functions_Class.IsBetweenLength(Username_TxtBox.Text, 5, 20);

bool ValidPassword = Data_Validation_Functions_Class.IsValidPassword(Password_TxtBox.Text, 5, 20);

bool NoneSelected = true;

foreach (RadioButton radioButton in Permission_GroupBox.Controls)
{
    if (radioButton.Checked)
    {
        NoneSelected = false;
        break;
    }
}

if (ValidName && ValidEmail && ValidPhoneNumber && ValidUsername && ValidPassword && NoneSelected != true)
{
    Confirm_Password_Pnl.Visible = true;
}
else
{
    // If any validation checks return false an error message box is displayed
    MessageBox.Show("One or more data fields is in incorrect format.\nPlease double check all fields are correct.\nPlease make sure your password contains:\nUppercase (1)\nSpecial Characters (1)\nNumbers (1)\nCharacters (5 - 20)");
}
```

Inserting the new user into the database

```
// Gets the permision selected for the new user and stores it as a string
string Permission = "";
if (Employee_Permision_RBtn.Checked == true)
{
    Permission = "E";
}
else if (Manager_Permision_RBtn.Checked == true)
{
    Permission = "M";
}

// Splits the name entered into first name and surname and stores each seperatley
string[] NameParts = Name_TxtBox.Text.Split(' ');
string FirstName = NameParts[0];
```

```
string Surname = string.Join(" ", NameParts.Skip(1));

// Generates a new random salt for the user
string Salt = Password_Hasher_Class.GenerateRandomSalt();

// Adds the new user to the database
string Query = "INSERT INTO EmployeeDetails (FirstName, Surname, Username, Password, Salt, Email,
PhoneNumber, Permission) " +
            $"VALUES ('{FirstName}', '{Surname}', '{Username_TxtBox.Text}',
'{Password_Hasher_Class.HashPassword(Password_TxtBox.Text, Salt)}', '{Salt}' , '{Email_TxtBox.Text}',
'{PhoneNumber_TxtBox.Text.Remove(0, 1)}', '{Permission}')";
Database_Opperations_Class.Connect_Read_Write(Query, "write");

MessageBox.Show($"{Name_TxtBox.Text} Succesfuly added.");

ClearAllFields();
```

Removing a user from the system

Updating the view of the database

The first thing you see when the form is loaded is a datagrid view containing all the users in the database, this view only contains the key information so the salt or password isnt shown, so we need to load the data into the field.

```
// Adds all the users to the grid view
private void Update_DataGrid_View()
{
    string Query = "SELECT EmployeeID, Firstname, Surname, Email, PhoneNumber " +
                "FROM EmployeeDetails ";
    string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");

    List<Employee_Search_Details> User_List = new List<Employee_Search_Details>();

    for (int x = 0; x < Results.GetLength(0); x++)
    {
        User_List.Add(new Employee_Search_Details(Results[x, 0], Results[x, 1], Results[x, 2], Results[x, 3],
Results[x, 4]));
        EmployeeIDs += Results[x, 0];
    }
    Database_Users_GridView.DataSource = User_List;
    EmployeeID_Selector_NumericUpDown.Maximum = int.Parse(Results[(Results.GetLength(0) - 1), 0]);
}
```

Deleting the selected user

Firstly we need to make sure that the user has selected a valid ID from the list of id's we made while loading the data into the grid view.

```
bool ValidID = EmployeeIDs.Contains(EmployeeID_Selector_NumericUpDown.Value.ToString());
```

If the ID is valid then we can go ahead and delete any trace of a user with that selected ID from all tables in the database making sure to show a confirmation message and updating the grid view.

```
//Delets all records with the selected ID
string Query = "DELETE FROM EmployeeDetails " +
            $"WHERE EmployeeID = '{EmployeeID_Selector_NumericUpDown.Value}';" +
            "DELETE FROM Messages " +
            $"WHERE EmployeeID = '{EmployeeID_Selector_NumericUpDown.Value}';" +
            "DELETE FROM ContractedShifts " +
            $"WHERE EmployeeID = '{EmployeeID_Selector_NumericUpDown.Value}';" +
            "DELETE FROM OvertimeShifts " +
            $"WHERE EmployeeID = '{EmployeeID_Selector_NumericUpDown.Value}';" +
            "DELETE FROM PaySlips " +
            $"WHERE EmployeeID = '{EmployeeID_Selector_NumericUpDown.Value}';";
Database_Opperations_Class.Connect_Read_Write(Query, "write");

MessageBox.Show($"User has been succesfuly deleted.");

Update_DataGrid_View();

// Resets the ID selector the the default
EmployeeID_Selector_NumericUpDown.Value = 1;
```

Editing a users details

Updating the grid view

This form uses a similar datagridview tool to display a list of users in the database so we are using the same update function as the removing user form for this one.

```
private void Update_Database()
{
   string Query = "SELECT EmployeeID, Firstname, Surname, Email, PhoneNumber " +
               "FROM EmployeeDetails ";
   string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");

   List<Employee_Search_Details> User_List = new List<Employee_Search_Details>();

// adds a new instance of the search details class to the list
   for (int x = 0; x < Results.GetLength(0); x++)
   {
      User_List.Add(new Employee_Search_Details(Results[x, 0], Results[x, 1], Results[x, 2], Results[x, 3],
Results[x, 4]));
      EmployeeIDs += Results[x, 0];
   }

   // Adds the compiled list to the datagrid view
   Database_Users_GridView.DataSource = User_List;

   EmployeeID_Selector_NumericUpDown.Maximum = int.Parse(Results[(Results.GetLength(0) - 1), 0]);
}
```

Selecting the user and loading their detials

Then the user will select the ID of the user they want to make edits too then press the select button, This will load all of that users details into the fields bellow allowing them to make changes.

```
bool ValidID = EmployeeIDs.Contains(EmployeeID_Selector_NumericUpDown.Value.ToString());
if (ValidID == true)
{
    Set_Fields_Enabled(true);

    SelectedID = EmployeeID_Selector_NumericUpDown.Value.ToString();

    // If the id is valid we select all the fields for that employeeID
    string Query = "SELECT FirstName, Surname, Username, Email, Password, PhoneNumber, Permission " +
                   "FROM EmployeeDetails " +
                   $"WHERE EmployeeID = '{SelectedID}'";
    string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");
    // Populates the fields with the data from the user with the selected ID
    Name_TxtBox.Text = $"{Results[0, 0]} {Results[0, 1]}";
    Email_TxtBox.Text = Results[0, 3];
    PhoneNumber_TxtBox.Text = $"+44{Results[0, 5].Substring(2)}";
    Username_TxtBox.Text = Results[0, 2];
    Password_TxtBox.Text = "Password";
    if (Results[0, 6] == "M")
    {
        Manager_Permision_RBtn.Checked = true;
    }
    else
    {
        Employee_Permision_RBtn.Checked = true;
    }
}
else
{
    MessageBox.Show("Please select a valid EmployeeID.");
}
```

Editing the password

In order to update the password the manager has to press the X button to clear the whole field before the passwordChar is turned off and they can enter a new password, this stops them from being able to see the users previous password.

```
private void New_Password_Btn_Click(object sender, EventArgs e)
{
    Password_TxtBox.Text = "";
    Password_TxtBox.ReadOnly = false;
    Password_TxtBox.UseSystemPasswordChar = false;
    Password_Changed = true;
}
```

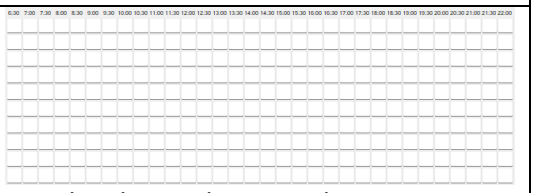
Adding shifts

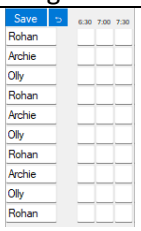
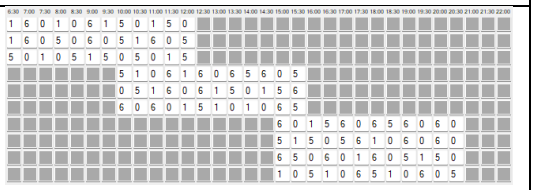
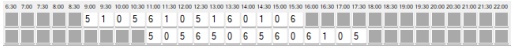
The add shift function essentially stores all the inputed data about the shift from the user, this is then formatted in the appropriate form to be uploaded to the database.

```
private void Add_Btn_Click(object sender, EventArgs e)
{
    // Gets all the data inputted and formats it
    string Selected_Day_Time = Date_Time_Picker.Value.ToString("ddd HH:mm");

    string[] Day_Time = Selected_Day_Time.Split(' ');
    string Time = Day_Time[1];

    // Splits the time into hours and minutes
    string[] Time_Parts = Time.Split(':');
    int hours = int.Parse(Time_Parts[0]);
    int minutes = int.Parse(Time_Parts[1]);

    decimal Hours = Hours_NumericUpDown.Value;

    // Converts the hours into a decimal form
    string Selected_Hours = Hours.ToString("0.0").TrimEnd('0').TrimEnd('.');
    // Converts the minutes into decimal and combines the hours and minutes
    string Selected_Time = (hours + (minutes / 60.0)).ToString();
    string Selected_Day = Day_Time[0];

    // Uploads the formatted values into the database
    string Query = "INSERT INTO OvertimeShifts (Hours, Day, StartTime, Date) " +
                   $"VALUES ('{Selected_Hours}', '{Selected_Day}', '{Selected_Time}',
'{Date_Time_Picker.Value.ToString().Substring(0, Date_Time_Picker.Value.ToString().Length - 12)}')";
    Database_Opperations_Class.Connect_Read_Write(Query, "write");

