Qualification Accredited

OCR
Oxford Cambridge and RSA

A LEVEL

*Exemplar Candidate Work*

# COMPUTER SCIENCE

**H446**
For first teaching in 2015

# H446/03 Summer 2017 examination series

# Set C – High Part 1

Version 1

# Contents – Part 1



**Please note: Due to file size limitations this exemplar has been split into two PDF files.**

# Introduction

These exemplar answers have been chosen from the summer 2017 examination series.

OCR is open to a wide variety of approaches and all answers are considered on their merits. These exemplars, therefore, should not be seen as the only way to answer questions but do illustrate how the mark scheme has been applied.

Please always refer to the specification (http://www.ocr.org.uk/Images/170844-specification-accredited-a-level-gce-computer-science-h446.pdf) for full details of the assessment for this qualification. These exemplar answers should also be read in conjunction with the sample assessment materials and the June 2017 Examiners' Report to Centres available on the OCR website http://www.ocr.org.uk/qualifications/.

The question paper, mark scheme and any resource booklet(s) will be available on the OCR website from summer 2018. Until then, they are available on OCR Interchange (school exams officers will have a login for this).

It is important to note that approaches to question setting and marking will remain consistent. At the same time OCR reviews all its qualifications annually and may make small adjustments to improve the performance of its assessments. We will let you know of any substantive changes.

**Please note: Due to file size limitations this exemplar has been split into two PDF files.**

# Exemplar 7 – Set C (High)

**Programming project (non exam assessment)**

Learners will be expected to analyse, design, develop, test, evaluate and document a program written in a suitable programming language.



## H446 (03) A Level Programming Project

This system has been developed to work with live radio servers so for the system to function correctly sensitive data has been used to properly test its functionality. Permission has been given for this data to be included in this document, and for it to be published online, by the owner of the data and managers of the radio station involved and its managing bodies. Any sensitive information, such as server passwords, have been blurred out for security reasons. This has been clearly explained beneath images where this has happened.

# Contents

# Analysis

## The Problem

After working part time at a radio station, it became clear it was hard for our station to find out our listener statistics, I then looked into this further, and after speaking to contacts in a lot of other radio stations, it became clear that many online radio stations up and down the country have the issue of monitoring their audience reach and figures, this is due to the sheer lack of information available about listener statistics. In fact, the only information radio stations can find out is an example screenshot from below:


As you can see, the information is very limited, the servers only report the current listener statistic, and that is about it.

If a computational approach was applied to this situation, a system could be created which periodically logs this information, and can then analyse and display the information back in a more useful and easy to read format. This would save a lot of time, especially due to the fact that some radio stations write down on paper their listener history to keep track of it, with this system it would all be automated, and really easy to see trends in listeners during certain shows enabling easy targeted advertising.

## My Clients/Stakeholders

The main stakeholders for my system would be online radio stations, in specifically [redacted]. I will speak to the station manager at [redacted], and target him as my end user. I will ask him for features and ideas of what he thinks will be useful to include in the system; then I will analyse them and add them to the list of my Key Solution features, this will then allow the application to be appealing and fitting for my end user, and other radio stations up and down the country.

They will use it to monitor their online stream statistics, and run reporting on them.

## Conversation with [redacted] *(Station Manager at [redacted])*

I had a conversation on the Phone with the Station Manager at my local radio station, he gave me some really good information and key features he wanted to be implemented into the system

**What is the problem?**

After speaking to [redacted], he identified the main problem as the lack of reporting available on listener statistics. He made the same conclusion as me above - that we can only find out the current listener amount and that is it. He said it is really limiting, and that having a system to fix this issue and help solve the problem would be really beneficial to the station.

**What are your key features you want out of the system?**

Ian said that in past experience when broadcasters can see their current listener statistics - when the figures drop, so does their on air performance. He said that the system needs to run in the background so that this doesn't happen, but should be able to be opened and looked at by management when needs be.

He also said that there needs to be graphs which are easy to view and change the ranges of to generate reports, this will enable the user to obtain useful information and to see key trends rather than raw data tables.

He also said there needs to be a way to see which shows are most popular, he also suggested if he could have a show timetable on there, so it is easy to analyse and see popular broadcasted shows. He said that the key feature would be seeing which hours in the week are the most popular, and a desirable feature would be to add the show names to that, and group the reports by individual shows.

He also mentioned the fact that he would like a live dashboard overview, so that he can see live what is happening when a presenter is broadcasting, he also said a desirable feature to this would be a user login to protect this information, as it cannot be released to the public, and must be held secure.

He also wanted the ability for a complete history report which can date back to any date. So that all of the past listener history is stored, so that any period of time can be brought up for reporting/analysis.

Finally the last desirable feature would be email notifications which can be setup to send of reports to management or himself if there are any issues with the system, and to automate the monthly listener reports which need to be sent off to various people.

The system in its base form will take statistics from an XML file, and archive them over time. It will then allow for this data to be analysed and different reports to be created, to view trends. This system with a few tweaks could be applied to many other analytics situations, from website traffic counting, to even lap times for cars in a race. For this project though I will make sure to keep my project geared towards a radio station's use only for reporting statistics from an online server.

## Problem Research

The first thing to compare is the system to a radio station which has no listener monitoring whatsoever, this means they would only get the current listener count, and that would be it (as you can see from the prior screenshot), the new proposed system would allow the analysis of this over time, allowing radio stations to see how effective their various shows are, and when best to run targeted adverts.

Another system which already exists is called 'Radio Toolbox' it can be found here: https://www.radiotoolbox.com *[Date accessed: 08/02/2016]*, this system allows for a live server overview, with various listener location graphs (as you can see from the screenshot below):

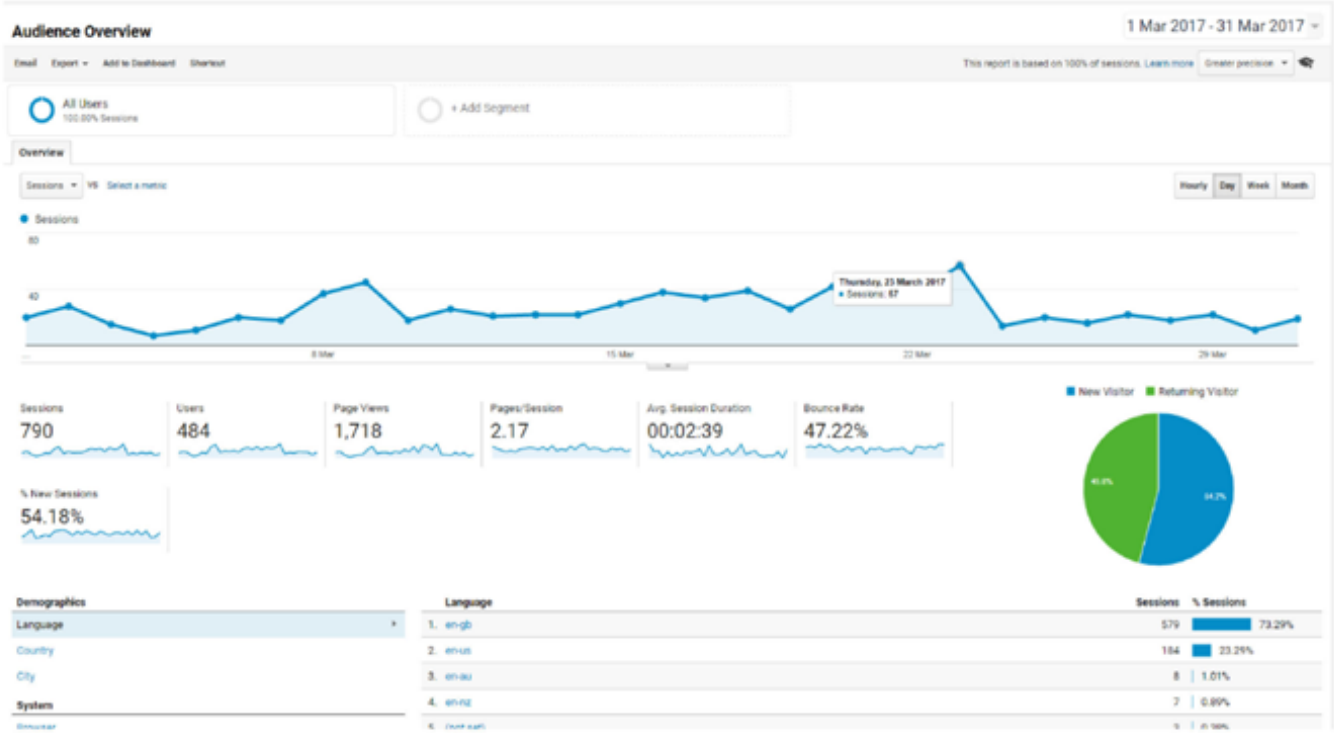

This system has some really handy features, such as email notifications when there are high listener peaks, or when there is a loss in connection, however it is still really basic, and does not support and history reviewing of statistics between certain time periods.

I will take the useful features of email notifications, and the viewing of current listeners on a graph so you can see where they are based. However I will make sure to improve on what this current system lacks, by allowing a view of listeners over a time frame, text and email notifications, and a secure login feature to protect the information.

When it comes to other systems, there are no other system out there which allows this monitoring of online radio web streams. I will use the positives from Radio Toolbox and improve on the negatives of the program. I am also going to look at some other analytics programs, while this are not related to

radio, they allow analysis of other statistics from various sources, I can then use the good and bad points from these, to help aid in my design decisions even more.

Google Analytics is a web based analytics system for websites, it allows you to see visitor traffic in real-time and over a time period to see how popular your website is, based on how many users have visited it. A screenshot of the dashboard overview is below:




Some of the good features of this dashboard is that the data is displayed in an easy to read format on a graph that displays the data point's value when your cursor is hovered over it. This will be really useful to have in my system as it will enable users to have a quick overview of data without having to read through tables. The graphs are also interactive, so you are able to zoom into a certain section, this would be nice to implement into the system, so if there are a lot of data points on the graph you can focus on a certain section in more detail, and have a feature where if you hover over a certain point it will display the points value as a tooltip. The dashboard also has a range selection as a calendar view, I will make sure to include this in my reports so it is easier to select two dates as the user can see a month calendar and the days of each date. I will make sure not to have huge tables of data as they are hard and confusing to read, and where tables are used I will make sure a graph is there to justify the data.

## Key Features of the Solution (System Goals)

*What key features will the application contain, and explain the choices*

After having conversations with the station manager/chief broadcast engineers from two different radio stations, I now have a set of key features which are specific to both radio stations. I will incorporate their specific features, along with the improvements and features I pointed out prior with Radio Toolbox. This will result in an application which will be feasible, and contain all of the required features needed by radio stations.

- **Live reporting of listener figures and current songs**
  This will allow a live overview for stations to see what their current statistics are, this would be used by the chief broadcast engineer or the broadcast quality management team, to see the radio station's current impact and statistics. This will be displayed in a live updating graph overview on a dashboard screen (like the Radio Toolbox example from earlier). The information will be taken from the server live, this is given in an easy to read XML format, as you can see below, this will then be

interpreted and turned into useful information. (Access to the XLM file requires HTTP authentication, so that will have to be added to the process/web-request which h the program will execute)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
  <SHOUTCASTSERVER>
    <CURRENTLISTENERS>418</CURRENTLISTENERS>
    <PEAKLISTENERS>468</PEAKLISTENERS>
    <MAXLISTENERS>500</MAXLISTENERS>
    <UNIQUELISTENERS>415</UNIQUELISTENERS>
    <AVERAGETIME>670</AVERAGETIME>
    <SERVERGENRE>Misc</SERVERGENRE>
    <SERVERURL>http://my_website.com</SERVERURL>
    <SERVERTITLE>Test Server</SERVERTITLE>
    <SONGTITLE>The Current Song</SONGTITLE>
    <STREAMHITS>150</STREAMHITS>
    <STREAMSTATUS>1</STREAMSTATUS>
    <STREAMPATH>/test.mp3</STREAMPATH>
    <BITRATE>128</BITRATE>
    <CONTENT>audio/mp3/wav</CONTENT>
    <VERSION>0.2.0.0.0.0.21.0</VERSION>
  </SHOUTCASTSERVER>
```

- **Email/Text Notifications**
  This will allow for notifications to be sent via email (built into C# using an SMTP server) or Text Notifications (sent using an api called Twillio – more documentation/information on this is available here: https://www.twilio.com) this will allow useful instant updates which can be set by the user. These could vary from Daily Summary reports, to peak listener reports even notifications if the server goes down/offline. This will be a desirable feature, as it is not essential of the functioning of the system.

- **History view available for any statistics**
  This will come in really useful, as it is a feature which stations have been unable to do. Allowing them to see which times during the week their station is most popular, along with which shows have the most listeners. This is really useful for when stations want to have target adverts, so it will allow them to easily show potential advertisers how much listener reach their station has.

- **All statistics will be periodically logged to a database**
  This will enable the ability to view history between any dates. There will be a log table in the database which will contain various fields, this will store the current statistics. Every 10 seconds the system will check the statistics and see if they have changed, if so it will write them as a new record into the database. If not it will wait another 10 seconds. This will drastically reduce the amount of data needing to be stored, however it will still offer the in-depth reporting that stations will be expecting.

- **Login system to secure all of the data and history**

This will secure all of the data, and prevent somebody who is visiting the station (or even another member of staff) from accessing information that they should not have access to, and also from changing any settings which would affect the operation of the program.

Passwords will be encrypted, and then stored in the database along with other user's information. This will also include an access system, to have various user levels of who can edit information, e.g. a 'normal user' who can manage and view statistics, and a 'admin' user who can change various settings and add/change users. This will be a desirable feature, as it is not essential of the functioning of the system.

- **Different layouts/ways of viewing data (graphs or raw table reports)**

This will enable users to customise their dashboard so they can have an overview which suits them. For example you could have various graphs of the most popular days, or peaks in listener figures. This will help as stations will have a customizability about what they see when on their dashboard, allowing them to access the information and statistics that they need as efficiently as possible.

All graphs will also have the ability to export the raw data along with printing off the graph, this will make it much easier when radio stations need to report their listener figures to advertisers or management.

- **System can run in the background on a computer, without interfering with anything**

This will be really useful, as the application will be able to minimise to the notifications taskbar, this is due to the fact that radio stations will need to use the computers for a lot of other work/tasks without the interfering with anything or getting in the way, until they need to see statistics.

## Limitations

*Any limitations which I could encounter when creating the solution/program*

After speaking to my client ( the Station Manager at ), it was clear that there were no other problems or issues then the ones identified above. This means that my system will only be focusing on the listener statistics, and monitoring reports. So will be made to meet the below Success Criteria only, and will not be expanding to reach anything else. This will also prevent the solution from expanding too rapidly, and snowballing into a project which will be unable to be completed in the allocated timeframe.

## Requirements for the Solution

*Detailed any hardware or software requirements for the solution*

The computer which connects to the server must have a stable internet connection (at least 1mbps download and 0.1mbps upload) and must have the required port your streaming server runs on unblocked and be allowed through a firewall.

The computer must also meet the following specifications:

Minimum processor rate = 1 GHz

Minimum ram = 512 MB

Minimum disk space = 10GB

## Success Criteria

*Detailed and justified success criteria for the proposed solution*

The following points are the success criteria which will make the system meet the goals set by my client, they are split into essential and desirable, this is due to the fact it is better to include all of the essential features, and then the desirable ones, so that the system firstly functions than can be improved with additional features from that point; so that if I do not have enough time to include all of the features, I will have at least covered the essential ones for the operation of the application.

Essential:

- **History view available for any statistics**

You will be able to view all statistics from when the system was first installed, to the current ones coming in.

- **All statistics will be periodically logged to a database**

The database will have a structure enabling data to be stored into it about all of the past listener history, while being as efficient with data storage as possible.

- **Different layouts/ways of viewing data (graphs or raw table reports)**

You will be able to pick from a selection of different reports you can run to obtain and view data from the database, these will be:

- **System can run in the background on a computer, without interfering with anything**

The system will be minimised to the toolbar, and will be running in the background. This means it will not be in the way of anything, and will not interfere with normal operation of the computer, when you double click on the icon the system will then open.

Desirable:

- **Login system to secure all of the data and history**

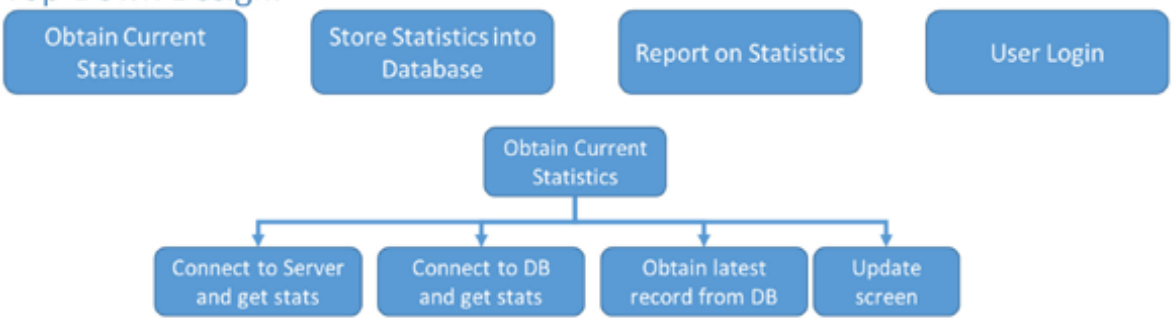
There will be a login screen when the system is opened, and users will be able to change passwords and add/manage/remove user accounts

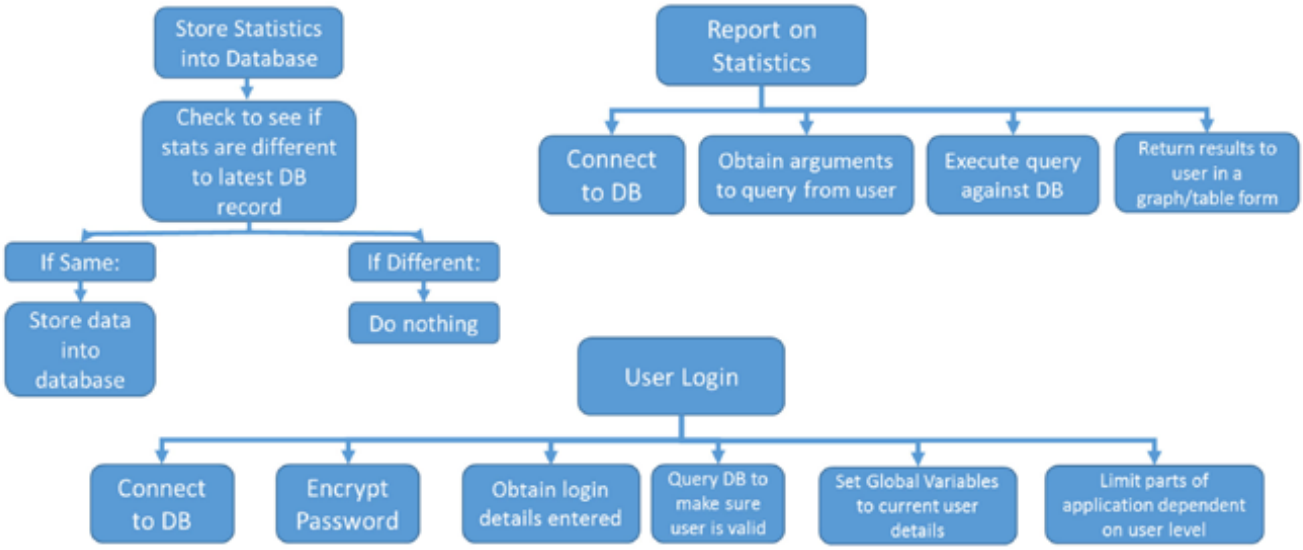
- **Email/Text Notifications**

The user will be able to setup custom email and/or text alerts to any mobile number or email address dependent on certain conditions which can be defined in the settings part of the application.

## Design

### Top-Down Design:






## Database Design:

- History view available for any statistics

The statistics which are obtained from the server are as follows: Current Listener Count and Current Song Name. They are returned in an XML format, and example of this is below:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
  <SHOUTCASTSERVER>
    <CURRENTLISTENERS>418</CURRENTLISTENERS>
    <UNIQUELISTENERS>415</UNIQUELISTENERS>
    <SONGTITLE>The Current Song</SONGTITLE>
  </SHOUTCASTSERVER>
```

This can then be easily parsed by C# in an xml format to obtain the required data. This will then have to be stored into a database. This will need to be designed to enable to storage of a large history of the data, along with it being easily queryable so custom reports can be produced.

- All statistics will be periodically logged to a database

The database will have a table which will store this information, it will be of the following structure:

| Name | PK/FK | Type | Null? |
|---|---|---|---|
| Log_ID | PK | Int (AutoIncrement) | No |
| Log_DateTime | | DateTime | No |
| Log_ListenerCount | | Int (10) | No |
| Log_CurrentSong | | String (MAX) | No |
| Log_ShowID | FK | Int | Yes |

This will enable for the data to be easily queryable. The system will check for information which has changed (i.e. if the current song has changed, or the listener count) and if it has it will log a new record to the database with the current date and time, this will conserve on space while still offering a fully searchable history report for any time frame.

- Login system to secure all of the data and history

This will be another table in the database, users will have multiple access levels, these will be stored as a integer and will be the following: 1 = Normal User, 2 = Management User and 3 = Admin User. The normal user will only have limited access to the dashboard overview, they can view data but will be

unable to generate reports, the Management user will have access to all of the reports and to create/manage other users, along with editing shows and times of them, the Admin Users will have access to all of what Management Users can along with access to application settings. Finally there will also be the last time the user has logged into the system. This will be stored in the following database table:

| Name | PK/FK | Type | Null? |
|---|---|---|---|
| User_ID | PK | Int (AutoIncrement) | No |
| User_Fullname | | String (MAX) | No |
| User_loginname | | String (MAX) | No |
| User_email | | String (MAX) | No |
| User_password | | String (MAX) | No |
| User_accesslevel | | Int(5) | No |
| User_lastlogin | | DateTime | No |

- Individual shows can be queried and compared

These shows will need to be stored in a timetable, this will need its own table to hold this information, which can then be referenced by the reporting functionality when they are generated. The shows will have a unique ID which can be used to reference them, a name so you can reference the show, the presenter who presents it, the day it is on, and the start and end time of each hour. This will allow the system to identify when a show is on. However, shows can change time and day so there needs to be a start and end date of the show, this allows for historical reporting back to when the show was running at a different time. Finally there is an active Boolean variable, this is set to false if the show is not currently broadcasting, and true if the show is current. This allows for easy searching of the system of active shows easily.

| Name | PK/FK | Type | Null? |
|---|---|---|---|
| Show_ID | PK | Int (AutoIncrement) | No |
| Show_Name | | String (MAX) | No |
| Show_Presenter | | String (MAX) | No |
| Show_Day | | String (MAX) | No |
| Show_Starttime | | Int | No |
| Show_Endtime | | Int | No |
| Show_StartDate | | DateTime | No |
| Show_EndDate | | DateTime | No |
| Show_Active | | Boolean | No |

As you can imagine, the amount of different reports which could be made are endless, this means that as I do not have enough time to create a bespoke reporting system (where the user can create custom reports) I will make 5 commonly used reports, and ask the user to enter the input data. I can then create a custom output which will best fit the type of data they are after.

The Reports will be the following:

**Show History Report**

This will take name of the show, and will produce a detailed report for every time the show has been on. It will give peak listeners, and average listeners. It will then allow you to drill down into each section of the show and view an hourly report.

**Song History Report**

This will allow you to select a certain song name, and then report on when it was played, along with the current listener statistics, along with what show played it.

**Popular Hours**

This will allow you to see over a custom date range the overview of listeners for each hour in the day, which can allow for easy comparison.

**Monthly Peak listeners report**

This will allow you to see the average and peak listener count for each month over the past year (since January), you can also look back over previous years

I was going to add a day-part overview report here, but the project would become too complex. As it is not an essential feature, and only a desirable one, I will omit it from this project, but may add it at a later date.

## Form Design:

First I will need to design the login screen, this will need to contain a place for my user to enter their username and password along with a login button. It will also need to contain a warning which states: "Access to this system is restricted, please enter your credentials below to gain access to the system." This will also have to have the radio station branding on it (as you can see from figure 1 and 2 below):

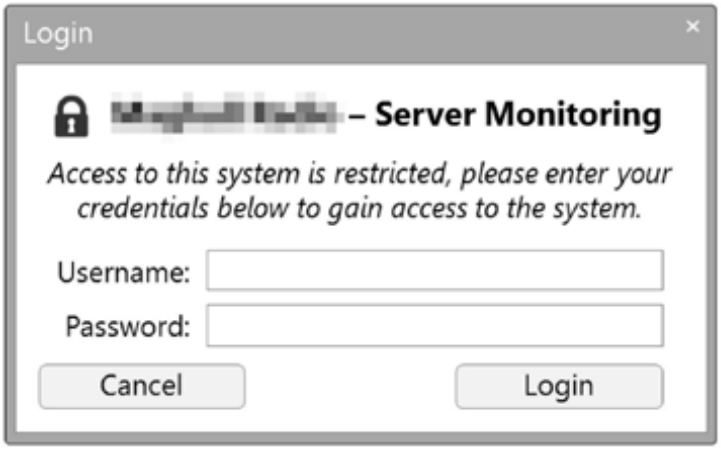



FIGURE 1 LOGIN FORM DESIGN WITHOUT BRANDING

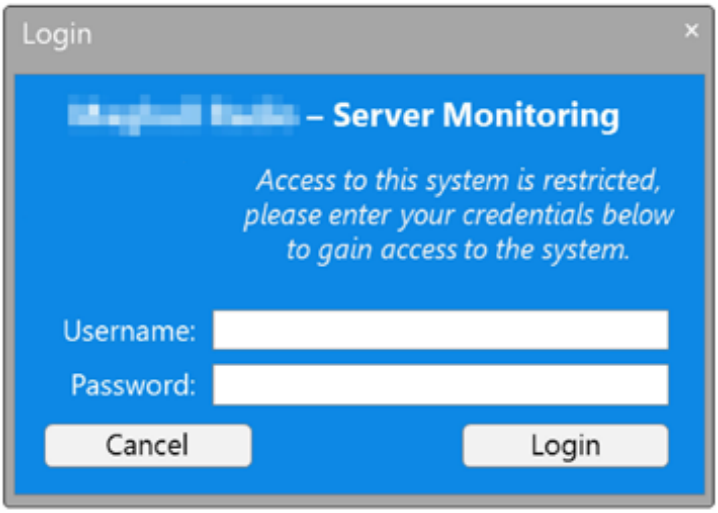
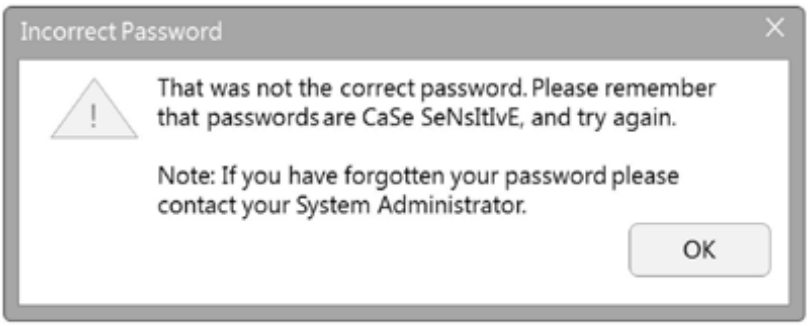



FIGURE 2 LOGIN FORM DESIGN WITH BRANDING

If the user enters an incorrect username and password, the following message box will be displayed:



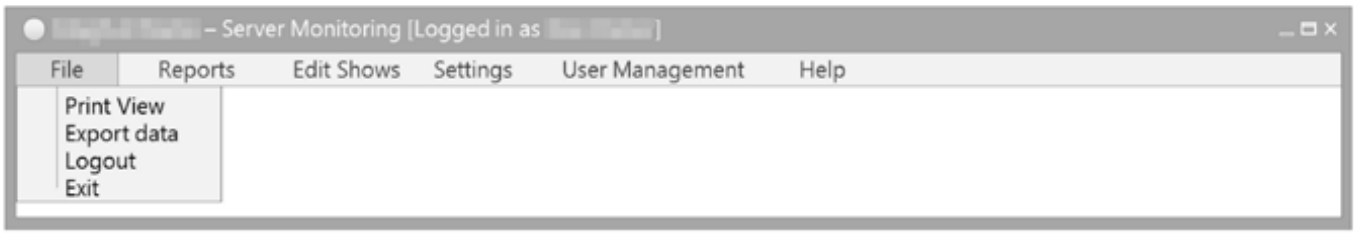

The custom branded form looks a lot nicer, and will look a lot more professional than having a white background on the screen. I will use this in development.

- Dashboard Screen

This screen will need to have an overview of what is currently happening this will contain the past 10 songs, the current listener statistics, the day's show schedule, and a brief history overview of the last 60 minutes and the last week of listener history.

The colours will match the consistent style of blue, light blue and orange for highlighting a field or item. The graphs will automatically update every minute along with the live statistics, and the currently playing song and show will be highlighted in orange.

This form will also have a drop down menu on it, these options can then open up into other forms top change various settings of the application, or run various reports.




This will have the following menu structure:

File (Dropdown, access to all users)

Print View (Prints off the current view – access to all users)

Export Data (Exports the current data as a CSV file – access to all users)

Logout (Logs out the current user – access to all users)

Exit (Exits the application permanently and stops all monitoring – access to level 3 'Admins' only)

Reports (Opens up the reporting form – access to levels 2 'management' and 3 'admin' only)

Edit Shows (Opens up the form to edit shows – access to levels 2 'management' and 3 'admin' only)

Settings (Opens up the Settings form – access to level 3 'admin' only)

User Management (Opens up the User Management form – access to level 3 'admin' only)

Help (Opens up a user help and troubleshooting guide – access to all users)

Each menu option is limited so only certain users (depending on the access level they have set) can access that feature. This is to protect the data, and the operation of the system. For example there would be no need for a manager to access the database settings, as that could affect the operation of the entire system, however the admin would need access to change settings. This can add a much-needed layer of security to the system. If a user does not have access to a certain feature, then that specific menu item will become hidden, so the user therefore cannot see or access it.