    // Uploads the view of the database the user can see
    Update_Database_View();
}
```

Testing

Algorithm to generate the daily rota

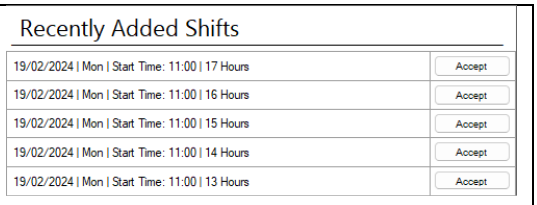
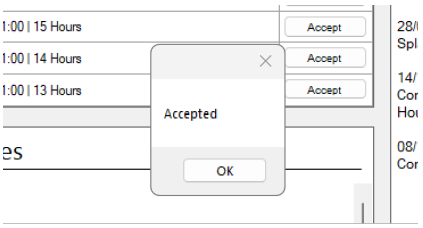
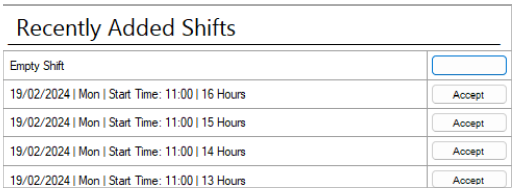
Purpose of test	Test Data and Description	Expected Outcome	Actual Outcome
Test grid generation with a given number of shifts	10 Shift records in the database EmployeeID, Hours, Day, StartTime 1, 6, Mon, 6.5 2, 6, Mon, 6.5 11, 6, Mon, 6.5 1, 6, Mon, 10 2, 6, Mon, 10 11, 6, Mon, 10 1, 6, Mon, 15 2, 6, Mon, 15 11, 6, Mon, 15 2, 6, Mon, 15	A grid with 32 columns and 10 rows	 A grid with 32 columns and 10 rows

	0 shifts on a given day	No grid to be shown	 No grid is displayed
Loading Users names into the first column	We'll alternate employeeID's, of three users, for the 10 shifts	First column should alternate names (Rohan, Archie, Olly)	 The names alternate as expected
Blocking out the text boxes that arent used to show the shift	A 10 hour shift starting at 9:30 in the database	This should block out 6 text boxes either side of the shift.	 6 text boxes are blacked out either side of the shift
	A 0 hour shift starting at any time	Every text box should be blocked out	 Every box is blocked out
	A 16.5 hour shift starting at 6:30	No text boxes should be blocked out	 No boxes are blocked out
Generating the string of random numbers	10 Shifts with varying times	Each row should be full of numbers following the rules -No more than 3 numbers in a row -No two of the same numbers consecutively	 All rows are full and follow the given rules
Saving a rota to the database	Pressing the save button and checking the database and reloading the matrix Matrix to be saved: 	Rota should be saved as a string with each row seperated with a ':' The matrix should be the same as before when reloading	String saved to database: :651051010150151650601060650606 1 :605650651060101010510616060606506 :515056056506560515016016106501050 (This string does also include the numbers that would be in the blacked out text boxes) (I have also split the string onto new lines after every colon to make it easier to read) Matrix after restarting the program and selecting the same day:  The same as the matrix from when the save button was pressed.
Re-Generating the matrix	Pressing the regenerate button and comparing the matrix before and after	The matrix after should be different to the one before	Matrix Before:  Matrix After:  The matrix before shows different numbers to the matrix after
Generating the matrix for different days	Testing if the shifts on a specific day are loaded when that day's button is pressed The shifts over Monday and	When the Thursday button is pressed there should be two shifts When the Friday. button is pressed there	Thursday Button Matrix:  Friday Button Matrix: 

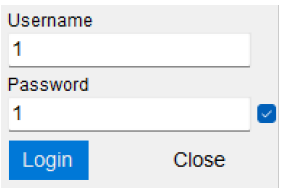
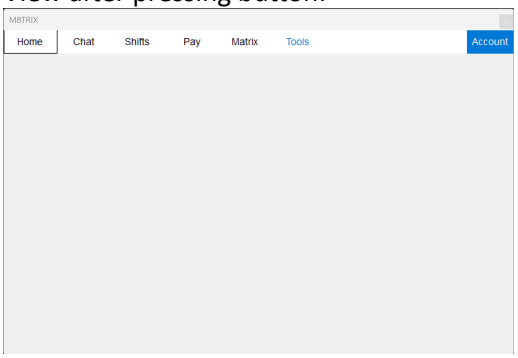
	Friday: ID Hours Day Time 31 1 7 Thu 9 32 2 7 Thu 11 33 1 7 Fri 9	should be one shift.	Loads the correct shifts when different day buttons are pressed
Loading shifts from both contracted and ovetime tables in the database	Attempt to load shifts from across both database tables Contracted Shifts: 1 7 Thu 9 2 7 Thu 11 Overtime Shifts: 16 11 5 Thu 11 19/02/2024	Both shifts from the Contracted table and overtime table should be loaded into the grid so 3 shifts should be loaded in total	Shifts Loaded:  All Shifts from both tables are loaded correctly.

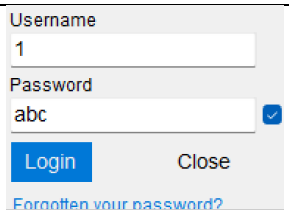
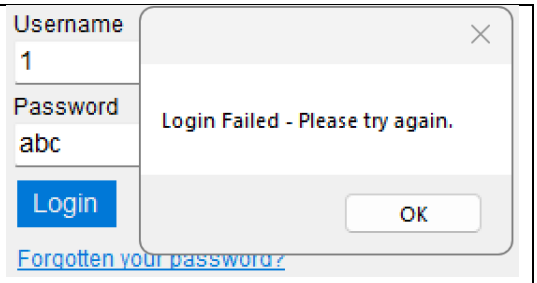
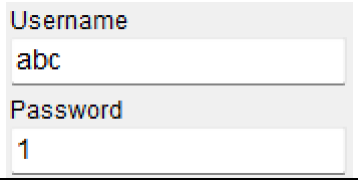
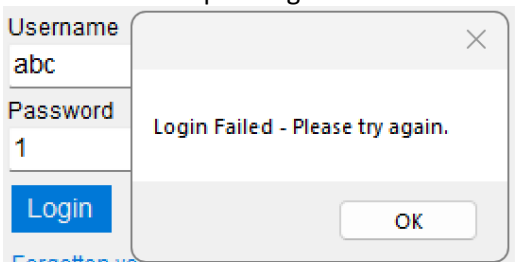
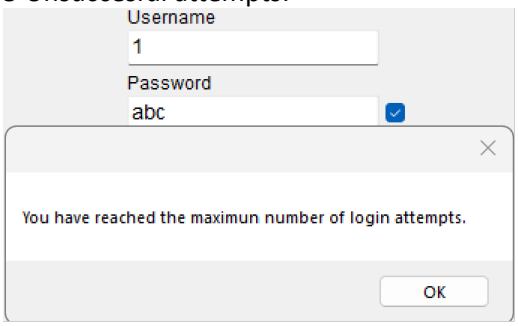
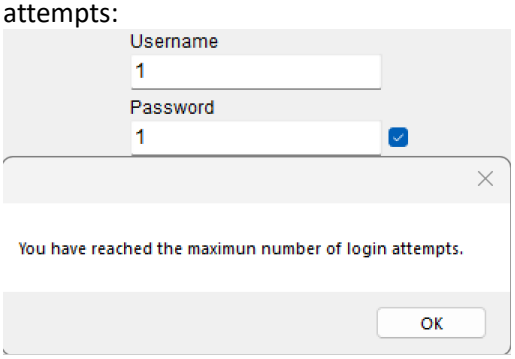
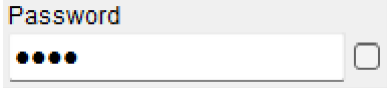
Home page with functional widgits

Purpose of test	**Test Data and Description**	**Expected Outcome**	**Actual Outcome**
Loading of the PaySlips	Payslips from database: EmployeeID Date Position Template Site HoursWorked 2 08/10/2023 Lifeguard Contracted Wadurs 7 2 14/11/2023 Lifeguard Contracted Splashpoint 6 2 28/01/2024 Lifeguard Casual Splashpoint 1 2 29/01/2024 Swim School Assistant Instructor Contracted Wadurs 2	The payslips from the database should be loaded into the text box on the home page.	Output on home page: Payslips 29/01/2024 . Swim School Assistant Instructor . Contracted . Wadurs . 2 Hours 28/01/2024 . Lifeguard . Casual . Splashpoint . 1 Hours 14/11/2023 . Lifeguard . Contracted . Splashpoint . 6 Hours 08/10/2023 . Lifeguard . Contracted . Wadurs . 7 Hours All the payslips are loaded into the text box.
Loading of the Messages	Last 10 Messages in database: Emp. ID Mes. ID Messg 43 11 72 2 44 11 73 3 45 11 74 4 46 11 75 5 47 11 76 6 48 11 77 7 49 11 78 8 50 11 79 9 51 11 80 10 52 11 81 hello	Loads the last 10 messages from the database.	Output on home page: Recent Messages Olly: 2 Olly: 3 Olly: 4 Olly: 5 Olly: 6 Olly: 7 Olly: 8 Olly: 9 Olly: 10 Olly: hello The last 10 messages were loaded into the text box on the home page.
Loading of the Shifts	The last 5 Overtime shifts: Shift ID Emp.ID Hours Day Time Date 43 43 NULL 13 Mon 11 19/02/2024 44 44 NULL 14 Mon 11 19/02/2024 45 45 NULL 15 Mon 11 19/02/2024 46 46 NULL 16 Mon 11 19/02/2024 47 47 NULL 17 Mon 11 19/02/2024	Loads the last 5 shifts into the tablelayoutpanel with accept buttons next to each shift	Output on home page: Recently Added Shifts 19/02/2024 | Mon | Start Time: 11:00 | 17 Hours Accept 19/02/2024 | Mon | Start Time: 11:00 | 16 Hours Accept 19/02/2024 | Mon | Start Time: 11:00 | 15 Hours Accept 19/02/2024 | Mon | Start Time: 11:00 | 14 Hours Accept 19/02/2024 | Mon | Start Time: 11:00 | 13 Hours Accept Loads the 5 most recent shifts each with their own accept button
Accepting a shift	Press the accept button and	EmployeeID of current	View of shifts before:

	see results Accepting shift number 47 in the database.	user (2) should be added into the employeeID column of the shift.	 View of shifts after:   Shift record in Database: 47 47 2 17 Mon 11 19/02/2024 The 47th shift has been updated with the correct employeeID and the shift has been removed from the home page.

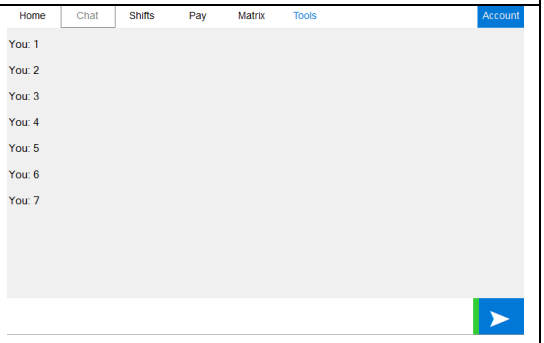
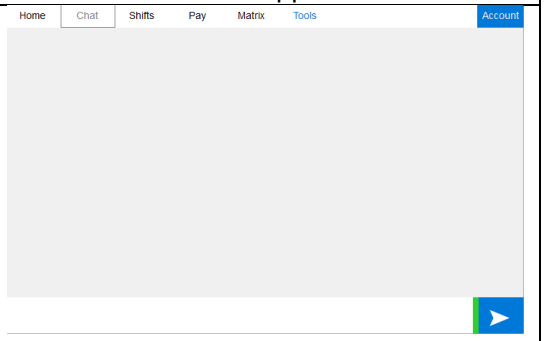
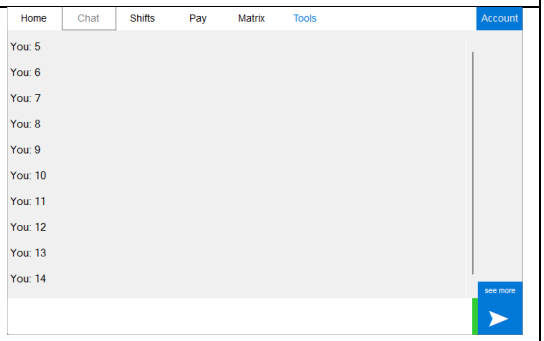
Secure login system

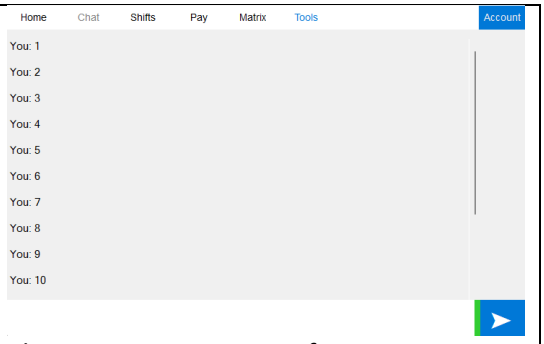
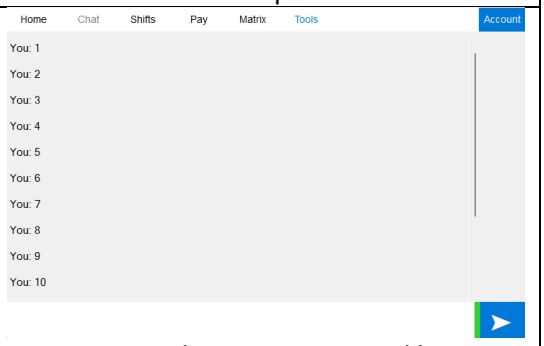
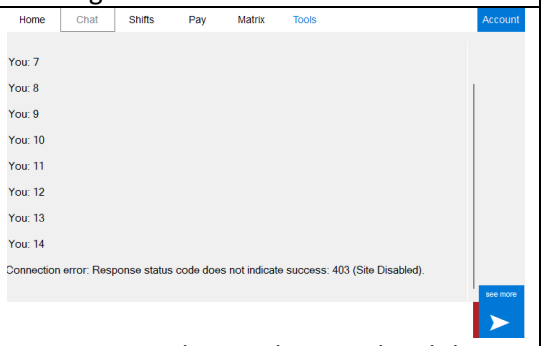
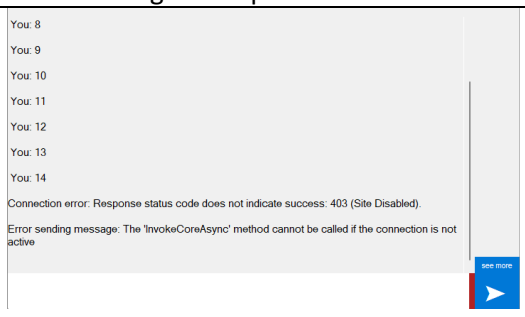
Purpose of test	**Test Data and Description**	**Expected Outcome**	**Actual Outcome**
Logging In	Using a username and password from the database Username: 1 Password: 1 	The Login page should be removed and the main page should be displayed	View after pressing button:  The login form is disposed and the main window is shown.
	Using a Username from the database but a random password Username: 1 Password: abc	An error message should be displayed.	Veiw after attemplting to login:

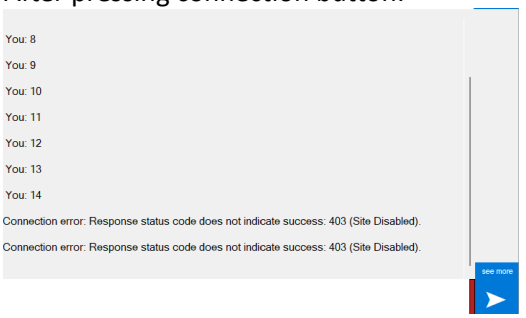
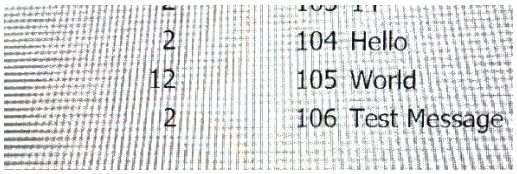
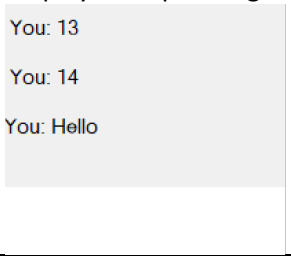
			 Login failed message displayed to user.
	Using a Password from the database but a random username Username: abc Password: 1 	An error message should be displayed.	View after attempted login:  Login failed message displayed to user.
Hitting Max LoginAttempts	Logging in with an invalid username / password 5 times.	After 5 unsuccesful logins a Max attempts message should be shown.	5 Unsuccesful attempts:  Error message is shown. User cant login
	Using a valid login after 5 unsuccessful attempts Valid Login Details Username: 1 Password: 1	The same Max attempts error message will be shown	Using a valid login after 5 unsuccessful attempts:  User still cant login after 5 unsuccessful attempts even with a valid login.
Show Password Button	Before and after pressing the show button password	Password should go from hidden to visible in the text box.	Before: 

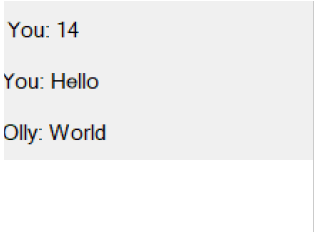
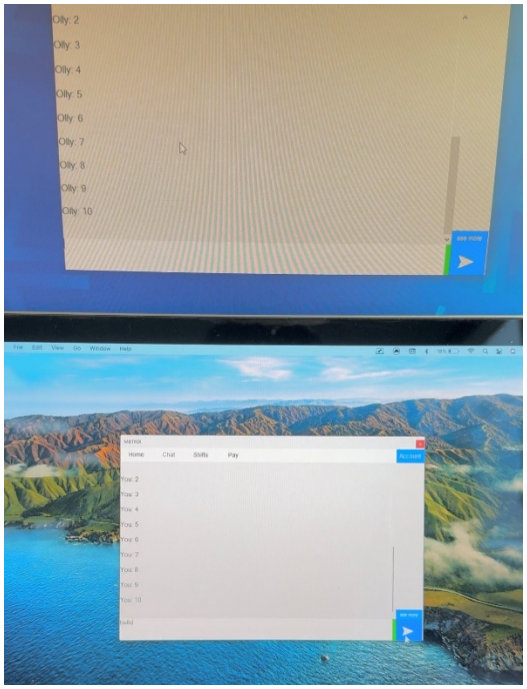
			After: 

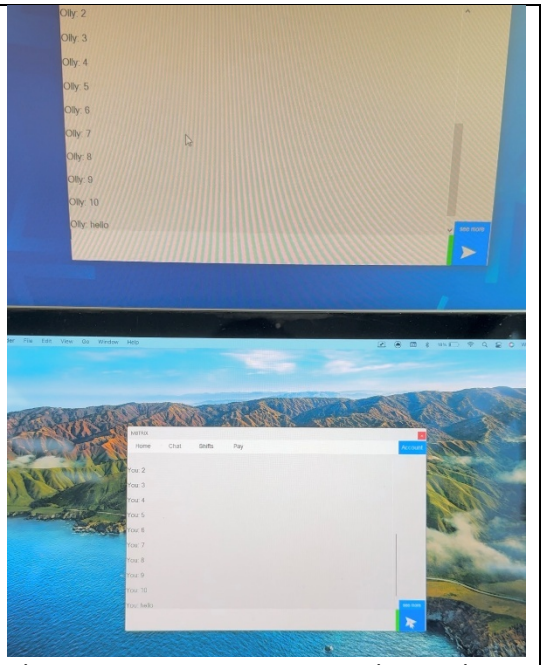
Multi-User real time messaging

Purpose of test	**Test Data and Description**	**Expected Outcome**	**Actual Outcome**
Loading of messages	10 or less messages	The messages will be loaded into the text box but the load more button wont appear.	 The messages are loaded and the load more button doesn't appear.
	0 Messages	No messages to be loaded	 No messages are loaded into the message box.
	More then 10 messages	The most recent 10 messages are loaded and the see more button is shown.	 The last 10 messages are loaded, messages 14 – 5. The see more button is shown too.

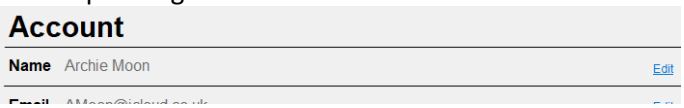
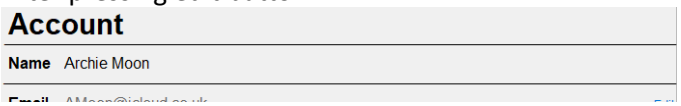
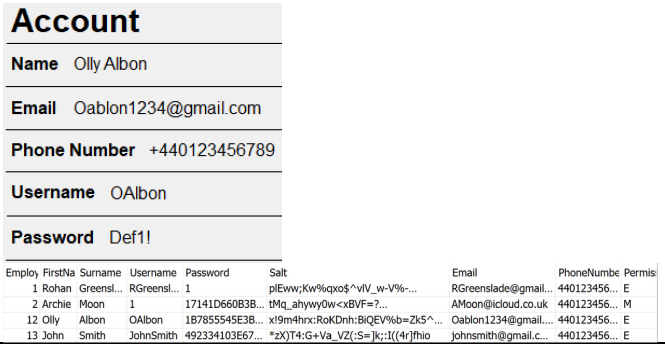
	Pressing the See More button	The previous 10 messages, or the last remaining messages, are loaded. See more button dissapears when all messages are loaded	 The remaining messages from message 4 to 1 are loaded. See more button dissapears.
Connecting to sever	Normal Connection	The connection indicator will show green when the form is loaded	 Connection indicator next to send button shows green.
	Eorror in connecting	The connection indicator will show red and an error message will be shown to the user.	 Connection indicator shows red and the error message is outputed.
	Sending a message while not conneted.	The message wont be sent and an error message will be shown.	 Error message is shown when the send message button is shown.

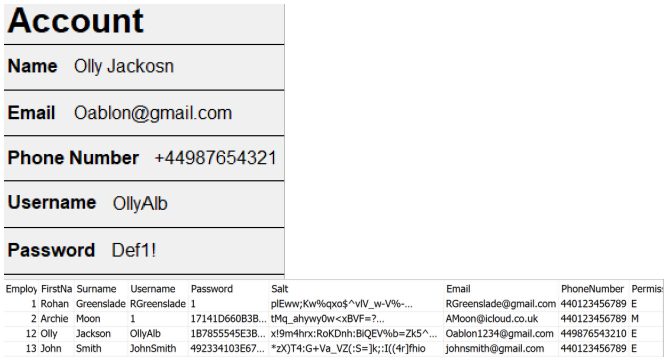
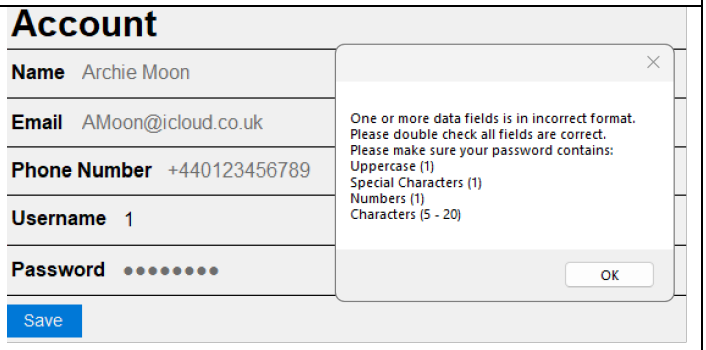
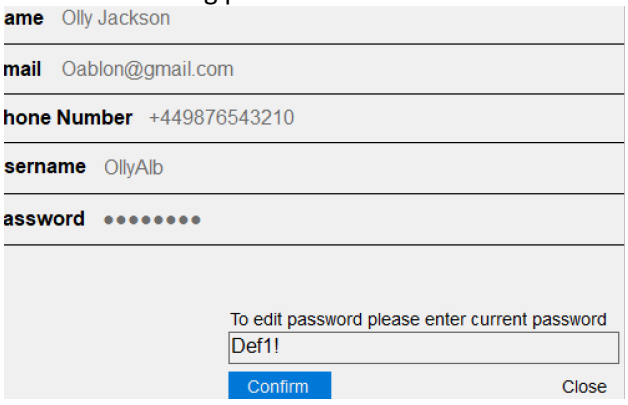
	Attempting to reconnect (Successful reconnect)	Connection indicator will turn green.	 The indicator shows green.
	Attempting to reconnect (Unsuccessful reconnect)	Connection indicator will show red and the error message will be shown.	After pressing connection button:  Indicator stays red and the appropriate error message is displayed.
Sending Messages	Storing the message locally. Message to send: ‘Hello’	When the send button Is pressed the message should be stored in the database with the ID of the sender.	Database after pressing send button: |	employeeID	Message ID	Message
13	2	102	13
14	2	103	14
15	2	104	Hello | Message is stored alongside the messageID and the employeeID
	Storing the message across devices. Message to send: ‘Test Message’	The message should be saved on both users databases.	Database of Sender: 			
---	---	---
2	104	Hello
12	105	World
2	106	Test Message	 Database of Reciever:  The message is saved into the database of both users.
	Displaying message to the sender. Message to send: ‘Hello’	When the message is sent it will be displayed to the sender as You: *message*	Display after pressing send button: 

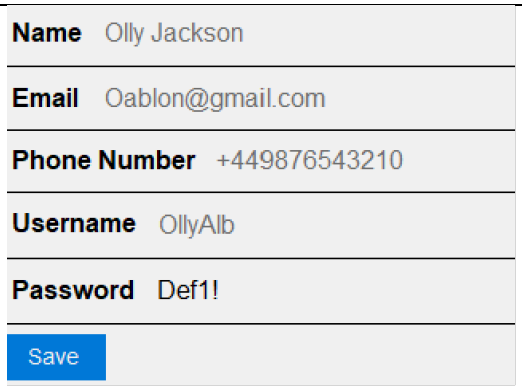
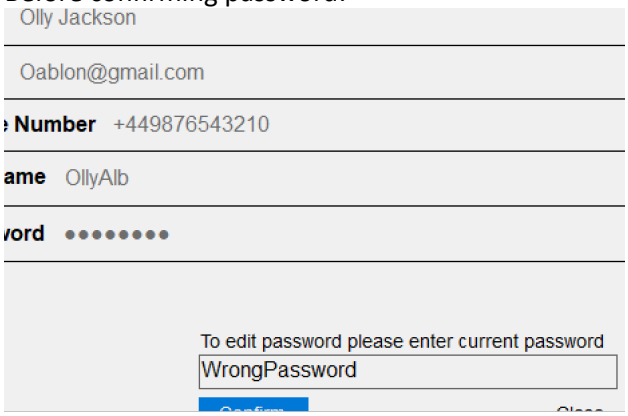
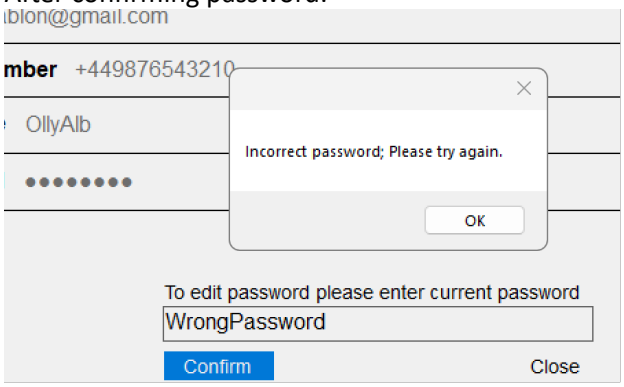
			Message displayes as expected
	Displaying message to other users. Message to send: ‘World’	When the message is received it will show as Senders Name: *message*	Users display after receiving a message:  Message is displayed as expected with the name of the sender.
	Real time sending across devices. Have the application open on two devices and record what happens when a message is sent on one.	The message should appear on the senders and the recievers device in the correct format for each user.	Before message sent:  (the message is being sent from the bottom display) After message sent:

			 The sent message appears on the senders screen and the recievers screen in the correct format and at the same time.

Account management

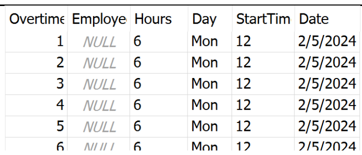
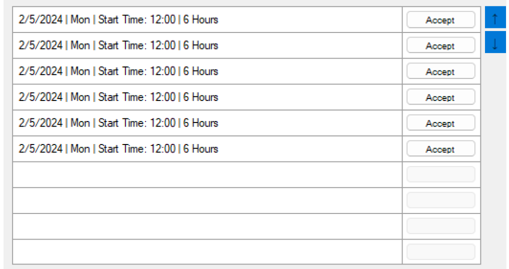
Purpose of test	**Test Data and Description**	**Expected Outcome**	**Actual Outcome**
Pressing Edit button	Pressing Name edit button	Should enable the name text box allowing the user to change the text.	Before pressing edit button:  After pressing edit button:  The text box becomes enabled and the edit button becomes invisible. This is the same for all fields.
Changing Details	All normal inputs For input field rules visit (Rules for inputs)	All changes should be updated in the database.	Details before: 

			Details after:  All the details changed are updated correctly in the database.
	Having one incorrect field Eg: incorrect password For input field rules visit (Rules for inputs)	An error message should be shown when any 1 field is wrong	 The correct error message is shown and no changes are saved For full example of each field being incorrect view (Adding a user) test results.
Enabling Password editing	To enable the password text box the user must enter their current password. This test will use the correct password.	The password text box should become enabled.	Before confirming password:  After confirming password:

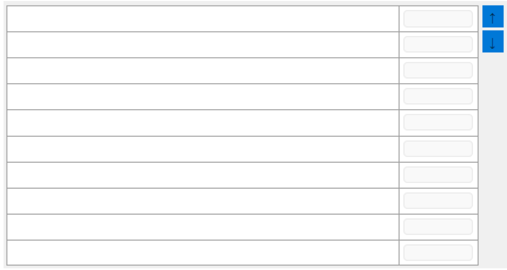
			 After pressing the button the password text box becomes enabled.
	This test will use the incorrect password.	-An error message should be displayed. -Password text box remains un-enabled	Before confirming password:  After confirming password:  After pressing the confirm button an error message is shown and the text box remains disabled.

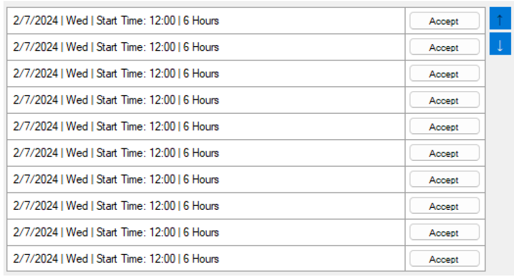
Viewing and accepting shifts

Purpose of test	Test Data and Description	Expected Outcome	Actual Outcome
Loading shifts from database	10 or less shifts Number of shifts: 6 Day: Monday	Should all load into the table for the correct day selected	Monday button press:

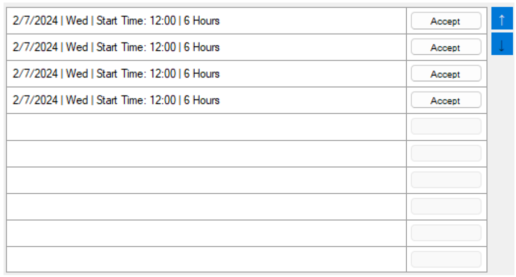
<table>
<tr>
<td></td>
<td> 
<table>
<tr><th>Overtime</th><th>Employe</th><th>Hours</th><th>Day</th><th>StartTim</th><th>Date</th></tr>
<tr><td>1</td><td>NULL</td><td>6</td><td>Mon</td><td>12</td><td>2/5/2024</td></tr>
<tr><td>2</td><td>NULL</td><td>6</td><td>Mon</td><td>12</td><td>2/5/2024</td></tr>
<tr><td>3</td><td>NULL</td><td>6</td><td>Mon</td><td>12</td><td>2/5/2024</td></tr>
<tr><td>4</td><td>NULL</td><td>6</td><td>Mon</td><td>12</td><td>2/5/2024</td></tr>
<tr><td>5</td><td>NULL</td><td>6</td><td>Mon</td><td>12</td><td>2/5/2024</td></tr>
<tr><td>6</td><td>NULL</td><td>6</td><td>Mon</td><td>12</td><td>2/5/2024</td></tr>
</table>
</td>
<td></td>
<td> 

All 6 shifts are loaded correctly and on the correct day.</td>
</tr>
<tr>
<td></td>
<td>0 shifts

Day: Tuesday</td>
<td>Nothing should be loaded into the table.</td>
<td>Tuesday button press:
 
No shifts are loaded into the table as expected.</td>
</tr>
<tr>
<td></td>
<td>More than 10 shifts

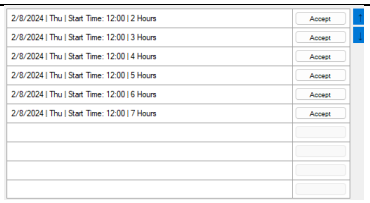
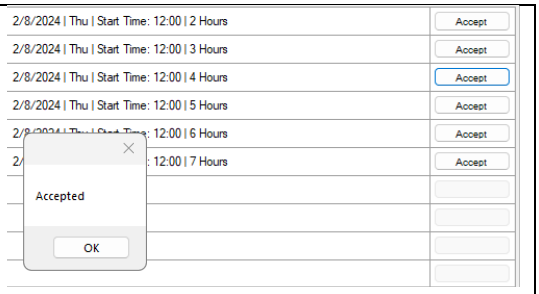
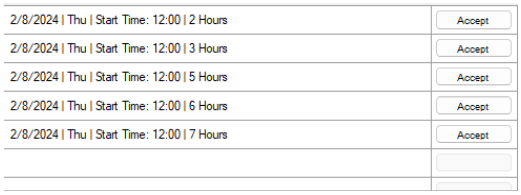
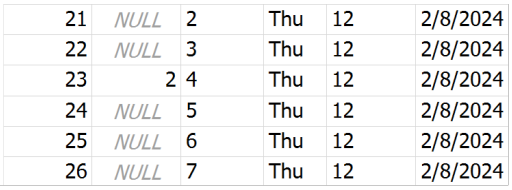
Number of Shifts: 14
Day Wednesday</td>
<td>10 shifts should be loaded into the table but the up down arrows should be enabled to allow the user to scroll to the second page.</td>
<td>Wednesday button press:
 

First 10 shifts are loaded, down button is enabled promting the user to scroll.

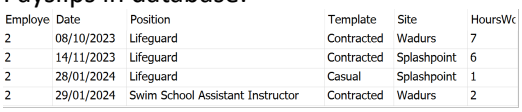
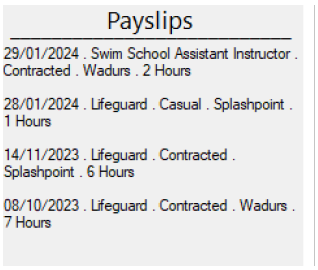
Down button pressed:
 

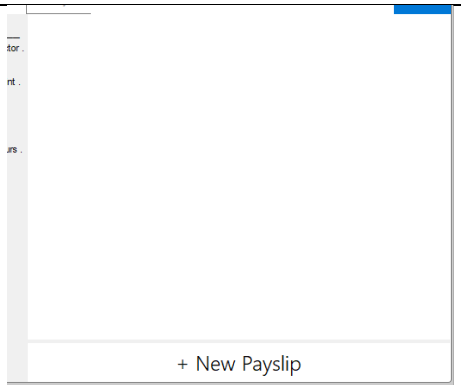
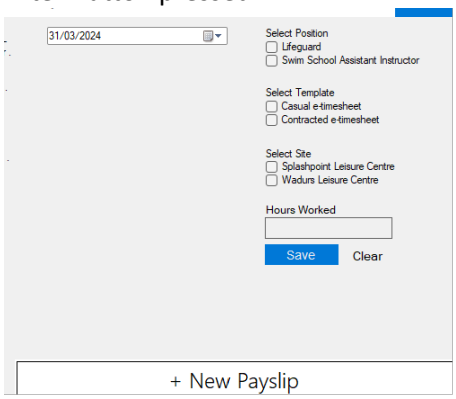
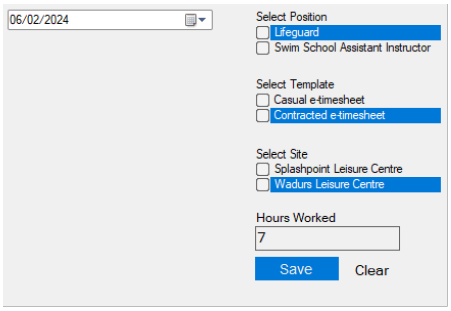
Loads the remaning shifts and enables the up arrow which will return the user to the screen shown above.</td>
</tr>
<tr>
<td>Accepting Shifts</td>
<td>Pressing accept button

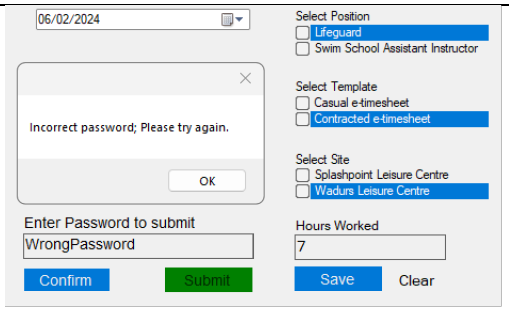
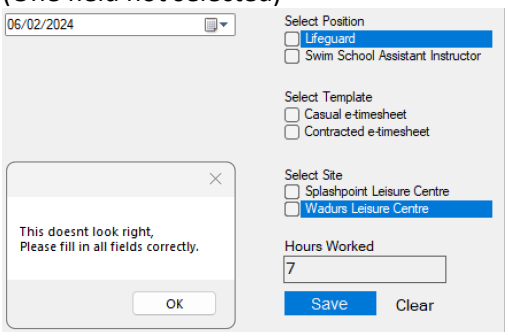
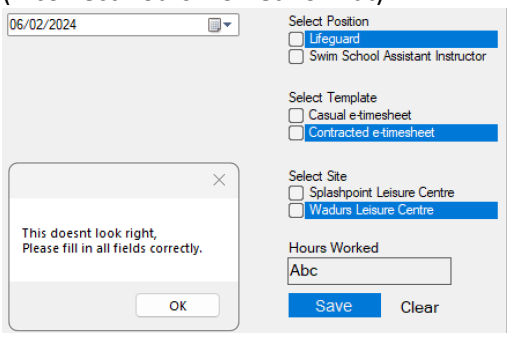
EmployeeID: 2
Shift selected: 4 hours shift on Thursday</td>
<td>Should show a shift accepted message and add the employeeID of the user to the record of the selected shift.</td>
<td>Pressing Accept button:</td>
</tr>
</table>

			 Accept message shown  Shift is removed from the list  EmployeeID 2 is added to the 4 hour shift record in the database

Payslip creation and storing

Purpose of test	**Test Data and Description**	**Expected Outcome**	**Actual Outcome**
Loading of the paylsips	Payslips in database: 	All payslips from the database should be formatted and loaded into the text box.	Form loaded:  All the payslips are loaded in the correct format.
New payslip button		Should bring up the payslip sub form within the window.	Before Button pressed:

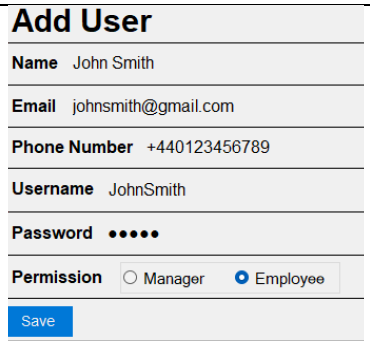
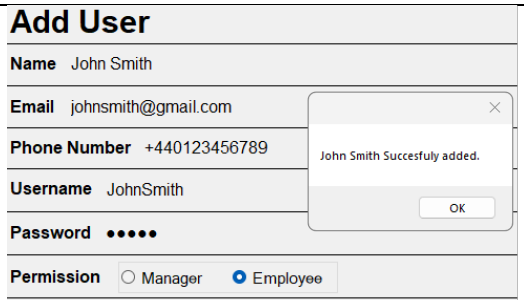
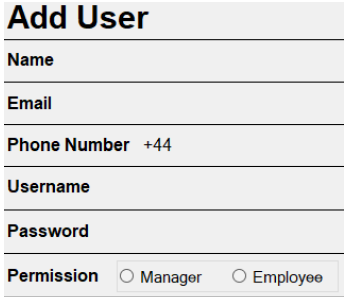
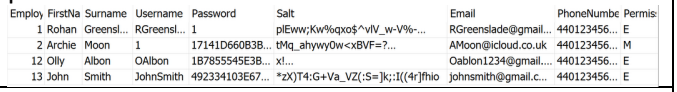
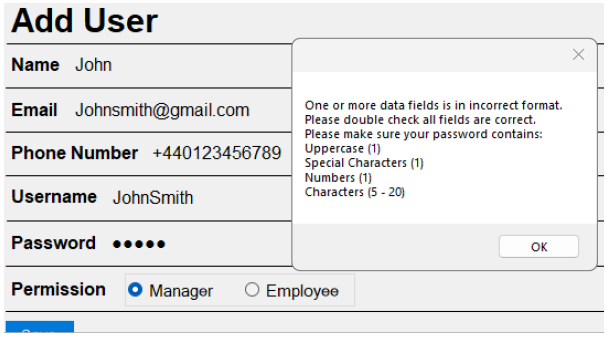
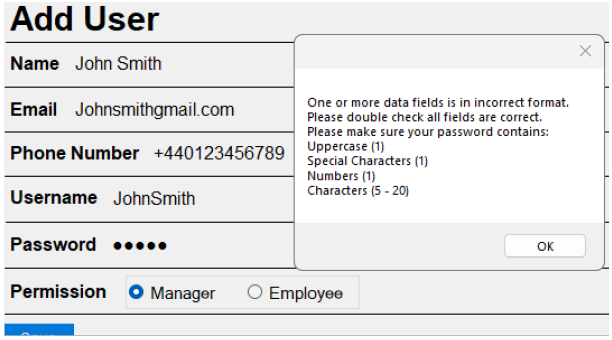
			 After Button pressed:  The input form for to create a payslip is shown in the window.
Creating a new payslip	Normal inputs with correct password confirmation. 	-A new payslip with all the inputed data should be added to the database -The new payslip will be added to the text box of loaded paylsips	Save Button pressed: Employe Date Position Template Site HoursWc 2 08/10/2023 Lifeguard Contracted Wadurs 7 2 14/11/2023 Lifeguard Contracted Splashpoint 6 2 28/01/2024 Lifeguard Casual Splashpoint 1 2 29/01/2024 Swim School Assistant Instructor Contracted Wadurs 2 2 06/02/2024 Lifeguard Contracted Wadurs 7 The payslips details are added to the database  The loaded payslips are updated to add the new one.
	Normal inputs with Incorrect Password confirmation.	An error message will be displayed and the user will have to re enter their password to save the payslip.	Confirm password button press:

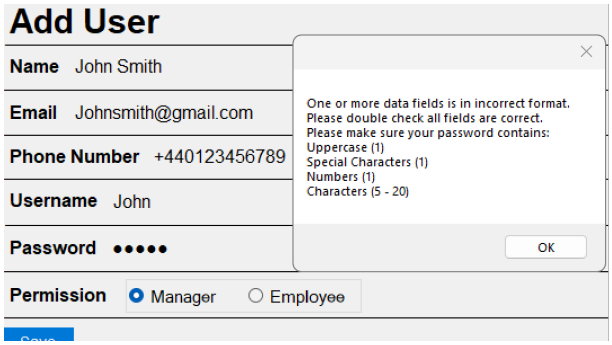
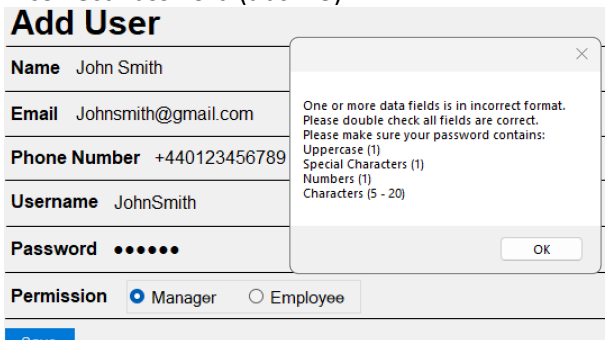
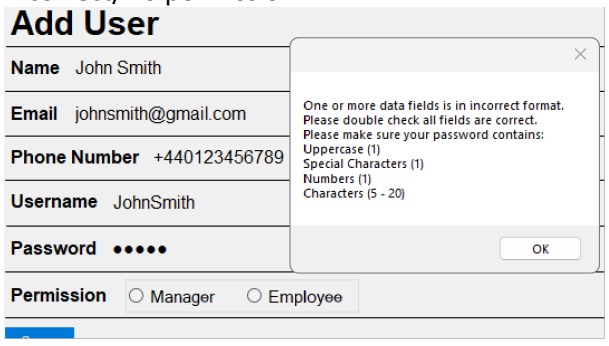
			 The correct error message is shown and the payslip isnt saved.
	Erronious inputs with correct password confirmation. (One field not selected or non decimal number in the hours worked field)	An error message should be shown and the payslip wont be saved.	Save Button pressed: (One field not selected)  Error message is shown stopping payslip from being saved. (Incorrect hours worked format)  Error message is shown stopping payslip from being saved. For full example of each field being incorrect view (Adding a user) test results.

Admin permission functions (adding, removing, editing users and adding shifts)

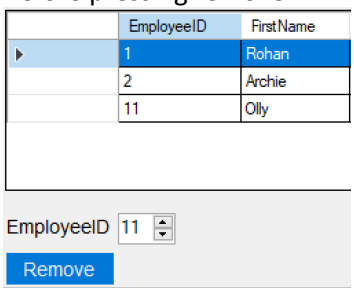
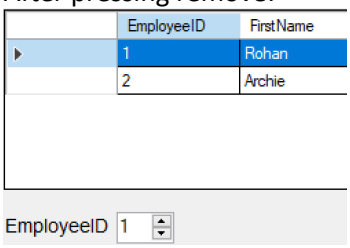
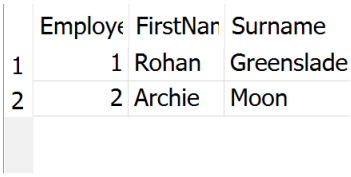
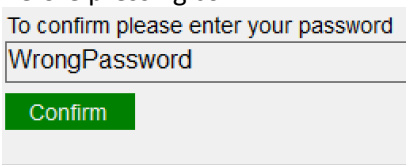
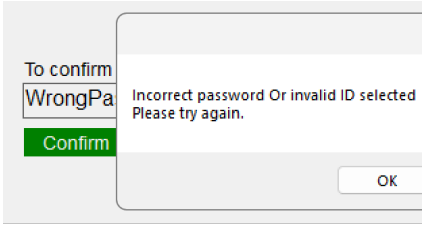
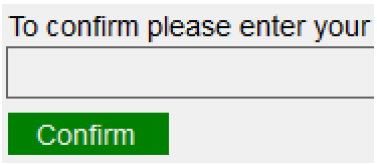
Adding Users

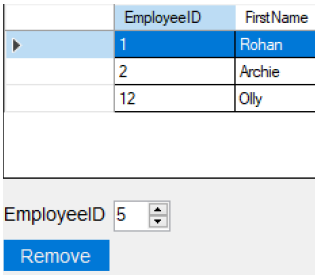
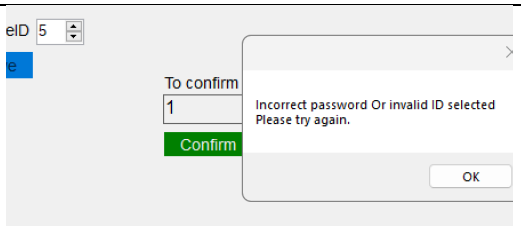
Purpose of test	**Test Data and Description**	**Expected Outcome**	**Actual Outcome**
Adding a user	All normal inputs Password: Def1!	-A user added message should be displayed	User added message is displayed:

	 For input field rules visit (Rules for inputs)	-The text fields should be cleared -The new user should be added to the employeeDetails table -A new salt should be generated and the Password in database should be hashed	 Input fields are cleared:  User is added to the database with a new salt and the password Is hashed: 
	Having one incorrect field For input field rules visit (Rules for inputs)	An error message should be shown when any 1 field is wrong	Incorrect Name:  Incorrect Email:  Incorrect PhoneNumber:

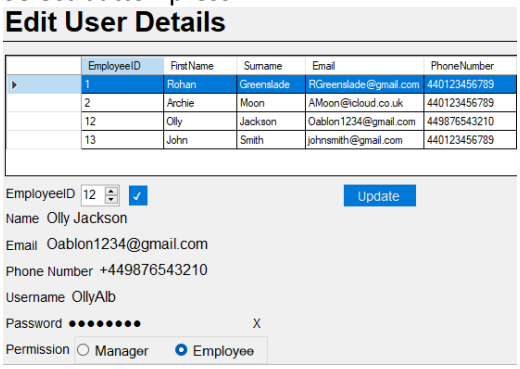
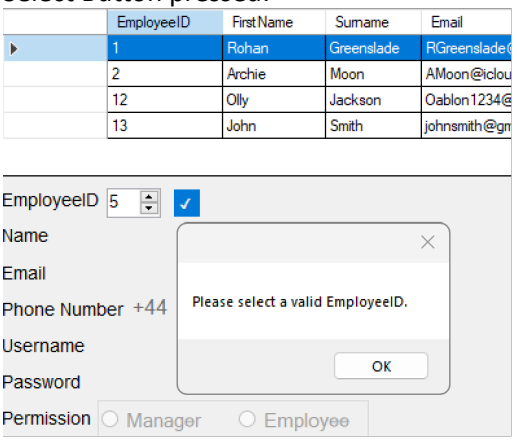
			 Incorrect Username:  Incorrect Password (abc123):  Incorrect/No permission:  In every instance on any one field being in the wrong format an error message is shown.

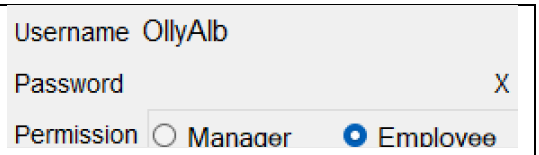
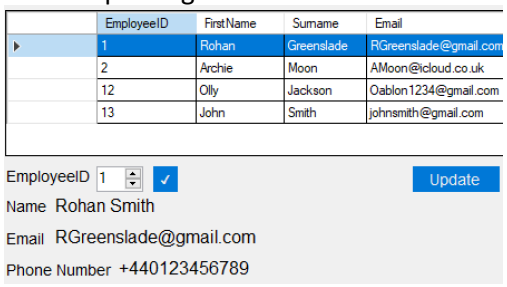
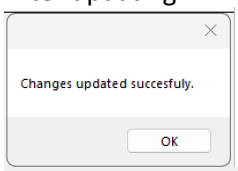
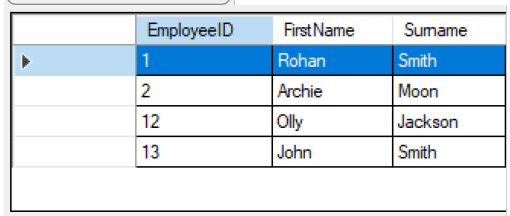
Removing Users

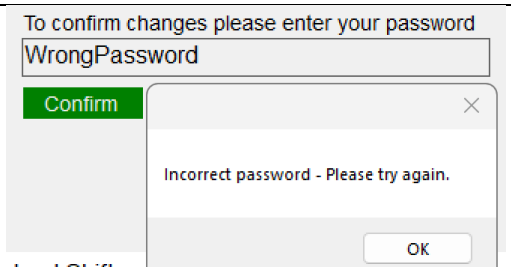
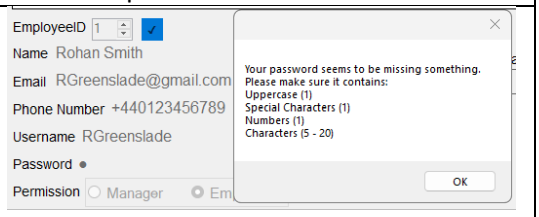
Purpose of test	Test Data and Description	Expected Outcome	Actual Outcome
Selecting and removing a Valid User	Selected User ID: 11 Password confirmation is correct for this test.(Normal)	-User will be promped to enter their password -User with ID 11 should be removed from the database -The gird view will be updated	Before pressing remove:  After pressing remove:  The grid view is updated with the selected user removed Database after removal:  The selected user is removed from the database.
	Selected User ID: 1 Passowrd confirmation is incorrect for this test (Erronious)	-An error message will be shown -The user will be prompted to confirm their passsword again	Before pressing confirm:  After pressing confirm:  Error message is dislayed.  User can re enter their password to

			confirm.
Selecting an Invalid ID	Selected ID: 5  *Password confirmation is valid	An error message saying the selected ID is invalid.	 Error message is shown when the invalid ID is selected and the remove button is pressed.

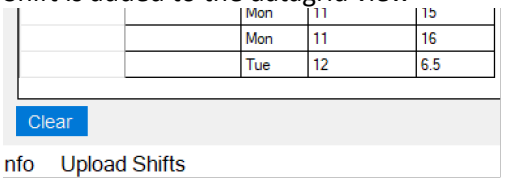
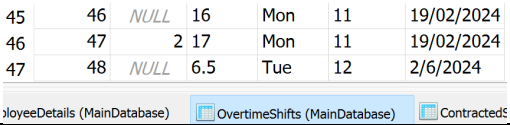
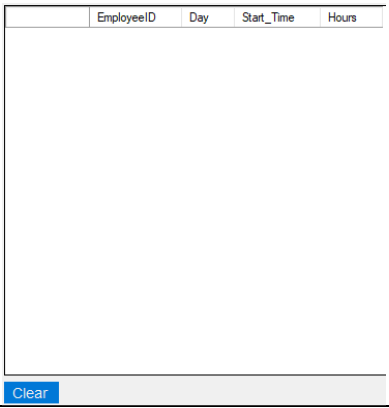
Editing Users

Purpose of test	**Test Data and Description**	**Expected Outcome**	**Actual Outcome**
Selecting an ID	Selecting a valid ID (ID: 12)	Should load the details, from the employee with the selected ID, into the text box fields	Select button press:  The details from employee with the ID 12 are loaded onto the correct fields.
	Selecting an invalid ID (ID: 5)	Should give us an invalid ID selected error message.	Select Button pressed:  Error message is shown indicating an invalid ID has been selected.
Editing the password	Pressing the new password (X button)	Will clear the entire passowrd field and enable it for editing.	New password button press:

			 Clears the previous password and allows a new password to be entered by the manager.
Updating Changes	Changing name from: Rohan Greenslade To Rohan Smtih Correct password confirmation	Should update the name in the database, and update the datagrid view with the new name	Before updating:  After updating:    The message box is displayed and the changes are updated in the data grid view and also in the database.
	Changing name from: Rohan Greenslade To Rohan Smtih Incorrect password confirmation	Should show an error message saying wrong password and the changes wont be saved.	

			 An error message is shown and the user will have to enter the correct passowrd in order to update the details
Updating changes with invalid inputs	Having any one field in incorrect format. Eg: incorrect password For input field rules visit (Rules for inputs)	Should show error message	 Shows the correct error message. This is the same for any incorrect field. For full example of each field being incorrect view (Adding a user) test results.

Adding Shifts

Purpose of test	Test Data and Description	Expected Outcome	Actual Outcome
Adding a shift	All normal inputs Date: 06/02/24 (Tuesday) Time: 12:00 Hours: 6.5	New record in database with the entered details. The new shift is added to the grid view	Shift is added to the datagrid view  The shift is added to the OvertimeShifts table: 
Clearing the shifts	Press the clear button and record results	-All shifts in the grid view are deleted -All shifts in the overtimeShifts Table are also deleted	All shifts in the dataGridView are deleted: 

			All shifts in the overtimeshifts table are deleted: OvertimeShiftID EmployeeID Hours Day StartTime Date

Evaluation

Objectives

Algorithm to generate the daily rota

Results

Partially Completed

Managers can save and load the daily rota, which is dynamically generated and includes both contracted and overtime shifts, for each day of the week. However the algorithm hasn't been entirely successful in making the entire matrix and therefore a compromise was made and instead it now just generates a sort of baseline that will need to be tweaked, thus still partially achieveing the goal of dramatically reducing the time taken and human errors in making the daily rotas.

Potential for future development

The one main goal for future development would be to fully automate this process and while that was unsuccsesful in this case, In part due to time constraints and overall machine power from using a virtual windows machine on mac, it is entirely possible with more time and effort being put into it so even after this project is completed I will still try to develop a fully working rota algorithm.

Client Response

Me: Although I wasn't able to fully automatically generate the rota, what are your thoughts on the partial generation and how it improve the rota making process?

Manager: Yeah of course the fully automated version would have been nicer but this is still a huge improvement on the current system and is quite impressive how it accounts for shifts that have been picked up too.

Me: would you push for further development of this feature in hopes to fully automate it in the future?

Manager: Yes one hundred percent having it fully automatic would be hughly beneficial so yeah I would definitely push for you to add that if it's possible.

Me: Would you recommend any other improvements?

Manager: On the whole not really a few more niche features might be nice such as being able to change the colour of a box and stuff like that.

Home page with functional widgits

Results

Fully Completed

The home page allows the users to have access to all the important information they might need such as seeing recent messages and shifts as well as their past payslips this is in my opinion a really useful inclusion as it combines some of the key aspects of this project into one form in a seamless way that fully works making it one of the most important objectives of the project.

Potential for future development

While there is some potential for future development such as maybe allowing the user to send messages directly from the home screen or when a shift is picked up loading in the next most recent one instead of just leaving it empty, I really don't think much more else needs to be done to this page without it getting too complex and too busy.