- Settings Screen

The administrators of the system will also need a facility to change the various settings of the application. The settings needed to be changed are listed below:

Database SQL Settings
- Server Host
- Server Port
- Server Username
- Server Password
- Database

ShoutCAST Settings
- Server Host
- Server Port
- Stream ID
- Server Username
- Server Password

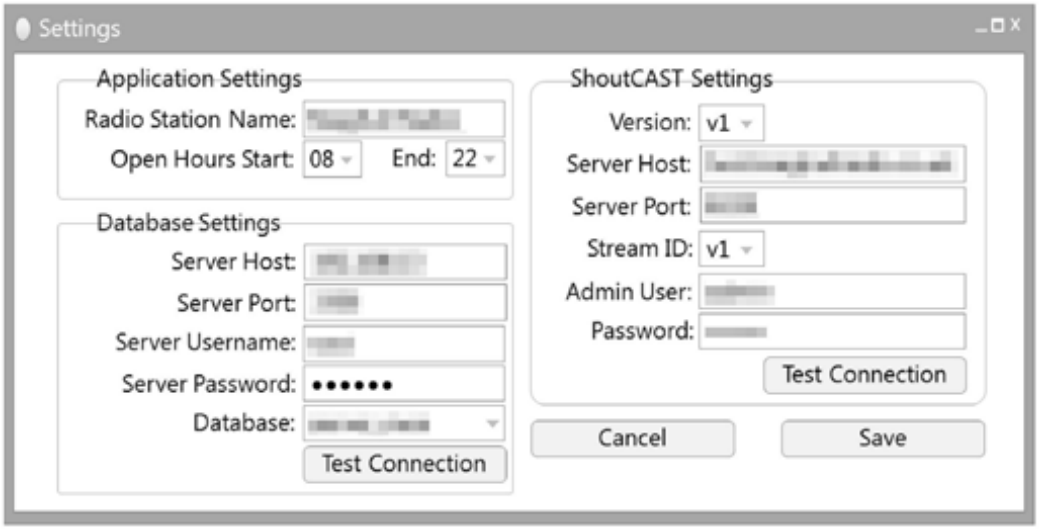
ShoutCAST Version (v1, v2)

Application Settings

Radio Station Name

Operating Times (Start Hour and End Hour)

I have then taken this and created a form design which contains all of the settings in a logical layout so they can be easily changed:

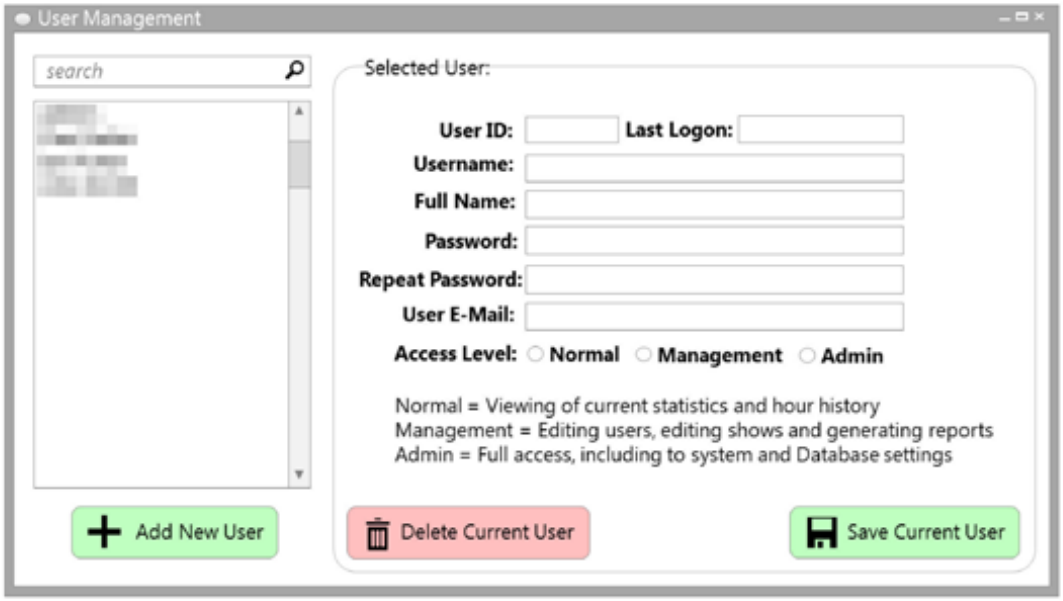



After the information has been filled in the user has the option of testing both of the 'database' and 'server' settings. This will be useful as you can double check the settings are correct before saving any information into the system. However as a back-up, when the save button is pressed, both of the connections will be tested anyway to make sure they are both working.

If the database has been created for the first time; and does not contain the required tables necessary for the application to function, they will be automatically created by executing an SQL command against the database to define all of the tables and structures for the records. The layout of this was detailed above under the database design section.

- User management screen

This screen will firstly have a searchable list of users, when you click on a user you can see their settings and information about them. This can allow you to delete users, or to change various settings to do with them. The User ID and Last Logon fields are read-only, and are just for reference. It will also contain their Username, their full name, their password, email and access level. This can easily be changed by entering new information for the user, then hitting 'Save'.

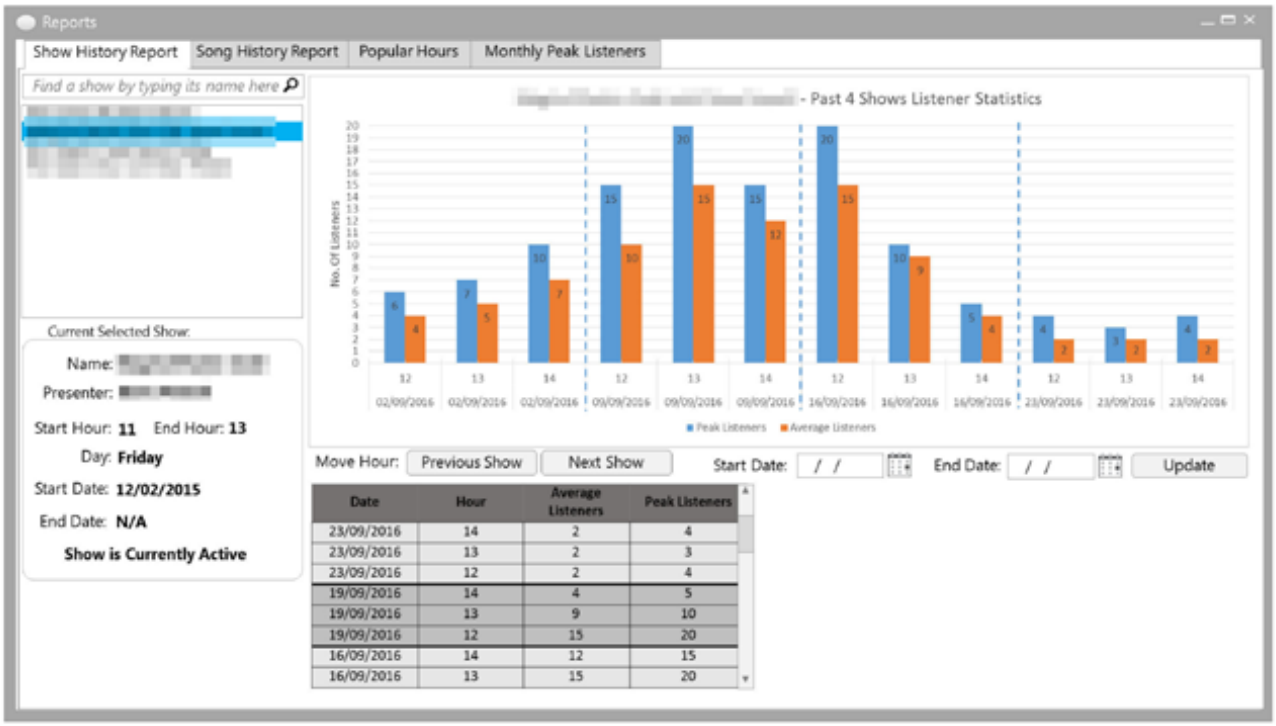



If you want to change the User's password you can start editing the current password field, then the 'retype password field' will become active, allowing you to change it. It will have an email for different notifications if enabled. And for easier user management. Finally it will have the 3 different access levels, which can be easily changed by pressing the appropriate checkbox. The access levels for each setting is detailed above.

- Reporting Screen

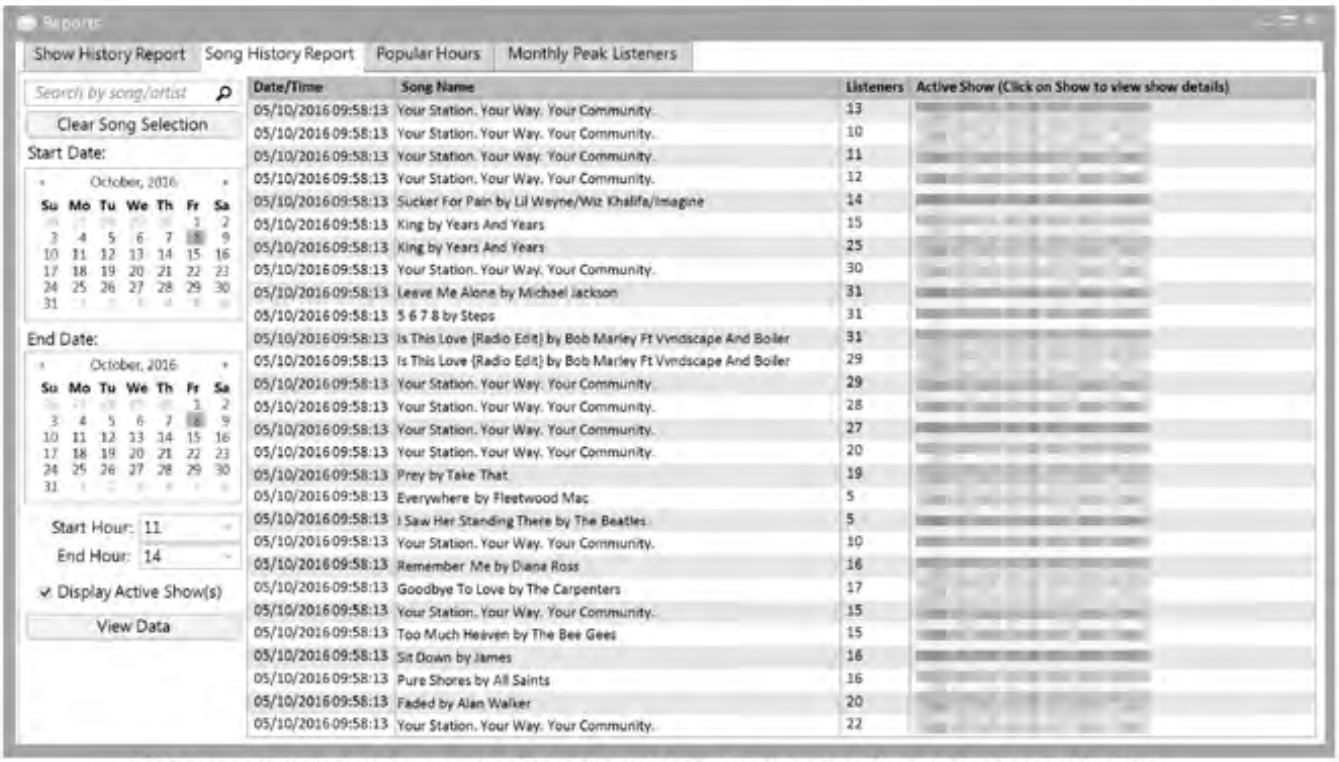
As mentioned above, the user will have a selection from 4 reports. To make this as easy to understand, I will have a tab-group as the main layout, allowing then for custom controls to be developed for each report. The tabs will say "Show History Report", "Song History Report", "Popular Hours" and "Monthly Peak Listeners". This will then allow the user to select one, enter the input data needed, and then generate the report dependent on the tab selected:

**Show History Report**



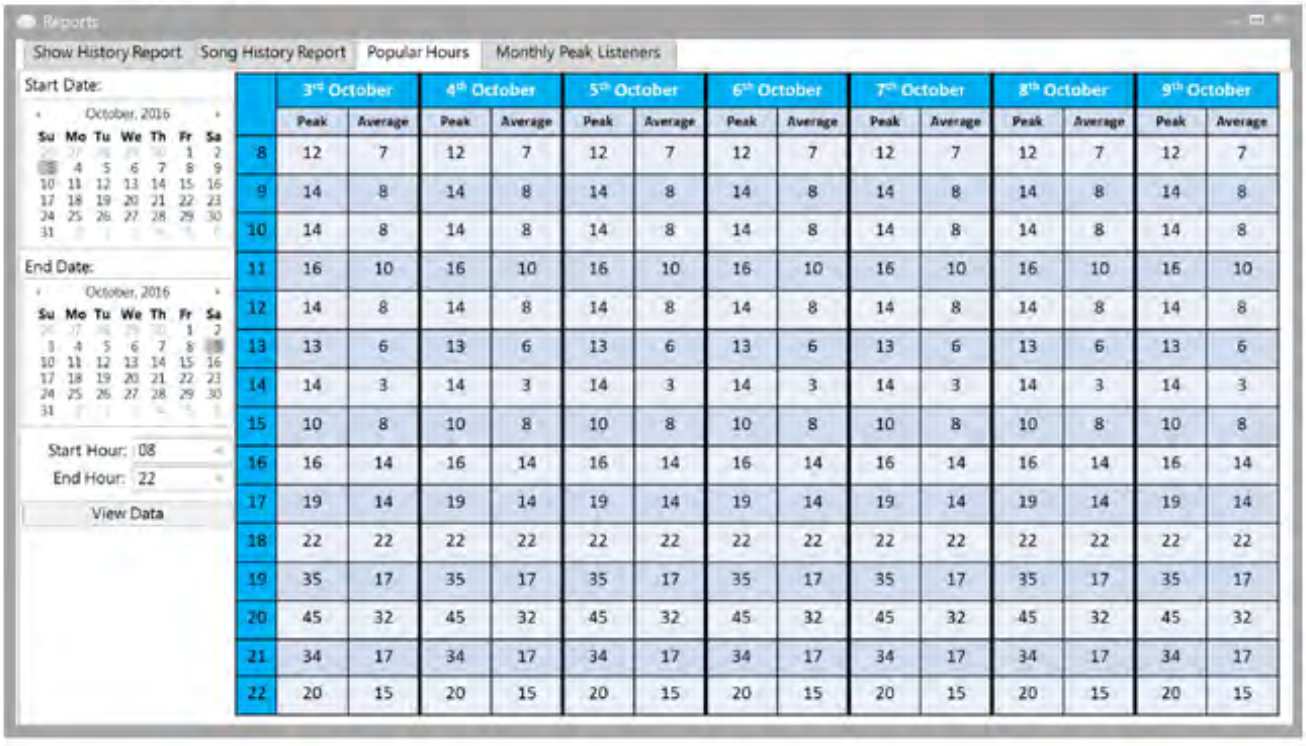

This screen will have a list of shows on the top left, which are searchable by entering the search criteria in the box above that will then filter the list showing the results which contain the criteria you entered. When you click on a show in the list box, it will then become the currently selected show. This will then update the information about the show (contained in the box at the bottom left for user reference). It will then obtain the listener statistics from the last couple of shows. This will contain the peak listener count for each hour, and the average listener count over that hour. There is also a table of raw data beneath this to make this too see raw data. You also have an option to define a date range of the shows that are displayed, by adjusting the values in 'Start Date' and 'End Date'. There will be validation in here to prevent the start date being set later than the end date. Once the update button has been pressed the graph and table will then update.

## Song History Report




This screen will display the raw data from the database between two time periods set by the two calendars and hour drop-down boxes.

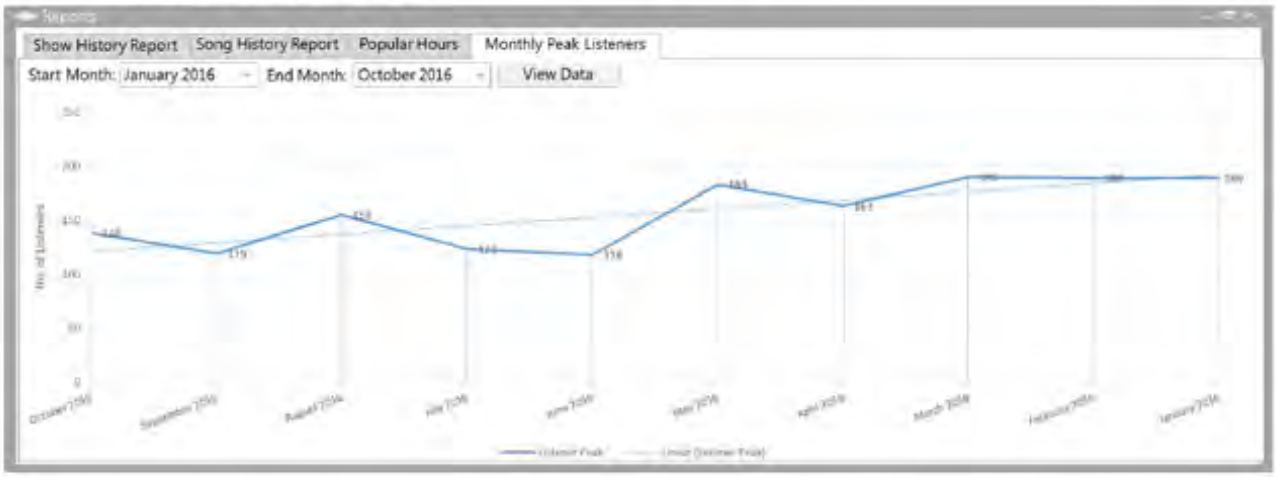
## Popular Hours



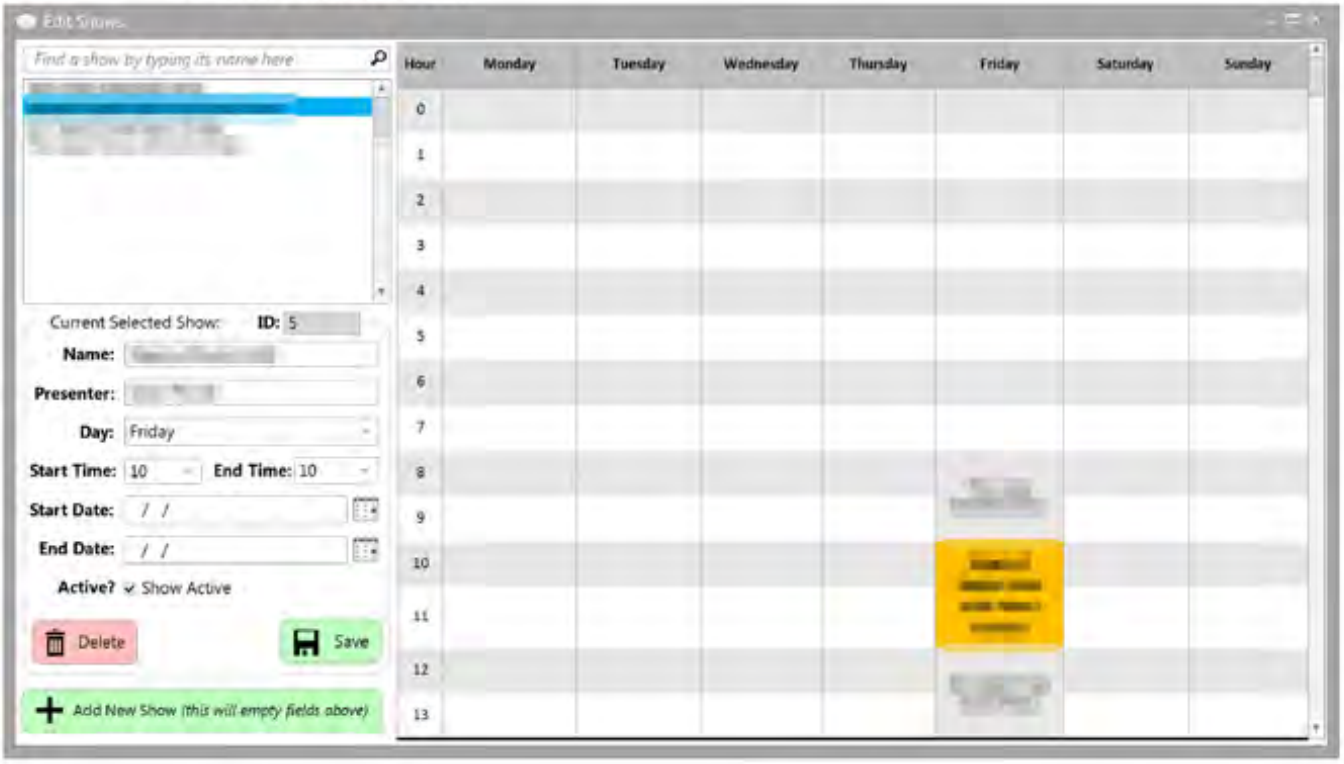

| | 3rd October | | 4th October | | 5th October | | 6th October | | 7th October | | 8th October | | 9th October | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Peak | Average | Peak | Average | Peak | Average | Peak | Average | Peak | Average | Peak | Average | Peak | Average |
| 8 | 12 | 7 | 12 | 7 | 12 | 7 | 12 | 7 | 12 | 7 | 12 | 7 | 12 | 7 |
| 9 | 14 | 8 | 14 | 8 | 14 | 8 | 14 | 8 | 14 | 8 | 14 | 8 | 14 | 8 |
| 10 | 14 | 8 | 14 | 8 | 14 | 8 | 14 | 8 | 14 | 8 | 14 | 8 | 14 | 8 |
| 11 | 16 | 10 | 16 | 10 | 16 | 10 | 16 | 10 | 16 | 10 | 16 | 10 | 16 | 10 |
| 12 | 14 | 8 | 14 | 8 | 14 | 8 | 14 | 8 | 14 | 8 | 14 | 8 | 14 | 8 |
| 13 | 13 | 6 | 13 | 6 | 13 | 6 | 13 | 6 | 13 | 6 | 13 | 6 | 13 | 6 |
| 14 | 14 | 3 | 14 | 3 | 14 | 3 | 14 | 3 | 14 | 3 | 14 | 3 | 14 | 3 |
| 15 | 10 | 8 | 10 | 8 | 10 | 8 | 10 | 8 | 10 | 8 | 10 | 8 | 10 | 8 |
| 16 | 16 | 14 | 16 | 14 | 16 | 14 | 16 | 14 | 16 | 14 | 16 | 14 | 16 | 14 |
| 17 | 19 | 14 | 19 | 14 | 19 | 14 | 19 | 14 | 19 | 14 | 19 | 14 | 19 | 14 |
| 18 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 |
| 19 | 35 | 17 | 35 | 17 | 35 | 17 | 35 | 17 | 35 | 17 | 35 | 17 | 35 | 17 |
| 20 | 45 | 32 | 45 | 32 | 45 | 32 | 45 | 32 | 45 | 32 | 45 | 32 | 45 | 32 |
| 21 | 34 | 17 | 34 | 17 | 34 | 17 | 34 | 17 | 34 | 17 | 34 | 17 | 34 | 17 |
| 22 | 20 | 15 | 20 | 15 | 20 | 15 | 20 | 15 | 20 | 15 | 20 | 15 | 20 | 15 |

H446 (03) A Level Programming Project 19

**Monthly Peak listeners report**




- Show timetable screen



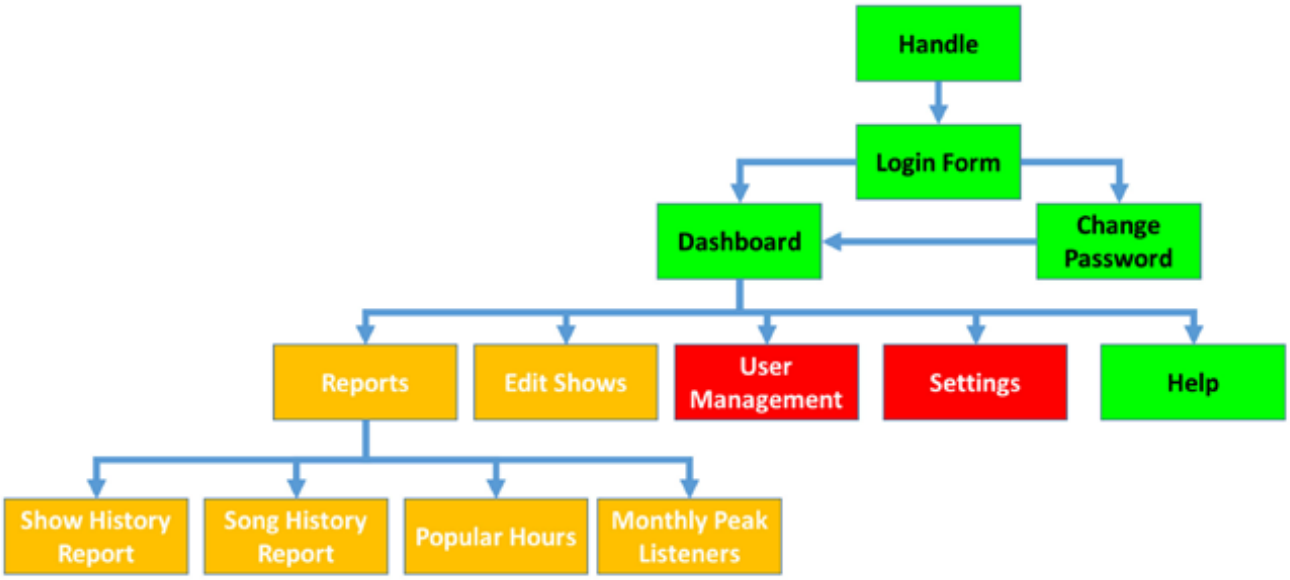

**Change Password**

This form will be very simple, it will display two text boxes, and will enable the user to change their password.

It will validate the password, and make sure the password they have typed and the repeated one match. It will then update the database for the user with their new password. It will then load the dashboard as if the user had logged in normally.

## Algorithms for each form/report

I need to create a layout of how all of the forms will relate to each other. This will be done via a top-down design:



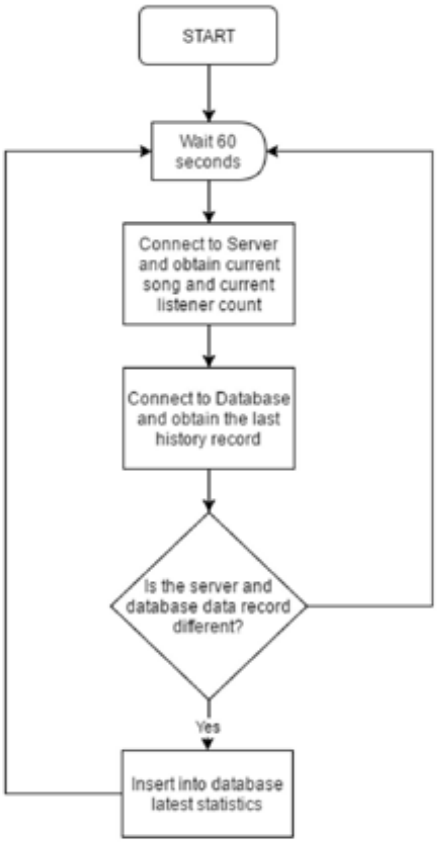

The colours depend on the user access level. Green means all users can access those forms, orange is only for management users, and red is only for administrator users. This makes it much easier to see what forms connect to each other, and will help with the code and algorithm section.

**Code/Algorithms:**

First thing to code is the handle form. This form will run in the background of the PC, but will have a notify icon icon displayed in the taskbar. This will allow for the application to constantly check the server and update the required statistics into the database without being obtrusive to the user.

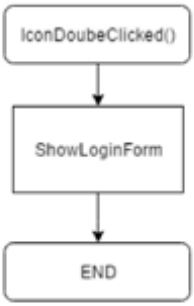
This form needs to do the following:

Obtain Current statistics from server, check if they have changed, and store into database



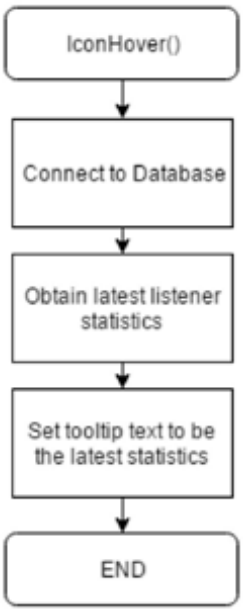

```
1  INT currentServerListeners
2  STRING currentServerSong
3  INT currentDatabaseListeners
4  STRING currentDatabaseSong
5
6  :START
7  WAIT 60 SECONDS
8
9  Server.Connect()
10 currentServerListeners = Server.CurrentListenerCount()
11 currentServerSong = Server.CurrentSong()
12 Server.CloseConnection()
13
14 Database.Connect()
15 currentDatabaseListeners = Database.LatestRecord.Listeners()
16 currentDatabaseSong = Database.LatestRecord.CurrentSong()
17
18 IF currentServerListeners != currentDatabaseListeners OR
   currentServerSong != currentDatabaseSong THEN
19     Database.InsertRecord(Time.Now(), currentServerListeners,
       currentServerSong)
20 END IF
21
22 GOTO START
```

When the icon is double-clicked on, open the login form for access to the system




```
1  FUNCTION iconDoubleClicked()
2      Show LoginForm
3  END FUNCTION
```

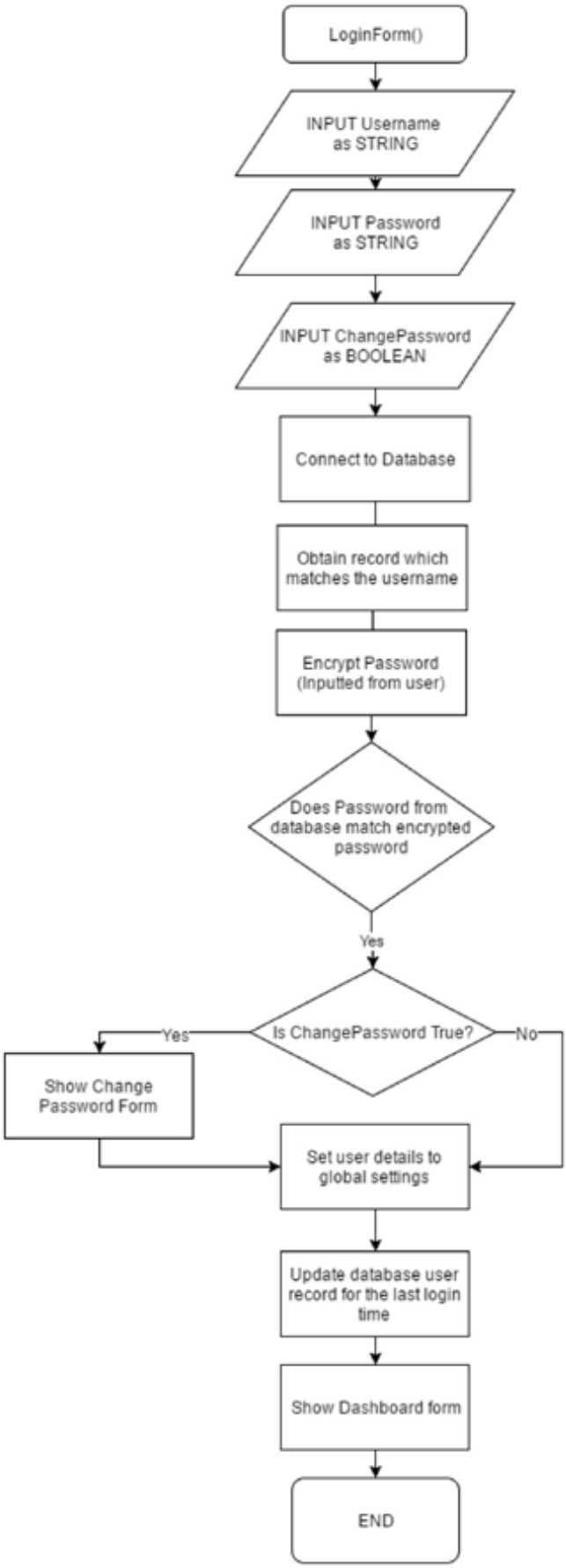
When the icon is hovered over, display the most recent statistics and time of entry




```
1  FUNCTION iconHover()
2      INT DatabaseListeners
3      STRING DatabaseCurrentSong
4      DATE DatabaseTime
5
6      Database.Connect()
7      DatabaseListeners = Database.LatestRecord.Listeners()
8      DatabaseCurrentSong =
       Database.LatestRecord.CurrentSong()
9      DatabaseTime = Database.LatestRecord.Time()
10
11     tooltip.Text = DatabaseTime + " | " +
       DatabaseListeners + " | " + DatabaseCurrentSong
12
13     Database.CloseConnection()
14 END FUNCTION
```

Before we go any further, we need a place where we can store settings across multiple forms. I will document this further in the development section, but for now I will presume any settings I create and reference in this pseudo code section, will be automatically generated. I will explain and talk through the process of this when in the development section, but for simplicity I will leave it out for now.

The next form to code is the Login form. This will handle access to the system. It will need to take a username and password from the user, connect to the database, then authorise the user, then set the user's settings to the global settings so all the forms can read that information. Then it will open the dashboard overview form. There will also be an option for the user to change their password, this will open up another form. Which after the password change process is complete, will take them back to the main screen.

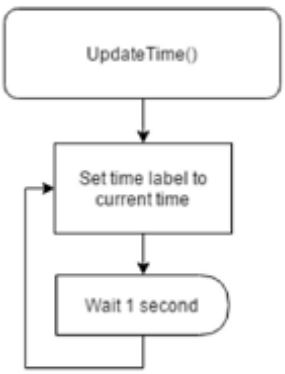

LoginForm()
INPUT Username as STRING
INPUT Password as STRING
INPUT ChangePassword as BOOLEAN
Connect to Database
Obtain record which matches the username
Encrypt Password (Inputted from user)
Does Password from database match encrypted password
Yes
Is ChangePassword True?
Yes
No
Show Change Password Form
Set user details to global settings
Update database user record for the last login time
Show Dashboard form
END

```
5  //Get inputs from user
6  Username = INPUT
7  Password = INPUT
8  ChangePassword = INPUT
9
10 //Connect to database
11 Database.Connect()
12 Database.RunQuery("SELECT * FROM USERS WHERE USERNAME ='" + Username + "'")
13
14 //Encrypt User's password
15 Password = Encrypt(Password)
16
17 //Check password against database password
18 IF Password = Database.GetValue("Password") THEN
19     IF ChangePassword = TRUE THEN
20         ChangePasswordForm.Open()
21     END IF
22
23     Settings.Username = Username
24     Settings.FullName = Database.GetValue("FullName")
25     Settings.AccessLevel = Database.GetValue("AccessLevel")
26
27     Database.SetValue("LoginTime") = Time.Now()
28
29     DashboardForm.Open()
30
31 ELSE
32     OUTPUT("Username or password is incorrect. Please try again.")
33 END IF
```

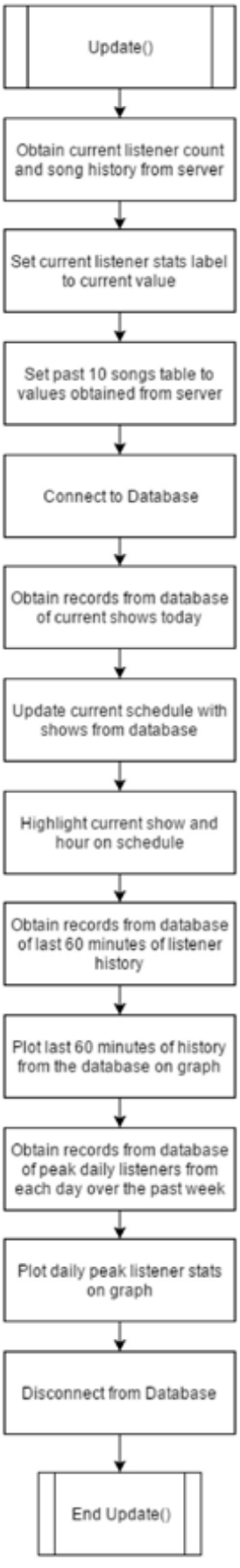
The next form is the Dashboard overview form. This will need to display the current schedule; current listener statistics; current time; past 10 songs; a graph of the last week of listener history; and a graph of the last 60 minutes of listener history. This will be updated every 60 seconds.

There will be two separate threads running, one will loop every second and will update the current time on the dashboard. The other will loop every 60 seconds and update the current schedule, current listener statistics; past 10 songs; the week listener history graph and the 60 minutes of listener history graph.


```
1  :START
2  WAIT 60 SECONDS
3
4  UpdateDashbaord()
5
6  GOTO START
```




```
1  :START
2  WAIT 1 SECOND
3
4  timeLabel = Time.Now()
5
6  GOTO START
```



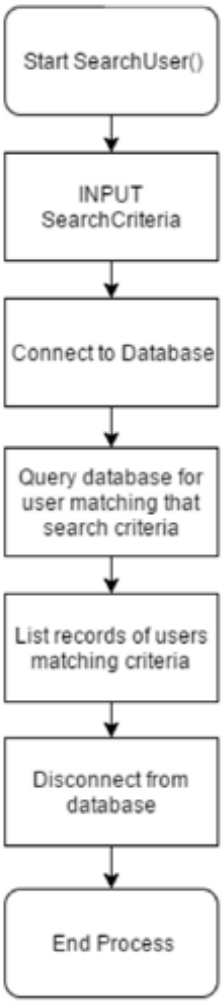

```
INT currentListeners
STRING ARRAY[10] songHistory
INT currentHour = Time.Now(Hour)

FUNCTION UpdateDashboard()

//Obtain current listener count and song history from Server
Server.Connect()
currentListeners = Server.CurrentListenerCount()
songHistory = Server.SongHistory().ToArray()
Server.CloseConnection()

//Set current listener count to label
currentListenersLabel = currentListeners
currentlySongLabel = songHistory[0]

//Loop through past 10 played songs and add to label
INT i = 0
FOREACH STRING song in SongHistory DO
    //This will cycle through each of the 10 labels and set it's text value to the
current song
    FindControl("songHistoryLabel" + i).Text = song
    i = i + 1
END FOREACH

//Get records from today's schedule
Database.Connect()
Database.RunQuery("SELECT * FROM `shows` WHERE `Show_Day` = '" + currentDay
+ "' ORDER BY `Show_Starttime` ASC")

//Remove all shows currently on schedule
ShowSchedulePanel.ClearAll()

//Loop through all shows in today's schedule
FOREACH RECORD show in Database.Records DO
    LABEL showLabel = new LABEL
    showLabel.Name = "showHour" + show['Show_Starttime']
    showLabel.Text = show['Show_Name'] + " with " + show['Show_Presenter']
    showLabel.RowSpan = show['Show_Endtime'].AsInteger() -
show['Show_Starttime'].AsInteger()
    showLabel.RowIndex = show['Show_Starttime'].AsInteger()
    showLabel.Parent = "ShowSchedulePanel"

// If the show is currently active, highlight it with the colour orange
    IF (currentHour >= show['Show_Starttime']) AND (currentHour <
show['Show_Endtime']) THEN
            showLabel.BackgroundColour = Orange
    END IF

END FOREACH
```

```
//Get records from last 60 minutes
STRING CurrentDate = Date.Now()
STRING CurrentTime = Time.Now()
STRING HourBeforeTime = Time.Now() - 3600 //Take an hour off the time

Database.RunQuery("SELECT * FROM log WHERE Log_DateTime between '" + CurrentDate + " " +
HourBeforeTime + "' and '" + CurrentDate + " " + CurrentTime + "'")

60MinuteGraph.ClearAll()

//Plot records on graph
FOREACH RECORD log in Database.Records DO
        STRING X = log['Log_DateTime']
        INT Y = log['Log_ListenerCount']

        60MinuteGraph.AddPoint(X,Y)

END FOREACH

//Get records from last week with peak listeners (filter by date, sort by peak, take highest record)
STRING WeekBeforeDate = Date.Now() - 7        //Take a week off the date
Database.RunQuery("SELECT DATE(Log_DateTime) AS Log_DateTime, MAX(Log_ListenerCount) AS
Log_ListenerCount FROM log WHERE Log_DateTime between '" + WeekBeforeDate + "' and '" +
CurrentDate + "' GROUP BY log.Log_DateTime;")

WeekGraph.ClearAll()

//Plot records on graph
FOREACH RECORD day in Database.Records DO
        STRING X = day['Log_DateTime']
        INT Y = day['Log_ListenerCount']

        WeekGraph.AddPoint(X,Y)

END FOREACH

//Disconnect from Database
Database.CloseConnection()

END FUNCTION
```

**User Management**

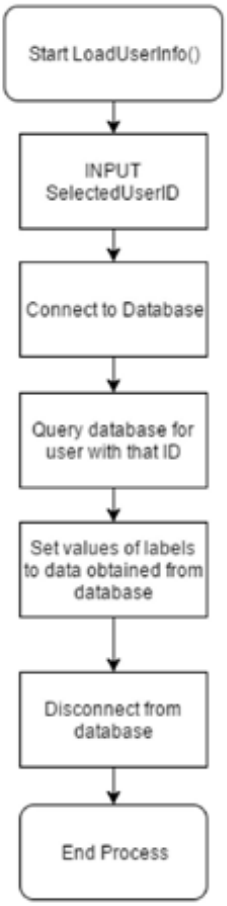
Now I need to create the user management form. This needs to contain a list of all active users, a way to search through them, when a user is highlighted it needs to retrieve their data and display it on the form. Then if the save button is pressed, it needs to update that user with the changed information. There also needs to be a way to add a new user to the database, and an option to delete the currently highlighted user.

This will be split into the following sections. Searching through users, loading a user's information from the database, updating a user, deleting a user, and adding a new user.




```
1  STRING SearchQuery
2
3  // Get search query from user
4  SearchQuery = INPUT
5
6  // Connect to database,
7  // and run query to return a list of users containing the query
8  Database.Connect()
9  Database.RunQuery("SELECT * FROM `users` WHERE `User_ID` LIKE '" +
   SearchQuery + "'")
10
11 // Cycle through each record returned, and add it to a list box for the user
12 // to select from
13 FOREACH RECORD user in Database.Records DO
14     usersList.Add(user['User_Fullname'] + " (" + user['User_loginname'] + ")")
15     usersList.TagLastItem(user['User_ID'])
16 END FOREACH
17
18 //Disconnect from Database
19 Database.CloseConnection()
```

**Loading a user's information from the database**




```
FUNCTION LoadUserInfo()

INT UserID

// Get search query from user
UserID = usersList.SelectedItem.Tag

// Connect to database,
// and run query to return a list of users containing the query
Database.Connect()
Database.RunQuery("SELECT * FROM `users` WHERE `User_ID` = '" + UserID + "'")

// Take the data obtained, and set it to the labels on the form
user = Database.SelectedRecord()

txtUserID.Text = user['User_ID']
txtLastLogon.Text = user['User_lastlogin']
txtUsername.Text = user['User_loginname']
txtFullName.Text = user['User_Fullname']
txtEmail.Text = user['User_email']
txtPassword.Text = ""
txtRepeatPassword.Text = ""

//Check user access level and check the correct radio box
IF user['User_accesslevel'] = 1 THEN
        checkBoxNormal.Checked = true
        checkBoxManagement.Checked = false
        checkBoxAdmin.Checked = false

ELSE IF user['User_accesslevel'] = 2 THEN
        checkBoxNormal.Checked = false
        checkBoxManagement.Checked = true
        checkBoxAdmin.Checked = false

ELSE IF user['User_accesslevel'] = 3 THEN
        checkBoxNormal.Checked = false
        checkBoxManagement.Checked = false
        checkBoxAdmin.Checked = true

ELSE
        checkBoxNormal.Checked = false
        checkBoxManagement.Checked = false
        checkBoxAdmin.Checked = false

END IF


//Disconnect from Database
Database.CloseConnection()

END FUNCTION
```

**Updating a user**


```
FUNCTION UpdateUserInfo()

// Determine user access level
int accessLevel = 1

IF checkBoxNormal.Checked = true THEN
    accessLevel = 1
ELSE IF checkBoxManagement.Checked = true THEN
    accessLevel = 2
ELSE IF checkBoxAdmin.Checked = true THEN
    accessLevel = 3
END IF

bool changePassword = false

// Determine if user has changed password
IF txtRepeatPassword.Text != "" THEN
    IF txtRepeatPassword.Text = txtPassword.Text THEN

            changePassword = true

    ELSE
            Output "The new password you have entered has not
been retyped correctly"
            BREAK
    END IF
END IF

// Connect to database,
// and run query to update user's data
Database.Connect()

IF changePassword = true THEN
    Database.RunQuery($"UPDATE `users` WHERE User_ID =
{txtUserID.Text} SET User_Fullname = {txtFullName.Text},
User_LoginName = {txtUsername.Text}, User_Email =
{txtEmail.Text}, User_Password = {Encrypt(txtPassword.Text)},
User_AccessLevel = {accessLevel};"
ELSE
    Database.RunQuery($"UPDATE `users` WHERE User_ID =
{txtUserID.Text} SET User_Fullname = {txtFullName.Text},
User_LoginName = {txtUsername.Text}, User_Email =
{txtEmail.Text}, User_AccessLevel = {accessLevel};"
 END IF

// Obtain user record back from database to check it has been successfully updated
user = Database.RunQuery("SELECT * from `users` WHERE User_ID = " + txtUserID.Text)

bool updateconfirm = true
// Check the obtained data against the current data
IF user['User_Fullname'] != txtFullName.Text THEN
        updateconfirm = false
ELSE IF user['User_LoginName'] != txtUsername.Text THEN
        updateconfirm = false
ELSE IF user['User_Email'] != txtEmail.Text THEN
        updateconfirm = false
```

```
ELSE IF user['User_AccessLevel'] != accessLevel THEN
        updateconfirm = false
END IF

// Display result to user
IF updateconfirm = true THEN
        Output "All user data has been successfully saved"
ELSE
        Output "There was an error saving the user data. Please try again."
END IF

//Disconnect from Database
Database.CloseConnection()

END FUNCTION
```

**Settings**

Now the settings form needs to be designed. This will need to store all of the settings to talk to the database, the server, and settings for the general application itself. Firstly the application will need to load the settings from the configuration files. Then if the user changes any database settings, when the user presses the 'database' dropdown, it will test the database connection, and then list all available databases. The Test Connection buttons will also test their respective database and server connections, this will make sure that valid data has been entered, and that there will be no errors when using the settings later on. The save button will then encrypt this data (the database connection string, and the users password). This will be simple reversible encryption, as the data does not need to be secure, just not readable by anyone who happens to open the configuration file. Once it is saved, the application will then be restarted to update the settings for the application.

The settings will be stored in a file (this will be located in the current running directory of the application). The raw text in the file will be as follows:

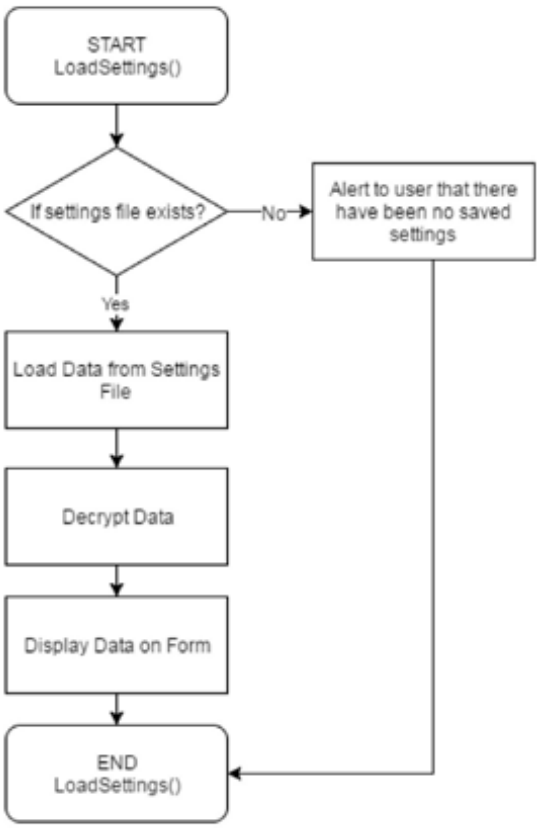
```
1 Name=[redacted]
2 Server=[redacted];User ID=[redacted];
3 Version=v1;[redacted];Port=[redacted];ID=[redacted];Username=[redacted];Password=[redacted];
```

The final two lines will contain sensitive information, both of these will be encrypted with base64 encoding, this will prevent anyone from reading the settings from the file, adding another layer of security to the application. After the final two lines had been Base64 encoded, it will be as follows:

```
1 Name=[redacted]
2 [redacted]
3 [redacted]
```

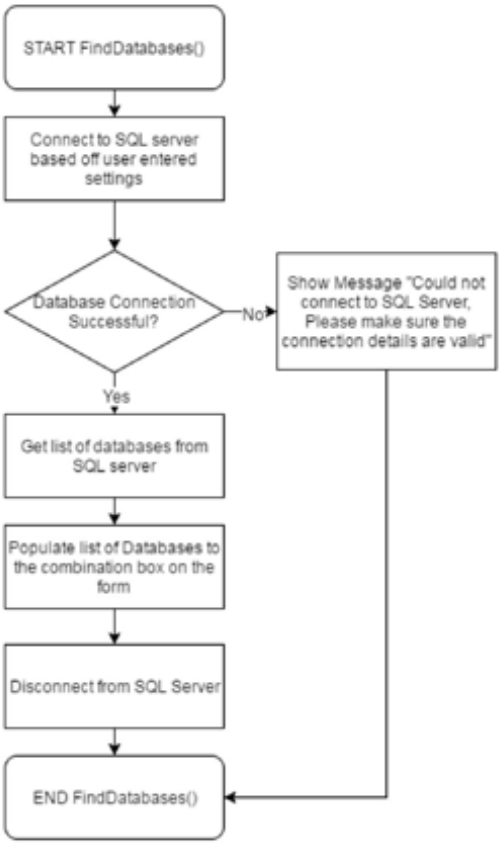
This will then be read by the program, line 1 are the general settings, line 2 are the SQL database settings, and line 3 are the radio server settings.

**Reading Settings from File**



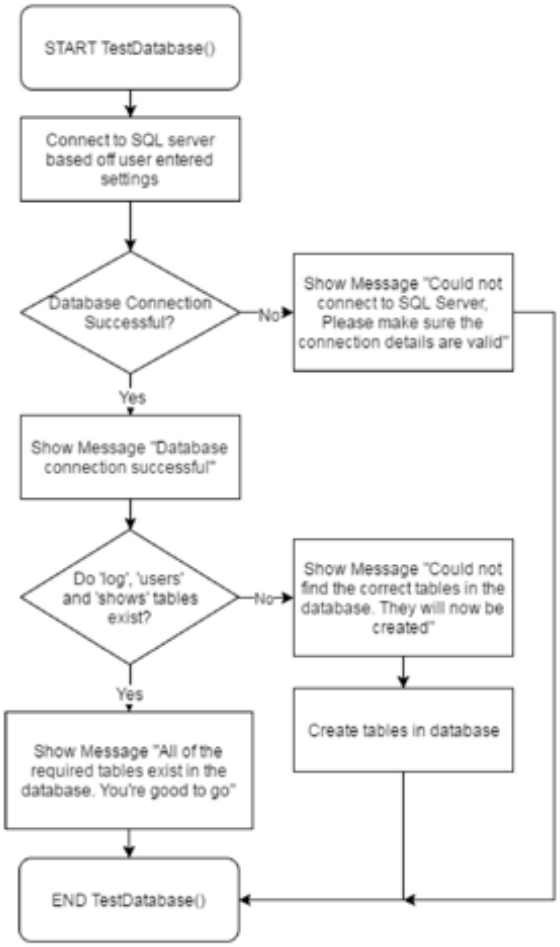

```
FUNCTION LoadSettings()

IF File.Exists("config.conf") THEN
        file = ReadFile("config.conf")

        //Set the general settings
        //Set the general variable to the first line
        string general = file[0]
        txtRadioStationName = general['Name']
        txtHoursStart = general['Start']
        txtHoursEnd = general['End']

        //Set the database settings
        //Set the database variable to the second line
        string database = Base64Deocde(file[1])
        txtDatabaseHost = database['Server']
        txtDatabasePort = database['Port']
        txtDatabaseUsername = database['User ID']
        txtDatabasePassword = database['Password']
        comboDatabase = database['Database']

        //Set the server settings
        //Set the server variable to the third line
        string server = Base64Deocde(file[2])
        txtServerVersion = server['Version']
        txtServerHost = server['Host']
        txtServerPort = server['Port']
        txtServerStreamID = server['ID']
        txtServerUsername = server['Username']
        txtServerPassword = server['Password']

        file.Close() //Close the file, so it can be edited if necessary
ELSE
        Output "There is no settings file saved. A file will be
        created when you press 'Save'"

END IF

END FUNCTION
```

**Update Available Databases (database dropdown)**




```
FUNCTION FindDatabases

// Attempt to connect to database
IF Database.TryConnect() = TRUE THEN
   // Connection was sucessful, so obtian list of databases
   Database.Connect()
   Database.RunQuery("show databases;")

   //Remove all databases currently listed, and update the list
   comboDatabase.DeleteAll()
   FOREACH RECORD db in Database.Records DO
            comboDatabase.Add = db['name']
   END FOREACH

   // Close connection to Database
   Database.CloseConnection()

ELSE
   OUTPUT "Could not connect to SQL Server. Please make sure
the connection details are valid"

END IF

END FUNCTION
```

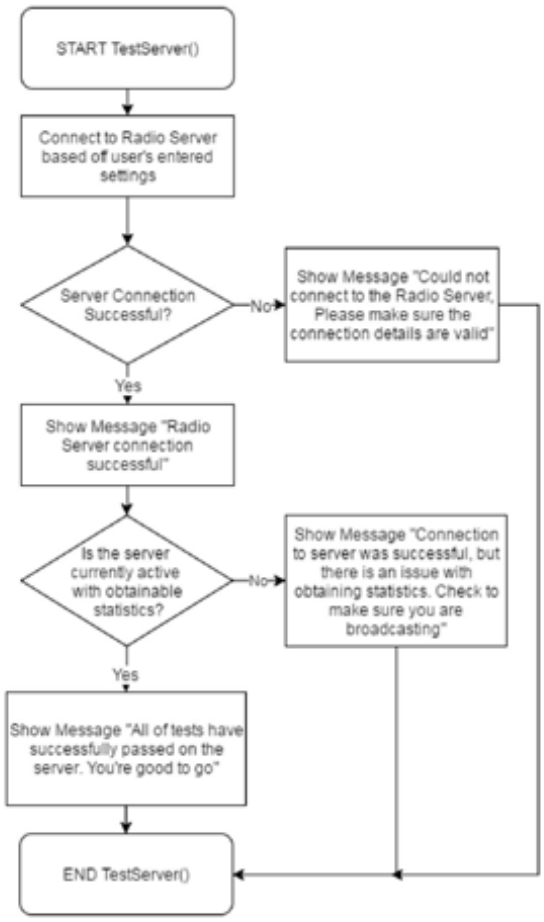
**Test Database**




```
FUNCTION TestDatabase

// Attempt to connect to database
IF Database.TryConnect() = TRUE THEN

        // Connection was successful, so obtain list of databases
        Database.Connect()
        Database.RunQuery("    SELECT COUNT(*) FROM
        information_schema.tables WHERE table_schema = '" +
        comboDatabase.Text + "' AND table_name = 'log' OR 'users'
        OR 'show';")


        IF Database.Record['count'] = 3 THEN
                OUTPUT "All of the required tables exist in the
                database. You're good to go."
        ELSE
                OUTPUT "Could not find the correct tables in the
                database. They will now be created"

                //CODE TO CREATE TABLES IN DATABASE

        // Close connection to Database
        Database.CloseConnection()

ELSE
        OUTPUT "Could not connect to SQL Server. Please make
        sure the connection details are valid"

END IF

END FUNCTION
```

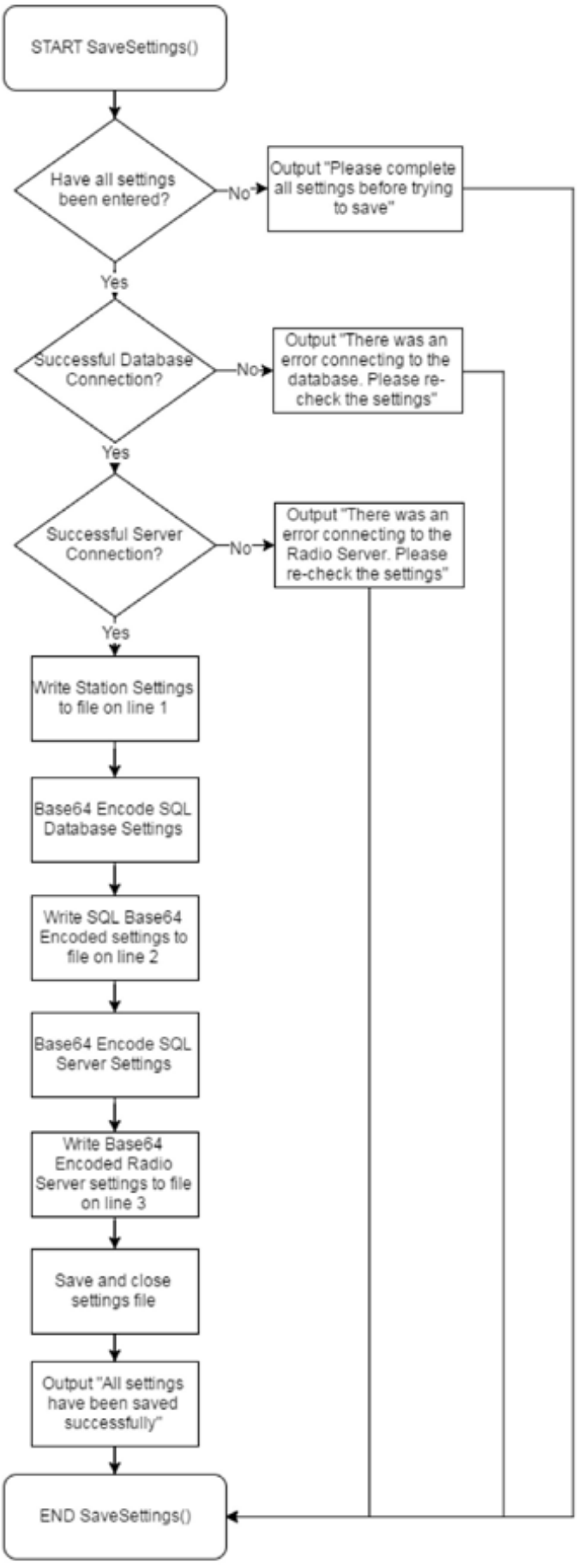
The code to create tables in the database will run a SQL query that will be a data export as SQL from the setup version of the database. This query will be created later on in the application's development.

**Test Server**




```
FUNCTION TestServer

// Set the server details, to the details entered in settings
Server.Version = txtServerVersion.Text
Server.Host = txtServerHost.Text
Server.Port = txtServerPort.Text
Server.StreamID = txtServerStreamID.Text
Server.Username = txtServerUsername.Text
Server.Password = txtServerPassword.Text

// Attempt to connect to server
IF Server.TryConnect() = TRUE THEN
        // Connection was sucessful
        OUTPUT "Radio Server connection was successful. Now
        performing tests on the server"

        //Connect to server and perform tests
        Server.Connect()
        Server.GetStatistics()

        //Check to make sure statistics have been returned from
        the server
        IF Server.Stats.Count > 0 THEN
                OUTPUT "All of the tests have successfully passed
                on the server. You're good to go."
        ELSE
                OUTPUT "Connection to the server was successful,
                but there is an issue obtain statistics. Check to
                make sure you are broadcasting."

        // Close connection to Server
        Server.CloseConnection()

ELSE
        OUTPUT "Could not connect to the Radio Server, Please
        make sure the connection details are valid"

END IF
```

**Save Settings**



```
FUNCTION SaveSettings()

// Check to see if all settings have been entered (by cycling through each textbox)
BOOLEAN valuesEntered = TRUE
FOREACH TEXTBOX ON FORM AS ITEM
        IF (ITEM.TEXT == "") THEN
                // No value entered for that text box, so set text to red, and change variable flag
                valuesEntered = FALSE
                ITEM.TextColor = Red
        ELSE
                // A value was entered so reset the colour of the textbox
                ITEM.TextColor = Black
        END IF
END FOREACH

//Check the combo box
IF(comboDatabase == "") THEN
        // No value was entered for database, set colour of text to red
        valuesEntered = FALSE
        comboDatabase.TextColor = Red
ELSE
        // A value was entered, so reset the colour of the text to black
        comboDatabase.TextColor = Black
END IF

IF(valuesEntered == FALSE) THEN
        OUTPUT "Please complete all settings before trying to save"
ELSE
        // All values have been entered so the database and server can be checked
        IF(TestDatabase() == TRUE) THEN
                IF(TestServer() == TRUE) THEN
                        // As all settings are fine and have been checked, save them to the file

                        // Check to see if config file exists, if not create a blank file
                        IF File.Exists("config.conf") == FALSE THEN
                                File.Create("config.conf")
                        END IF

                        //Open the file to write to
                        file = OpenFile("config.conf")

                        // Write the radio station settings to file on line 1
                        file.Write[1] = "Name=" + txtRadioStationName + ";Start=" + txtHoursStart +
                        ";End=" + txtHoursEnd + ";"

                        // Write the database settings to file on line 2 and encode them
                        file.Write[2] = Base64Encode("Server=" + txtDatabaseHost + ";Port=" +
                        txtDatabasePort + ";User ID=" + txtDatabaseUsername + ";Password=" +
                        txtDatabasePassword + ";Database =" + comboDatabase + ";"

                        // Write the Radio Server settings to file on line 3 and encode them to a file
                        file.Write[3] = Base64Encode("Version=" + txtServerVersion + ";Host=" +
                        txtServerHost + ";Port=" + txtServerPort + ";ID=" + txtServerStreamID +
                        ";Username=" + txtServerUsername + ";Password=" + txtServerPassword + ";"

                        // Now all settings have been saved, save and close the config file
                        File.Save()
```

```
                File.Close()

                // Output to the user the settings were successfully saved
                OUTPUT "All settings have been saved successfully"

            ELSE
                // Error connecting to Radio Server, so alert user
                OUTPUT "There was an error connecting to the Radio Server. Please re-check
                the settings"
            END IF
        ELSE
            // Error connecting to SQL Server, so alert user
            OUTPUT "There was an error connecting to the database. Please re-check the settings"
        END IF
END IF

END FUNCTION
```

**Edit Shows**

The edit shows section is something which would be needed in the actual application, however I will not be documenting the development of this section as it is not part of the user requirements, however it will still be needed for the system to function. The system will read shows from the database, and this form will allow them to be edited. The main aim of the project is to overview analytics, rather than creating a show timetable. I will presume that the shows in the database can and have already been edited.

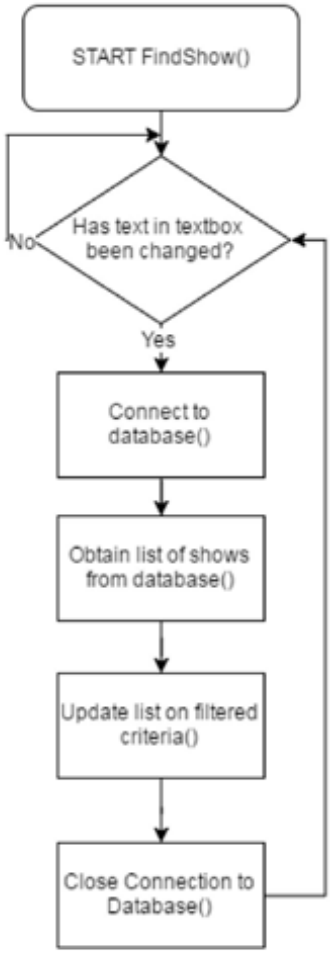
**Reports**

As mentioned earlier, the reports are split up into 4 different sections, these will be in 4 different tabs on the same form to allow for the user to easily navigate between them. Each of these will be designed separately.

**Show History Report**

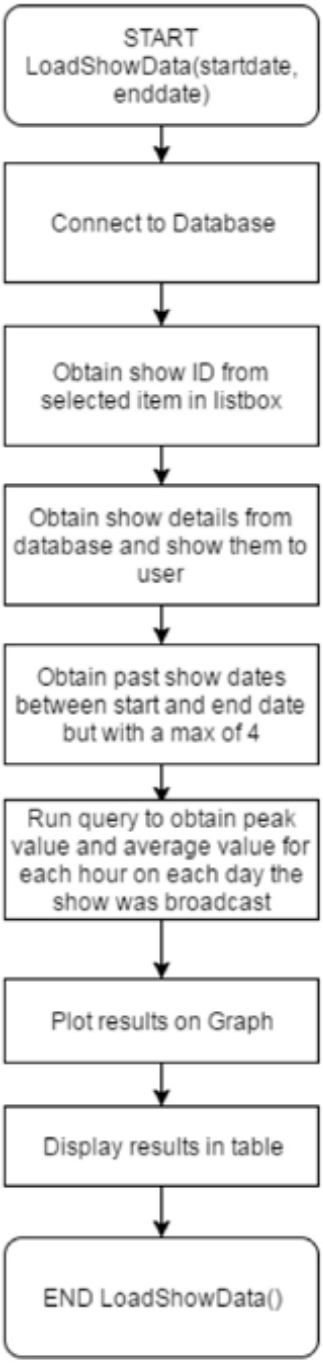
This form will search though the shows (obtained from the show table in the database) it will update the list of shows when you type a value in the search box. When a show is found it will obtain the data of the show, including what weekday it is on, and the hours. It will then plot a graph for the past previous 4 shows based off the most recent show which was broadcast. It will also create a table to give the raw data it is plotting as well. This will be split into two functions, these are 'Searching for a show' and 'Loading show Data'.