Client Response

Me: Does this achieve what you were expecting from a 'home page'?

Manager: Yes above and beyond that even I was expecting to be able to see messages and maybe like news updates but the fact that you can pick up shifts is really impressive.

Me: How do you feel about being able to see all this information on one page?

Lifeguard: Its actually really useful its annoying having to switch between apps so often to check shifts or payslips so having that all in one place is really nice.

Me: As someone who is going to make good use out of functionality like this do you have any comments on how it could be improved?

Lifeguard: I think its pretty good how it is to be honest it might be nice to see a few more shifts at once or maybe even select what day using like a filter thing or something but other than that no.

Secure login system

Results

Fully Completed

The login system created for this project allows for fully secure login and password storage using password hashing with a salt to store the users passwords this protects against database breaches as their password isnt stored as plain text. It follows a really simple desgin too with a username and password field followed by a login and a close button this makes it as simple as possible to access with the program.

Potential for future development

As a feature, with limited room for development beyond the basic login page, I feel there really cant be much more I can do with this the only thing would be to make the password storage even more

secure by not storing it at all by using serices such as microsofts login system to store and manage the passwords security. I could also maybe explore options for the user to reset their password themselves.

Client Response

Me: Does my login system achieve what you'd expect from a login system?

Manager: Yeah the login system is really good I like the added layers of security you have added too they are nice although an even more secure method would always be appriciated with such sensitive information accessible if there is a breach but on the whole its really good.

Multi-User real time messaging

Results

Fully Completed

My program allows users to send and receive messages across a signalR server hosted microsoft azure which allows users to connect to the server from any device across different networks. Users can also load previous messages at the click of the button allowing them to scroll back through the chat. However one big flaw was noticed, when sending messages between different devices the message sent does appear on both users screens (the sender and any recievers) however it isnt saved to the database of any logged ou users, this is because the database file is stored locally on the users computer and therefore the database of the sender will be updated with the new message so it can be loaded in the future but this wont happen for receiving users that arent logged in when the message was sent.

Potential for future development

The issue of savin messages to the database of non logged in users is a clear aim for future development but this can be relatively easily rectified by moving the database either to some sort of shared file system, using a firestore/firebase solution, or even to the azure system to update it using the server. This would have been possible having spotted this issue earlier in development however at the time it was spotted it was too late to impliment this change sufficiently as the majority of forms need to access a database.

Client Response

Me: What are you opinions on my messaging system?

Manager: overall its really fluid I mean its simple and easy to use however of course as you were explaining to me earlier the issue with the database stuff Is obviously something that we would like to see fixed in the long term but I mean you can still send and receive messages sent in real time and load any previous messages but yeah once its fully working it will be amazing.

Account management

Results

Fully Completed

The account management page allows users to edit every individual detail of their acccount (excluding their salt and their permission) all they have to do is press the edit button next to whatever field they want to change and it allows them to edit it. All inputs are checked to see if they

are in the correct format before they are saved and the user must enter their password in order to be able to edit their password for added security.

Potential for future development

In terms of potential development I don't see much in the way of features I can add the general idea of this form is reasonably simple just allow a user to edit their account details why implimenting some security features. It is however possible to have stricter security measures and tighter requirements for the password and username.

Client Response

Me: Are you happy with the account management functionality?

Manager: Yeah I mean it does all that it can, cant ask for anything more.

Me: Is this account page intuitive to you? Would you be able to use this without any guidance?

Lifeguard: Yeah it couldn't be simpler I mean you just press edit and edit what you want. Having to enter your password to change it is a nice added feature it just adds that extra layer of security.

Viewing and accepting shifts

Results

Fully Completed

This functionality allows for the users to select a day and then all the shifts for that day that have been uploaded, they can then chose to accept any shift they choose and this will be updated in the database. And that shift will be removed from being able to be selected

Potential for future development

It might also be nice for users to be able to select multiple shifts at the same time as well as this it would also be nice to ensure that they cant select two shifts on the same day that overlap to avoid the issues this would cause.

Client Response

Me: How does this method of picking up shifts compare to the current method in place.

Manager: Its honestly so much better once that issue is fixed this would be an amazing way of picking up shifts, it takes the need for us to manually add users to the matrix out of the equation which will save a hell of a lot of time and reduce the amount of errors so having this is a great edition.

Me: How do you feel about this new way of picking up shifts?

Lifeguard: This looks way easier than having to see what shifts are available then having to scroll through the messages and see if its already been taken then ask to be put on that shift and hope that whoever has made the rota sees it.

3NF SQLite database

Results

Fully Completed

My database which has been normalized to the third normal form allows for most of the functionality of my project with it being used in almost every form, having this in the third normal form is really useful as it helps imrpove the efficiency of SQL querys as well as readability and maintainablility of my database which allows it to easily be scaled up.

Potential for future development

While this objective has been fully completed to the goals set out there is still need for improvement while the database itself meets the objectives it still has the flaw of not being able to updated for any non current users, for this to be solved I will have to host the database in a way that allows users to access it from multiple devices I can do this through using my existing azure server to host a MySQL database and allow users to read and write to it through that.

Client Response

Me: How is the experience of the database backed features?

Manager: Yeah it seems quite good actually I mean you can tell that all the changes I make are updated and stored to be used in the program in the future which allows for a wide range of functionality that is key to the application.

Payslip creation and storing

Results

Fully Completed

The payslip tab allows users to see their previous payslips as well as create new ones, they have to enter a date, select a position a location and a template before entering the number of hours they worked. Then they can press save, confirming their password before the payslip is saved to the database.

Potential for future development

In terms of future development the obvious step would be to integrate this with some sort of payroll one way we could do this is to use an api that can send an email to a specific adress, twilio has an api that can do this, this can be used to forward any payslips to the finance team who can take it from there. In terms of development in the way of features it might be nice to be able to search for a payslip using filter such as the date or the location.

Client Response

Me: How do you feel about this method of creating a payslip?

Manager: Its quite nice actualy, its pretty similar to what we use so that'll be nice for integration and yeah it could probably pretty easily be integrated with hr and payroll.

Me: Would there need to be anything else in order for this to be done?

Manager: Not too much really they would have to be able to receive and view their final payslips with all of the details and stuff but I don't really see the need for anything more than that.

Admin permission functions (adding, removing, editing users and adding shifts)

Results

Fully Completed

This is actually a collection of forms that allow for the manager to perform the basic needs of the system.

Using these forms they can add a new user by entering all the details of the user they want to add then pressing add and confirming their password.

They can edit any existing users details to do this they select the ID of the user which loads their details onto the screen for the manager to then change and save the changes they make.

They can also delete any user in a similar way by selecting the ID of the user they want to delete and then confiming their passowrd and removing the user.

This objective also covers adding shifts, this works similarly to adding a user they enter the details of the shift and simply press save adding it to thw database.

I put all of these under the sam objective as they all fall under the category of database manipulation and all posses the same basic functionality of updating / adding to a table in the database.

Potential for future development

To develop this further would involve just adding more funtionality to this group of forms such as adding and removing contracted shifts as well as being able to edit or remove specific shifts doing this would give managers more control over the database without actually having to interact with it directly which is always good. It would also be nice to include some statistic funtionality such as being able to view specific users shift patterns or how much overtime the pick up all of this could be done.

Client Response

Me: What are your thought on these functions that allow you to maintain your users and shifts?

Manager: These all look really good its nice to have these available in the project and not have to go through the hastle of working the database or anything like that.

Me: What do you think about the aspect of future development I have mentioned?

Manager: yeah those are really important I think, How it is not is good but having that extra sort of ability to change the specifics would be really good and the idea of being able to see like stats and figures is nice too as we currently have nothing like that.

Independant evaluation

Conclusion

Overall im really happy with how this project has turned out and I belive that it has really helped my develop practical skills as well as my understanding of the development process of applications like this.

One way I would take this project further would be to possible migrate the entire thing to web forms or some sort of web based application to allow users to access it from a wider range of devices

instead of restricting them to a laptop but winforms at the time was more familiar to me and allowed me to focus more on feature implimentation instead of having to learn a whole new template.

Althought there was a few unfortunate compromises with the rota generation and the unfortunate issue with the databases, both of which can and will be fixed in future, im still really pleased with how the whole program works and belive that it has achieved the goal of creating a centralized single application and at the same time saving both time and money solving a real life problem in my place of work and with future development I would definitley look to trying to get them to actually take my application on board and consider using it.

Appendix

Code Listings

Non objective forms (Toolbars)

Main Window Toolbar