**Searching for Show**




```
FUNCTION FindShow()

// The code will be ran whenever the text in the txtShowQuery text box is
changed by the user
WHEN txtShowQuery IS UPDATED

    // Connect to database,
    // and run query to return a list of users containing the query
    Database.Connect()

    // Obtain all of the records which contain the search criteria in the name
    of the show, or the presenters name
    Database.RunQuery("SELECT * FROM `shows` WHERE `Show_Name`
    LIKE '%" + txtShowQuery + "%' OR `Show_Presenter` LIKE '%" +
    txtShowQuery + "%'")

    //Loop through each of the records obtained, and add them to a list box
    for the user to select
    FOREACH RECORD show in Database.Records DO

        //Put each show in the format of "Show Name with Presenter
        (Monday Start Time - End Time)"
        listBoxShows.Add = show['Show_Name'] + " with " +
        show['Show_Presenter'] + " (" + show['Show_Day'] + " " +
        show['Show_Starttime] + " - " + show['Show_Endtime] + ")"

        //Add a tag to each of the shows so they can be referenced
        when they are selected
        listBoxShows.AddTag = show['Show_ID']

    END FOREACH

    //Close the connection to the database
    Database.Close()

END WHEN

END FUNCTION
```

**Loading show Data**



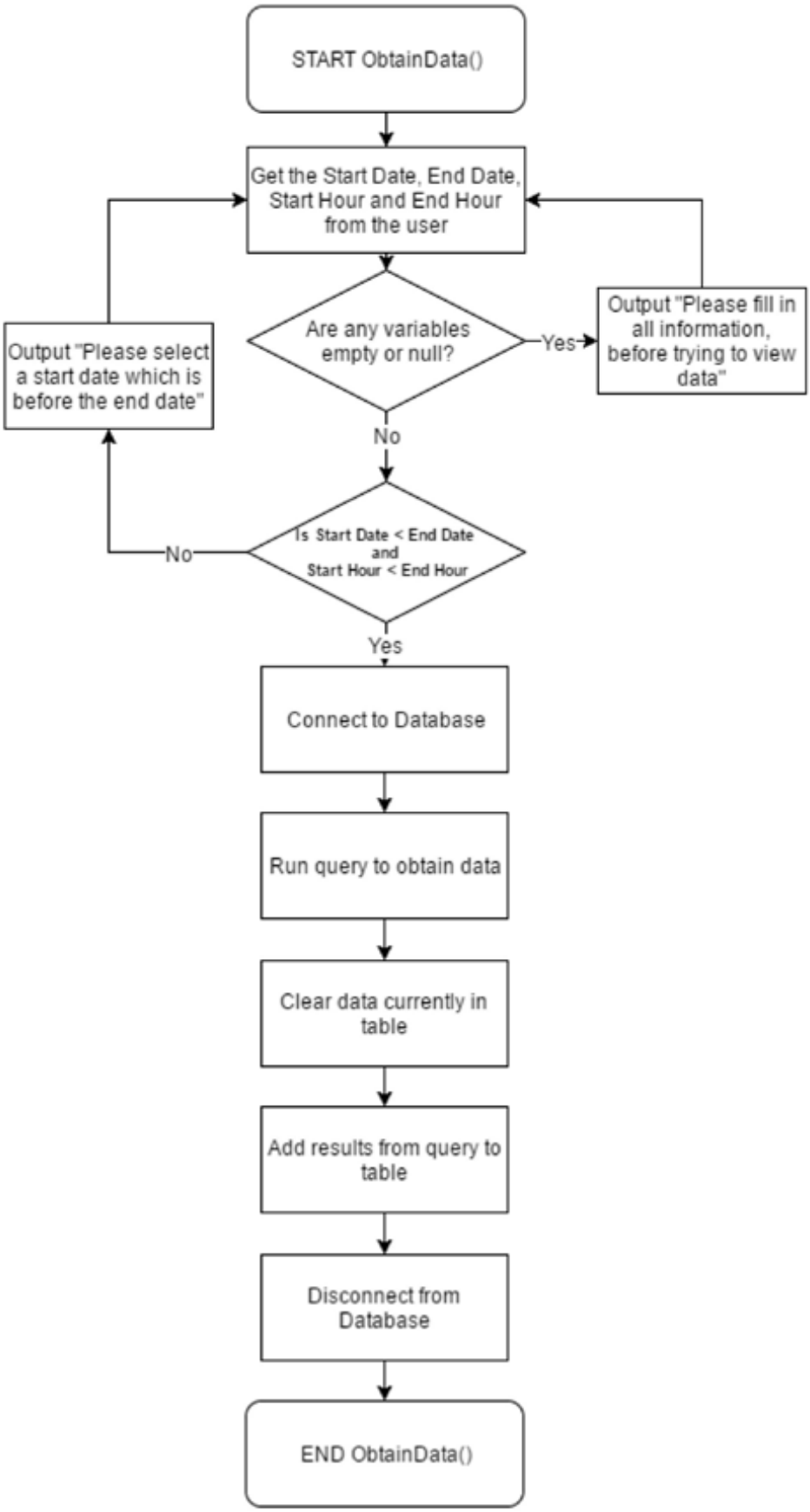

```
FUNCTION LoadShowData(startdate, enddate)

        // Connect to database,
        // and run query to return a list of users containing the query
        Database.Connect()

        //Get current show ID from listBoxShows
        int showID = listBoxShows.SelectedShow.Tag

        //Obtain show data from database and show to the user
        Database.RunQuery("SELECT * FROM `shows` WHERE `Show_ID` = '" +
        showID + "';")

        //Display data to user
        txtShowName = show['Show_Name']
        txtShowPresenter = show['Show_Presenter']
        txtShowStartHour = show['Show_Starttime']
        txtShowEndHour = show['Show_Endtime']
        txtShowDay = show['Show_Day']
        txtShowStartDate = show['Show_StartDate']
        txtShowEndDate = show['Show_EndDate']

        IF(show['Show_Active'] = TRUE) THEN
                txtShowActive = "Show is Currently Active"
        ELSE
                txtShowActive = "Show is not active"
        END IF

        // This array will hold the dates of shows to check
        Array dates = new Array[4]

        //Check to see if enddate and startdate variables have been passed
        IF(startdate != "" AND enddate !="") THEN
                // Obtain maximum 4 dates of past shows between the dates
                specified
                // This function will set the dates array to contain the dates
                which the day has occurred over the past number of weeks,
                between the start and end dates, with a maximum of 4 weeks
                dates = Time.FindDatesOfDay(show['Show_Day'], startdate,
                enddate, 4)
        ELSE
                //No dates have been specified, so obtain past 4 dates from
                today
                dates = Time.FindDatesOfDay(show['Show_Day'], 4)
        END IF

        //Create new graph and set title
        HistoryGraph = new BarGraph
        HistoryGraph.Title = "Past Show hour history"

        //Cycle through each date and obtain statistics for it
        FOREACH date in dates THEN
                Database.RunQuery("SELECT HOUR(Log_DateTime) AS `Hour`,
                MAX(Log_ListenerCount) AS ListenerPeak,
                ROUND(AVG(Log_ListenerCount),0) AS ListenerAverage  FROM
                log WHERE Log_DateTime between '" + startdate + " 00:00:00'
```

```
            and '" + enddate + " 23:59:59' GROUP BY
            HOUR(log.Log_DateTime);

            // Plot this information on a graph
            HistoryGraph.addNewSeries(date)
            HistoryGraph.addData(Database.QueryResult)

            // Add this information to a table
            table.addData(Database.QueryResult)
        END FOR

        //Close the connection to the database
        Database.Close()

    END FUNCTION
```

**Song History Report**

The next report to design is the Song history report, this will give a raw read-out of data from the database between two dates which have been specified. This will be split into two sections, "Obtain raw data" which will list all of the data recorded between the two dates and hours specified; and "Search by song" this will return all occurrences of a song which has been played.

**Obtain Raw Data**



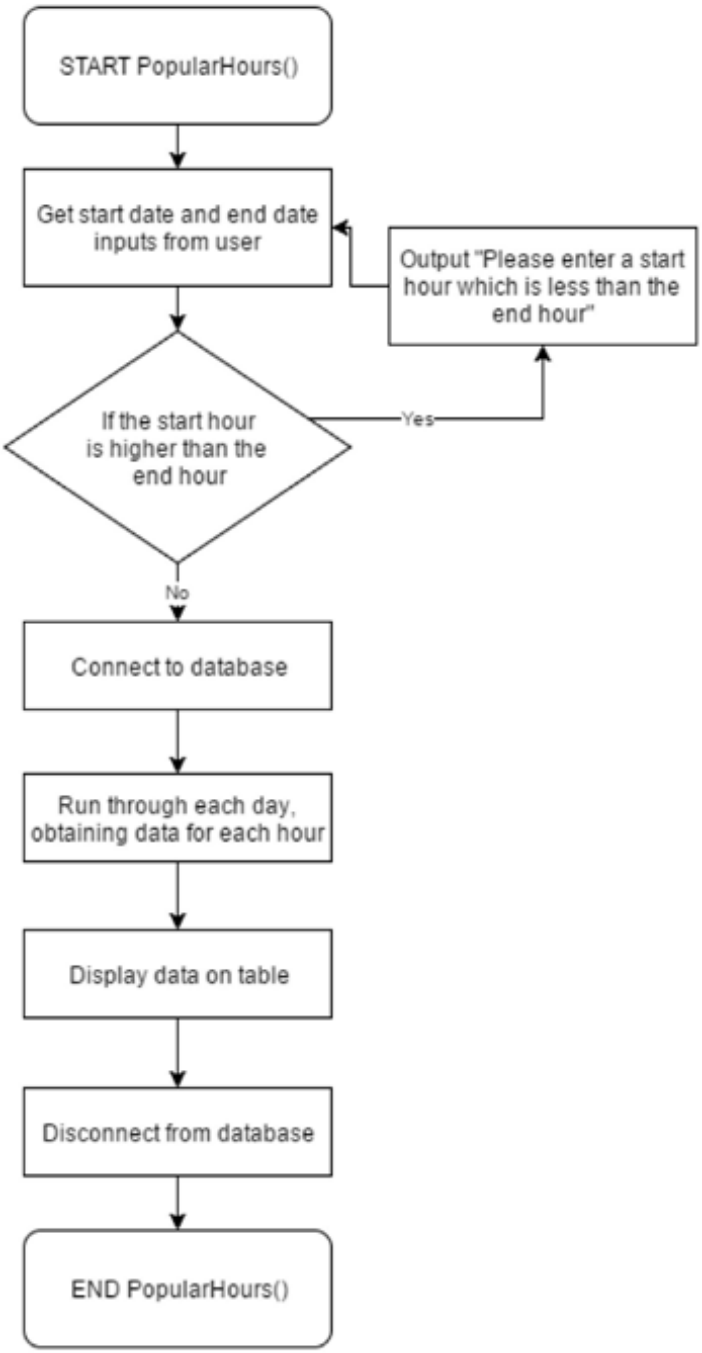
```
FUNCTION ObtainData()

        //Get the information the user has entered, and set it as variables
        startDate = calendarStartDate.Date
        endDate = calendarEndDate.Date

        //Append ':00:00' to the end of the start time to make sure it is in a time format, rather then just
        the hour the user has entered
        startTime = txtStartHour + ":00:00"

        //Append ':59:59' to the end of the end time to make sure it is also in a time format, rather then
        just the hour the user has entered
        endTime = txtEndHour + ":59:59"

        //Check to make sure all variables have been entered
        IF(startDate != "" AND endDate != "" AND startTime != ":00:00" AND endTime !=":59:59") THEN

                //This means all of the data has been entered correctly, now we need to check to make
                sure the start time and date is less than the end time and date
                IF(startDate < endDate AND startTime < endTime) THEN

                        //This means the dates are correct, so the query can now be ran
                        Database.Connect()
                        Database.RunQuery("SELECT * FROM log WHERE Log_DateTime between '" +
                        startDate + " " + startTime + "' and '" + endDate + " " + endTime + "';")

                        //Now we need to clear the data currently in the table, and set it to be the
                        results obtained from running the query
                        resultsTable.Clear()
                        resultsTable.Data = Database.Results

                        //Now we can close the connection to the database
                        Database.CloseConnection()
                ELSE
                        //This means the dates entered are not in the correct order
                        OUTPUT "Please select a start data which is before the end date"
                END IF
        ELSE
                //This means the data has not all been entered
                OUTPUT "Please fill in all information before trying to view data"
        END IF

END FUNCTION
```

**Search by Song**


```
FUNCTION SearchBySong()
WHEN txtSearchCriteria is updated THEN

        //Connect to database
        Database.Connect()

        //Run query to obtain data from database where the song name contains in any
        location the search criteria entered
        Database.RunQuery("SELECT * FROM log WHERE Log_CurrentSong LIKE '%" +
        txtSearchCriteria + "%';")

        //Clear the data currently in the table
        resultsTable.Clear()

        //Set the table to display the results obtained from the database
        resultsTable.Data = Database.Results

        //Disconnect from the database
        Database.CloseConnection()

END WHEN
END FUNCTION
```

**Popular Hours**

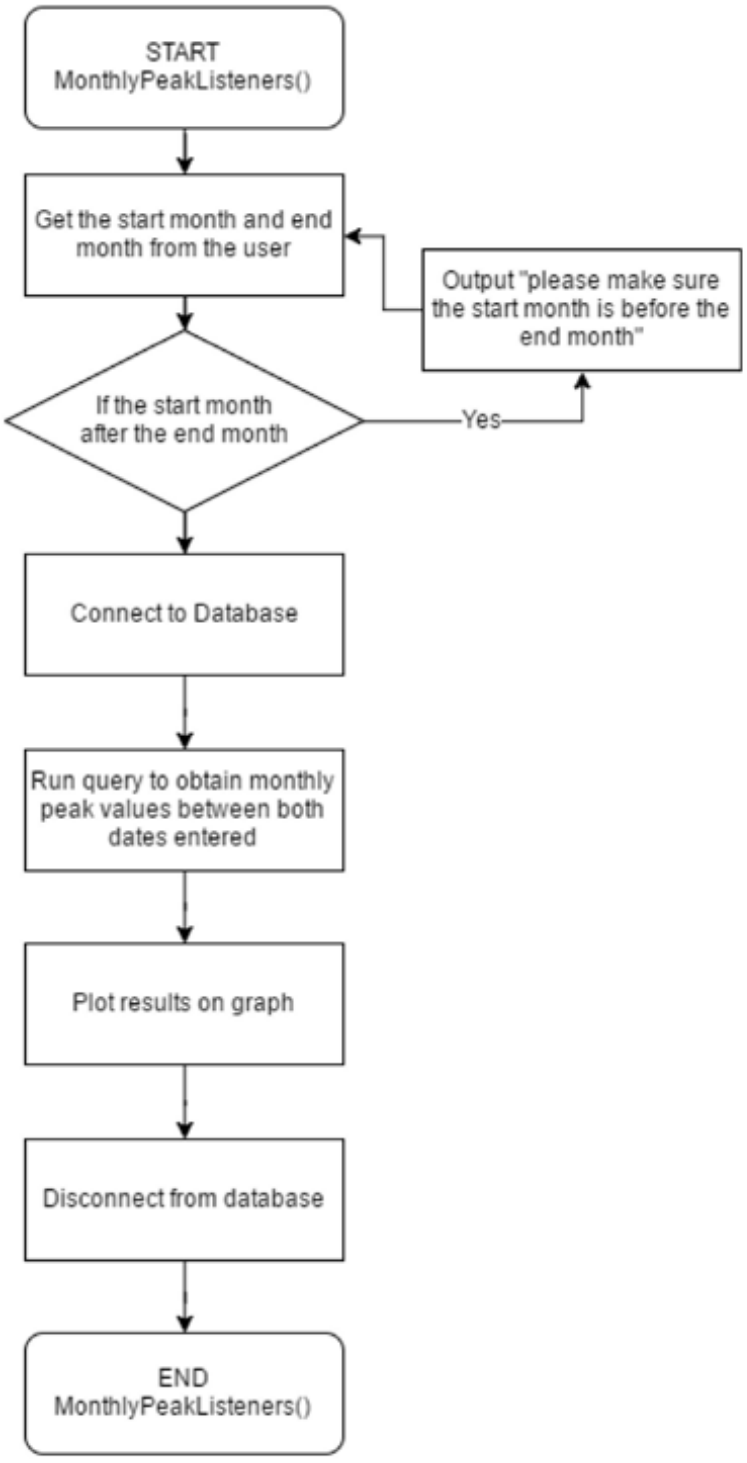
This report will return the statistics for the individual hours over the past week. It will start off with a start date, and an end date, reporting data for hours between.



```
FUNCTION PopularHours()
        startDate = calendarStartDate.Date

        //Append ':00:00' to the end of the start time to make sure it is in a time format, rather then just
        the hour the user has entered
        startTime = txtStartHour + ":00:00"

        //Append ':59:59' to the end of the end time to make sure it is also in a time format, rather then
        just the hour the user has entered
        endTime = txtEndHour + ":59:59"

        //Check to make sure all variables have been entered
        IF(startDate != "" AND startTime != ":00:00" AND endTime !=":59:59") THEN
                //This means all of the data has been entered correctly, now we need to check to make
                sure the start time and date is less than the end time and date
                IF(startTime < endTime) THEN
                        //This means the dates are correct, now we can obtain data for each day of the
                        week, by running a loop

                        //Connect to database
                        Database.Connect()

                        //Clear the table, ready to add new data
                        resultsTable.Clear()

                        //Start the loop to obtain data
                        WHILE(INT i = 0; i < 7; i++) THEN
                                // This will run the query for each day
                                Database.RunQuery("SELECT HOUR(Log_DateTime) AS `Hour`,
                                MAX(Log_ListenerCount) AS ListenerPeak,
                                ROUND(AVG(Log_ListenerCount),0) AS ListenerAverage FROM log
                                WHERE Log_DateTime between '" + DATE(startDate + i) + " " + startTime
                                + "' and '" + DATE(startDate + i) + " " + endTime + "' GROUP BY
                                HOUR(log.Log_DateTime);")

                                // Now we can add these results as a new set of columns to the table
                                resultsTable.AddHeader(DATE(startDate + i))
                                resultsTable.AddData(Database.Results)
                                resultsTable.CreateNewColumn()
                        END WHILE

                        //Now we can close the connection to the database
                        Database.CloseConnection()
                ELSE
                        //This means the times entered are not in the correct order
                        OUTPUT "Please select a start hour which is before the end hour"
                END IF
        ELSE
                //This means the data has not all been entered
                OUTPUT "Please fill in all information before trying to view data"
        END IF

END FUNCTION
```

**Monthly Peak Listeners**

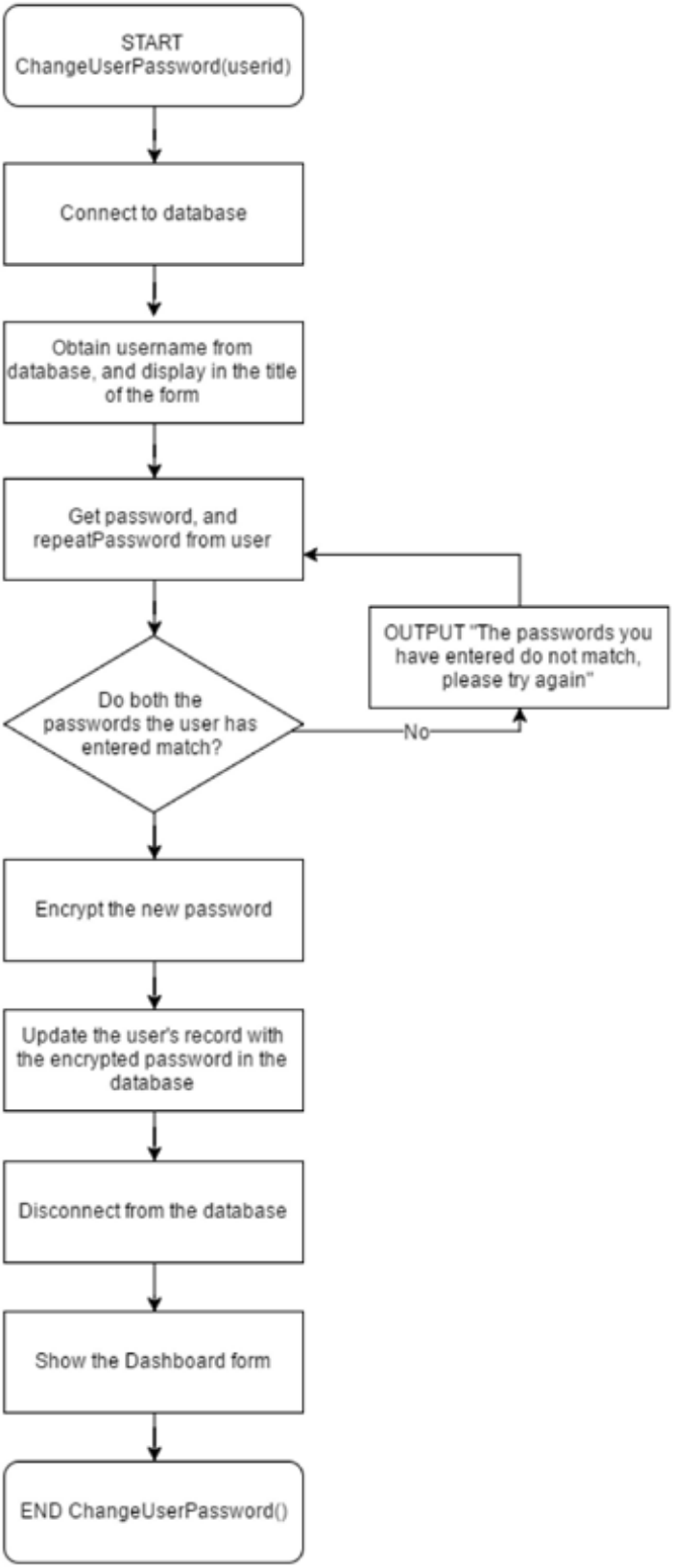
This is the final reporting option, this will take two months entered and allow you to see the peak listeners for each month, and then it will plot this on a graph for the user to see.



```
FUNCTION MonthlyPeakListeners()

    //Obtain the start and end date from the user (convert the value they selected into YYYY-MM so
    then add on '-01 00:00:00' to make it start from the first second of the month)
    startDate = DATE(comboStartMonth) + "-01 00:00:00"
    endDate = DATE(comboEndMonth) + "-01 00:00:00"

    //Check to make sure the start month is before the end month
    IF startDate < endDate THEN
        //This means the dates are the correct way around, we can now proceed with obtaining
        the required date

        //Connect to database
        Database.Connect()

        //Run query to obtain the peak listener values for each month between the two dates
        the user specified
        Database.RunQuery("SELECT CONCAT(MONTHNAME(Log_DateTime), " ",
        YEAR(Log_DateTime)) AS 'Month', MAX(Log_ListenerCount) AS 'Peak' FROM log WHERE
        Log_DateTime between '" + startDate + "' and '" + endDate + "' GROUP BY
        MONTH(log.Log_DateTime);")

        //Clear the data currently on the graph
        resultsGraph.Clear()

        //Plot the results on the graph
        ResultsGraph.addNewSeries(startDate + " - " + endDate)
        HistoryGraph.addData(Database.QueryResult)


        //Disconnect from the database
        Database.CloseConnection()
    ELSE
        OUTPUT "Please make sure the start month is before the end month"
    END IF

END FUNCTION
```

**Change Password**

This is the final form which needs to be pseudo-coded, this will take an input of a user ID, then ask the user for a new password. After confirming they have entered their password correctly, it will then update it in the database, then log the user into the dashboard overview screen.



```
FUNCTION ChangeUserPassword(userId)

        //Connect to database
        Database.Connect()

        //Obtain Username of current user
        Database.RunQuery("SELECT User_FullName, User_LoginName FROM users WHERE User_ID = "
        + userId)

        //Set records returned to new variable
        user = Database.Results

        //Display results in form title with the format "Change Password for user Dan Parker (admin)"
        Title = "Change Password for " + user['User_FullName'] + " (" + user['User_LoginName'] + ")"

        WHEN SaveButtonPressed THEN
                //When the user has pressed the save button

                //Check both of their passwords match
                IF(txtPassword == txtRepeatPassword) THEN
                        // Both of the passwords they entered are the same, so update them

                        //Encrypt password
                        newPassword = Encrypt(txtPassword)

                        //Update password in database
                        Database.RunQuery("UPDATE users SET User_Password = '" + newPassword + "'
                        WHERE User_ID = " + userId)

                        //Close connection to database
                        Database.CloseConnection()

                        //Display confirmation message to user
                        OUTPUT "Your password was successfully updated."

                        //Open the dashboard
                        Dashboard.Show()

                ELSE
                        //The user hasn't entered the same password twice
                        OUTPUT "The passwords you have entered do not match. Please try again."

                END IF
        END WHEN
END FUNCTION
```

Now I have a good design overview, I am going to use the Agile Methodology. This is developing the project in sprints and iterations. This will mean I can use stepwise refinement, so I can make further design changes as I am developing the application, this is mainly due to the fact while pseudo code and flowcharts give you a good overview of what you are doing (and how the code will work). A lot of practices are language specific, meaning when it comes to making it in C#, which is object orientated, a few things may have to be changed along the way.

## Test Plan

I will be testing the application throughout the development process by performing Alpha testing to test each module of the application at a time. This will enable me to make sure that the application will successfully work when it all comes together at the end. Because of this I can't define any test data to use, as that will depend on each individual module, however I will explain a test plan that will enable thorough testing of each module.

I will make sure to check that all inputs and outputs into and out of each function and method are valid and are expected, by displaying message boxes at various points in each function or method for debugging purposes this will be White Box testing where the internal code can be seen and modified. I will then also test each function and method when added to its relevant control on a form, simulating an end user, making sure the feedback they get is useful and relevant, along with making sure each component on each form functions as expected, this will be ran as black box testing, without being able to see the internal code, and documenting the final results.

After development I will perform a destruction test, along with a general usability test. The destruction test will mainly check to prevent against SQL Injection caused by failure to escape user inputs or correctly passing them into an SQL query. This will be done by using the following string "-- '-- that would cause an SQL syntax error, if the inputs have not been handled correctly. I will also be testing situations such as if the application looses connection to the database, or if it looses internet connection, all of these are real situations that could happen when the application is running.

Finally I will perform general usability testing to make sure that validation is active on all user inputs, to make sure if any are not complete, or contain invalid data, that it is handled properly.

I will create a table for general usability and destruction testing, it will outline the test number, its description and the expected outcome. When it comes to performing the tests after development I will also have extra columns to hold the actual outcome, and improvements that will need to be made to correct the application if any errors are found. This table is outlined below:

| Test Number | Test Description | Test Data | Expected Result |
|---|---|---|---|
| 1 | SQL Injection | Pass the string "-- '-- into every textbox | The system should handle this without error |
| 2 | Deleting all users | | The last user should not be removed |
| 3 | Running multiple instances of the one application on the same PC | | It should run fine with no errors |
| 4 | Running multiple instances of the one application across the network | | It should run fine with no errors |
| 5 | Internet connection dropout | | It should try again within 30 seconds |

| | | | |
|---|---|---|---|
| 6 | SQL Connection dropout | | It should try again within 30 seconds |
| 7 | Leave application running for 48 hours | | It should run fine and store data for 48 hours |
| 8 | SQL Injection On Song Name | Use a packet injector to change the name of the song to the SQL test escape string of "-- '-- | It should insert the record correctly into the database with no errors |

## Development

Before any development can begin, the first thing needing to be setup is the database. Currently as I am developing this application in College I am using a MySQL server which I am running at home. I then have mapped this to my own domain on port 2083, which is different to the default MySQL port of 3306. This will allow database access via connecting to [redacted]. However for development purposes, I have installed a copy of MySQL on my laptop, enabling me to connect to it via localhost:3306. This will be the best option, as I can always connect to the database for development purposes, and can have access to it while in college. When it comes to final deployment and installation it will be the user's responsibility to acquire a MySQL database running on a server. [redacted] already has 2 dedicated SQL servers, so the system's database will be added onto that, running under a separate set of credentials, so it will not interfere with any other data already on the database.

Now I will setup the SQL server locally on my PC, I will be using an open source program called HeidiSQL to manage and administer the database. To setup the server I will need to create a new database, I will call this 'MAGMONDB' (this stands for '[redacted] Monitoring Database').

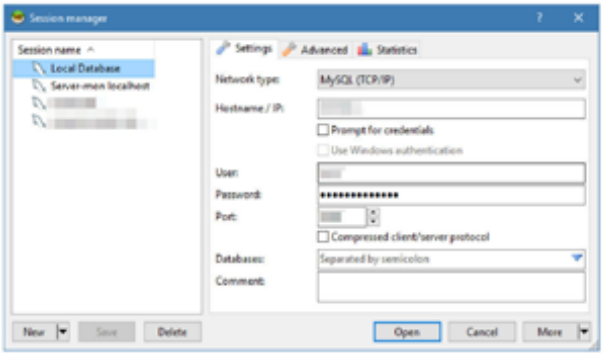



Figure 3. Connecting to the local databse which will be used for development purposes

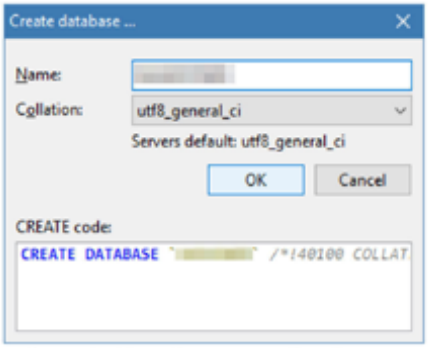
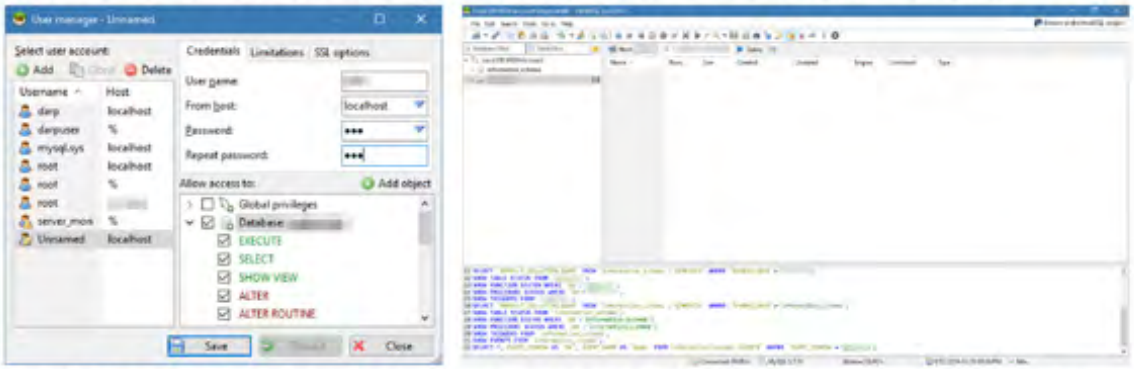



Figure 4. Creating a new database on the server

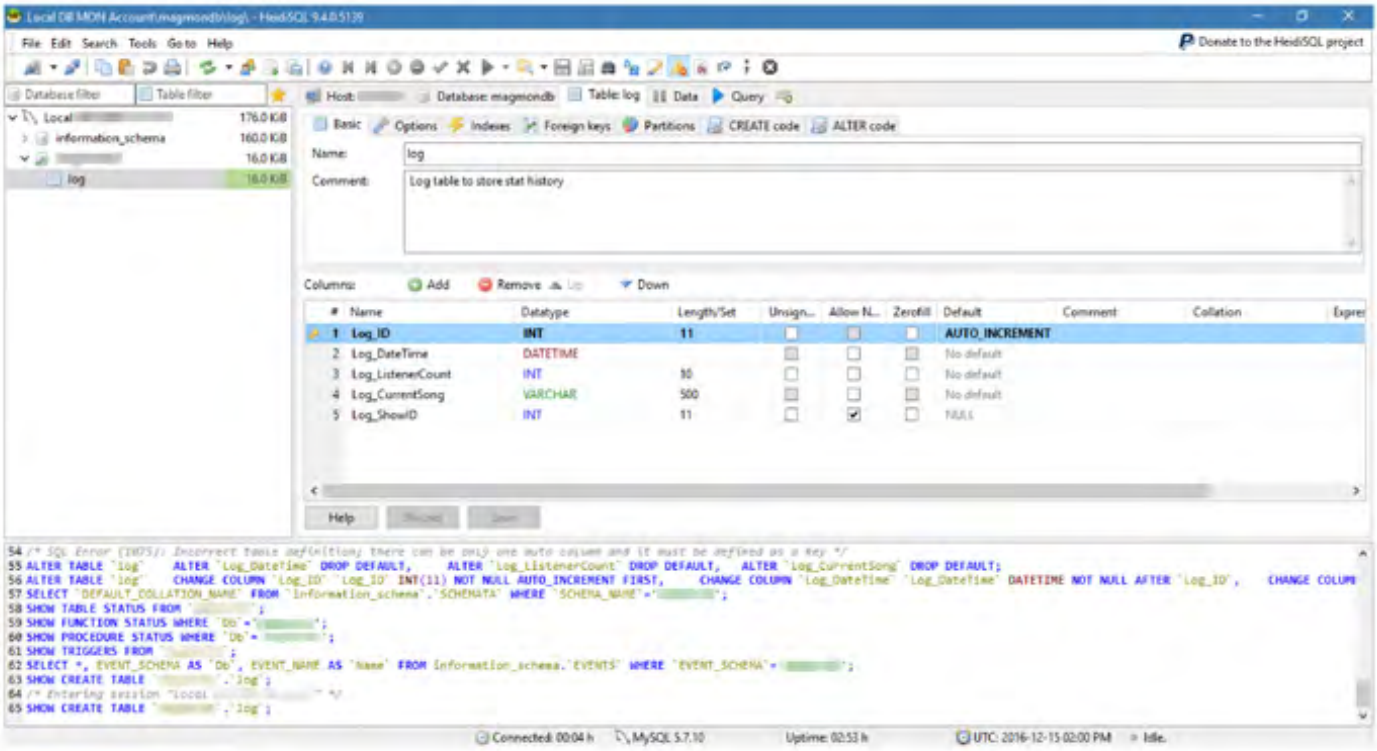
Now I need to create a new user to access to data in that database, this will have a username of 'mon' and a password of 'mon'. I have also given it all privileges on the [redacted] database. This will be changed before the application is published, as leaving your application with extra unneeded database privileges is a huge security risk. However it is okay for development purposes now. As you can see from below, I have added a new user, and I am now logged in as that user, looking at the database:

**H446 (03) A Level Programming Project** 52

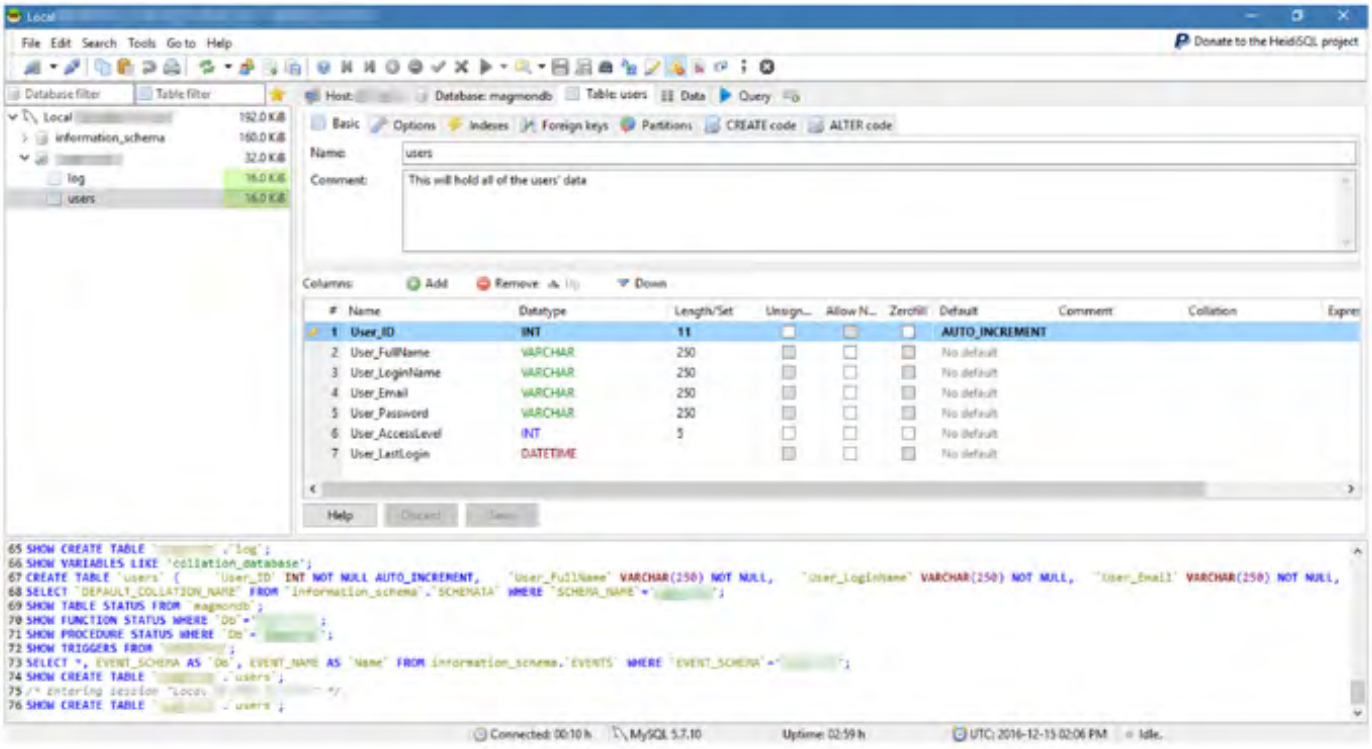
Now I need to add the various tables to the database, based on the design section from earlier.



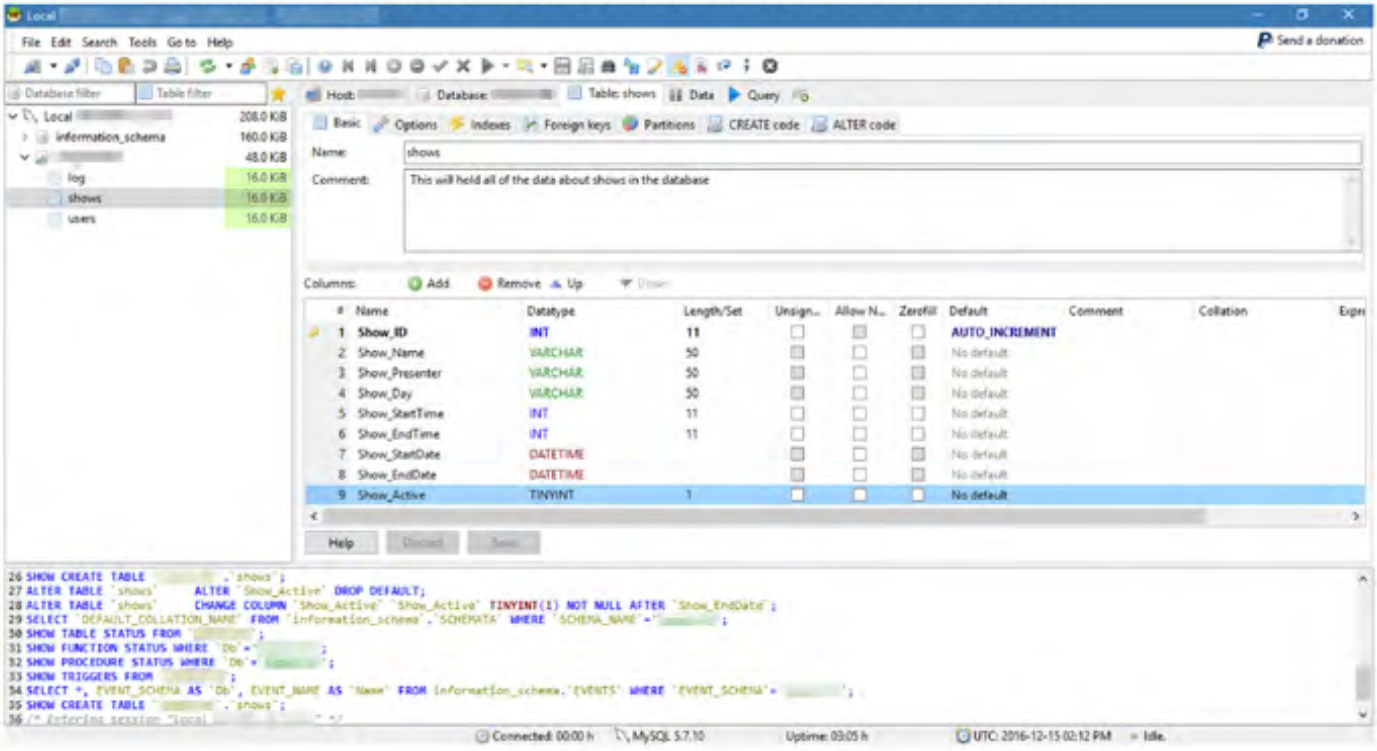
The first table to add is the log table, this will hold all of the statistical history and can be queried when running reports.



Now the log table has been setup, I can move onto setting up the users table, which will hold all of the user's data:

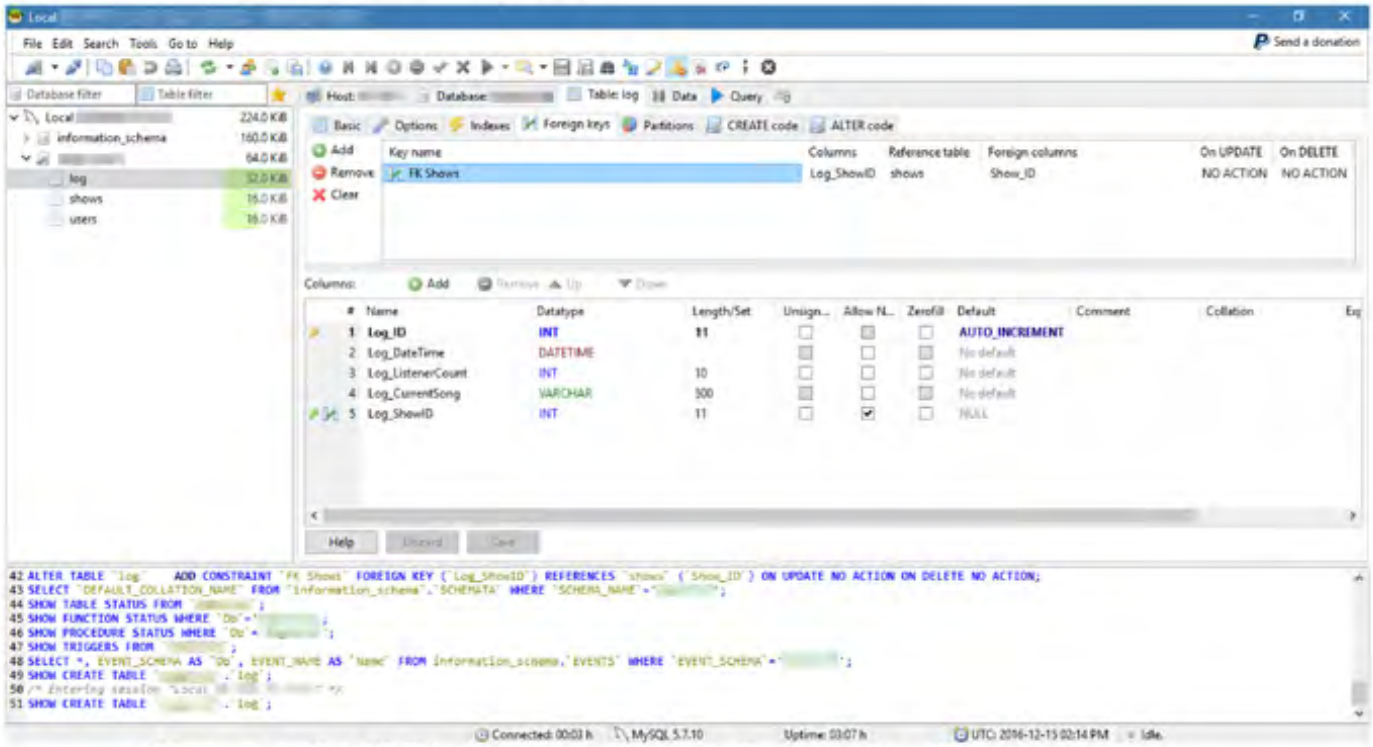



The only table left to create is the show table which will hold all of the data about shows.



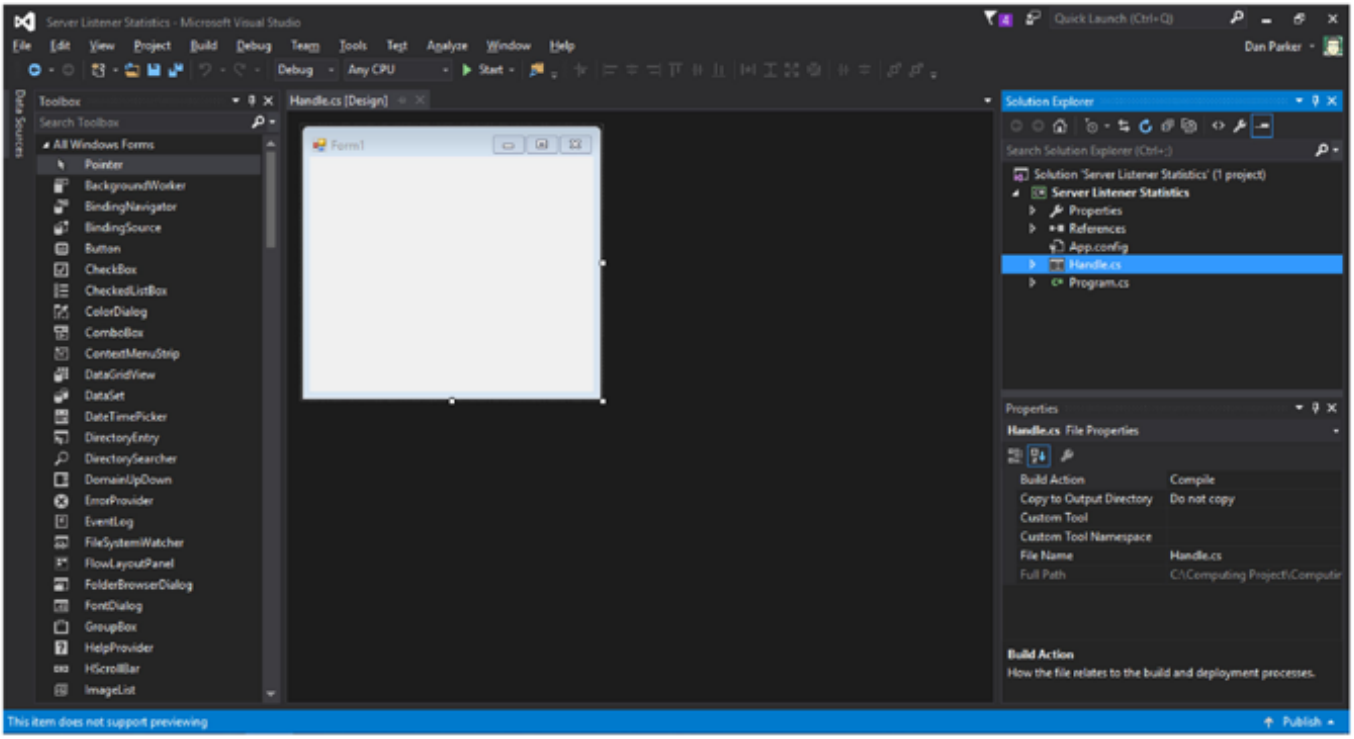
**H446 (03) A Level Programming Project** 54

Now all of the tables have been made, the only thing left with the database is to setup the relationships between the log table, and the shows table. This will link the Foreign Key of Show_ID from the shows table, to the Log_ShowID field of the log table.




As you can see from the image above, the relationships have now been setup successfully.

Now the database has been setup according to the design, we can now begin to create our application which will manage and store data in it.




This is the default screen after starting the project. This will become the Handle form. This will not be visible, and will run the global update code in the background. It will also have a notify icon in the taskbar which will enable the user to access the login form.

As this form should not been seen by the user I have set the following settings:

FormBorderStyle = None (This will hide any toolbars around the form)

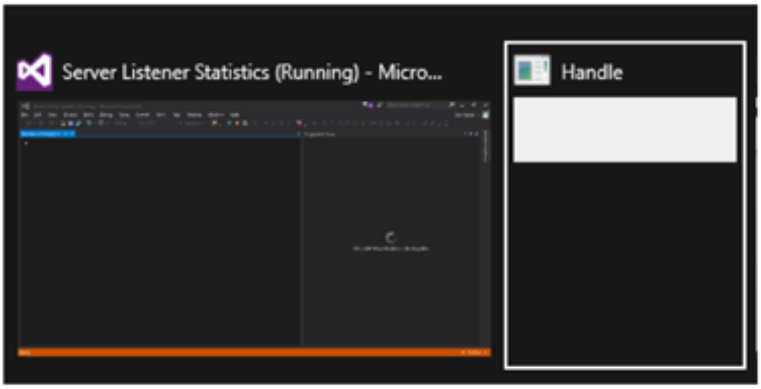
Opacity = 0% (This will make the form invisible, so it can't been seen)

ShowIcon = False (This will hide the icon on the form)

ShowInTaskbar = False (This will prevent the icon displaying on the taskbar)

Size = 0, 0 (This will make the form as small as possible)

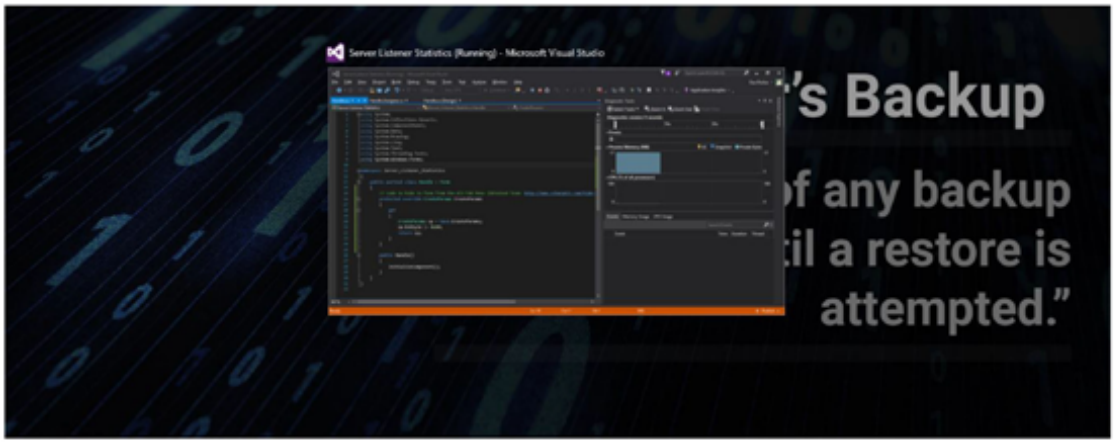
When this is ran it works great and you can't see the form. The only issue is that if you press 'Alt' + 'Tab' on windows you can see it as a window you can switch to, as it looks below:



To prevent this from displaying in the alt-tab menu, I will now need to add code which will set the window style to a system tool window. Since I have never done this before I had to do a bit of research, I came across this link, which gave a snippet of code to do just this job: http://www.csharp411.com/hide-form-from-alttab/

```
namespace Server_Listener_Statistics
{
    public partial class Handle : Form
    {
        // Code to hide to form from the Alt-Tab Menu (Obtained from: http://www.csharp411.com/hide-form-from-alttab/)
        protected override CreateParams CreateParams
        {
            get
            {
                CreateParams cp = base.CreateParams;
                cp.ExStyle |= 0x80;
                return cp;
            }
        }

        public Handle()
        {
            InitializeComponent();
        }
    }
}
```

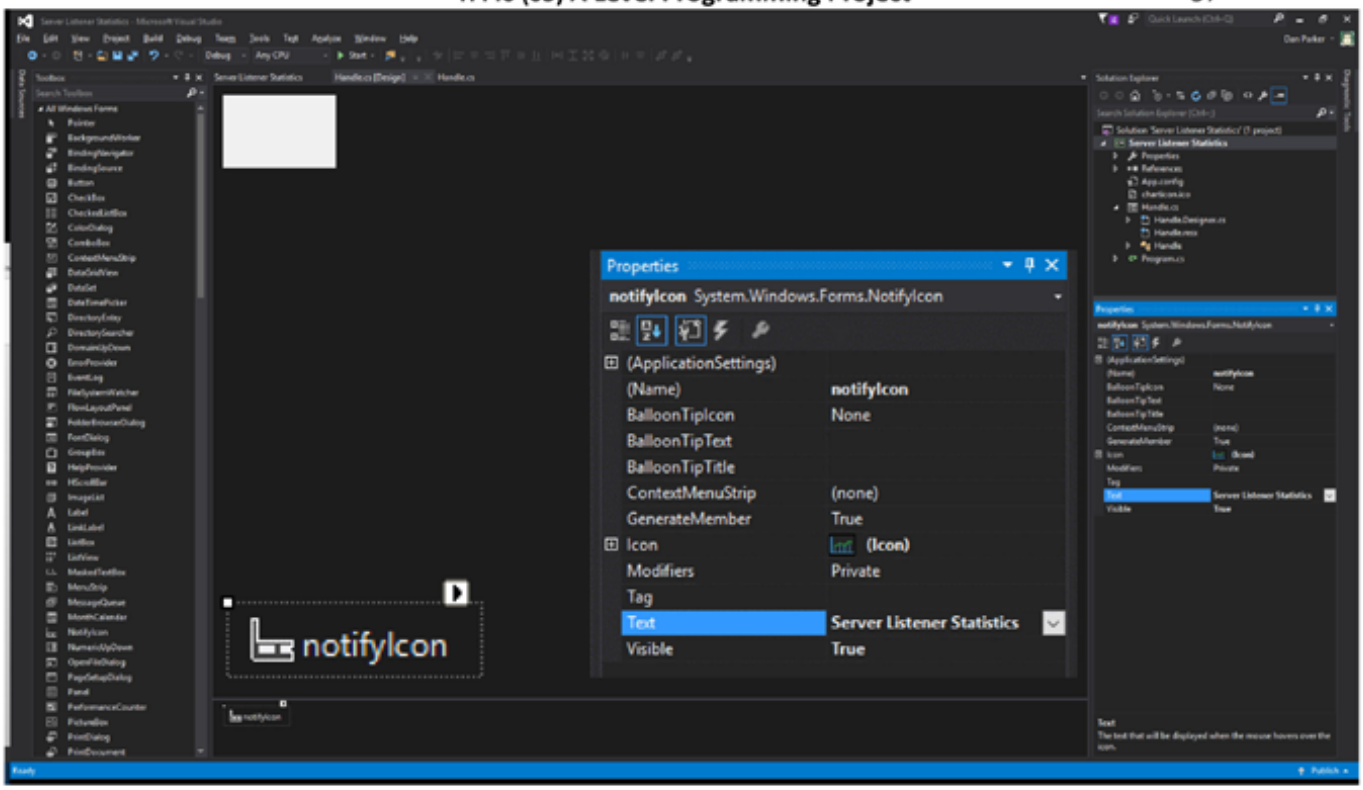
After implementing this code, and re-debugging the application, it successfully worked, and the form was not visible in the alt-tab menu, or anywhere else on the system, as you can see from the screenshot below:




This will now help hit the essential criteria of the system running in the background without interfering with the user, the next element to add is the notify icon, this is a form control in Visual Studio which will enable the user to click on the icon in the system toolbar to access the statistics and login to the system.

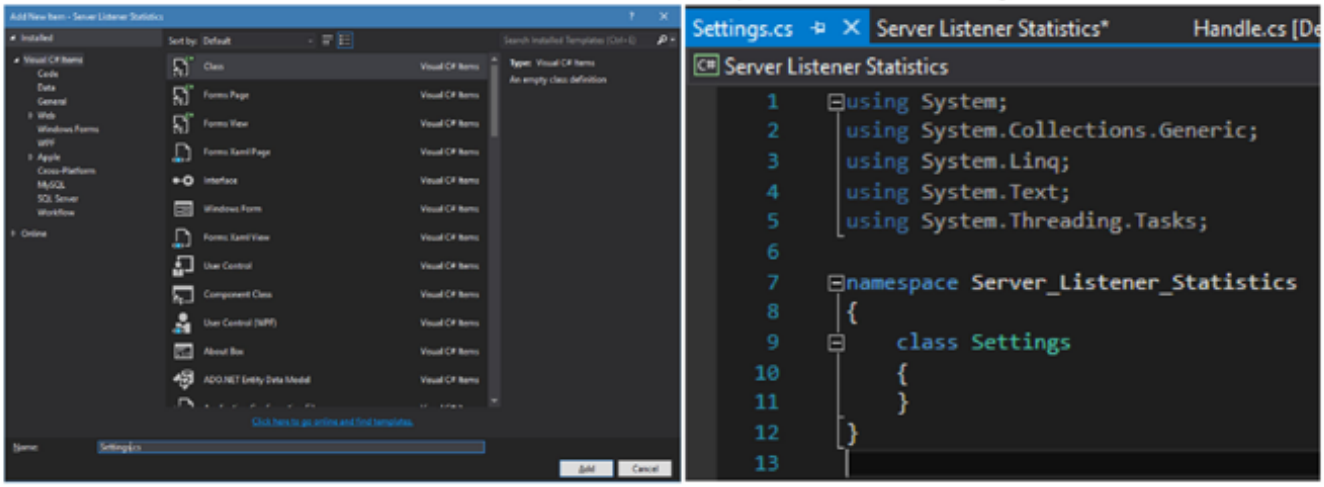
The notify icon will be added to the handle form, this will then act as the main launcher for opening any other forms, and recording statistics into the database.

The tooltip doesn't need any menu options, as hovering over it will return the latest database statistics, and clicking it will open up the user login form. This has now been added to the form, and an icon has been set which I will use as the icon throughout the rest of the application, as you can see below:




When the application is debugged you can now see this icon the taskbar:


Now the notification icon has been setup, I now need to write the global class which will hold the settings for the application. This is due to the fact all of the settings will be the same across multiple forms, all of which need access to them. I will make a new public class called 'settings'.




This class will have to hold the database settings, radio server settings, and the general radio station settings. I will make a public method which will enable me to update the settings from the Config file from anywhere in the application, however I will also have a private and public variable for each setting. This means that the variables will be read-only to any other function or method in the code, other than the 'updateSettings' function which I will now make. This will enable better security, and will prevent somebody from trying to forcefully change application settings, and will also prevent other code updating settings that it is not meant to change.

```
Settings.cs | Server Listener Statistics | Handle.cs [Design] | Handle.cs
Server Listener Statistics - Server_Listener_Statistics.Settings

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Server_Listener_Statistics
8  {
9      class Settings
10     {
11         // Define local private variables
12         // Database Settings
13         private static string dbConnectionStringLocal = "";
14
15         // Radio Server Settings
16         private static string serverHostLocal = "";
17         private static string serverPortLocal = "";
18         private static string serverVersionLocal = "";
19         private static string serverIDLocal = "";
20         private static string serverUsernameLocal = "";
21         private static string serverPasswordLocal = "";
22
23         // General Radio Settings
24         private static string radioNameLocal = "";
25         private static int radioStartHourLocal = 0;
26         private static int radioEndHourLocal = 23;
27
28
29         // Define global public read only variables
30         // Database Settings
31         public static string dbConnectionString { get { return dbConnectionStringLocal; } }
32
33         // Radio Server Settings
34         public static string serverHost { get { return serverHostLocal; } }
35         public static string serverPort { get { return serverPortLocal; } }
36         public static string serverVersion { get { return serverVersionLocal; } }
37         public static string serverID { get { return serverIDLocal; } }
38         public static string serverUsername { get { return serverUsernameLocal; } }
39         public static string serverPassword { get { return serverPasswordLocal; } }
40
41         // General Radio Settings
42         public static string radioName { get { return radioNameLocal; } }
43         public static int radioStartHour { get { return radioStartHour; } }
44         public static int radioEndHour { get { return radioEndHour; } }
45
46
47     }
48 }
```

Now all of the settings have been defined, it is now time to test and make sure they are read only. As they are all identical in definition, then I only need to test one to know that all of them are working. I am going to give the dbConnectionStringLocal variable the value of "teststring" for testing purposes. I will then make that display in the message box when the handle form loads.

Iteli-sense in visual studio has already picked up on the global variables of the settings class (but is not displaying the private ones as they are not accessible)

```
MessageBox.Show(Settings.);

    dbConnectionString        string Settings.dbConnectionString { get; }
    Equals
    radioEndHour
    radioName
    radioStartHour
    ReferenceEquals
    serverHost
    serverID
    serverPassword
```

Also, in the handle scope, if I try to assign a value to the dbConnectionString variable, it will not let me as it's read only.

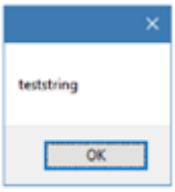
```
MessageBox.Show(Settings.dbConnectionString);
Settings.dbConnectionString = "Test Value";
```

string Settings.dbConnectionString { get; }

Property or indexer 'Settings.dbConnectionString' cannot be assigned to -- it is read only

This is good as the protection is working, the only thing left to test is the message box when the form opens to see if it displays the value of dbConnectionString. I have set it to "teststring" in the local variable definition in the settings class.

```
class Settings
{
    // Define local private variables
    // Database Settings
    private static string dbConnectionStringLocal = "teststring";
```




Now when the form runs you can see a message box appears with the text "teststring" inside. This means everything is working successfully, and I can now move onto the global method to update the settings from the configuration file.