```
using System;
using System.Drawing;
using System.Windows.Forms;
using Winforms;

namespace Moon_Archie_Winforms_NEA
{
    public partial class Main_WindowToolbar : Form
    {
        private bool Form_Loading = false;
        public Main_WindowToolbar()
        {
            InitializeComponent();
        }
        private void Main_Window_Toolbar_Frm_Load(object sender, EventArgs e)
        {
            if (LoginInfo.Permissions == "E")
            {
                Tools_Btn.Visible = false;
                Matrix_Btn.Visible = false;
            }
        }
        private void Account_Btn_Click(object sender, EventArgs e)
        {
            Button_Colour_Reset();
            LoadForm(new Account_Frm());
        }
        private void Chat_Btn_Click(object sender, EventArgs e)
        {
            Button_Colour_Reset();
            LoadForm(new Chat_Frm());
            Chat_Btn.ForeColor = Color.Gray;
        }
        private void Home_Btn_Click_1(object sender, EventArgs e)
        {
```

```
            Button_Colour_Reset();
            LoadForm(new Home_Frm());
            Home_Btn.ForeColor = Color.Gray;
        }
        private void Pay_Btn_Click(object sender, EventArgs e)
        {
            Button_Colour_Reset();
            LoadForm(new Pay_Calculator_Frm());
            Pay_Btn.ForeColor = Color.Gray;
        }
        private void Tools_Btn_Click(object sender, EventArgs e)
        {
            Button_Colour_Reset();
            LoadForm(new Tools_Toolbar_Frm());
            Tools_Btn.ForeColor = Color.Gray;
        }
        private void Matrix_Btn_Click(object sender, EventArgs e)
        {
            Button_Colour_Reset();
            LoadForm(new Matrix_Frm());
            Matrix_Btn.ForeColor = Color.Gray;
        }
        private void Shifts_Btn_Click(object sender, EventArgs e)
        {
            Button_Colour_Reset();
            LoadForm(new Shifts_Frm());
            Shifts_Btn.ForeColor = Color.Gray;
        }
        private void Button_Colour_Reset()
        {
            Home_Btn.ForeColor = SystemColors.ControlText;
            Chat_Btn.ForeColor = SystemColors.ControlText;
            Shifts_Btn.ForeColor = SystemColors.ControlText;
            Pay_Btn.ForeColor = SystemColors.ControlText;
            Tools_Btn.ForeColor = SystemColors.Highlight;
            Matrix_Btn.ForeColor = SystemColors.ControlText;
        }
        private void LoadForm(Form formToLoad)
        {
            // uses a form loading flag to stop another form from being loaded while there is one already
loading
            Form_Loading = true;

            // clears any loaded forms
            Clear_Panel();

            formToLoad.TopLevel = false;
            Main_Window_View_Pnl.Controls.Add(formToLoad);
            formToLoad.BringToFront();
            formToLoad.Show();

            Form_Loading = false;
        }
        private void Clear_Panel()
        {
```

```
            // stops the form from being disposed if there is a form in the middle of loading
            if (Form_Loading)
            {
                return;
            }
            foreach (Control control in Main_Window_View_Pnl.Controls)
            {
                if (control is Form form)
                {
                    form.Dispose();
                }
            }
        }
    }
}
```

Manager tools toolbar

```
using System;
using System.Windows.Forms;

namespace Moon_Archie_Winforms_NEA
{
    public partial class Tools_Toolbar_Frm : Form
    {
        private bool Form_Loading = false;
        public Tools_Toolbar_Frm()
        {
            InitializeComponent();
        }
        private void Tools_Toolbar_Frm_Load(object sender, EventArgs e)
        {

        }
        private void Add_User_Btn_Click(object sender, EventArgs e)
        {
            LoadForm(new Add_User_Frm());
        }
        private void Remove_User_Btn_Click(object sender, EventArgs e)
        {
            LoadForm(new Remove_User_Frm());
        }

        private void Edit_User_Info_Btn_Click(object sender, EventArgs e)
        {
            LoadForm(new Edit_User_Info_Frm());
        }

        private void Upload_Shifts_Btn_Click(object sender, EventArgs e)
        {
            LoadForm(new Upload_Shifts_Frm());
        }
        private void LoadForm(Form formToLoad)
```

```
        {
            // uses a form loading flag to stop another form from being loaded while there is one already
loading
            Form_Loading = true;

            // clears any loaded forms
            Clear_Panel();

            formToLoad.TopLevel = false;
            Tool_Pnl.Controls.Add(formToLoad);
            formToLoad.BringToFront();
            formToLoad.Show();

            Form_Loading = false;
        }
        private void Clear_Panel()
        {
            // stops the form from being disposed if there is a form in the middle of loading
            if (Form_Loading)
            {
                return;
            }
            foreach (Control control in Tool_Pnl.Controls)
            {
                if (control is Form form)
                {
                    form.Dispose();
                }
            }
        }
    }
}
```

Algorithm to generate the daily rota

```
using System;

using System.Collections.Generic;

using System.Drawing;

using System.Windows.Forms;


namespace Moon_Archie_Winforms_NEA

{

    public partial class Matrix_Frm : Form

    {

        // Constants defininf grid dimensions and global string
```

```
private const int NumColumns = 33;
private const int ColumnWidth = 20;
private const int FirstColumnWidth = 55;
private int NumRows = 0;

private string SelectedDay = "";
private string StringToSave = "";

public Matrix_Frm()
{
   InitializeComponent();
}

private TextBox[,] TextBoxes;

private void Matrix_Frm_Load(object sender, EventArgs e)
{
}

private void CreateDynamicGrid(int rows)
{
   Random random = new Random();

   NumRows = rows;

   // The 2D array of TextBoxes that holds the grid
   TextBoxes = new TextBox[NumColumns, NumRows];

   // Loops through each row
   for (int row = 0; row < NumRows; row++)
   {
```

```
            string generatedString = "";
            if (CheckForSavedRota(SelectedDay) == true)
            {
                // if no rota is saved in the databse we generate a new string for each row
                generatedString = GenerateString(random);
                StringToSave = StringToSave + ":" + generatedString;
            }
            else
            {
                // if there is a saved rota we get each rows string from the database
                string Query = "SELECT Rota " +
                               "FROM DailyRotas " +
                               $"WHERE Day = '{SelectedDay}'";
                string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");
                string Rota = Results[0, 0];

                string[] substrings = Rota.Split(new char[] { ':' }, StringSplitOptions.RemoveEmptyEntries);

                generatedString = substrings[row];
            }
            // Loops through each column
            for (int col = 0; col < NumColumns; col++)
            {
                // Calculates the x-coordinate based on the index of the column
                // This is as the first column will have larger text boxes meaning the second column will
need to be adjusted acordingly
                int xCoordinate = col == 0 ? col * FirstColumnWidth : FirstColumnWidth + col *
ColumnWidth;

                // Creates a new text box
                TextBox textBox = new TextBox
                {
```

```
            // Sets the size based on weather its the first column or not
            Size = new Size(col == 0 ? FirstColumnWidth : ColumnWidth, 0),

            // Sets the location of each text box
            Location = new Point(xCoordinate, (row * 21) + 20),

            // Sets the name for each text box with the x and y coordinates
            Name = $"TextBox_{col}_{row}",
        };
        Controls.Add(textBox);

        // Stores it in the 2D array so it can be manipulated if needed
        TextBoxes[col, row] = textBox;
        ChangeTextOfTextBox(col, row, generatedString[col].ToString());
      }
    }
}
private void ChangeTextOfTextBox(int col, int row, string newText)
{
    // Makes sure the column and row entered wont throw an index error
    if (col >= 0 && col < NumColumns && row >= 0 && row < NumRows)
    {
      // Sets the text of the specified text box in the 2D array to the given text
      TextBoxes[col, row].Text = newText;
    }
}
private void ChangeBackColorOfTextBox(int col, int row, Color newColor)
{
    if (col >= 0 && col < NumColumns && row >= 0 && row < NumRows)
    {
      TextBoxes[col, row].BackColor = newColor;
```

```
        }
    }
    private void Load_Employee_Names_and_Generate(string Day)
    {
        // Selects the firstname from the given employeeID that is ascociated with all shifts, overtime and contracted, on a given day
        string Query = "SELECT EmployeeDetails.FirstName " +
                    "FROM EmployeeDetails " +
                    "JOIN ContractedShifts ON EmployeeDetails.EmployeeID = ContractedShifts.EmployeeID " +
                    $"WHERE ContractedShifts.Day = '{Day}' " +
                    "UNION ALL " +
                    "SELECT EmployeeDetails.FirstName " +
                    "FROM EmployeeDetails " +
                    "JOIN OvertimeShifts ON EmployeeDetails.EmployeeID = OvertimeShifts.EmployeeID " +
                    $"WHERE OvertimeShifts.Day = '{Day}'";
        string[,] results = Database_Opperations_Class.Connect_Read_Write(Query, "Read");

        CreateDynamicGrid(results.Length);

        for (int i = 0; i < results.Length; i++)
        {
            // Calls the change text function for the text boxes in the first column and sets them to the name of the employee
            ChangeTextOfTextBox(0, i, results[i, 0]);
        }
    }
    private void BlockOut_TextBoxes(string Day)
    {
        // The query that selects the start time and ammount of hours for every shift on a given day
        string Query = "SELECT StartTime, Hours " +
```

```
                "FROM ContractedShifts " +
                $"WHERE Day = '{Day}' " +
                "UNION ALL " +
                "SELECT StartTime, Hours " +
                "FROM OvertimeShifts " +
                $"WHERE Day = '{Day}'";
        string[,] results = Database_Opperations_Class.Connect_Read_Write(Query, "Read");

        // Loops through each row in the results array
        for (int i = 0; i < results.GetLength(0); i++)
        {
            // Calculates the value for the starting index for each shift based off the start time
            // Performs: ( Start time - Earliest start time ) * 2 to get the apropriate index
            double calculatedValue = (double.Parse(results[i, 0]) - 6.5) * 2;

            results[i, 0] = calculatedValue.ToString();

            // Calculates the end index for the shift
            // Each index (box) is 30 mins worth so the end is start index (calculated above) + (number of hours * 2)
            calculatedValue = (double.Parse(results[i, 0]) + (double.Parse(results[i, 1])) * 2) + 1;

            results[i, 1] = calculatedValue.ToString();

        }
        // Loops through each row in the results array
        for (int i = 0; i < results.GetLength(0); i++)
        {
            // Sets the boxes from index 0 to the start index to DarkGrey
            for (int j = 1; j <= int.Parse(results[i, 0]); j++)
            {
```

```
            ChangeBackColorOfTextBox(j, i, Color.DarkGray);
            ChangeTextOfTextBox(j, i, "");
        }
        // Sets the boxes from the end index to the final index to DarkGrey
        for (int k = int.Parse(results[i, 1]); k <= 32; k++)
        {
            ChangeBackColorOfTextBox(k, i, Color.DarkGray);
            ChangeTextOfTextBox(k, i, "");
        }
    }
}
private void ClearGrid()
{
    List<TextBox> textBoxList = new List<TextBox>();

    foreach (Control control in Controls)
    {
        // Check if the control is a text box
        TextBox textBox = control as TextBox;
        if (textBox != null)
        {
            // If it is it adds it to our list
            textBoxList.Add(textBox);
        }
    }
    // Loops through each textbox in the list
    foreach (var textBox in textBoxList)
    {
        // Removes the text box from the controls and UI
        Controls.Remove(textBox);
```

```
        // Disposes of any additional resources used by the text box
        textBox.Dispose();
    }
}
public string GenerateString(Random random)
{
    char[] characters = { '0', '1', '5', '6' };
    string result = "";

    for (int i = 0; i < 33; i++)
    {
        char nextChar;

        // If the previous two characters were non-zero numbers, the next character must be '0'
        if (i >= 3 && result[i - 1] != '0' && result[i - 2] != '0' && result[i - 2] != '0')
        {
            nextChar = '0';
        }
        else
        {
            nextChar = characters[random.Next(characters.Length)];
        }

        // Ensure there are no consecutive identical numbers
        if (i > 0 && result[i - 1] == nextChar)
        {
            // Generate a new character until it's different from the previous one
            while (result[i - 1] == nextChar)
            {
                nextChar = characters[random.Next(characters.Length)];
            }
```

```
        }

        // Append the character to the result
        result += nextChar;
    }

    return result;
}
public bool CheckForSavedRota(string Day)
{
    // Selects the count from the database entered when the function is called
    string Query = "SELECT Rota " +
                   "FROM DailyRotas " +
                   $"WHERE Day = '{Day}'";
    string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");

    // Checks if the count returned is 0
    if (string.IsNullOrEmpty(Results[0, 0]))
    {
        // Returns true if count is 0
        return true;
    }
    // Returns true if count is more than 0
    else { return false; }
}
private void MakeButtonsVisible()
{
    Save_Btn.Visible = true;
    ReGenerate_Btn.Visible = true;
    Time_Title_Pnl.Visible = true;
}
```

```
    private void Save_Btn_Click(object sender, EventArgs e)
    {
        if (StringToSave != "")
        {
            // if there is a rota to save we save it to the database and then reset it so we know there is a
saved one
            string Query = "UPDATE DailyRotas " +
                        $"SET Rota = '{StringToSave}' " +
                        $"WHERE Day ='{SelectedDay}'";
            Database_Opperations_Class.Connect_Read_Write(Query, "write");
            StringToSave = "";
        }
        else
        {
            MessageBox.Show("The current rota is already saved.");
        }
    }

    private void ReGenerate_Btn_Click(object sender, EventArgs e)
    {
        // deletes any saved rotas on the selected day
        string Query = "UPDATE DailyRotas " +
                    "SET Rota = NULL " +
                    $"WHERE day = '{SelectedDay}'";
        Database_Opperations_Class.Connect_Read_Write(Query, "write");

        // generates a new grid or random numbers
        ClearGrid();
        Load_Employee_Names_and_Generate(SelectedDay);
        BlockOut_TextBoxes(SelectedDay);
    }
```

```
private void Monday_Matrix_Btn_Click(object sender, EventArgs e)
{
   SelectedDay = "Mon";
   StringToSave = "";
   MakeButtonsVisible();
   ClearGrid();
   Load_Employee_Names_and_Generate(SelectedDay);
   BlockOut_TextBoxes(SelectedDay);
}
private void Tuesday_Matrix_Btn_Click(object sender, EventArgs e)
{
   SelectedDay = "Tue";
   StringToSave = "";
   MakeButtonsVisible();
   ClearGrid();
   Load_Employee_Names_and_Generate(SelectedDay);
   BlockOut_TextBoxes(SelectedDay);
}
private void Wednesday_Matrix_Btn_Click(object sender, EventArgs e)
{
   SelectedDay = "Wed";
   StringToSave = "";
   MakeButtonsVisible();
   ClearGrid();
   Load_Employee_Names_and_Generate(SelectedDay);
   BlockOut_TextBoxes(SelectedDay);
}
private void Thursday_Matrix_Btn_Click(object sender, EventArgs e)
{
   SelectedDay = "Thu";
```

```
    StringToSave = "";
    MakeButtonsVisible();
    ClearGrid();
    Load_Employee_Names_and_Generate(SelectedDay);
    BlockOut_TextBoxes(SelectedDay);
}
private void Friday_Matrix_Btn_Click(object sender, EventArgs e)
{
    SelectedDay = "Fri";
    StringToSave = "";
    MakeButtonsVisible();
    ClearGrid();
    Load_Employee_Names_and_Generate(SelectedDay);
    BlockOut_TextBoxes(SelectedDay);
}
private void Saturday_Matrix_Btn_Click(object sender, EventArgs e)
{
    SelectedDay = "Sat";
    StringToSave = "";
    MakeButtonsVisible();
    ClearGrid();
    Load_Employee_Names_and_Generate(SelectedDay);
    BlockOut_TextBoxes(SelectedDay);
}
private void Sunday_Matrix_Btn_Click(object sender, EventArgs e)
{
    SelectedDay = "Sun";
    StringToSave = "";
    MakeButtonsVisible();
    ClearGrid();
    Load_Employee_Names_and_Generate(SelectedDay);
```