The configuration file will be stored in the same directory the application is running from, this will be the debug folder now, and the Program Files folder when the application is deployed on a computer. The file is called "config.conf" and is a plain text file. As mentioned earlier in the design it will be 3 lines long, the first will be the general settings (Radio station name, Start Hour and End Hour); the second will be the SQL Database connection string (Base64 encrypted for security); and the third will be the Radio server settings (again Base64 encrypted for security).

```
//Define global method to update settings
public static void obtainSettings()
{
    if (File.Exists(@"config.conf"))
    {
        StreamReader sr = new StreamReader(@"config.conf");

        //Obtain general settings
        string[] generalSettings = sr.ReadLine().Split(';'); //This will take the first line and split it by the ';' symbol
        // Each vairable will need to be split again by the '=' symbol, to return the text to the right of the equals sign,
        // this is as each variable starts with a name, then an equals sign, then the variables value
        radioNameLocal = generalSettings[0].Split('=')[1];
        radioStartHourLocal = int.Parse(generalSettings[1].Split('=')[1]);
        radioEndHourLocal = int.Parse(generalSettings[2].Split('=')[1]);

        // Obtain SQL connection string
        dbConnectionStringLocal = DecryptFromBase64(sr.ReadLine()); //As this is already in the correct SQL connection string format, it can be left as it is

        // Obtain radio server settings
        string[] radioServerSettings = DecryptFromBase64(sr.ReadLine()).Split(';'); // This will take the 3rd line and split it by the ';' symbol

        serverVersionLocal = radioServerSettings[0].Split('=')[1];
        serverHostLocal = radioServerSettings[1].Split('=')[1];
        serverPortLocal = radioServerSettings[2].Split('=')[1];
        serverIDLocal = radioServerSettings[3].Split('=')[1];
        serverUsernameLocal = radioServerSettings[4].Split('=')[1];
        serverPasswordLocal = radioServerSettings[5].Split('=')[1];

        // Close the file so it can be written to
        sr.Close();
    }
}
```

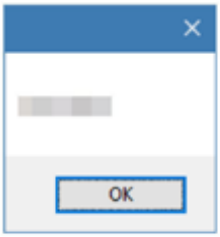
I will also need to create 2 more functions, one to Encrypt text to Base 64, and another to Decrypt text from Base 64. It is a really simple process, you convert the input string to a byte array, then convert that to a base 64 string, and you do the opposite to go back again, as you can see below:

```
// Define global functions to encrypt test to base 64, and decrypt text from base 64
public static string EncryptToBase64(string input)
{
    byte[] byteArray = System.Text.ASCIIEncoding.ASCII.GetBytes(input);
    return Convert.ToBase64String(byteArray);
}

public static string DecryptFromBase64(string input)
{
    byte[] byteArray = Convert.FromBase64String(input);
    return System.Text.ASCIIEncoding.ASCII.GetString(byteArray);
}
```

It is really simple to test, I will make a message box display showing the text entered which has been encrypted then decrypted which should return the same value.

```
MessageBox.Show(Settings.DecryptFromBase64(Settings.EncryptToBase64(  )));
```

This displays the following output, which means both of the functions are now working successfully.



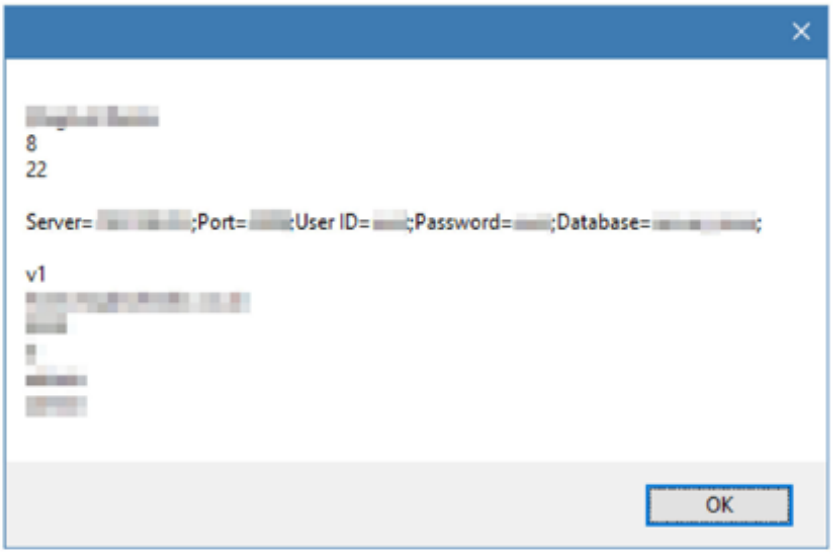

I now need to test and make sure the settings are being read from the file successfully, for this I will create an example Config file which will contain all of the needed settings. I will then call the obtainSettings function, and display the settings via a message box on the screen.

The settings I have entered are as follows:

Name= Start=08;End=22;
Server= ;Port= ;User ID= Password= ;Database= ;
Version= ;Host= ;Port= ID= ;Username= ;Password= ;

When they are Base 64 encrypted it looks like this:
Name= ;Start=08;End=22;
U2VydmVyPTE5Mi4xNjguMC4zO1BvcnQ9MzMwNjtVc2VyIElEPXJvb3Q7UGFzc3dvcmQ9cm9vdDtEYXRhY
mFzZT1zZXJ2ZXJfbW9uOw==
VmVyc2lvbj12MTtIb3N0PWhvc3QubWFnaHVsbHJhZGlvLmNvLnVrO1BvcnQ9ODIyODtJRD0wO1VzZXJuY
W1lPWFkbWluO1Bhc3N3b3JkPWFkbWluOw==

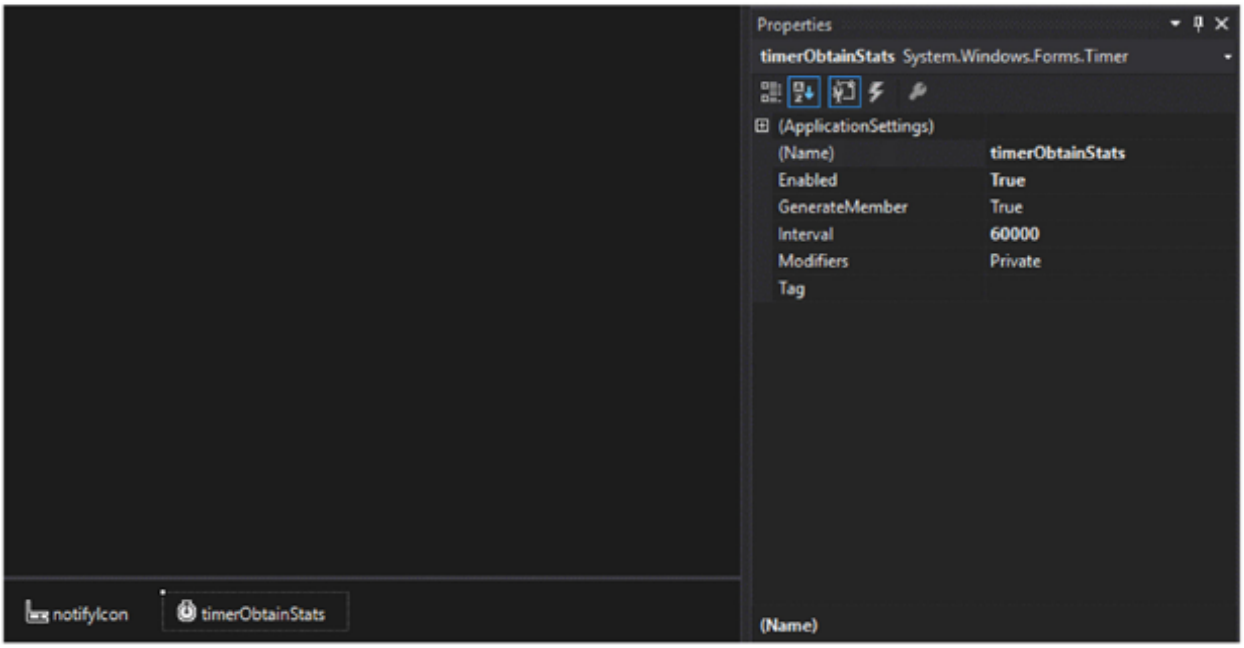
This has been saved to a file called 'config.conf' in the debug directory of my application.
When the application is ran the following message box is displayed:




This matches with the plain text settings above, so now all of the settings are being read and referenced correctly.

Now the settings have been implemented, I can now connect to the radio server and obtain the current song and listener count. This will then be checked against the latest record in the database, if they are different then a new record will be added containing the new information. This will be handled by using a timer.

This is setup like a normal component on the handle form. And the interval has been set to 60,000 milliseconds, this means every 60 seconds the code in the timer's tick event will run, as you can see below:

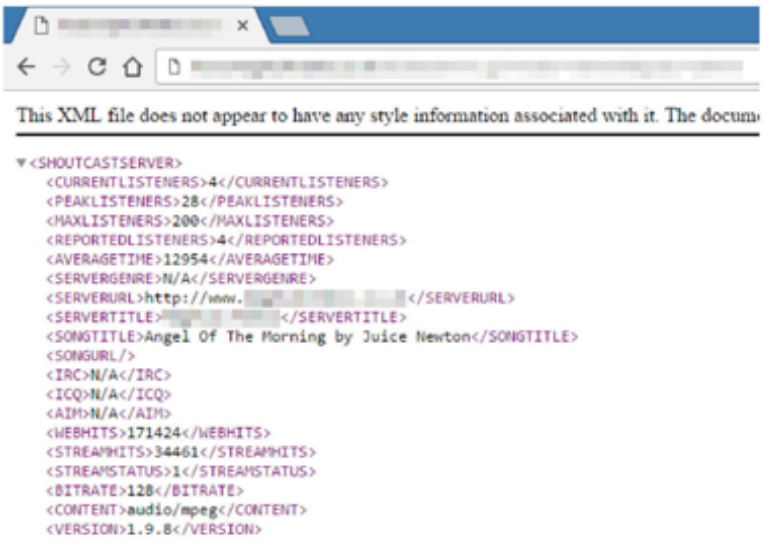



To make it easier when connecting to the server, I have created a new public variable in the settings class, this is the server connection string, and it will hold all of the information entered as a URL which can be navigated to, to pull the server information from.

```
public static string serverConnectionURL { get { return
$"http://{serverHostLocal}:{serverPortLocal}/admin.cgi?mode=viewxml&pass={serverPasswordL
ocal}"; } }
```

I will then test this URL creation to make sure it is in the correct format, by opening it in a web browser using Process.Start when the form loads for testing purposes.

When the application is ran, the following page opens in a web browser:

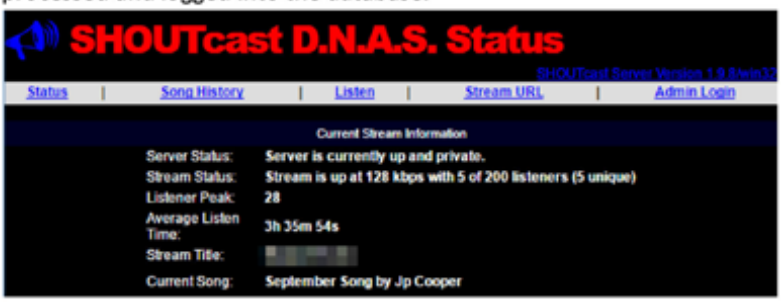


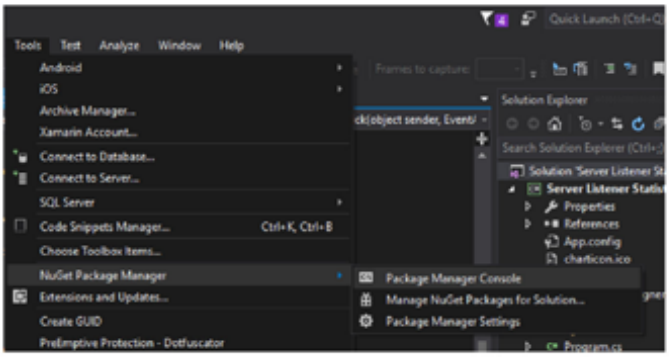

This means that the test has succeeded, and we can now proceed to taking the XML information, and parsing it to return the listener count, and the current song.

I will be using the WebClient object in C# to connect to the XML file, and download it as a string. I can then parse that string for the data I need. This code will go at the top of the timer.Tick event which will run every 60 seconds.

```
private void timerObtainStats_Tick(object sender, EventArgs e)
{
      // Connect to server and obtain statistics
      // Create a new web client to use for the connection to the server
      WebClient client = new WebClient();

      // Set the web headers user agent to the Mozilla firefox browser (as all
      others appear to not work) It's the default internet headers on a computer,
      so it will connect to the server without any issues
      client.Headers["User-Agent"] = "Mozilla/4.0 (Compatible; Windows NT 5.1;
      MSIE 6.0) " + "(compatible; MSIE 6.0; Windows NT 5.1; " + ".NET CLR
      1.1.4322; .NET CLR 2.0.50727)";

      // Store the entire XML result in a variable called result
      string result = client.DownloadString(Settings.serverConnectionURL);

      // Now split to obtain listener count and current song
      // The current song is between two tags of '<SONGTITLE>' and '</SONGTITLE>'
      as it's C# each split needs an array of characters, so it will split a
      string by a string delimiter
      string currentSong = result.Split(new string[] { "<SONGTITLE>" },
      StringSplitOptions.None)[1].Split(new string[] { "</SONGTITLE>" },
      StringSplitOptions.None)[0];

      // Now do the same for the current listener count between
      '<CURRENTLISTENERS>' and '</CURRENTLISTENERS>' also parse it to an integer,
      so it's easier to handle later on
      int currentListenerCount = int.Parse(result.Split(new string[] {
      "<CURRENTLISTENERS>" }, StringSplitOptions.None)[1].Split(new string[] {
      "</CURRENTLISTENERS>" }, StringSplitOptions.None)[0]);

      // Show a message box for testing purposes to make sure the data is being
      obtained correctly
      MessageBox.Show($"{currentSong} - {currentListenerCount}");
}
```

When ran, this results in the following message box being displayed, to the right is a snapshot of the current server info as well. Both of them match, meaning the system is now pulling in the correct data from the server, which can now be processed and logged into the database.

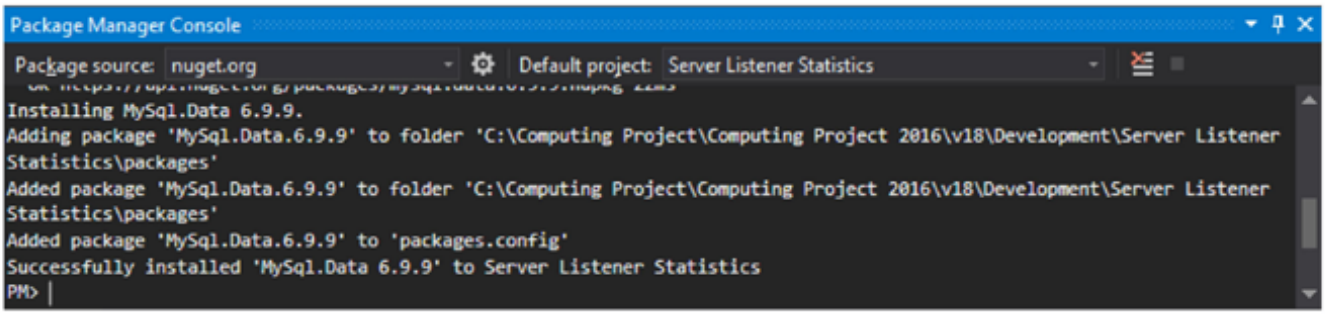



To connect to the database, a reference is needed in the project, this is the MySQL Client.dll file. It will allow us to establish a connection to the database. This can be added using the package manager of Visual Studio.

The package manager console can be opened from Tools -> NuGet Package Manager -> Package Manager Console

The following command can then be ran to download and install all of the MySQL dependencies:

```
Install-Package MySql.Data
```




As you can see from the final message output to the console, the MySql.Data package has been successfully installed to the project. This dependency will also automatically be included when the project is compiled and deployed.

Now all that is left is to add the reference to the top of the handle file, so the MySQL DLL's are imported and can be used within the scope of the rest of the class. This can be done by adding the following line to the top of the file:

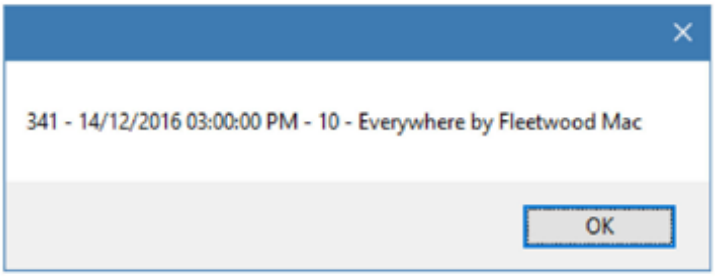
```
using MySql.Data.MySqlClient;
```

Now we have the DLL's to communicate with the SQL server, we can now establish a connection to it, then compare the most recent record, see if it differs from the information we currently have, and if it is different insert a new record into the table.

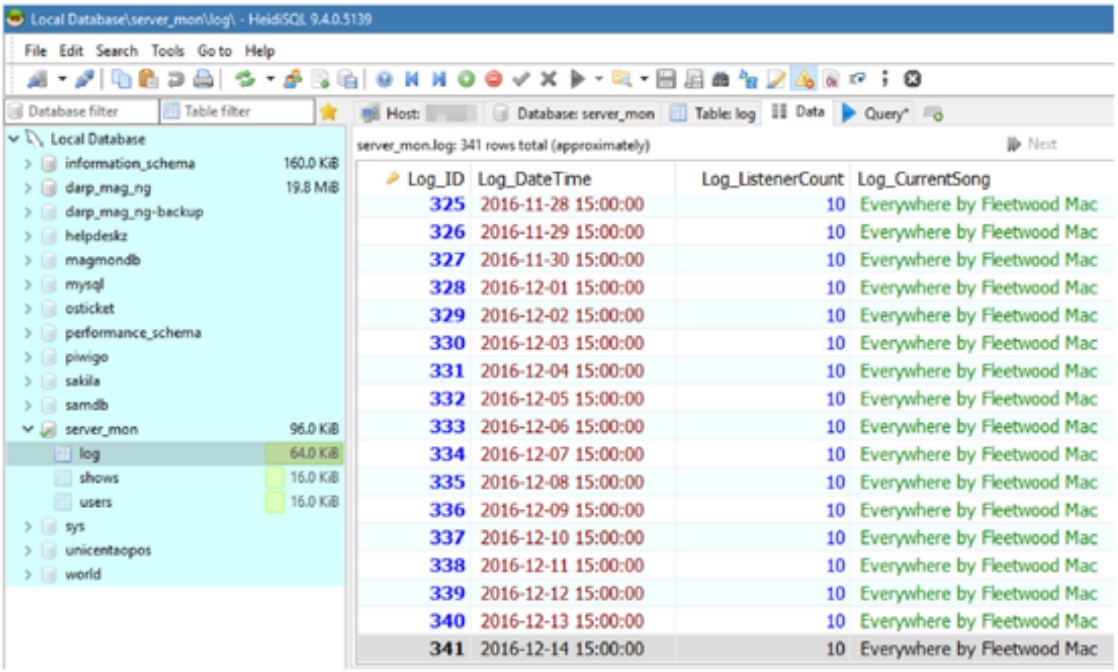
```
// Define MySQL Connection details
MySqlConnection mySqlConn = new MySqlConnection(Settings.dbConnectionString);

// Connect to database
mySqlConn.Open();

// Compare latest record
// Create command to hold query
MySqlCommand command = mySqlConn.CreateCommand();
command.CommandText = "SELECT * FROM log ORDER BY Log_ID DESC LIMIT 1";

// Create new reader to hold results of query
IDataReader reader = command.ExecuteReader();

// Check to make sure values have been returned
if (reader.Read())
{
    MessageBox.Show($"{reader["Log_ID"]} - {reader["Log_DateTime"]} - {reader["Log_ListenerCount"]} - {reader["Log_CurrentSong"]}");
}

// Close the reader so it can be used again with a different set of results
reader.Close();

// Close connection to the database for security reasons
mySqlConn.Close();
```

The code above will create a new connection object from the connection string from the settings file, it will then create a new command which will the run the query against the database. That will then be fed into a reader, which will read through the returned results one by one. Then a message box will be displayed, to test the code is working properly. The SQL command will return the latest record which has been entered into the table.

When the code is ran the following message box appears:




This matches the latest record in the log table (as you can see below) so I can now proceed to comparing this information and seeing if the current statistics match the latest recorded ones.




For the system to communicate with the database, it will need to use the reference to 'MySql.Data.MySqlClient' that we imported earlier at the top of the class.

Here is the code which I will use to connect to the database, find the latest saved record, check to see if there is a difference between the current song name or the current listener count, if so, then work out what current show should be broadcasting, then insert a new record into the database with the current information. This code will go inside the timer.Tick method, after the current code which is already in there

```
// Define MySQL Connection details
MySqlConnection mySqlConn = new MySqlConnection(Settings.dbConnectionString);

// Connect to database
mySqlConn.Open();

// Compare latest record
// Create command to hold query
MySqlCommand command = mySqlConn.CreateCommand();
command.CommandText = "SELECT * FROM log ORDER BY Log_ID DESC LIMIT 1";
```

```
// Create new reader to hold results of query
IDataReader reader = command.ExecuteReader();

// Check to make sure values have been returned
if (reader.Read())
{
        // Check to see if the current statistics are different from the latest database
        statistics
        if ((reader["Log_ListenerCount"].ToString() != currentListenerCount.ToString()) ||
        (reader["Log_CurrentSong"].ToString() != currentSong))
        {
                // Close reader so it can be used again
                reader.Close();

                // If they are different, we first need to work out the current show which
                is on
                string currentShowID = "null";

                command.CommandText = "SELECT Show_ID from shows WHERE Show_Day =
                DAYNAME(NOW()) AND Show_Starttime <= HOUR(NOW()) AND Show_Endtime >
                HOUR(NOW()) AND (Show_StartDate IS NULL OR Show_StartDate < NOW()) AND
                (Show_EndDate IS NULL OR Show_EndDate > NOW());";

                // Update reader to contain new result
                reader = command.ExecuteReader();

                // Check to make sure a show was returned
                if (reader.Read())
                {
                        currentShowID = reader["Show_ID"].ToString();
                }
                reader.Close();

                // Now insert a new record into the database with the new information
                command.CommandText = $"INSERT INTO log(Log_DateTime, Log_ListenerCount,
                Log_CurrentSong, Log_ShowID) VALUES(NOW(), {currentListenerCount},
                \"{currentSong}\", {currentShowID});";

                // Execute command against database
                command.ExecuteNonQuery();
        }


}

// Close the reader so it can be used again with a different set of results
reader.Close();

// Close connection to the database for security reasons
mySqlConn.Close();
```

There are 3 SQL queries executed within this code block, I will explain what each of these do, as they are all called exactly the same in the code.

```
SELECT * FROM log ORDER BY Log_ID DESC LIMIT 1
```

This obtains the latest record from the database's log table. This is done by taking all of the fields, and as each record has an incrementing ID, then we can sort that in descending order, so you are left with the most recent record added to the database.

```
SELECT Show_ID from shows WHERE Show_Day = DAYNAME(NOW()) AND Show_Starttime
<= HOUR(NOW()) AND Show_Endtime > HOUR(NOW()) AND (Show_StartDate IS NULL OR
Show_StartDate < NOW()) AND (Show_EndDate IS NULL OR Show_EndDate > NOW());
```

This returns the show ID of the current show. It is done by using the SQL server's local time (which will be on the same machine running the SQL server so times will be in sync). This is then taken and converted into the current day, from those shows which are on today, it is then filtered to only contain shows which have a start time less than or equal to the current hour. It is then also filtered for shows

which have an end time which is greater than the current hour. Finally the start dates and end dates are also checked (this is for if you had a show which started in January, but you wanted to already have it programmed into your schedule, or if you had to change the presenter or the name of the show, but still wanted to keep historical copies using the correct data). This will then return the current show which is on, otherwise it will return nothing. As there can never be 2 shows on at the same time, and validation will be done when inputting shows into the system, there is no need for a validation check, other than to see if a show has been returned.

The variable currentShowID is a string, while this may seem counter-intuitive at first, the information obtained is being fed back into the database via another SQL query. In SQL any string of numbers without quotes around is treated as an integer. This means whether you stored it as a string or integer would make no difference when updating the database. The only difference occurs when you need to update a field as null, in SQL you need to put the string 'null' in its place. To save time it is therefore easier and more efficient to store the showID as a string, so if there is no show identified, it will still hold its default value of "null", which will then be interpreted as the SQL NULL type when inserted into the database.

```
$"INSERT INTO log(Log_DateTime, Log_ListenerCount, Log_CurrentSong,
Log_ShowID) VALUES(NOW(),{currentListenerCount}, \"{currentSong}\",
{currentShowID});"
```

This will insert a new record into the log table of the database. This is a string argument as part of the MySQL.Command object. This means I can use C#'s new String Interpolation feature, where if you prefix a string with the '$' symbol, it will allow you to reference variables if you surround them with the '{' and '}' symbols. As you can see above, '{currentListenerCount}' will be replaced with the value of the current listener count. The currentSong variable is surrounded with quotes, this is so when the SQL query is ran, it is treated like a whole continuous text string. They are also prefixed with the '\' symbol so they are escaped, as the query itself is surrounded with quotes, so you need to escape any quotes you put inside so they are treated as text.

This code makes up the Timer Tick event which is ran every 60 seconds. To test this I am going to leave it running for an hour and see what results are saved into the database. After this time period you can see the results below:

| | Log_DateTime | | Log_CurrentSong | L... |
|---|---|---|---|---|
| 386 | 2016-12-21 17:26:09 | 2 | Your Station. Your Way. Your Community. | (NULL) |
| 387 | 2016-12-22 16:43:59 | 3 | Don't Wanna Know by Maroon 5 Ft Kendrick ... | (NULL) |
| 388 | 2016-12-22 16:44:58 | 3 | Your Station. Your Way. Your Community. | (NULL) |
| 389 | 2016-12-22 16:45:58 | 3 | Desire by Years And Years | (NULL) |
| 390 | 2016-12-22 16:49:58 | 3 | Get Down (You're The One For M by Backstr... | (NULL) |
| 391 | 2016-12-22 16:52:58 | 3 | I Feel It Coming by The Weekend | (NULL) |
| 392 | 2016-12-22 16:57:58 | 3 | Me Myself And I by G-Eazy Ft Bebe Rexha | (NULL) |
| 393 | 2016-12-22 17:00:58 | 3 | Crazy For You by Let Loose | (NULL) |
| 394 | 2016-12-22 17:04:58 | 3 | Black Beatles by Rae Sremmurd Ft Gucci Mane | (NULL) |
| 395 | 2016-12-22 17:08:58 | 3 | Crazy In Love by Beyonce Ft Jay Z | (NULL) |
| 396 | 2016-12-22 17:12:58 | 3 | How Deep Is Your Love by Take That | (NULL) |
| 397 | 2016-12-22 17:16:58 | 3 | Crazy (Radio) by Gnarls Barkley | (NULL) |
| 398 | 2016-12-22 17:19:58 | 3 | Your Station. Your Way. Your Community. | (NULL) |
| 399 | 2016-12-22 17:20:58 | 3 | Figure 8 (The Alias Radio Edit by Elle Goulding | (NULL) |
| 400 | 2016-12-22 17:24:58 | 3 | All I Want For Christmas Is You by Mariah Carey | (NULL) |
| 401 | 2016-12-22 17:28:58 | 3 | Dance With My Father by Luther Vandross | (NULL) |
| 402 | 2016-12-22 17:32:58 | 3 | Livin' La Vida Loca by Ricky Martin | (NULL) |
| 403 | 2016-12-22 17:36:58 | 3 | One Dance by Drake Ft Wizkid And Kyla | (NULL) |
| 404 | 2016-12-22 17:39:58 | 3 | You're Beautiful by James Blunt | (NULL) |
| 405 | 2016-12-22 17:42:58 | 3 | Your Station. Your Way. Your Community. | (NULL) |
| 406 | 2016-12-22 17:43:58 | 3 | Sexual by Neiked | (NULL) |
| 407 | 2016-12-22 17:47:58 | 3 | Let The Sun Shine by Labrinth | (NULL) |

This has worked successfully. The only issue with this is that the apostrophe is being replaced with ''' to fix this I will modify the following line where the currentSong is set.

Original:

```
string currentSong = result.Split(new string[] { "<SONGTITLE>" },
StringSplitOptions.None)[1].Split(new string[] { "</SONGTITLE>" },
StringSplitOptions.None)[0];
```

New:

```
string currentSong = result.Split(new string[] { "<SONGTITLE>" },
StringSplitOptions.None)[1].Split(new string[] { "</SONGTITLE>" },
StringSplitOptions.None)[0].Replace("&#x27;", "'");
```

After this and running testing for a few more minutes, you can now see from the screenshot below that now apostrophes now appear as apostrophes:

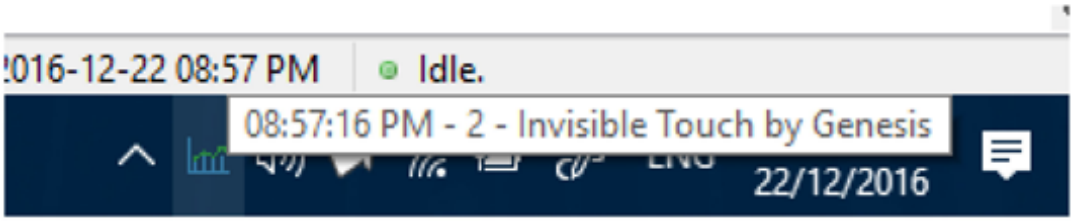
| | | | | |
|---|---|---|---|---|
| 448 | 2016-12-22 20:28:20 | 2 | Dan's test Song | 14 |
| 449 | 2016-12-22 20:32:25 | 3 | Holding Out For A Hero by Bonnie Tyler | 14 |
| 450 | 2016-12-22 20:33:24 | 3 | Holding Out for a 'Hero' | 14 |
| 451 | 2016-12-22 20:35:24 | 3 | Take On Me by A Ha | 14 |
| 452 | 2016-12-22 20:39:24 | 3 | Your Station. Your Way. Your Community. | 14 |
| 453 | 2016-12-22 20:40:24 | 3 | Livin' On A Prayer by Bon Jovi | 14 |

The final part to implement with this section is for the user to be able to see the last recorded added to the database, this can be set to the text of the notification icon. So when the user hovers over it they will see the most recent record.

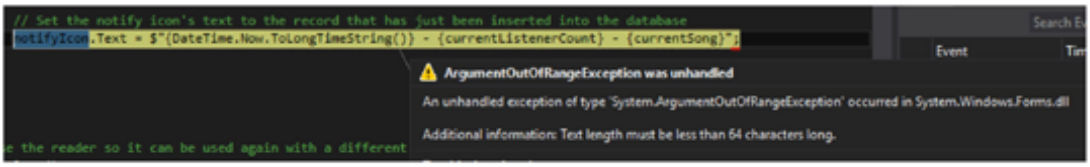
This simply needs another line of code beneath the one where it updates the database with the new information:

```
// Set the notify icon's text to the record that has just been inserted into the database
notifyIcon.Text = $"{DateTime.Now.ToLongTimeString()} - {currentListenerCount} -
{currentSong}";
```

When the application is ran, it produces the following tooltip text when the notification icon is hovered over:




This works great, however on some occasions when being updated an error is thrown:




This is because the length of the song name can vary, while this is not a problem when storing the data into the database, a notify icon can only have a maximum of 63 characters. I will have to modify the update code, and change it so it checks the length of the string about to be set to the label of the notify icon, and it if is over 63 characters to shorten it to 60 characters long, and add an ellipsis (...) onto the end of it to show the song name has been shortened.