```
            BlockOut_TextBoxes(SelectedDay);
        }
    }
}
```

Home page with functional widgits

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using Winforms;

namespace Moon_Archie_Winforms_NEA
{
    public partial class Home_Frm : Form
    {
        public Home_Frm()
        {
            InitializeComponent();
        }

        // Summary: Loads data from the appropriate databases into the widget panels
        private void Home_Frm_Load(object sender, EventArgs e)
        {
            // Checks if any of the databases are empty
            // If they arent empty then we can call the appropriate loader
            if (DatabaseIsEmpty("PaySlips")) {}
            else { PaySlip_Window_Loader(); }

            if (DatabaseIsEmpty("Messages")) {}
            else { Message_Loader(); }
```

```
    if (DatabaseIsEmpty("OvertimeShifts")) {}
    else { Shift_Loader(); }
}

// Summary: Function that checks if a database is empty
// Parameters: string "DatabaseName"
// Returns: Bool (true/false)
private bool DatabaseIsEmpty(string DatabaseName)
{
    // Selects the count from the database entered when the function is called
    string Query = "SELECT COUNT(*) " +
                   $"FROM {DatabaseName}";
    // Calls the Database_Opperations_Class to execute the query
    string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "Read");

    // Checks if the count returned is 0
    if (int.Parse(Results[0, 0]) == 0)
    {
        // Returns true if count is 0
        return true;
    }
    // Returns true if count is more than 0
    else {  return false; }
}

// Summary: Loads the data from the payslips table in the database
private void PaySlip_Window_Loader()
{
    // Emptys any text from the text box
    Payslip_TxtBox.Text = "";
```

```
        // Selects the count from the payslips table where the employeeID is the same as the logged
in user
        string query = "SELECT Count(*) " +
                    "FROM PaySlips " +
                    $"WHERE EmployeeID = '{LoginInfo.EmployeeID}'";
        string[,] Results = Database_Opperations_Class.Connect_Read_Write(query, "Read");
        int Count = int.Parse(Results[0, 0]);

        // Selects every field from the payslips table where the employeeID is the same as the logged
in user
        query = "SELECT * " +
            "FROM PaySlips " +
            $"WHERE EmployeeID = '{ LoginInfo.EmployeeID}'";
        Results = Database_Opperations_Class.Connect_Read_Write(query, "Read");

        // Loops for the count of the number of rows in the table
        for (int x = 0; x < Count; x++)
        {
           // Formats all the data and adds it to the text box
           Payslip_TxtBox.Text = $"{Results[x, 1]} . {Results[x, 2]} . {Results[x, 3]} . {Results[x, 4]} .
{Results[x, 5]} Hours \n\n{Payslip_TxtBox.Text}";
        }
    }

    // Summary: Loads the data from the messages table in the database
    private void Message_Loader()
    {
        // Selects the number of messages in the database
        string Query = "SELECT Count(*) " +
                    "FROM Messages";
        string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");
```

```
        // Creates an int to store how many times the loader will need to be looped and the index to
start at
        int Loop_Count = 0;
        int Start_Index = 0;

        // Creates a string with the messages to add from the database
        string Messages_To_add = "";

        // If the count minus 9 >= 0, this means that there are more than 10 messages to load
        if (int.Parse(Results[0, 0]) - 9 >= 0)
        {
            // The loop count is then set to 10 as we load 10 messages at a time
            Loop_Count = 10;

            // The start index is set to the count - 10 (as the index value is always 1 less as it starts from
0)
            Start_Index = int.Parse(Results[0, 0]) - 10;
        }
        // If there are less than 10 messages
        // We need to work out how many times to loop to avoid an index error
        else
        {
            // Loop count is set to the number of rows in the database
            Loop_Count = int.Parse(Results[0, 0]);

            // The start index is set to 0
            Start_Index = 0;
        }

        // Selects the message and employee name
        // From the employeeID of that message and stores them in a 2D array
        Query = "SELECT EmployeeDetails.FirstName, Messages.Message " +
```

```
                "FROM Messages " +
                "JOIN EmployeeDetails ON Messages.EmployeeId = EmployeeDetails.EmployeeId;";
        Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");

        // Loops from x = 0 to x = Loop_Count
        for (int x = 0; x < Loop_Count; x++)
        {
            // Checks if the Name is the same as the name of the user
            if (Results[Start_Index, 0] == LoginInfo.First_Name)
            {
                // The message is added to string of all messages to add (You: 'Message')
                Messages_To_add = $"{Messages_To_add} You: {Results[Start_Index, 1]} \n\n";
            }
            else
            {
                // The message is added to string of all messages to add (Employee Name: 'Message')
                Messages_To_add = $"{Messages_To_add} {Results[Start_Index, 0]}: {Results[Start_Index,
1]}\n\n";
            }
            // The index count is incrimented
            Start_Index++;
        }
        // The messages to add are added at the top of the text box
        Messages_RichTxtBox.Text = Messages_To_add + Messages_RichTxtBox.Text;
    }

    //Summary: Loads the data from the OvertimeShifts table in the database
    private void Shift_Loader()
    {
        // Initializes a new list to hold the details of a shift
        List<string> Shift_Details = new List<string>();
```

```
        // Selects the Day Date start time and hours from the overtimeshifts table where no
employeeID has been specified
        string Query = "SELECT OvertimeShiftID, Date, Day, StartTime, Hours " +
                    "FROM OvertimeShifts " +
                    "WHERE EmployeeID IS NULL";
        string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");

        // An integer i is set to 0 so we can incriment the indexes
        int i = 0;

        // Loops for x = (Number of rows in database - 1) to x = (length - 6), so it loops 5 times.
        for (int x = Results.GetLength(0) - 1; x > (Results.GetLength(0) - 6) && x >= 0; x--)
        {
            // Gets the integer value of the time
            int Hours = (int)double.Parse(Results[x, 3]);

            // Gets the decimal value of the time
            double DecimalPart = double.Parse(Results[x, 3]) - Hours;

            // Calculate minutes from the decimal value
            int Minutes = (int)(DecimalPart * 60);

            // Formats the hours and minutes into HH:mm format
            string FormattedTime = string.Format("{0:00}:{1:00}", Hours, Minutes);

            // Adds the database fields and the formatted time into the list
            Shift_Details.Add($"{Results[x, 1]} | {Results[x, 2]} | Start Time: {FormattedTime} |
{Results[x, 4]} Hours");

            // Access' the label and button in the current row i
            Label Label = (Label)Shift_View_Tbl.GetControlFromPosition(0, i);
```

```
            Button Button = (Button)Shift_View_Tbl.GetControlFromPosition(1, i);

            // Updates the label with the string from the list
            Label.Text = Shift_Details[i];

            // Set the Tag property of the button to the shift ID
            Button.Tag = Results[x, 0];

            // Update button Text and handle the click event
            Button.Text = "Accept";
            Button.Click += (sender, e) =>
            {
                // Get the shift ID from the Tag property of the button
                string shiftID = Button.Tag.ToString();

                // Selects the EmployeeID from the OvertimeShifts table Where the shiftID is equal to the one chosen
                Query = "SELECT EmployeeID " +
                        "FROM OvertimeShifts " +
                        $"WHERE OvertimeShiftID = {shiftID}";
                // Execute the query to update the shift
                Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");
                if (Results[0, 0] == "")
                {
                    MessageBox.Show("Accepted");

                    // Update the shift in the database using the shiftID
                    Query = "UPDATE OvertimeShifts " +
                            $"SET EmployeeID = {LoginInfo.EmployeeID} " +
                            $"WHERE OvertimeShiftID = {shiftID}";
```

```
                // Execute the query to update the shift
                Database_Opperations_Class.Connect_Read_Write(Query, "write");

                Button.Text = "";
                Label.Text = "Empty Shift";
            }
            else
            {
                MessageBox.Show("This shift has already been taken.\nPlease refresh this page to see updated view.");
            }
        };
        // i is incrimented
        i++;
    }
  }
 }
}
```

Secure login system

```
using System;
using System.Windows.Forms;
using Winforms;

namespace Moon_Archie_Winforms_NEA
{
   public partial class Login_Frm : Form
   {
      public int attempts = 0;
      public Login_Frm()
      {
         InitializeComponent();
      }
      private void Login_Frm_Load(object sender, EventArgs e)
      {
      }
      private void Password_Reset_PopUp_Lbl_Click(object sender, EventArgs e)
      {
         // shows the password reset prompt label
         Password_Reset_Info_Lbl.Visible = true;
```

```
    }
    private void Login_Btn_Click(object sender, EventArgs e)
    {
        if (attempts == 5)
        {
            MessageBox.Show("You have reached the maximun number of login attempts.");
            Username_TxtBox.Text = "";
            Password_TxtBox.Text = "";
        }
        else
        {
            // stores the entered details in variables
            string username = Username_TxtBox.Text;
            string password = Password_TxtBox.Text;

            // selects the count of how many entries in the database match the username and password
entered
            DBSQLite_Class.DBSQLite_Class db = new DBSQLite_Class.DBSQLite_Class(@"data source =
MainDatabase.db");
            string query = "SELECT Count(*) FROM EmployeeDetails " +
                    "WHERE Username = '" + username + "'";
            string[,] results = db.GetRecords(query);

            query = "SELECT EmployeeID, FirstName, Surname, Username, Password, Salt, Email,
PhoneNumber, Permission " +
                "FROM EmployeeDetails " +
                "WHERE Username = '" + username + "'";
            string[,] UserInfo = db.GetRecords(query);

            if (int.Parse(results[0, 0]) > 0 && password != "" && username != "")
            {
                if (Password_Hasher_Class.VerifyPassword(password, UserInfo[0, 5], UserInfo[0, 4]) == true)
                {
                    LoginInfo.EmployeeID = UserInfo[0, 0];
                    LoginInfo.First_Name = UserInfo[0, 1];
                    LoginInfo.Surname = UserInfo[0, 2];
                    LoginInfo.Username = UserInfo[0, 3];
                    LoginInfo.Password = UserInfo[0, 4];
                    LoginInfo.Salt = UserInfo[0, 5];
                    LoginInfo.Email = UserInfo[0, 6];
                    LoginInfo.PhoneNumber = UserInfo[0, 7];
                    LoginInfo.Permissions = UserInfo[0, 8];

                    Main_WindowToolbar Main_WindowToolbar = new Main_WindowToolbar();
                    this.Hide();
                    Main_WindowToolbar.ShowDialog();
                    this.Close();
                }
                else
                {
                    MessageBox.Show($"Login Failed - Please try again.");
                    Username_TxtBox.Text = "";
                    Password_TxtBox.Text = "";
                    attempts++;
                    return;
```

```
                }
            }
            else
            {
                MessageBox.Show($"Login Failed - Please try again.");
                Username_TxtBox.Text = "";
                Password_TxtBox.Text = "";
                attempts++;
                return;
            }
        }
    }
    private void showPassword_CheckBox_CheckedChanged(object sender, EventArgs e)
    {
        if (showPassword_CheckBox.Checked == true)
        {
            Password_TxtBox.UseSystemPasswordChar = false;
        }
        else
        {
            Password_TxtBox.UseSystemPasswordChar = true;
        }
    }
    private void Close_Btn_Click(object sender, EventArgs e)
    {
        this.Close();
    }
  }
}
```

Multi-User real time messaging

Server-Side

Client-Side

```
using System;
using System.Drawing;
using System.Threading.Tasks;
using System.Windows.Forms;
using Microsoft.AspNetCore.SignalR.Client;
using Winforms;

namespace Moon_Archie_Winforms_NEA
{
  public partial class Chat_Frm : Form
  {
    //public int ID_Length = LoginInfo.EmployeeID.Length;
    public bool connected = false;
    private HubConnection hubConnection;
    public Chat_Frm()
    {
```

```
    InitializeComponent();

    // Creates the signalR connection and tells it what host to connect to
    hubConnection = new HubConnectionBuilder()
        .WithUrl("https://moonarchieserversidenea.azurewebsites.net/chatHub")
        .Build();

    // Event handler for when a message is recieved
    hubConnection.On<string>("ReceiveMessage", (Message) =>
    {
        // Splits the name and the ID into seperate strings
        // Stores the ID as a string
        string[] Message_And_ID_Strings = Message.Split(':');
        string Id_From_Message = Message_And_ID_Strings[1];

        // Connects to the database and gets the name if the employee with that id
        string Query = "SELECT FirstName FROM EmployeeDetails " +
                       $"WHERE EmployeeID = '{Id_From_Message}'";
        string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");

        // If the ID of the user isnt the same as the ID of the sender
        // The message is displayed with the name of the sender
        if (LoginInfo.EmployeeID != Id_From_Message)
        {
            // Stores the Message as a string
            Message = Message_And_ID_Strings[0];

            // Displays the Message (Name + : + Message Text) in the text box
            DisplayMessage(Results[0, 0] + $": {Message}\n", Messages_TxtBox);
        }
         // If there is a connection it adds the new message to the  database
         // Any messages sent when there is no connection will not be stored
        if (connected == true)
        {
            Query = "INSERT INTO Messages (EmployeeID, Message) " +
            $"VALUES ('{LoginInfo.EmployeeID}',  '{MessageInput_TxtBox.Text}');";
            Database_Opperations_Class.Connect_Read_Write(Query, "Write");
        }
    });
}
private async void Chat_Frm_Load(object sender, EventArgs e)
{
    // Gets the number of messages in the database
    string Query = "SELECT Count(*) " +
                   "FROM Messages";
    string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");

    // Stores the count in the Message info static class
    Message_Info_Class.Message_Count = int.Parse(Results[0, 0]);

    // Calls the message loader to load the last 10 previous messages
    Message_Loader();

    // Starts the signalR connection when the form is opened
    await ConnectAsync();
```

```
    }
    private async Task ConnectAsync()
    {
        try
        {
            // Starts the SignalR connection
            await hubConnection.StartAsync();
            // Sets the connection button to green if connection is made
            Connect_Btn.BackColor = Color.LimeGreen;
            connected = true;
        }
        catch (Exception error)
        {
            if (error.Message == "The HubConnection cannot be started if it is not in the Disconnected
state.")
            {
                // If the error is because the connection is already made connection the button stays green
                Connect_Btn.BackColor = Color.LimeGreen;
            }
            else
            {
                // If there is a connection error for another reason the button turns red
                // The error message caught is displayed in the feed
                DisplayMessage($"Connection error: {error.Message}\n", Messages_TxtBox);
                Connect_Btn.BackColor = Color.Firebrick;
            }
        }
    }
    private async Task SendMessageAsync(string Message)
    {
        try
        {
            // Calls the end message method on the server
            await hubConnection.InvokeAsync("SendMessage", Message);

            // Splits the name and the ID into seperate strings
            string[] Message_And_ID_Strings = Message.Split(':');
            Message = Message_And_ID_Strings[0];

            // Displays the Message (You + : + Message Text) in the text box
            DisplayMessage($"You: {Message}\n", Messages_TxtBox);
        }
        catch (Exception ex)
        {
            // Displays any error messages that come from sending the message
            DisplayMessage($"Error sending message: {ex.Message}\n", Messages_TxtBox);
        }
    }
    private void DisplayMessage(string Message, RichTextBox Message_TxtBox)
    {
        // Displays the message in text box
        Message_TxtBox?.BeginInvoke(new Action(() => { Message_TxtBox.AppendText(Message +
Environment.NewLine); }));
    }
    private async void Message_Send_Btn_Click(object sender, EventArgs e)
```

```
{
    // Adds the id of the sender to the end of the message when the message is sent
    string Message = $"{MessageInput_TxtBox.Text}:{LoginInfo.EmployeeID}";

    // Calls the send message function and passes in the message and clears the text box
    await SendMessageAsync(Message);

    // Clears the message box
    MessageInput_TxtBox.Clear();
}
private async void Connect_Btn_Click(object sender, EventArgs e)
{
    // Attempts to connect to the server when the connect button is pressed
    await ConnectAsync();
}

private void Load_Messages_Btn_Click(object sender, EventArgs e)
{
    // Calls the message loader to load more previous messages
    Message_Loader();
}

private void Message_Loader()
{
    // Creates an int to store how many times the loader will need to be looped
    int Loop_Count = 0;

    // Creates a string with the messages to add from the database
    string Messages_To_add = "";

    // If the count minus 9 >= 0, this means that there are more than 10 messages to load
    // The loop count is then set to 10 as we load 10 messages at a time
    if (Message_Info_Class.Message_Count - 9 >= 0)
    {
        // Message count is updated for the next itteration
        Message_Info_Class.Message_Count = Message_Info_Class.Message_Count - 10;

        // The new start index is set to the new message count
        Message_Info_Class.Start_Index = Message_Info_Class.Message_Count;
        Loop_Count = 10;
    }
    // If there are less than 10 messages
    // We need to work out how many times to loop to avoid an index error
    else
    {
        // Start index is set to 0
        Message_Info_Class.Start_Index = 0;

        // The loop count is set for the count of messages
        Loop_Count = Message_Info_Class.Message_Count;

        // The load messages button is made invisible as there are no more messages to load
        Load_Messages_Btn.Visible = false;
    }
```

```
            // Selects the message and employee name
            // From the employeeID of that message and stores them in a 2D array
            string Query = "SELECT EmployeeDetails.FirstName, Messages.Message " +
                           "FROM Messages " +
                           "JOIN EmployeeDetails ON Messages.EmployeeId = EmployeeDetails.EmployeeId;";
            string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");

            // The index is set to the start index calculated above
            int index = Message_Info_Class.Start_Index;

            // Loops from x = 0 to x = Loop_Count
            for (int x = 0; x < Loop_Count; x++)
            {
                // If the Name is the same as the name of the user
                if (Results[index, 0] == LoginInfo.First_Name)
                {
                    // The message is added to string of all messages to add (You: 'Message')
                    Messages_To_add = $"{Messages_To_add} You: {Results[index, 1]} \n\n";
                }
                else
                {
                    // The message is added to string of all messages to add (Employee Name: 'Message')
                    Messages_To_add = $"{Messages_To_add} {Results[index, 0]}: {Results[index, 1]}\n\n";
                }
                // The index count is incrimented
                index++;
            }
            // The messages to add are added at the top of the text box
            Messages_TxtBox.Text = Messages_To_add + Messages_TxtBox.Text;
        }
    }
}
```