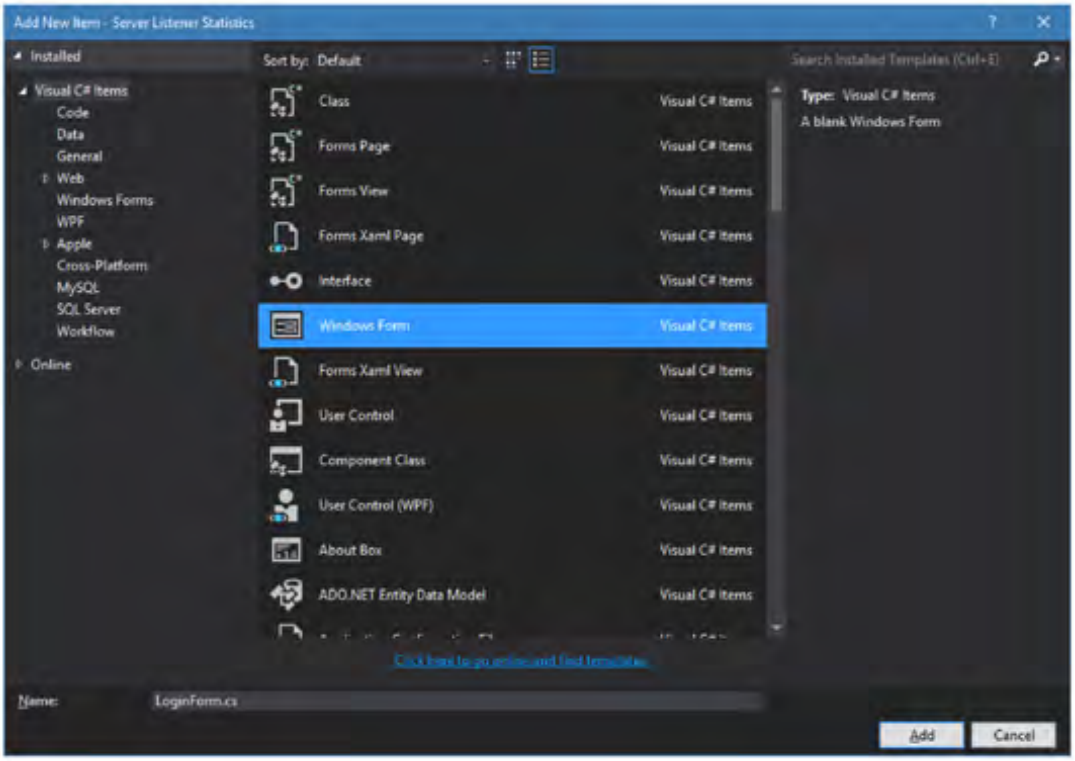
This new code is displayed below:

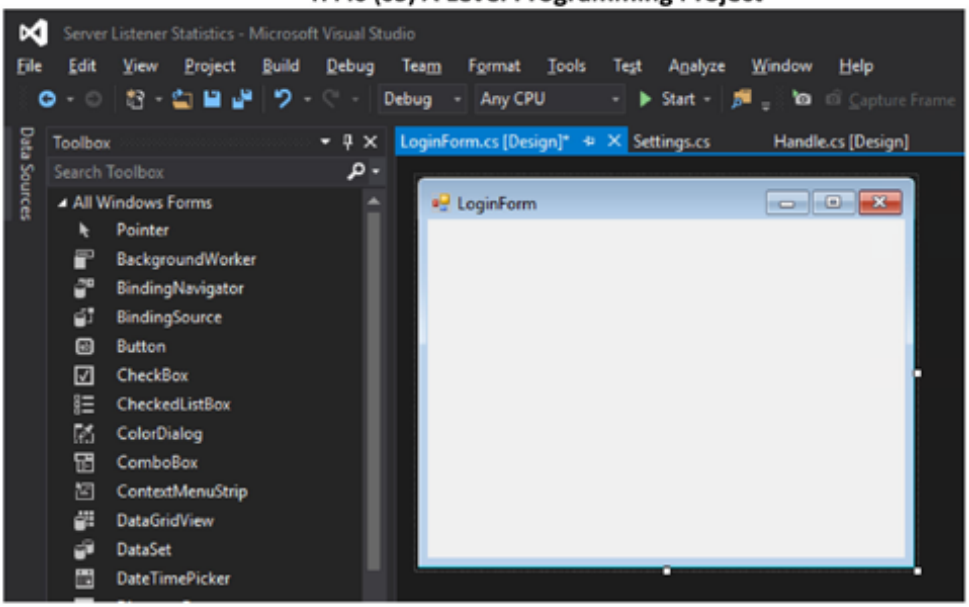
```
//Create a variable to hold the text for the notification icon
string notifyText = $"{DateTime.Now.ToLongTimeString()} - {currentListenerCount} - {currentSong}";

// Check to see if the length is over 63 charachters long
if (notifyText.Length > 63)
{
    // The text is too long, we need the first 60 charachters, and add on an elipsis to the end
    notifyText = $"{notifyText.Substring(0, 60)}...";
}

// Set the notify icon's text to the record that has just been inserted into the database
notifyIcon.Text = notifyText;
```

I will now see what happens if the song "Long Way From Home (Radio) by Fedde Le Grand, Sultan And Ned S" is played, as this will take it over the 63 character limit.

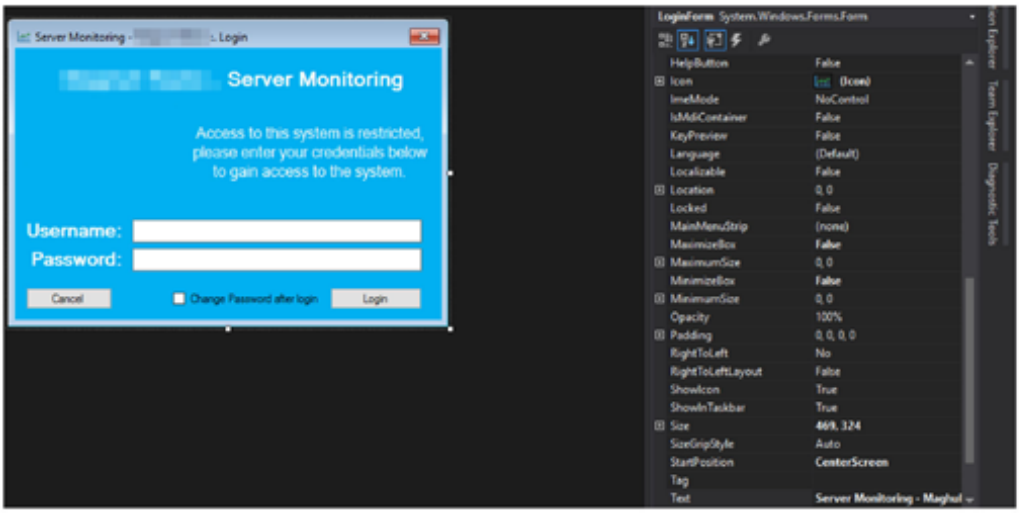

This now works, and is successfully recording all of the data into the database.

The next form to make is the login form, this will open when the user double clicks on the notification icon.






Now the form is added, I can add components and lay it out according to the design earlier.




The components have all been added, the password textbox has "Use System Password Char" set to True, so when the user enters their password it is hidden behind dots. The form has had a custom icon set to match the rest of the application's icons. It has also had "Maximise Box" and "Minimize Box" set to false, along with "FormBorderStyle" set to "FixedSingle". This will prevent the window from being adjusted or minimised. The accept button has also been set to btnLogin, so that when the enter key is pressed, it is the same as pressing the login button.

Now the code can be written to process the user's login, the first thing which is needed is the reference for the MySQL client, this will be required for all forms which need database access, and will be added at the very top of the file.

```
using MySql.Data.MySqlClient;
```

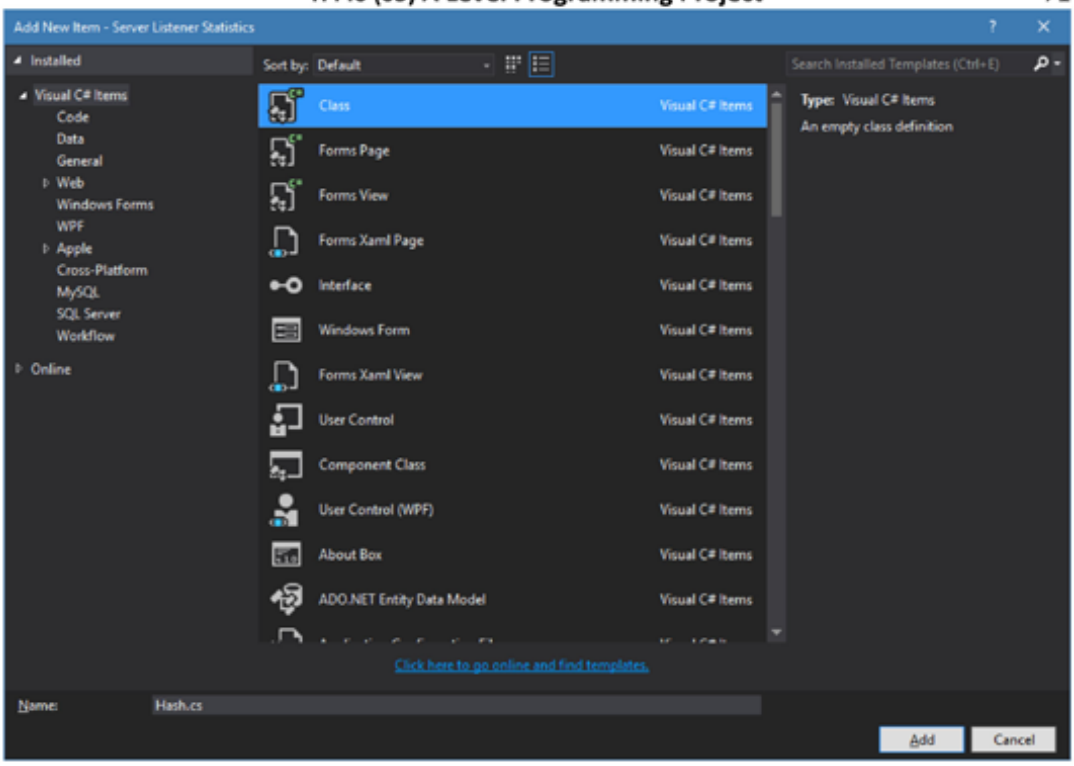
As I did with the Handle form, I will need to define the MySql connection object again. This time it will be above the login form method, so it can be accessed from any child method in the form.

When the form loads I will open a connection to the database. This will be done in the form load method, which will be executed as soon as the form loads.

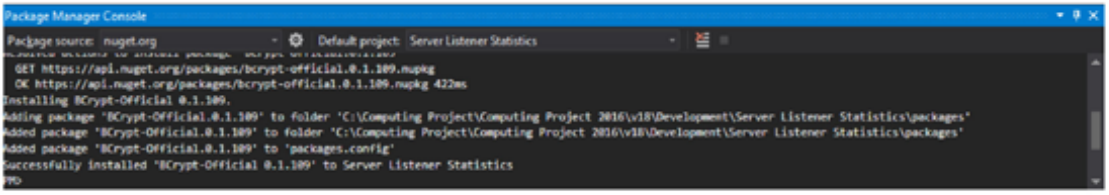
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MySql.Data.MySqlClient;

namespace Server_Listener_Statistics
{
    public partial class LoginForm : Form
    {

        // Define MySQL Connection details
        MySqlConnection mySqlConn = new MySqlConnection(Settings.dbConnectionString);

        public LoginForm()
        {
            InitializeComponent();
        }

        private void LoginForm_Load(object sender, EventArgs e)
        {
            // Open the connection to the databse when the form loads
            mySqlConn.Open();

        }
    }
}
```

Before I proceed any further I need to decide on a form of hashing for the password, the settings were encrypted to the configuration file as they need to be able to be reversed and used. However the passwords will need to be hashed into the database, this is one way so they cannot be reverted back into their plain text form. This will increase the security of the application.

I will create a new class called Hash. This will contain all of the required functions and methods to hash a password, and validate them.




I will be using the bCrypt Nuget package to manage password hashing. This will be installed like the SQL package was by running the command "Install-Package BCrypt-Official" in the Package Manager Console




As you can see from the console output, the BCrypt package was successfully installed, I can now use it in my application. I will add it as an import at the top of the Hash class, so that I can use it to hash passwords, by adding the following line of code:

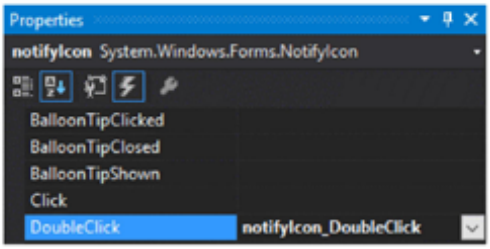
```
using BCrypt.Net;
```

To use this in my application I will need to create 2 functions, one which will generate a salt and hash the password, and another which will take a user's entered password, and check to see if it matches the hashed password.

```
8   namespace Server_Listener_Statistics
9   {
10      class Hash
11      {
12          // Hash password
13          public static string hashPassword(string userPassword)
14          {
15              // This will return a hash (with a salt applied) which can then be stored into a database
16              return BCrypt.Net.BCrypt.HashPassword(userPassword, BCrypt.Net.BCrypt.GenerateSalt(12));
17          }
18
19
20          // Verify password
21          public static bool verifyPassword(string hash, string userPassword)
22          {
23              // This will verify the user's entered password by extrapolating the salt from the hash,
24              // and then hashing the user's password to make sure both of them match
25              return BCrypt.Net.BCrypt.Verify(userPassword, hash);
26          }
27
28      }
29  }
```

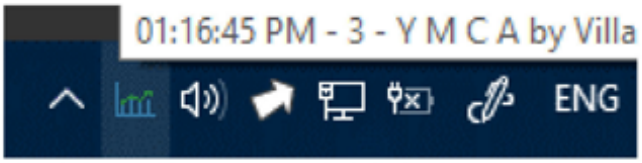
I decided to use the BCrypt package as it is one of the most efficient, reliable and secure packages out there, it is also worked on by a team of developers, so it will be a lot more robust and secure than any cryptography algorithm I was to create myself.

To test that the hashing is working successfully, I will first need to setup the method which will open the login form when the Notification Icon is double clicked on. This can be done by defining a new method in the 'DoubleClick' event of the notify Icon.

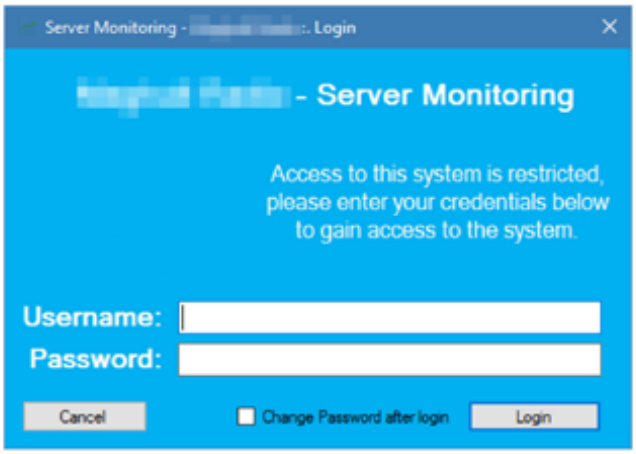



```
private void notifyIcon_DoubleClick(object sender, EventArgs e)
{
    // Create a new instance of the login form and show it to the user
    LoginForm loginForm = new LoginForm();
    loginForm.Show();
}
```

This now means when the icon in the taskbar is double-clicked on, the login form will be displayed as you can see below:




This is the Notification Icon which is double clicked on to open the login form




This is the form which is then displayed after the icon has been double clicked

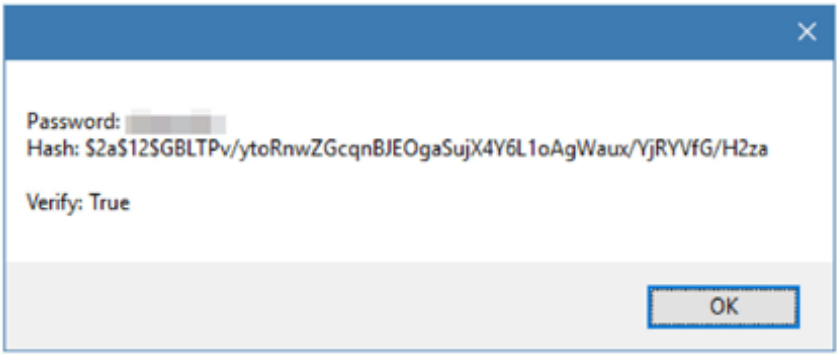
Now I have a working form which opens when the icon is double-clicked on. I now need to test the password hashing function, to make sure it is working correctly.

This can be done by adding some code for testing purposes to the login form's load method, this will be executed when the form is opened. It will simply display a message box, showing an example password, then the hashed version, and then use the validate function of the Hash class to return if the password verification was successful.

```
private void LoginForm_Load(object sender, EventArgs e)
        {
            // Open the connection to the databse when the form loads
            mySqlConn.Open();
```

```
        // The next two lines are for testing purposes to check the password
        hashing is successful
        string passwordHash = Hash.hashPassword( [redacted] );
        MessageBox.Show($"Password: [redacted] \nHash: {passwordHash}\n\nVerify:
        {Hash.verifyPassword(passwordHash, [redacted] )}");
    }
```

When the login form opens, the following message box is displayed:




This means that the hashing function is successfully working, so I can now move onto logging users into the system.

For the system to know which user is currently logged in, I need to create some more global settings, these will hold the user's access level, full name and their username. This will enable various controls to be enabled or disabled depending on the user's access level when the form is loaded.

As I did with the other settings, there was a private writable variable, and a public read-only variable. This is to prevent other parts of the code from changing the values, which should not be changing them.

The new variables are as follows, and are added to the 'Settings' public class:

```
// Private Local User login details
private static string userNameLocal = "";
private static string fullNameLocal = "";
private static int accessLevelLocal = 0;

// Read-only public User login details
public static string userName { get { return userNameLocal; } }
public static string fullName { get { return fullNameLocal; } }
public static int accessLevel { get { return accessLevelLocal; } }
```

I will need to create a method to update the local variables, which will take 3 inputs, and then assign them to the variables (again still in the settings class)

```
// Create method to update the user's login details
public static void updateLoginDetails(string userName, string fullName, int accessLevel)
{
        userNameLocal = userName;
        fullNameLocal = fullName;
        accessLevelLocal = accessLevel;
}
```

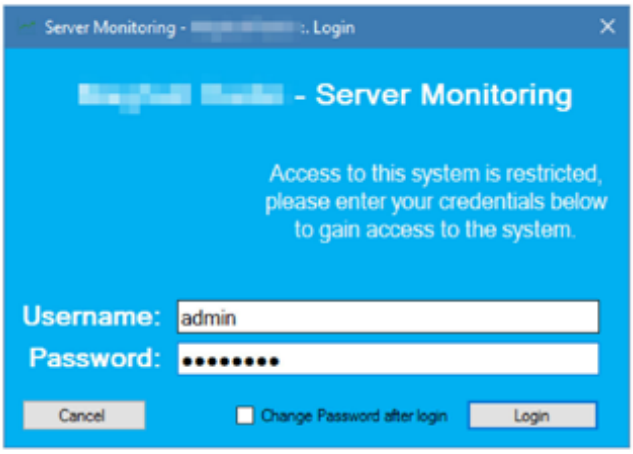
Now I can create the code for when the user presses the login button:

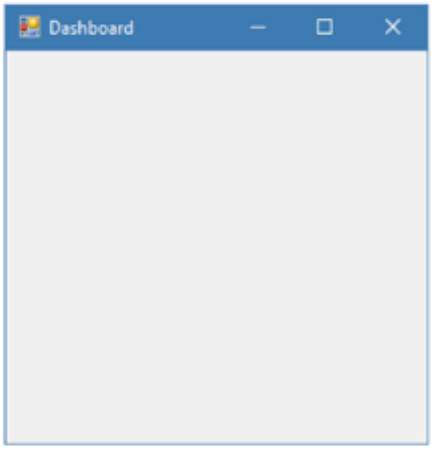
```
private void btnLogin_Click(object sender, EventArgs e)
{
        // Create a new MySQL command to query the database for the username the user has
        entered
        MySqlCommand command = mySqlConn.CreateCommand();
        command.CommandText = $"SELECT * FROM Users WHERE `User_LoginName` =
        \"{txtUsername.Text}\"";
```

```
// Create a data reader to hold results of the query
IDataReader reader = command.ExecuteReader();

// Check to make sure the reader returned a value
if (reader.Read())
{
       // As the username exists in the database, check to make sure the password
       the user entered is correct
       if (Hash.verifyPassword(reader["User_Password"].ToString(),
       txtPassword.Text))
       {
              // User has entered the correct password, update the details in the
              global settings
              Settings.updateLoginDetails(txtUsername.Text,
              reader["User_FullName"].ToString(),
              int.Parse(reader["User_AccessLevel"].ToString()));

             // Check to see if the user wants to change their password
             if (cbChangePassword.Checked)
             {
                    // If the checkbox is checked, then the user wants to change
                    their password, open the change password form
                   ChangePassword changePassword = new ChangePassword();
                   changePassword.ShowDialog();
             }

             // Update database with new login time
              command.CommandText = $"UPDATE users SET `User_LastLogin` = NOW()
              WHERE `User_ID` = {reader["User_ID"].ToString()}";

             // Close the reader so another command can be run
             reader.Close();

             // Run query to update database
             command.ExecuteNonQuery();

             // Show the dashboard form
             Dashboard dashboard = new Dashboard();
             dashboard.Show();

             // Close connection to the database
             mySqlConn.Close();

             // Hide the login form, so the user only sees the dashboard form
             this.Hide();

              // Reset the controls on the login form to be blank values for when
              it is displayed again
             txtUsername.Text = "";
             txtPassword.Text = "";
             cbChangePassword.Checked = false;

        }
        else
        {
            // The password was incorrect, display an error message to the user
             MessageBox.Show("The username or password is incorrect. Please try
             again", "Invalid Credentials!", MessageBoxButtons.OK,
             MessageBoxIcon.Error);

             // Reset the values of the controls on the login form for the user
             to try again
            txtPassword.Text = "";
            txtUsername.Focus();
            txtUsername.SelectAll();
        }
    }
    else
    {
```

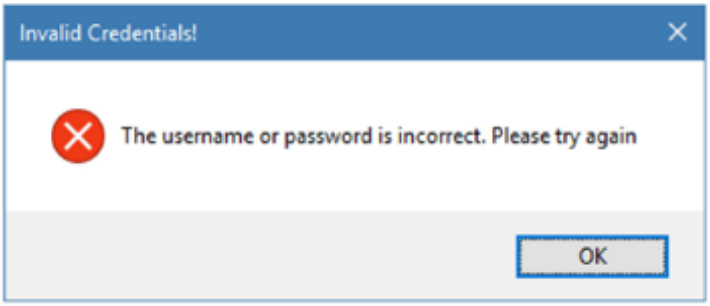
```
            // The username was not found, display an error message to the user
            MessageBox.Show("The username or password is incorrect. Please try again",
            "Invalid Credentials!", MessageBoxButtons.OK, MessageBoxIcon.Error);

            // Reset the values of the controls on the login form for the user to try
            again
              txtPassword.Text = "";
              txtUsername.Focus();
              txtUsername.SelectAll();
        }

        // Close the data reader so it can be used again
        reader.Close();

    }
```

Now the code can be tested, I have already updated my test user record in the database with a hashed password of 'password'. So the login details should be 'admin' and 'password'.




As you can see, when the correct details are entered, the dashboard form is displayed. If the incorrect details are entered, the following error message appears:




This also happens if no data is entered for either text box. The only issue occurs when a user tries to type a quote in the text field. This causes an SQL syntax error, so I will need to escape all single and double quotes with a backwards slash in front of them. This can be done by modifying the following line of code:
Original:

```
command.CommandText = $"SELECT * FROM Users WHERE `User_LoginName` =
\"{txtUsername.Text}\"";
```

New code:

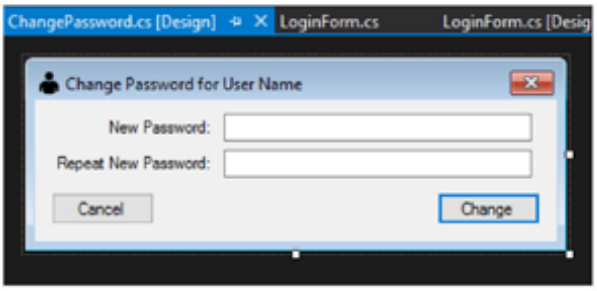
```
command.CommandText = $"SELECT * FROM Users WHERE `User_LoginName` = @username";
command.Parameters.AddWithValue("@username", txtUsername.Text);
```

This is using the SQL command parameters instead of passing it in normally. This means it is treated as a value rather than any command which can be executed, meaning there is no risk of any SQL injection, or any errors caused by SQL syntax.

Finally, after logging into the application, you can see that the database record for the user has been updated with the current time as the last login time for the user.

| | User_Full... | Use... | User_Email | User_Password | | User_LastLogin |
|---|---|---|---|---|---|---|
| 1 | [redacted] | admin | [redacted] | $2a$12$Ru0mwUj6gGKZ/xGz6Ri48OBfJ5sXctLcPU6JCo0nVXg4u/SSr/OyC | 3 | 2016-12-30 15:24:38 |

Now I can create the change password form. This when loaded from the login form, will take the username settings from the global application settings. Then ask the user for a new password, it will then check both passwords match, if so it will update the settings in the database, if not it will ask the user to try again. It will then hand back to the login form, which will then open up the dashboard



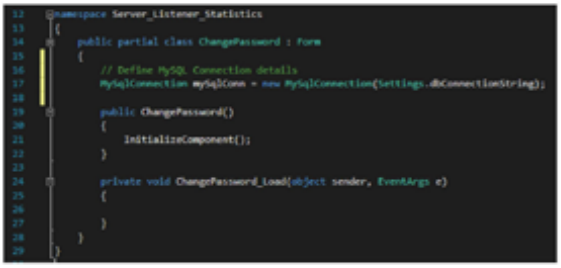

As you can see, the form contains two text boxes for the user to type in their new password, and to repeat it for confirmation. It also has 2 buttons, one to accept the password change, and another one to cancel it.

The form itself has had its form border style set to "Fixed Dialog" so that the minimise and maximise buttons disappear, and the form can not be resized. It has also had its icon change to a picture of a user; along with the accept and cancel buttons being set to the relevant Change and Cancel buttons on the form. This means when the user presses the escape key, the cancel button will be pressed, and the same with the enter key, the accept button will be pressed. Finally, both of the text boxes on the form have their "Use System Password Char" set to True, so that the password the user types in his hidden behind dots for security.

Now the form has been designed, I can move onto the coding behind it. Firstly as the form is connecting to the database, the MySQL import is needed at the top of the file:

```
using MySql.Data.MySqlClient;
```

I will also add the MySQL connection string to the top of the file to be referenced later

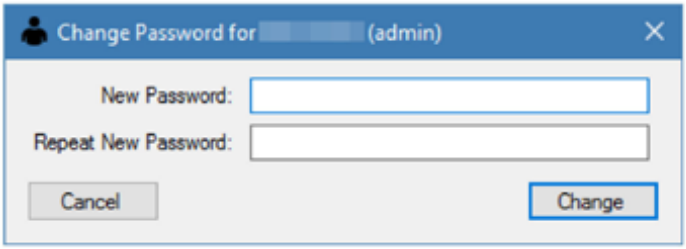


Next I will need the form to obtain the settings from the global settings class, and update the title of the form to match the currently logged in user.

This will be done in the load event of the change password form.

```
private void ChangePassword_Load(object sender, EventArgs e)
        {
            this.Text = $"Change Password for {Settings.fullName} ({Settings.userName})";
        }
```

When the form is ran (by logging in with the 'Change Password' checkbox checked, the following is displayed:

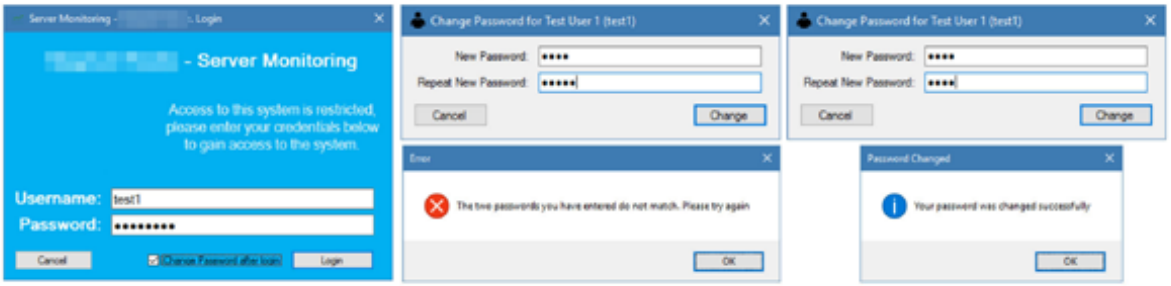



This is correctly displaying the current user's full name and password. Now I can add the code when the 'Change' button is pressed.

```
private void btnChange_Click(object sender, EventArgs e)
{
    // Check to make sure the password the user entered is the same as the repeat password
    if (txtNewPassword.Text == txtRepeatNewPassword.Text)
    {
        // Both of the passwords the user has enetered match, update the user's password in the database

        // Create a new command which will hash and update the user's password
        MySqlCommand command = mySqlConn.CreateCommand();
        command.CommandText = $"UPDATE users SET `User_Password` = \"{Hash.hashPassword(txtNewPassword.Text)}\" WHERE `User_LoginName` = \"{Settings.userName}\"";

        // Run the command
        command.ExecuteNonQuery();

        // Display a success message to the user
        MessageBox.Show("Your password was changed successfully", "Password Changed", MessageBoxButtons.OK, MessageBoxIcon.Information);

        // Close the form
        this.Hide();
    }
    else
    {
        // The two passwords did not match, alert the user
        MessageBox.Show("The two passwords you have entered do not match. Please try again", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

    }
}
```

There is no need to do any user validation with the change password form, due to the fact the form is only displayed if the user was previously successfully authenticated. This code will check both passwords entered match, if they do it will then update the value for the current logged in user in the database and then show a success message to the user. If the passwords do not match, an error message is displayed to the user to tell them to try again.

This can now be tested by changing the passwords of some test users.

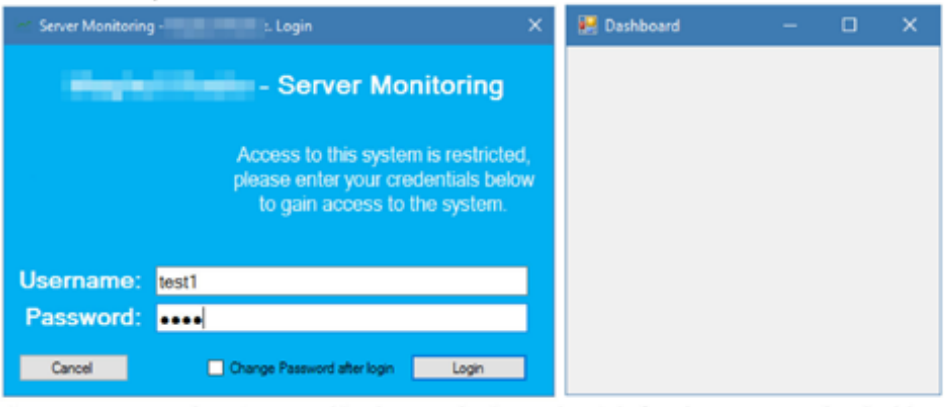



Hash in database:
**$2a$12$Gl.a7gY3AYF7Yvg2jxHgPuj0GMXcHwDvNNhx0RtFCXK9onW.5AKo.**

Hash in database:
**$2a$12$Gl.a7gY3AYF7Yvg2jxHgPuj0GMXcHwDvNNhx0RtFCXK9onW.5AKo.**

Hash in database:
**$2a$12$5bfOYR8GsCJ3TIEp.XNhX.39.mQ1t5OcX1nmK2zpLeOCGDLGAmJeG**

As you can see. The password was successfully updated in the database; as the hash on the right is different to the one before the password change. Now all that is left to test is to make sure I can log in with the new password:




As you can see, when I entered in the new login credentials for the test user the dashboard opened, so now the change password function is successfully working.

Now the next form to develop is the Dashboard form. This will display an overview of current statistics, based off the form I designed in the design section earlier.

Firstly it will need to display in the title the name of the currently logged in user, this is really easy at that is saved in the settings class. All I need to do is add the following line of code to the form's Dashboard_Load event.