Account management

```
using System;
using System.Linq;
using System.Windows.Forms;
using Winforms;

namespace Moon_Archie_Winforms_NEA
{
    public partial class Account_Frm : Form
    {
        // Stores weather the password has been changed
        public static bool Password_Changed = false;
        public Account_Frm()
        {
            InitializeComponent();
        }
        private void Account_Frm_Load(object sender, EventArgs e)
        {
            // All the data is loaded into the account details fields from the static logininfo class
            // Some fields are formatted
```

```
            Name_TxtBox.Text = LoginInfo.First_Name + " " + LoginInfo.Surname;
            Email_TxtBox.Text = LoginInfo.Email;
            PhoneNumber_TxtBox.Text = "+" + LoginInfo.PhoneNumber;
            Username_TxtBox.Text = LoginInfo.Username;
            // A placeholder string otherwise the field will show the hashed string which is far too long
            Password_TxtBox.Text = "Password";
        }
        private void Save_Btn_Click(object sender, EventArgs e)
        {
            // Uses the data validation class to check each field is appropriate and can be added to the database
            bool ValidName = Data_Validation_Functions_Class.IsNumberOfWords(Name_TxtBox.Text, 2);
            bool ValidEmail = Data_Validation_Functions_Class.IsValidEmial(Email_TxtBox.Text);
            bool ValidPhoneNumber =
Data_Validation_Functions_Class.IsValidPhoneNumber(PhoneNumber_TxtBox.Text);
            bool ValidUsername = Data_Validation_Functions_Class.IsBetweenLength(Username_TxtBox.Text, 5,
20);

            // Checks if all the booleans are true
            if (ValidName && ValidEmail && ValidPhoneNumber && ValidUsername)
            {
                // Any pressed edit buttons become visible again and Text fields become read only
                Set_Edit_Btn_Visibility(true);
                Enable_TxtBox_Fields(false);

                Save_Btn.Visible = false;

                // The password text box reverts back to SystemPasswordChar
                Password_TxtBox.UseSystemPasswordChar = true;

                // Splits the Name entered into the text box into firstname and surname
                string[] NameParts = Name_TxtBox.Text.Split(' ');

                // Firstname and surname variables are updated
                string FirstName = NameParts[0];
                string Surname = string.Join(" ", NameParts.Skip(1));

                string Query;
                // Checks if the password has been changed
                if (Password_Changed == true)
                {
                    bool ValidPassword = Data_Validation_Functions_Class.IsValidPassword(Password_TxtBox.Text,
5, 20);
                    if (ValidPassword)
                    {
                        // The new password is hashed and stored locally
                        string Password = Password_Hasher_Class.HashPassword(Password_TxtBox.Text,
LoginInfo.Salt);

                        // All changes are updated in the SQLite database and within the loginInfo static class
                        Query = $"UPDATE EmployeeDetails " +
                            $"SET FirstName = '{FirstName}', Surname = '{Surname}', Username =
'{Username_TxtBox.Text}', Password = '{Password}', PhoneNumber =
'{PhoneNumber_TxtBox.Text.Substring(1)}' " +
                            $"WHERE Username = '{LoginInfo.Username}' AND Password = '{LoginInfo.Password}'";
                        Database_Opperations_Class.Connect_Read_Write(Query, "write");
```

```
                    // The remaining data is also updated in the static class, regardless of if its changed
                    LoginInfo.Email = Email_TxtBox.Text;
                    LoginInfo.PhoneNumber = PhoneNumber_TxtBox.Text.Substring(1);
                    LoginInfo.Username = Username_TxtBox.Text;
                    LoginInfo.First_Name = FirstName;
                    LoginInfo.Surname = Surname;

                    // Password text box is set back to placeholder string
                    Password_TxtBox.Text = "Password";

                    LoginInfo.Password = Password;
                }
                else
                {
                    MessageBox.Show("One or more data fields is in incorrect format.\nPlease double check all
fields are correct.\nPlease make sure your password contains:\nUppercase (1)\nSpecial Characters
(1)\nNumbers (1)\nCharacters (5 - 20)");

                }
            }
            else
            {
                // Query is the same as above but excludes the password
                // The password isnt included in this query as it hasnt been changed
                Query = $"UPDATE EmployeeDetails " +
                    $"SET FirstName = '{FirstName}', Surname = '{Surname}', Username =
'{Username_TxtBox.Text}', PhoneNumber = '{PhoneNumber_TxtBox.Text.Substring(1)}' " +
                    $"WHERE Username = '{LoginInfo.Username}' AND Password = '{LoginInfo.Password}'";
                // The according query is performed
                Database_Opperations_Class.Connect_Read_Write(Query, "write");

                // The remaining data is also updated in the static class, regardless of if its changed
                LoginInfo.Email = Email_TxtBox.Text;
                LoginInfo.PhoneNumber = PhoneNumber_TxtBox.Text.Substring(1);
                LoginInfo.Username = Username_TxtBox.Text;
                LoginInfo.First_Name = FirstName;
                LoginInfo.Surname = Surname;

                // Password text box is set back to placeholder string
                Password_TxtBox.Text = "Password";
            }
        }
        else
        {
            // If any validation checks return false an error message box is displayed
            MessageBox.Show("One or more data fields is in incorrect format.\nPlease double check all fields
are correct.\nPlease make sure your password contains:\nUppercase (1)\nSpecial Characters (1)\nNumbers
(1)\nCharacters (5 - 20)");
        }
    }
    private void EditName_Btn_Click(object sender, EventArgs e)
    {
        // enables the name text box for editing
        EditName_Btn.Visible = false;
```

```
    Save_Btn.Visible = true;
    Name_TxtBox.Enabled = true;
}
private void EditEmail_Btn_Click(object sender, EventArgs e)
{
    // enables the email text box for editing
    EditEmail_Btn.Visible = false;
    Save_Btn.Visible = true;
    Email_TxtBox.Enabled = true;
}
private void EditPhoneNumber_Btn_Click(object sender, EventArgs e)
{
    // enables the PhoneNumber text box for editing
    EditPhoneNumber_Btn.Visible = false;
    Save_Btn.Visible = true;
    PhoneNumber_TxtBox.Enabled = true;
}
private void EditUsername_Btn_Click(object sender, EventArgs e)
{
    // enables the username text box for editing
    EditUsername_Btn.Visible = false;
    Save_Btn.Visible = true;
    Username_TxtBox.Enabled = true;
}
private void EditPassword_Btn_Click(object sender, EventArgs e)
{
    // enables the password text box for editing
    EditPassword_Btn.Visible = false;
    Confirm_Password_Pnl.Visible = true;
}

///
// Code bellow is for the confirm password panel and its controls
///

private void Close_Confirm_Password_Btn_Click(object sender, EventArgs e)
{
    Confirm_Password_Pnl.Visible = false;
    Confirm_Password_TxtBox.Text = "";
    Save_Btn.Visible = false;
    EditPassword_Btn.Visible = true;
}
private void Confirm_Password_Btn_Click(object sender, EventArgs e)
{
    // Uses the verify function to compare the password they entered to the stored one
    string Password = Confirm_Password_TxtBox.Text;

    if (Password_Hasher_Class.VerifyPassword(Password, LoginInfo.Salt, LoginInfo.Password))
    {
        Password_Changed = true;
        Save_Btn.Visible = true;
        Confirm_Password_TxtBox.Text = "";
        Confirm_Password_Pnl.Visible = false;
        Password_TxtBox.Text = Password;
```

```
            // Reveals the hidden password and allows it to be edited
            Password_TxtBox.UseSystemPasswordChar = false;
            Password_TxtBox.Enabled = true;
        }
        else
        {
            // Displays an error message if the passwords do not match
            MessageBox.Show("Incorrect password; Please try again.");
            Confirm_Password_TxtBox.Text = "";
        }
    }
    private void Set_Edit_Btn_Visibility(bool state)
    {
        // Makes all edit buttons visible or invisible
        EditName_Btn.Visible = state;
        EditEmail_Btn.Visible = state;
        EditPhoneNumber_Btn.Visible = state;
        EditPassword_Btn.Visible = state;
        EditUsername_Btn.Visible = state;
    }
    private void Enable_TxtBox_Fields(bool state)
    {
        // Makes all text boxes enabled or read only
        Name_TxtBox.Enabled = state;
        Email_TxtBox.Enabled = state;
        PhoneNumber_TxtBox.Enabled = state;
        Username_TxtBox.Enabled = state;
        Password_TxtBox.Enabled = state;
    }
  }
}
```

Viewing and accepting shifts

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using Winforms;

namespace Moon_Archie_Winforms_NEA
{
  public partial class Shifts_Frm : Form
  {
    public int Shift_Loader_Start_Index;
    public int Shift_Loader_End_Index;
    public int Shift_Loader_Shift_Count;
    public string Selected_Day;
    public Shifts_Frm()
    {
      InitializeComponent();
    }
    private void Monday_Shifts_Btn_Click(object sender, EventArgs e)
    {
```

```
    Selected_Day = "Mon";
    TableLayoutPanel_Clear();
    Main_Control_Pnl.Visible = true;
    Initialize_Shift_Loader("Mon");
    Shift_Loader("Mon");
}
private void Tuesday_Shifts_Btn_Click(object sender, EventArgs e)
{
    Selected_Day = "Tue";
    TableLayoutPanel_Clear();
    Main_Control_Pnl.Visible = true;
    Initialize_Shift_Loader("Tue");
    Shift_Loader("Tue");
}
private void Wednesday_Shifts_Btn_Click(object sender, EventArgs e)
{
    Selected_Day = "Wed";
    TableLayoutPanel_Clear();
    Main_Control_Pnl.Visible = true;
    Initialize_Shift_Loader("Wed");
    Shift_Loader("Wed");
}
private void Thursday_Shifts_Btn_Click(object sender, EventArgs e)
{
    Selected_Day = "Thu";
    TableLayoutPanel_Clear();
    Main_Control_Pnl.Visible = true;
    Initialize_Shift_Loader("Thu");
    Shift_Loader("Thu");
}
private void Friday_Shifts_Btn_Click(object sender, EventArgs e)
{
    Selected_Day = "Fri";
    TableLayoutPanel_Clear();
    Main_Control_Pnl.Visible = true;
    Initialize_Shift_Loader("Fri");
    Shift_Loader("Fri");
}
private void Saturday_Shifts_Btn_Click(object sender, EventArgs e)
{
    Selected_Day = "Sat";
    TableLayoutPanel_Clear();
    Main_Control_Pnl.Visible = true;
    Initialize_Shift_Loader("Sat");
    Shift_Loader("Sat");
}
private void Sunday_Shifts_Btn_Click(object sender, EventArgs e)
{
    Selected_Day = "Sun";
    TableLayoutPanel_Clear();
    Main_Control_Pnl.Visible = true;
    Initialize_Shift_Loader("Sun");
    Shift_Loader("Sun");
}
private void Shifts_Frm_Load(object sender, EventArgs e)
```

```
        {
        }
        private void TableLayoutPanel_Clear()
        {
            foreach (Control control in Shift_View_Tbl.Controls)
            {
                // Clear the text of all controls
                control.Text = "";

                // Check if the control is a button
                if (control is Button)
                {
                    control.Enabled = false;
                }
            }
        }
        private void Initialize_Shift_Loader(string Day)
        {
            string Query = "SELECT OvertimeShiftID, Date, Day, StartTime, Hours " +
                           "FROM OvertimeShifts " +
                           $"WHERE EmployeeID IS NULL AND Day = '{Day}'";
            string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");
            Shift_Loader_Shift_Count = Results.GetLength(0) - 1;

            // Decides weather we can load all of the shifts as there is 10 or less
            if (Shift_Loader_Shift_Count <= 9)
            {
                Shift_Loader_Start_Index = 0;
                Shift_Loader_End_Index = Shift_Loader_Shift_Count;
                Page_Down_Btn.Enabled = false;
                Page_Up_Btn.Enabled = false;
            }
            // or weather we load the first just 10
            else
            {
                Shift_Loader_Start_Index = 0;
                Shift_Loader_End_Index = 9;
                Page_Down_Btn.Enabled = true;
            }
        }
        private void Shift_Loader(string Day)
        {
            // Selects the Day Date start time and hours from the overtimeshifts table where no employeeID has
been specified
            string Query = "SELECT OvertimeShiftID, Date, Day, StartTime, Hours " +
                           "FROM OvertimeShifts " +
                           $"WHERE EmployeeID IS NULL AND Day = '{Day}'";
            string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");

            List<string> Shift_Details = new List<string>();

            int i = 0;

            for (int x = Shift_Loader_Start_Index; x <= Shift_Loader_End_Index;  x++)
            {
```

```
            // Gets the integer value of the time
            int Hours = (int)double.Parse(Results[x, 3]);

            // Gets the decimal value of the time
            double DecimalPart = double.Parse(Results[x, 3]) - Hours;

            // Calculate minutes from the decimal value
            int Minutes = (int)(DecimalPart * 60);

            // Formats the hours and minutes into HH:mm format
            string FormattedTime = string.Format("{0:00}:{1:00}", Hours, Minutes);

            // Adds the database fields and the formatted time into the list
            Shift_Details.Add($"{Results[x, 1]} | {Results[x, 2]} | Start Time: {FormattedTime} | {Results[x, 4]}
Hours");

            Label Label = (Label)Shift_View_Tbl.GetControlFromPosition(0, i);
            Button Button = (Button)Shift_View_Tbl.GetControlFromPosition(1, i);

            Label.Text = Shift_Details[i];

            Button.Tag = Results[x, 0];
            Button.Enabled = true;

            // Update button Text and handle the click event
            Button.Text = "Accept";
            Button.Click += (sender, e) =>
            {
               // Get the shift ID from the Tag property of the button
               string shiftID = Button.Tag.ToString();

               // Selects the EmployeeID from the OvertimeShifts table Where the shiftID is equal to the one
chosen
               Query = "SELECT EmployeeID " +
                       "FROM OvertimeShifts " +
                       $"WHERE OvertimeShiftID = {shiftID}";
               Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");
               if (Results[0, 0] == "")
               {
                  MessageBox.Show("Accepted");

                  // Update the shift in the database using the shiftID
                  Query = "UPDATE OvertimeShifts " +
                          $"SET EmployeeID = {LoginInfo.EmployeeID} " +
                          $"WHERE OvertimeShiftID = {shiftID}";
                  Database_Opperations_Class.Connect_Read_Write(Query, "write");

                  Main_Control_Pnl.Visible = false;
               }
            };
            i++;
         }
      }
      private void Page_Down_Btn_Click(object sender, EventArgs e)
      {
```

```
            Page_Up_Btn.Enabled = true;
            Page_Down_Btn.Enabled = false;
            // Loads the second set of 10
            Shift_Loader_Start_Index = 10;
            Shift_Loader_End_Index = Shift_Loader_Shift_Count;
            TableLayoutPanel_Clear();
            Shift_Loader(Selected_Day);
        }
        private void Page_Up_Btn_Click(object sender, EventArgs e)
        {
            Page_Down_Btn.Enabled = true;
            Page_Up_Btn.Enabled = false;
            // Loads the first 10
            Shift_Loader_Start_Index = 0;
            Shift_Loader_End_Index = 9;
            TableLayoutPanel_Clear();
            Shift_Loader(Selected_Day);
        }
    }
}
```

3NF SQLite database

Payslip creation and storing

Main Form and past payslips

```
using System;
using System.Windows.Forms;
using Winforms;

namespace Moon_Archie_Winforms_NEA
{
    public partial class Pay_Calculator_Frm : Form
    {
        PaySlip_Window PaySlip_Window = new PaySlip_Window();

        public Pay_Calculator_Frm()
        {
            InitializeComponent();
        }

        private void new_Payslip_Btn_Click(object sender, EventArgs e)
        {
            // adds the new payslip form into the mini window
            PaySlip_Window.SendToBack();
            PaySlip_Window.BringToFront();
            PaySlip_Window.Show();
        }

        private void Pay_Calculator_Frm_Load(object sender, EventArgs e)
```

```
        {
            Payslip_TxtBox.Text = "";
            DBSQLite_Class.DBSQLite_Class db = new DBSQLite_Class.DBSQLite_Class(@"data source =
MainDatabase.db");
            string query = "SELECT Count(*) " +
                        "FROM PaySlips " +
                        "WHERE EmployeeID = '" + LoginInfo.EmployeeID + "'";
            string[,] results = db.GetRecords(query);
            int count = int.Parse(results[0, 0]);

            query = "SELECT * " +
                  "FROM PaySlips " +
                  "WHERE EmployeeID = '" + LoginInfo.EmployeeID + "'";
            results = db.GetRecords(query);

            // gets all the data from the database and formats it into a string showing the details of each payslip
            for (int x = 0; x < count; x++)
            {
                Payslip_TxtBox.Text = results[x, 1] + " . " + results[x, 2] + " . " + results[x, 3] + " . " + results[x, 4] +
" . " + results[x, 5] + " Hours \n\n" + Payslip_TxtBox.Text;
            }

            PaySlip_Window.TopLevel = false;
            Payslip_View_Pnl.Controls.Add(PaySlip_Window);
        }
    }
}
```

New payslip sub -window

```
using System;
using System.Windows.Forms;
using Winforms;

namespace Moon_Archie_Winforms_NEA
{
    public partial class PaySlip_Window : Form
    {
        public PaySlip_Window()
        {
            InitializeComponent();
        }

        private void PaySlip_Window_Load(object sender, EventArgs e)
        {
            Site_CheckBox.CheckOnClick = false;
            Template_CheckBox.CheckOnClick = false;
            Position_CheckBox.CheckOnClick = false;
        }

        private void Save_Btn_Click(object sender, EventArgs e)
        {
            // checks if all the fields are in the correct format
            bool Hours_Is_Double;
```

```
            try
            {
                double.Parse(Hours_Worked_TxtBox.Text);
                Hours_Is_Double = true;
            }
            catch
            {
                Hours_Is_Double = false;
            }
            if (Position_CheckBox.SelectedIndex.ToString() == "-1" ||
Template_CheckBox.SelectedIndex.ToString() == "-1" || Site_CheckBox.SelectedIndex.ToString() == "-1" ||
Hours_Is_Double != true)
            {
                MessageBox.Show("This doesnt look right,\nPlease fill in all fields correctly.");
            }
            else
            {
                Confirm_Password_Pnl.Visible = true;
            }
        }

        private void Confirm_Password_Btn_Click(object sender, EventArgs e)
        {
            // makes the user enter their password in order to be able to submit the payslip
            string password = Confirm_Password_TxtBox.Text;
            if (Password_Hasher_Class.VerifyPassword(password, LoginInfo.Salt, LoginInfo.Password))
            {
                Submit_Btn.Enabled = true;
                Confirm_Password_TxtBox.Text = "";
                Confirm_Password_Btn.Visible = false; Confirm_Password_Lbl.Visible = false;
Confirm_Password_TxtBox.Visible = false;
            }
            else
            {
                // shows an error message if they get the password wrong
                MessageBox.Show("Incorrect password; Please try again.");
                Confirm_Password_TxtBox.Text = "";
            }
        }

        private void Submit_Btn_Click(object sender, EventArgs e)
        {
            // gets each of the selected fields from the radio buttons
            string Position;
            if (Position_CheckBox.SelectedIndex.ToString() == "0") { Position = "Lifeguard"; }
            else { Position = "Swim School Assistant Instructor"; }

            string Template;
            if (Template_CheckBox.SelectedIndex.ToString() == "0") { Template = "Casual"; }
            else { Template = "Contracted"; }

            string Site;
            if (Site_CheckBox.SelectedIndex.ToString() == "0") { Site = "Splashpoint"; }
            else { Site = "Wadurs"; }
```

```
            // adds the payslip to the database
            DBSQLite_Class.DBSQLite_Class db = new DBSQLite_Class.DBSQLite_Class(@"data source =
MainDatabase.db");
            string query = "INSERT INTO PaySlips (EmployeeID, Date, Position, Template, Site, HoursWorked) " +
                        "VALUES ('" + LoginInfo.EmployeeID + "', '" + Date_Picker.Text + "', '" + Position +"', '" +
Template +"', '" + Site +"', '" + Hours_Worked_TxtBox.Text + "');";
            db.UpdateInsertDeleteRecords(query);

            // Clear all the fields and makes the submit button invisible
            ClearFields();
            Submit_Btn.Visible = false;
        }
        private void ClearFields()
        {
            Hours_Worked_TxtBox.Text = "";
            Position_CheckBox.SelectedIndex = -1;
            Template_CheckBox.SelectedIndex = -1;
            Site_CheckBox.SelectedIndex = -1;
        }
        private void Clear_Btn_Click(object sender, EventArgs e)
        {
            ClearFields();
        }
    }
}
```

Admin permission functions (adding, removing, editing users and adding shifts)

Adding Users

```
using System;
using System.Windows.Forms;
using Winforms;
using System.Linq;

namespace Moon_Archie_Winforms_NEA
{
    public partial class Add_User_Frm : Form
    {
        public Add_User_Frm()
        {
            InitializeComponent();
        }
        private void Save_Btn_Click(object sender, EventArgs e)
        {
            // Uses the data validation class to check each field is appropriate and the user can proceed to
confirming their password
            bool ValidName = Data_Validation_Functions_Class.IsNumberOfWords(Name_TxtBox.Text, 2);
            bool ValidEmail = Data_Validation_Functions_Class.IsValidEmial(Email_TxtBox.Text);
            bool ValidPhoneNumber =
Data_Validation_Functions_Class.IsValidPhoneNumber(PhoneNumber_TxtBox.Text);
            bool ValidUsername = Data_Validation_Functions_Class.IsBetweenLength(Username_TxtBox.Text, 5,
20);
            bool ValidPassword = Data_Validation_Functions_Class.IsValidPassword(Password_TxtBox.Text, 5,
20);
```

```
bool NoneSelected = true;

foreach (RadioButton radioButton in Permission_GroupBox.Controls)
{
    if (radioButton.Checked)
    {
        NoneSelected = false;
        break;
    }
}

        if (ValidName && ValidEmail && ValidPhoneNumber && ValidUsername && ValidPassword &&
NoneSelected != true)
        {
            // The confirm password panel becomes visible
            Confirm_Password_Pnl.Visible = true;
        }
        else
        {
            // If any validation checks return false an error message box is displayed
            MessageBox.Show("One or more data fields is in incorrect format.\nPlease double check all fields
are correct.\nPlease make sure your password contains:\nUppercase (1)\nSpecial Characters (1)\nNumbers
(1)\nCharacters (5 - 20)");
        }
    }
    private void Close_Confirm_Password_Btn_Click(object sender, EventArgs e)
    {
        // The panel becomes invisible and the text field is cleared
        Confirm_Password_Pnl.Visible = false;
        Confirm_Password_TxtBox.Text = "";
    }
    private void Confirm_Password_Btn_Click(object sender, EventArgs e)
    {
        string Password = Confirm_Password_TxtBox.Text;
        // Checks if the password entered by the person adding the user is correct
        if (Password_Hasher_Class.VerifyPassword(Password, LoginInfo.Salt, LoginInfo.Password) == true)
        {
            // Gets the permision selected for the new user and stores it as a string
            string Permission = "";
            if (Employee_Permision_RBtn.Checked == true)
            {
                Permission = "E";
            }
            else if (Manager_Permision_RBtn.Checked == true)
            {
                Permission = "M";
            }

            // Splits the name entered into first name and surname and stores each seperatley
            string[] NameParts = Name_TxtBox.Text.Split(' ');
            string FirstName = NameParts[0];
            string Surname = string.Join(" ", NameParts.Skip(1));

            // Generates a new random salt for the user
```

```
            string Salt = Password_Hasher_Class.GenerateRandomSalt();

            // The query that adds the new user to the database
            // The hashed password is only calculated here when it is added so its not stored within the code
            string Query = "INSERT INTO EmployeeDetails (FirstName, Surname, Username, Password, Salt,
Email, PhoneNumber, Permission) " +
                        $"VALUES ('{FirstName}', '{Surname}', '{Username_TxtBox.Text}',
'{Password_Hasher_Class.HashPassword(Password_TxtBox.Text, Salt)}', '{Salt}' , '{Email_TxtBox.Text}',
'{PhoneNumber_TxtBox.Text.Remove(0, 1)}', '{Permission}')";
            Database_Opperations_Class.Connect_Read_Write(Query, "write");

            // Shows a name box showing the new user has been added succesfuly
            MessageBox.Show($"{Name_TxtBox.Text} Succesfuly added.");

            ClearAllFields();

            Confirm_Password_Pnl.Visible = false;
        }
        else
        {
            // If the user's (the one adding the new user) password is incorrect an error message is displayed
            MessageBox.Show("Incorrect password - Please try again.");
            Confirm_Password_TxtBox.Text = "";
        }
    }
    private void Add_User_Frm_Load(object sender, EventArgs e)
    {

    }
    private void ClearAllFields()
    {
        // Subroutine that can be called to clear every field in the form
        Name_TxtBox.Text = "";
        Username_TxtBox.Text = "";
        Password_TxtBox.Text = "";
        Email_TxtBox.Text = "";
        Confirm_Password_TxtBox.Text = "";
        PhoneNumber_TxtBox.Text = "+44";
        Manager_Permision_RBtn.Checked = false;
        Employee_Permision_RBtn.Checked = false;
    }
  }
}
```

Removing Users

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using Winforms;

namespace Moon_Archie_Winforms_NEA
{
```

```
public partial class Remove_User_Frm : Form
{
    public string EmployeeIDs = "";
    public Remove_User_Frm()
    {
        InitializeComponent();
    }

    private void Remove_User_Frm_Load(object sender, EventArgs e)
    {
        Update_DataGrid_View();
    }

    private void Remove_Btn_Click(object sender, EventArgs e)
    {
        // The confirm password panel becomes visible
        Confirm_Password_Pnl.Visible = true;
        Remove_Btn.Enabled = false;
    }

    private void Close_Confirm_Password_Btn_Click(object sender, EventArgs e)
    {
        // The panel becomes invisible and the text field is cleared
        Confirm_Password_Pnl.Visible = false;
        Confirm_Password_TxtBox.Text = "";
        Remove_Btn.Enabled = true;
    }

    private void Confirm_Password_Btn_Click(object sender, EventArgs e)
    {
        bool ValidID = EmployeeIDs.Contains(EmployeeID_Selector_NumericUpDown.Value.ToString());

        string Password = Confirm_Password_TxtBox.Text;
        // Checks if the password entered by the person adding the user is correct
        if (Password_Hasher_Class.VerifyPassword(Password, LoginInfo.Salt, LoginInfo.Password) == true
&& ValidID == true)
        {
            string Query = "DELETE FROM EmployeeDetails " +
                        $"WHERE EmployeeID = '{EmployeeID_Selector_NumericUpDown.Value}';" +
                        "DELETE FROM Messages " +
                        $"WHERE EmployeeID = '{EmployeeID_Selector_NumericUpDown.Value}';" +
                        "DELETE FROM ContractedShifts " +
                        $"WHERE EmployeeID = '{EmployeeID_Selector_NumericUpDown.Value}';" +
                        "DELETE FROM OvertimeShifts " +
                        $"WHERE EmployeeID = '{EmployeeID_Selector_NumericUpDown.Value}';" +
                        "DELETE FROM PaySlips " +
                        $"WHERE EmployeeID = '{EmployeeID_Selector_NumericUpDown.Value}';";
            Database_Opperations_Class.Connect_Read_Write(Query, "write");

            // Shows a confirmation that the user has been removed
            MessageBox.Show($"User has been succesfuly deleted.");

            // Updates the Data grid view to show the updated list of employees
            Update_DataGrid_View();
```

```
            // Resets the ID selector the the default
            EmployeeID_Selector_NumericUpDown.Value = 1;

            // Makes the confirm password panel invisible again
            Confirm_Password_Pnl.Visible = false;
            Confirm_Password_TxtBox.Text = "";
            Remove_Btn.Enabled = true;
        }
        else
        {
            // If the user's (the one adding the new user) password is incorrect an error message is displayed
            MessageBox.Show("Incorrect password Or invalid ID selected\nPlease try again.");
            Confirm_Password_TxtBox.Text = "";
        }
    }
    private void Update_DataGrid_View()
    {
        string Query = "SELECT EmployeeID, Firstname, Surname, Email, PhoneNumber " +
                       "FROM EmployeeDetails ";
        string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");

        // List to store instances of a custom class containing employee details
        List<Employee_Search_Details> User_List = new List<Employee_Search_Details>();
        for (int x = 0; x < Results.GetLength(0); x++)
        {
            // Create a new Employee_Search_Details object and adds it to the list
            User_List.Add(new Employee_Search_Details(Results[x, 0], Results[x, 1], Results[x, 2], Results[x, 3],
Results[x, 4]));

            // adds the employees id to the list of valid ID's
            EmployeeIDs += Results[x, 0];
        }
        // Set the DataSource of the DataGridView to the list of employee details
        Database_Users_GridView.DataSource = User_List;
        EmployeeID_Selector_NumericUpDown.Maximum = int.Parse(Results[(Results.GetLength(0) - 1), 0]);
    }
  }
}
```

Editing Users

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows.Forms;

using Winforms;


namespace Moon_Archie_Winforms_NEA
```

```
{
    public partial class Edit_User_Info_Frm : Form
    {
        public string EmployeeIDs = "";
        public bool Password_Changed = false;
        public string SelectedID = "";
        public Edit_User_Info_Frm()
        {
            InitializeComponent();
        }

        private void Edit_User_Info_Frm_Load(object sender, EventArgs e)
        {
            Update_Database();
        }
        private void Update_Database()
        {
            string Query = "SELECT EmployeeID, Firstname, Surname, Email, PhoneNumber " +
                           "FROM EmployeeDetails ";
            string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");

            // List to store instances of a custom class containing employee details
            List<Employee_Search_Details> User_List = new List<Employee_Search_Details>();
            for (int x = 0; x < Results.GetLength(0); x++)
            {
                // Create a new Employee_Search_Details object and adds it to the list
                User_List.Add(new Employee_Search_Details(Results[x, 0], Results[x, 1], Results[x, 2],
Results[x, 3], Results[x, 4]));
                EmployeeIDs += Results[x, 0];
            }
            // Set the DataSource of the DataGridView to the list of employee details
```

```
        Database_Users_GridView.DataSource = User_List;
        EmployeeID_Selector_NumericUpDown.Maximum = int.Parse(Results[(Results.GetLength(0) -
1), 0]);
    }

    private void Select_Btn_Click(object sender, EventArgs e)
    {
        Password_TxtBox.UseSystemPasswordChar = true;
        Update_Database();
        // checks if the ID selected is valid
        bool ValidID =
EmployeeIDs.Contains(EmployeeID_Selector_NumericUpDown.Value.ToString());
        if (ValidID == true)
        {
            Set_Fields_Enabled(true);
            SelectedID = EmployeeID_Selector_NumericUpDown.Value.ToString();
            // Loads the details of the selected employee into the fields in the form
            string Query = "SELECT FirstName, Surname, Username, Email, Password, PhoneNumber,
Permission " +
                        "FROM EmployeeDetails " +
                        $"WHERE EmployeeID = '{SelectedID}'";
            string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");
            Name_TxtBox.Text = $"{Results[0, 0]} {Results[0, 1]}";
            Email_TxtBox.Text = Results[0, 3];
            PhoneNumber_TxtBox.Text = $"+44{Results[0, 5].Substring(2)}";
            Username_TxtBox.Text = Results[0, 2];
            Password_TxtBox.Text = "Password";
            if (Results[0, 6] == "M")
            {
                Manager_Permision_RBtn.Checked = true;
            }
            else
```

```
            {
                Employee_Permision_RBtn.Checked = true;
            }
        }
        else
        {
            MessageBox.Show("Please select a valid EmployeeID.");
        }
    }

    private void New_Password_Btn_Click(object sender, EventArgs e)
    {
        // clears the placeholder and enables the password text field
        Password_TxtBox.Text = "";
        Password_TxtBox.ReadOnly = false;
        Password_TxtBox.UseSystemPasswordChar = false;
        Password_Changed = true;
    }

    private void Update_Btn_Click(object sender, EventArgs e)
    {
        Set_Fields_Enabled(false);
        Password_TxtBox.UseSystemPasswordChar = true;
        // Uses the data validation class to check each field is appropriate and the user can proceed to confirming their password
        bool ValidName = Data_Validation_Functions_Class.IsNumberOfWords(Name_TxtBox.Text, 2);
        bool ValidEmail = Data_Validation_Functions_Class.IsValidEmial(Email_TxtBox.Text);
        bool ValidPhoneNumber = Data_Validation_Functions_Class.IsValidPhoneNumber(PhoneNumber_TxtBox.Text);
        bool ValidUsername = Data_Validation_Functions_Class.IsBetweenLength(Username_TxtBox.Text, 5, 20);
```

```
        if (ValidName && ValidEmail && ValidPhoneNumber && ValidUsername)
        {
            // The confirm password panel becomes visible
            Confirm_Password_Pnl.Visible = true;
        }
        else
        {
            // If any validation checks return false an error message box is displayed
            MessageBox.Show("One or more data fields is in incorrect format.\nPlease double check all fields are correct.\nPlease make sure your password contains:\nUppercase (1)\nSpecial Characters (1)\nNumbers (1)\nCharacters (5 - 20)");
        }
    }

    private void Close_Confirm_Password_Btn_Click(object sender, EventArgs e)
    {
        Set_Fields_Enabled(true);
        Confirm_Password_Pnl.Visible = false;
        Confirm_Password_TxtBox.Text = "";
    }

    private void Confirm_Password_Btn_Click(object sender, EventArgs e)
    {
        string Password = Confirm_Password_TxtBox.Text;
        // Checks if the password entered by the person adding the user is correct
        if (Password_Hasher_Class.VerifyPassword(Password, LoginInfo.Salt, LoginInfo.Password) == true)
        {
            // Gets the permision selected for the new user and stores it as a string
            string Permission = "";
            if (Employee_Permision_RBtn.Checked == true)
            {
```

```
            Permission = "E";
        }
        else if (Manager_Permision_RBtn.Checked == true)
        {
            Permission = "M";
        }
        // Splits the Name entered into the text box into firstname and surname
        string[] NameParts = Name_TxtBox.Text.Split(' ');

        // Firstname and surname variables are updated
        string FirstName = NameParts[0];
        string Surname = string.Join(" ", NameParts.Skip(1));

        string Query = "";
        // Checks if the password has been changed
        if (Password_Changed == true)
        {
            bool ValidPassword =
Data_Validation_Functions_Class.IsValidPassword(Password_TxtBox.Text, 5, 20);
            if (ValidPassword == true)
            {
                Query = "SELECT Salt " +
                        "FROM EmployeeDetails " +
                        $"WHERE EmployeeID = '{SelectedID}'";
                string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");
                // The new password is hashed and stored locally
                string New_Password = Password_Hasher_Class.HashPassword(Password_TxtBox.Text,
Results[0, 0]);

                // All changes are updated in the SQLite database and within the loginInfo static class
                Query = "UPDATE EmployeeDetails " +
```

```
                    $"SET FirstName = '{FirstName}', Surname = '{Surname}', Username =
'{Username_TxtBox.Text}', Password = '{Password}', PhoneNumber =
'{PhoneNumber_TxtBox.Text.Substring(1)}', Permission = '{Permission}' " +
                    $"WHERE EmployeeID = '{SelectedID}'";
                // The according query is performed
                Database_Opperations_Class.Connect_Read_Write(Query, "write");

                ClearAllFields();

                MessageBox.Show("Changes updated succesfuly.");
            }
            else
            {
                MessageBox.Show("Your password seems to be missing something.\nPlease make sure
it contains:\nUppercase (1)\nSpecial Characters (1)\nNumbers (1)\nCharacters (5 - 20)");
            }
        }
        else
        {
            // Query is the same as above but excludes the password
            // The password isnt included in this query as it hasnt been changed
            Query = $"UPDATE EmployeeDetails " +
                $"SET FirstName = '{FirstName}', Surname = '{Surname}', Username =
'{Username_TxtBox.Text}', PhoneNumber = '{PhoneNumber_TxtBox.Text.Substring(1)}', Permission =
'{Permission}' " +
                $"WHERE EmployeeID = '{SelectedID}'";

            // The according query is performed
            Database_Opperations_Class.Connect_Read_Write(Query, "write");

            ClearAllFields();

            MessageBox.Show("Changes updated succesfuly.");
```

```
            }
            Set_Fields_Enabled(true);
            Confirm_Password_Pnl.Visible = false;
            Confirm_Password_TxtBox.Text = "";
            Update_Database();

        }
        else
        {
            // If the user's (the one adding the new user) password is incorrect an error message is
displayed
            MessageBox.Show("Incorrect password - Please try again.");
            Confirm_Password_TxtBox.Text = "";
        }
    }
    private void ClearAllFields()
    {
        // Subroutine that can be called to clear every field in the form
        Name_TxtBox.Text = "";
        Username_TxtBox.Text = "";
        Password_TxtBox.Text = "";
        Email_TxtBox.Text = "";
        Confirm_Password_TxtBox.Text = "";
        PhoneNumber_TxtBox.Text = "+44";
        Manager_Permision_RBtn.Checked = false;
        Employee_Permision_RBtn.Checked = false;
    }
    private void Set_Fields_Enabled(bool state)
    {
        Name_TxtBox.Enabled = state;
        Email_TxtBox.Enabled = state;
```

```
            PhoneNumber_TxtBox.Enabled = state;
            Username_TxtBox.Enabled = state;
            Password_TxtBox.Enabled = state;
            Manager_Permision_RBtn.Enabled = state;
            Employee_Permision_RBtn.Enabled = state;
            EmployeeID_Selector_NumericUpDown.Enabled = state;
            New_Password_Btn.Visible = state;
            Update_Btn.Visible = state;
            Select_Btn.Enabled = state;
        }
    }
}
```

Adding Shifts

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

namespace Moon_Archie_Winforms_NEA
{
    public partial class Upload_Shifts_Frm : Form
    {
        public Upload_Shifts_Frm()
        {
            InitializeComponent();
        }

        private void Upload_Shifts_Frm_Load(object sender, EventArgs e)
        {
            Update_Database_View();
        }
        private void Update_Database_View()
        {
            string Query = "SELECT EmployeeID, Day, StartTime, Hours " +
                           "FROM OvertimeShifts " +
                           "WHERE EmployeeID IS NULL";
            string[,] Results = Database_Opperations_Class.Connect_Read_Write(Query, "read");

            // List to store instances of a custom class containing the shifts details
            List<Overtime_Shift_Info_Class> Shifts_List = new List<Overtime_Shift_Info_Class>();
            for (int x = 0; x < Results.GetLength(0); x++)
            {
                // Create a new Overtime_Shift_Info object and adds it to the list
```

```
            Shifts_List.Add(new Overtime_Shift_Info_Class(Results[x, 0], Results[x, 1], Results[x, 2], Results[x,
3]));
        }
        // Set the DataSource of the DataGridView to the list of the shifts details
        Current_Shifts_DataGridView.DataSource = Shifts_List;
    }

    private void Clear_Btn_Click(object sender, EventArgs e)
    {
        // Delets all the shifts from the database
        string Query = "DELETE FROM OvertimeShifts ";
        Database_Opperations_Class.Connect_Read_Write(Query, "write");
        Update_Database_View();
    }

    private void Add_Btn_Click(object sender, EventArgs e)
    {
        // Formatts all the data so its ready for the database
        string Selected_Day_Time = Date_Time_Picker.Value.ToString("ddd HH:mm");

        string[] Day_Time = Selected_Day_Time.Split(' ');
        string Time = Day_Time[1];

        string[] Time_Parts = Time.Split(':');
        int hours = int.Parse(Time_Parts[0]);
        int minutes = int.Parse(Time_Parts[1]);

        decimal Hours = Hours_NumericUpDown.Value;

        // stores the formatted inputs
        string Selected_Hours = Hours.ToString("0.0").TrimEnd('0').TrimEnd('.');
        string Selected_Time = (hours + (minutes / 60.0)).ToString();
        string Selected_Day = Day_Time[0];

        // Adds the shift to the database
        string Query = "INSERT INTO OvertimeShifts (Hours, Day, StartTime, Date) " +
                       $"VALUES ('{Selected_Hours}', '{Selected_Day}', '{Selected_Time}',
'{Date_Time_Picker.Value.ToString().Substring(0, Date_Time_Picker.Value.ToString().Length - 12)}')";
        Database_Opperations_Class.Connect_Read_Write(Query, "write");

        Update_Database_View();
    }
  }
}
```

References and sources

Related Topic	Description	Link

Password Hashing	This useful video explained how each type of password hashing works as well as the importance of performing different functions. This lead to a better understanding of how my algorithm should run and and the features that It shouldcontain.	“Password storage tier list: encryption, hashing, salting, bcrypt and beyond” (https://youtu.be/qgpsIBLvrGY?si=p2g3-6vHpjcRMI45)
SignalR	This video gave me a good base to create my signalR chat messaging app and while some of the code I used is very similar to the one in the video I also, through the video, gained a better understanding and added / customised my own methods.	“Create your own chat app: SignalR mastery in c# & ASP.NET” (https://youtu.be/_gJ0NbNKKZg?si=CHJSxjmGtEIkOyuz)
Dynamic text box creation	I used this video to understand how to generate a text box dynamically using c# code which I do when generating the template for the rota.	“How to create a text box dynamically in c#” (https://youtu.be/E4Mu9hhyaz0?si=uPdFtdoJWyLPAsXU)
Human error statistic charts	These images are used to show the need for my application to remove the factor of human error in data driven environments	Left Image: Datastore365 (https://www.datastore365.co.uk/human-error-is-the-biggest-cause-of-data-loss-in-office-365/) Right Image: International Association of Privacy Professional (https://iapp.org/news/a/data-indicates-human-error-prevailing-cause-of-breaches-incidents/)