```
// Set title of form to the currently logged in user
this.Text = $"{Settings.radioName} - Server Monitoring [Logged in as {Settings.fullName}]";
```

Now when the dashboard form opens it has the following title:

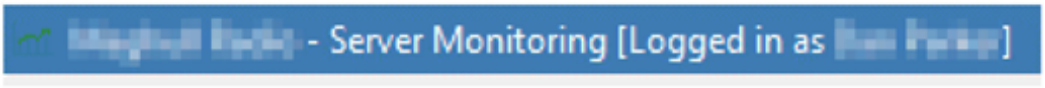

This can now be tested for the other two accounts "Test 1" and "Test 2".



This works and will update for all of the other users.

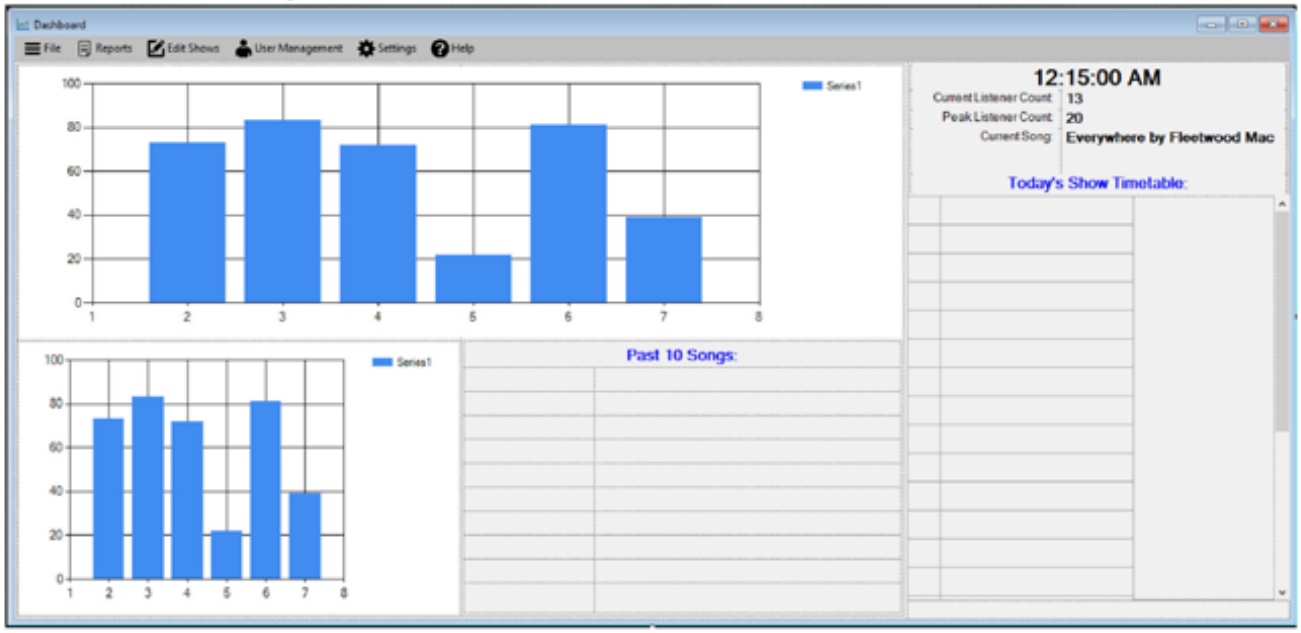
Next I need to define two properties on the dashboard form, firstly I will give it the same icon as all of the other forms, by setting that in the Icon option. And I will also set the form's window state to Maximized, so that when it opens it will be full screen on any sized display.

You can now see the new icon on the form:




Now I need to add the various components to the form to allow it to match the form design I created earlier. This will be a menu strip; a status bar; two graphs and two table layout panels.

These are arranged as shown below:




This will allow the dashboard to scale correctly. There are blank placeholders on the screen, as when the form first loads, labels will be dynamically created and added to the correct rows of the table layout panel for the timetable hours, and for the placeholder labels for the past 10 songs.

Firstly however, there needs to be a timer added which is responsible solely for updating the time label on the form. It will be set to enabled, and will have an interval of 1000milliseconds which is every second. This will be called "TimerUpdateTime" and will have the following inside its Timer.Tick method:

```
// Update the time label on the form with the current time
lblCurrentTime.Text = DateTime.Now.ToString();
```

Now I can add the method which will add hour labels to each row on the timetable layout. However before I do this, I will add a colours section to the global settings class, so if any colour schemes need to be changed later on down the line, it can easily be done.

```
// Define global editable colour settings
public class colours
{
        public static Color LightBlue = Color.FromArgb(222, 235, 247);
        public static Color Blue = Color.FromArgb(0, 176, 240);
        public static Color Orange = Color.FromArgb(255, 192, 0);
        public static Color LightGrey = Color.FromArgb(241, 240, 240);
}
```

Now I can add a method which will add the individual hour marks to the schedule, this will be a public method but will only be called when the information needs to be refreshed or updated.

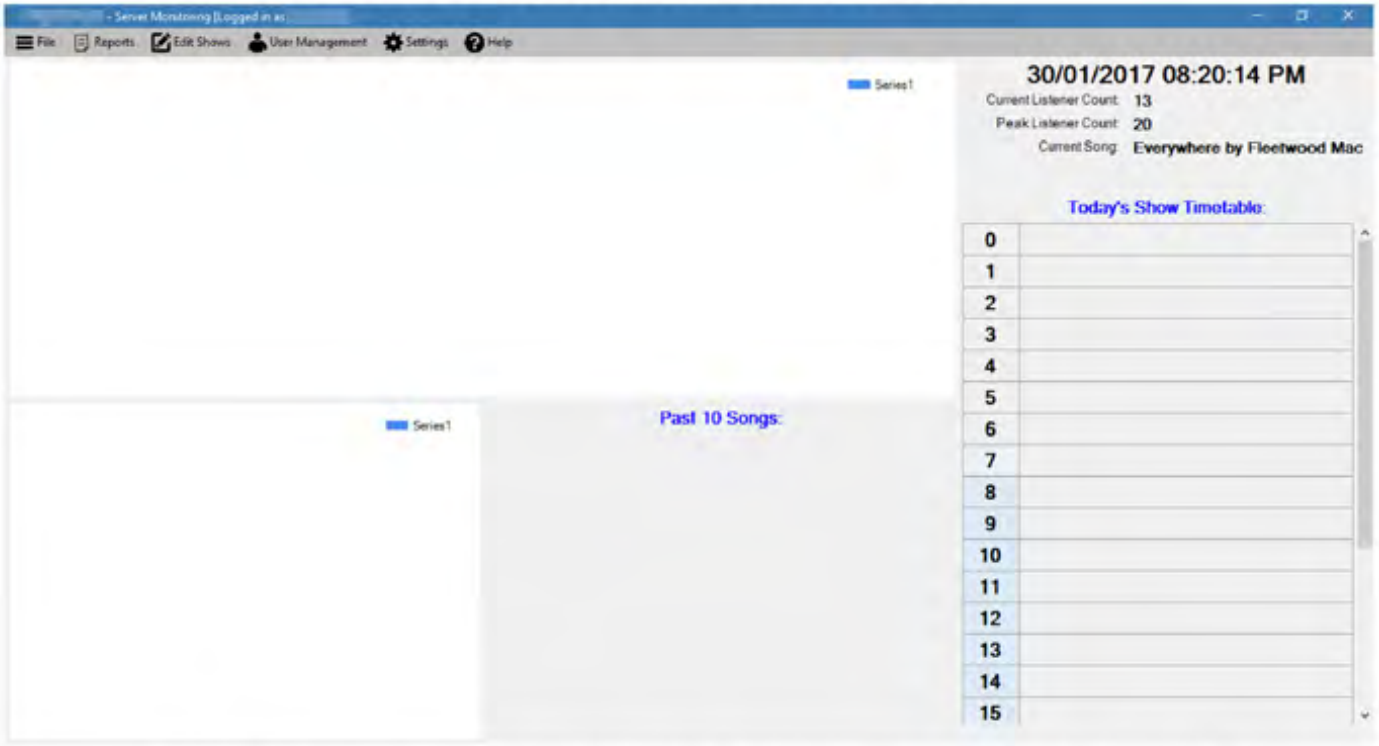
```
// Hour Marks
private void addHourMarks()
{
    // Add labels from 0 to 23 to the show timetable
    for (int i = 0; i < 24; i++)
    {
        // Create a new label object to hold current hour
        Label lblTitle = new Label();

        // Set the text to be the current hour
        lblTitle.Text = i.ToString();

        // Align text to centre, set Auto-Size to false, set the margain to 0, set the Dock style to fill,
        // and set the back colour to the Light Blue colour defined in the settings
        lblTitle.TextAlign = ContentAlignment.MiddleCenter;
        lblTitle.AutoSize = false;
        var margin = lblTitle.Margin;
        margin.All = 0;
        lblTitle.Margin = margin;
        lblTitle.Dock = DockStyle.Fill;
        lblTitle.BackColor = Settings.colours.LightBlue;

        // Give the label a name of 'lblScheduleHour' followed by the hour it represents so it can be referenced later
        // and set the font to by Mircoroft Sans Serif, Size 14 in bold
        lblTitle.Name = "lblScheduleHour" + i;
        lblTitle.Font = new Font("Microsoft Sans Serif", 14, FontStyle.Bold);

        // Check to see if the hour is within the radio operating hours, if it is not
        // set the background colour to be a Light Grey from the settings
        if (i < Settings.radioStartHour || i > Settings.radioEndHour)
        {
            lblTitle.BackColor = Settings.colours.LightGrey;
        }

        // Add the label to the timetable layout panel, in the first coloumn in the row of the hour it represents
        tableLayoutSchedule.Controls.Add(lblTitle, 0, i);

    }
}
```

I will now add a call to run this method on the form's load event, so it is ran when the form is first opened. I will also set the text of the time label to be the current time, so that it displays the correct time when the form first opens:

```
private void Dashboard_Load(object sender, EventArgs e)
{

        // Add hours to timetable
        addHourMarks();

        //Set label to current time
        lblCurrentTime.Text = DateTime.Now.ToString();

}
```

**H446 (03) A Level Programming Project** 80

When the dashboard is opened by a user logging in the following is displayed:



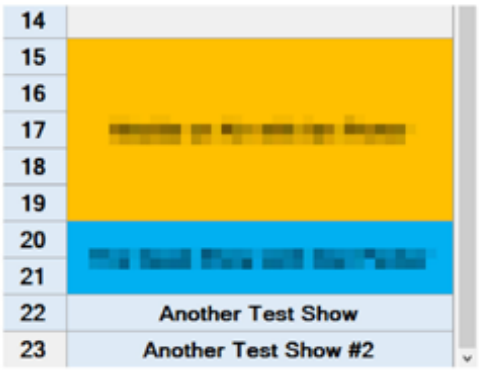

Now I can work on populating the timetable to list the current shows for that current day. I will firstly create a function that will add a label for a show with the specified length to the correct position on the schedule. Then I will integrate this to obtain information from the database.

```
// Add show to timetable
private void addShow(string ShowName, int StartHour, int EndHour, Color Colour)
{
    // Create a new label object to hold the show name
    Label lblShow = new Label();

    // Set the text to be the value of ShowName which was passed into the method
    lblShow.Text = ShowName.ToString();

    // Align text to centre, set Auto-Size to false, set the margain to 0, set the Dock style to fill,
    // and set the back colour to the colour passed into the method called Colour
    lblShow.TextAlign = ContentAlignment.MiddleCenter;
    lblShow.AutoSize = true;
    lblShow.Dock = DockStyle.Fill;
    lblShow.BorderStyle = BorderStyle.None;
    var showmargin = lblShow.Margin;
    showmargin.All = 0;
    lblShow.Margin = showmargin;
    lblShow.BackColor = Colour;

    // Give the label a name of 'lblShow' followed by the start hour and the end hour so it can be referenced later
    // and set the font to by Mircoroft Sans Serif, Size 13 in bold
    lblShow.Name = "lblShow" + StartHour + EndHour;
    lblShow.Font = new Font("Microsoft Sans Serif", 13, FontStyle.Bold);

    // Calculate the length of the program, so the number of rows the label needs to span can be calculated
    int HourSpan = EndHour - StartHour;

    // Add the label to the timetable and set the rowspan to the integer variable 'HourSpan'
    tableLayoutSchedule.Controls.Add(lblShow, 1, StartHour);
    tableLayoutSchedule.SetRowSpan(lblShow, HourSpan);

}
```

To test the code I will add a few test shows to the timetable, there is no need to validate any data being passed into the method, as it will be obtained from the database so the data will have already been validated before the record was inserted. This can be done by adding the following lines to the end of the form's load method just for testing purposes:

```
// Add test shows
addShow("[redacted]", 15, 20, Settings.colours.Orange);
addShow("[redacted]", 20, 22, Settings.colours.Blue);
addShow("Another Test Show", 22, 23, Settings.colours.LightBlue);
addShow("Another Test Show #2", 23, 24, Settings.colours.LightBlue);
```

This produces the following in the show timetable when the form is opened:




This shows that all of the shows have been added successfully, and that the add show method is working correctly.

Now I can implement this with the show timetable obtained from the database.

I will need to add another import reference at the top of the class as I will be accessing the database, this is as follows:

```
using MySql.Data.MySqlClient;
```

I will also need to create a new connection to the database with the following line added at the top of the form class:

```
// Define MySQL Connection details
MySqlConnection mySqlConn = new MySqlConnection(Settings.dbConnectionString);
```

Next I need to open a connection to the database, this will be done with the following line added to the form's load method:

```
// Open connection to the database
mySqlConn.Open();
```

Now I can create another method called updateTimetable, this will clear all of the labels in the form, then it run a query to obtain a list of all shows that day, then it will add them to the schedule highlighting the current show in orange, along with the current hour in orange as well:

```
// Update show timetable
public void updateTimetable()
{
        // Suspend the layout to prevent flickering while updating
        tableLayoutSchedule.SuspendLayout();

        // Clear all of the components from the timetable
        tableLayoutSchedule.Controls.Clear();

        // Add hour marks to the timetable
        addHourMarks();

        // Get current day
        string currentDay = DateTime.Now.DayOfWeek.ToString();

        // Get current hour
        int currentHour = DateTime.Now.Hour;

        // Create a new command to list all active shows that are on today
        MySqlCommand command = mySqlConn.CreateCommand();
```

```
        command.CommandText = $"SELECT * FROM `shows` WHERE `Show_Day` = '{currentDay}'
        AND (Show_StartDate IS NULL OR Show_StartDate < NOW()) AND (Show_EndDate IS NULL
        OR Show_EndDate > NOW()) ORDER BY `Show_Starttime` ASC";

        IDataReader reader = command.ExecuteReader();

        while (reader.Read())
        {
                // The show name needs to have a regex replace of any new line or
                terminator character due to a space happening prefixed to start of string
                when read from the database
                string ShowName = Regex.Replace(reader["Show_Name"].ToString(),
                @"\t|\n|\r", "");

                // Create variables to hold show information
                string ShowPresenter = reader["Show_Presenter"].ToString();
                int ShowStartTime = int.Parse(reader["Show_Starttime"].ToString());
                int ShowEndTime = int.Parse(reader["Show_Endtime"].ToString());

                // Create a temporary variable to hold the colour of the show (this will be
                changed if show is currently live)
                Color currentShowColor = Settings.colours.Blue;

                // The ShowEndTime needs to have 1 taken away from it, so two hours are not
                highlighted if it is the final hour of the show
                if ((currentHour >= ShowStartTime) && (currentHour <= ShowEndTime - 1))
                {
                        currentShowColor = Settings.colours.Orange;
                }
                else
                {
                        currentShowColor = Settings.colours.Blue;
                }

                //Add show to schedule
                addShow($"{ShowName} with {ShowPresenter}", ShowStartTime, ShowEndTime,
                currentShowColor);

        }

        // Highlight current hour
        tableLayoutSchedule.Controls.Find($"lblScheduleHour{currentHour.ToString()}",
        false)[0].BackColor = Settings.colours.Orange;

        // Resume the layout so the user can see the updated shows
        tableLayoutSchedule.ResumeLayout();

        // Close the data reader so it can be used again
        reader.Close();
}
```

This code contains the following SQL query, which I will test externally before implementing it:

```
SELECT * FROM `shows` WHERE `Show_Day` = '{currentDay}' AND (Show_StartDate IS NULL OR
Show_StartDate < NOW()) AND (Show_EndDate IS NULL OR Show_EndDate > NOW()) ORDER BY
`Show_Starttime` ASC
```

This gives the following results when ran for Friday:

| | Show_Name | Show_Presenter | Sho... | ... | ... | Sho... | Sho... | |
|---|---|---|---|---|---|---|---|---|
| 15 | [redacted] | [redacted] | Friday | 9 | 11 | (NULL) | (NULL) | 1 |
| 16 | [redacted] | [redacted] | Friday | 12 | 15 | (NULL) | (NULL) | 1 |
| 8 | [redacted] | [redacted] | Friday | 15 | 16 | (NULL) | (NULL) | 1 |

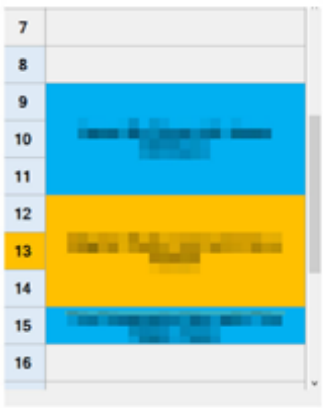
Even though the shows have a unique ID, you can see they have been ordered in chronological order based off time, so that they will follow in the schedule. Now I know the query is working, I can test the updateTimetable function. I will do this by calling the function every time the help button is pressed, just for testing purposes.

When I load the form and press the Help button, the following time table is displayed:


This matches up with the information that was queried earlier, so the function is working great. I can now remove the test hours I added on the load of the form.

I have changed the data of one of the shows, so that the first show ends at 12am, rather than 11am, I will press the Help button again to make sure it updates properly.

| | Show_Name | Show_Presenter | Sho... | ... | ... | Sho... | Sho... | |
|---|---|---|---|---|---|---|---|---|
| 15 | [redacted] | [redacted] | Friday | 9 | 12 | (NULL) | (NULL) | 1 |
| 16 | [redacted] | [redacted] | Friday | 12 | 15 | (NULL) | (NULL) | 1 |
| 8 | [redacted] | [redacted] | Friday | 15 | 16 | (NULL) | (NULL) | 1 |




This is the new data it should display in the timetable. As you can see to to the left, it has updated to change to the correct display.

Now the timetable is successfully working, I can now move onto getting the graphs working. There are 2 on the form, the weekly listener graph, which will display the past 7 days of listener statistics, including the peak and average listener values for that day. There will also be another graph which will display the current listener and peak values for that minute.

I will start off on the 7 day graph, first I will need to create another query that will select data from the past 7 days, and then calculate the peak listener value, along with the average listener value, and then group it on each days date. Luckily I can use the 'group by' and 'sort by' functions in SQL.

The query I will use is as follows:

```
SELECT CONCAT(DAY(Log_DateTime), '/', MONTH(Log_DateTime), '/', YEAR(Log_DateTime)) AS
'Date', ROUND(AVG(Log_ListenerCount), 0) AS 'Average', MAX(Log_ListenerCount) AS 'Peak'
FROM log WHERE Log_DateTime between AddDate(NOW(), -7) and NOW() GROUP BY
DAY(log.Log_DateTime) ORDER BY Log_DateTime ASC;
```

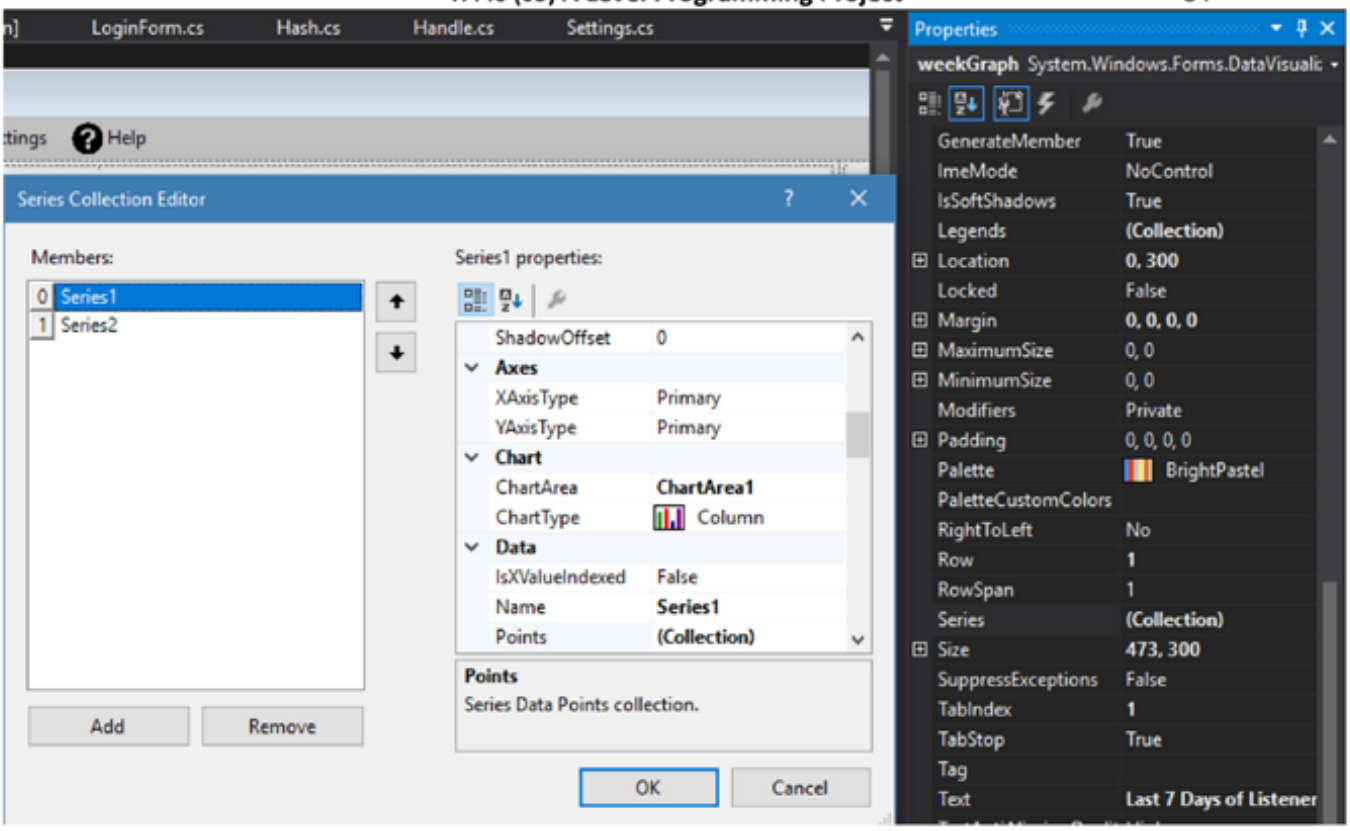
When ran it returns the following data:

| Date | Average | Peak |
|---|---|---|
| 30/1/2017 | 4 | 6 |
| 31/1/2017 | 4 | 6 |
| 1/2/2017 | 4 | 7 |
| 2/2/2017 | 3 | 4 |
| 3/2/2017 | 3 | 4 |

This only contains 5 records, as I have left the system running for the past 5 days to allow for test data to be collected. It is also in reverse order so that when it is plotted on the graph, it goes in chronological order, as it plots points from left to right.

Before this data is plotted on the graph, it needs to be setup correctly to display the data in a useful form.

Firstly I need to add another series to the graph, this is as I will be displaying two sets of data on the one graph. This can be done by opening the series collection under the graph's properties as you can see below. There is no need to name them or define any settings, as this can be done in code later on:




Now I can create a function called 'setupWeekGraph' this will define all of the options needed on the graph to enable the data to be displayed correctly.

```
private void setupWeekGraph()
{
        // Setup X Axis
        weekGraph.ChartAreas[0].Axes[0].Title = "Day";
        weekGraph.ChartAreas[0].Axes[0].IsLabelAutoFit = true;
        weekGraph.ChartAreas[0].Axes[0].LabelAutoFitStyle =
        LabelAutoFitStyles.LabelsAngleStep30;
        weekGraph.ChartAreas[0].Axes[0].LabelStyle.Enabled = true;

        // Setup Y Axis
        weekGraph.ChartAreas[0].Axes[1].Title = "No. Listeners";
        weekGraph.ChartAreas[0].Axes[1].IsLabelAutoFit = true;
        weekGraph.ChartAreas[0].Axes[1].LabelAutoFitStyle =
        LabelAutoFitStyles.LabelsAngleStep30;
        weekGraph.ChartAreas[0].Axes[1].LabelStyle.Enabled = true;

        // Setup the series to hold the average listener count as a normal line graph in
        blue with a bold line with markers
        weekGraph.Series[0].ChartType = SeriesChartType.Line;
        weekGraph.Series[0].MarkerStyle = MarkerStyle.Circle;
        weekGraph.Series[0].LegendText = "Average Listener Count";
        weekGraph.Series[0].BorderWidth = 4;
        weekGraph.Series[0].Color = Settings.colours.Blue;
        weekGraph.Series[0].MarkerColor = Settings.colours.Blue;
        weekGraph.Series[0].MarkerSize = 8;
        weekGraph.Series[0].LabelForeColor = Settings.colours.Blue;

        // Setup the series to hold the peak listener count as a step line graph in orange
        with a thick line without any markers
        weekGraph.Series[1].ChartType = SeriesChartType.StepLine;
        weekGraph.Series[1].LegendText = "Peak Listener Count";
        weekGraph.Series[1].BorderWidth = 4;
        weekGraph.Series[1].Color = Settings.colours.Orange;
```

```
        weekGraph.Series[1].LabelForeColor = Settings.colours.Orange;

}
```

I have added this method to the event which is ran when the help button is clicked on in the toolbar. This is just for testing purposes, so I can check to see if the code to update the graph is running correctly. Now I can test the setupWeekGraph method. This should update the title of the graph, and add the axis labels as detailed above.


After running the method, the following graph is displayed to the left. Now as the graph is being setup correctly, I can now add the data obtained from the graph.

Before I add the data to the graph, as I am going to be plotting a date value on the x axis, I will need to take the data obtained from the SQL query, and then put this into a dictionary structure. The graph can then plot this as a key value pair, with the date on the X Axis and the value of listener count on the Y Axis.

Now I can make another method called 'updateWeekGraph' which will clear the data on both parts of the series, then it will run the SQL query, obtaining the new data to plot. This will then be put into 2 dictionaries, one for the average listener count, and another one for the peak listener count. They will then both be plotted onto their own series to then be displayed on the graph.

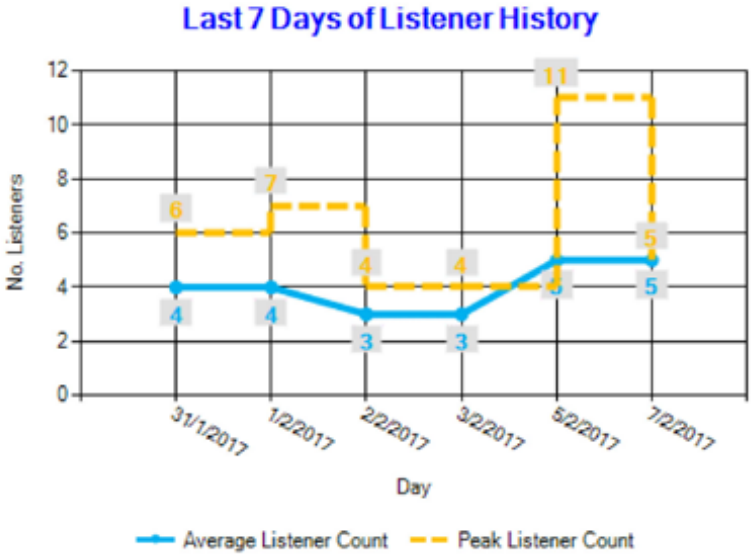
```
private void updateWeekGraph()
{
        // Clear points on graph for both Average and Peak listener counts
        weekGraph.Series[0].Points.Clear();
        weekGraph.Series[1].Points.Clear();

        // Run query against database to obtain points to plot
        MySqlCommand command = mySqlConn.CreateCommand();
        command.CommandText = "SELECT CONCAT(DAY(Log_DateTime), '/', MONTH(Log_DateTime),
        '/', YEAR(Log_DateTime)) AS 'Date', ROUND(AVG(Log_ListenerCount), 0) AS 'Average',
        MAX(Log_ListenerCount) AS 'Peak' FROM log WHERE Log_DateTime between
        AddDate(NOW(), -7) and NOW() GROUP BY DAY(log.Log_DateTime) ORDER BY Log_DateTime
        ASC;";

        // Run the query, and obtain the data back as a DataReader
        IDataReader reader = command.ExecuteReader();

        // Create 2 dictionaries to hold the Average and Peak listener counts
        Dictionary<string, int> dictionaryAverage = new Dictionary<string, int>();
        Dictionary<string, int> dictionaryPeak = new Dictionary<string, int>();

        // Cycle through each record returned by the query
        while (reader.Read())
        {
                // Add the date and the value for each Average value to the Average
                listener count dictionary
                dictionaryAverage.Add(reader["Date"].ToString(),
                int.Parse(reader["Average"].ToString()));

                // Add the date and the value for each Peak value to the Peak listener
                count dictionary
                dictionaryPeak.Add(reader["Date"].ToString(),
                int.Parse(reader["Peak"].ToString()));
        }

        // Cycle through each value for the average points, and plot it on the graph on
        the 1st series
        foreach (KeyValuePair<string, int> point in dictionaryAverage)
        {
                weekGraph.Series[0].Points.AddXY(point.Key, point.Value);
        }
```

```
        // Cycle through each value for the peak points, and plot it on the graph on the
        2nd series
        foreach (KeyValuePair<string, int> point in dictionaryPeak)
        {
                weekGraph.Series[1].Points.AddXY(point.Key, point.Value);
        }

        // Close the reader so more data can be queried from the database
        reader.Close();
}
```

Before I test this code I will add to the Dashboard's load event a call to run the method 'setupWeekGraph' so it is setup when the form opens.

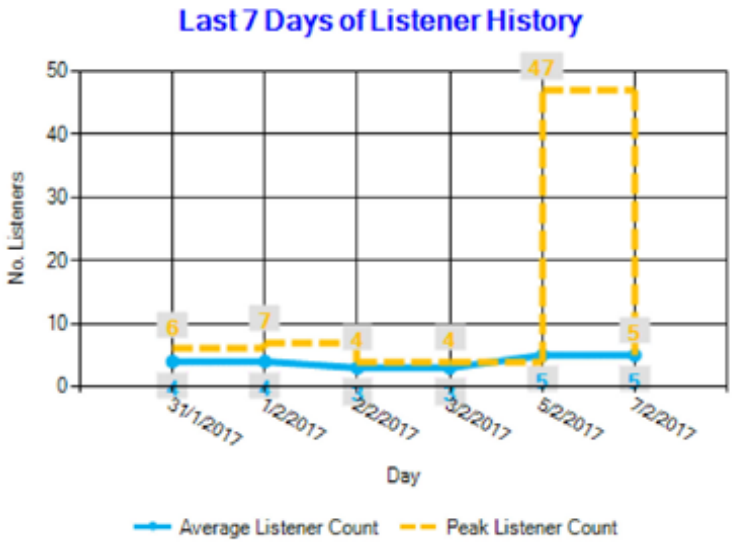
Now I can add the 'updateWeekGraph' method to the press of the help button in the menu bar, again to test that it is obtaining data correctly. Below is a screenshot of the data the query returned, and the graph the application then plotted:

| Date | Average | Peak |
|---|---|---|
| 31/1/2017 | 4 | 6 |
| 1/2/2017 | 4 | 7 |
| 2/2/2017 | 3 | 4 |
| 3/2/2017 | 3 | 4 |
| 5/2/2017 | 5 | 11 |
| 7/2/2017 | 5 | 5 |



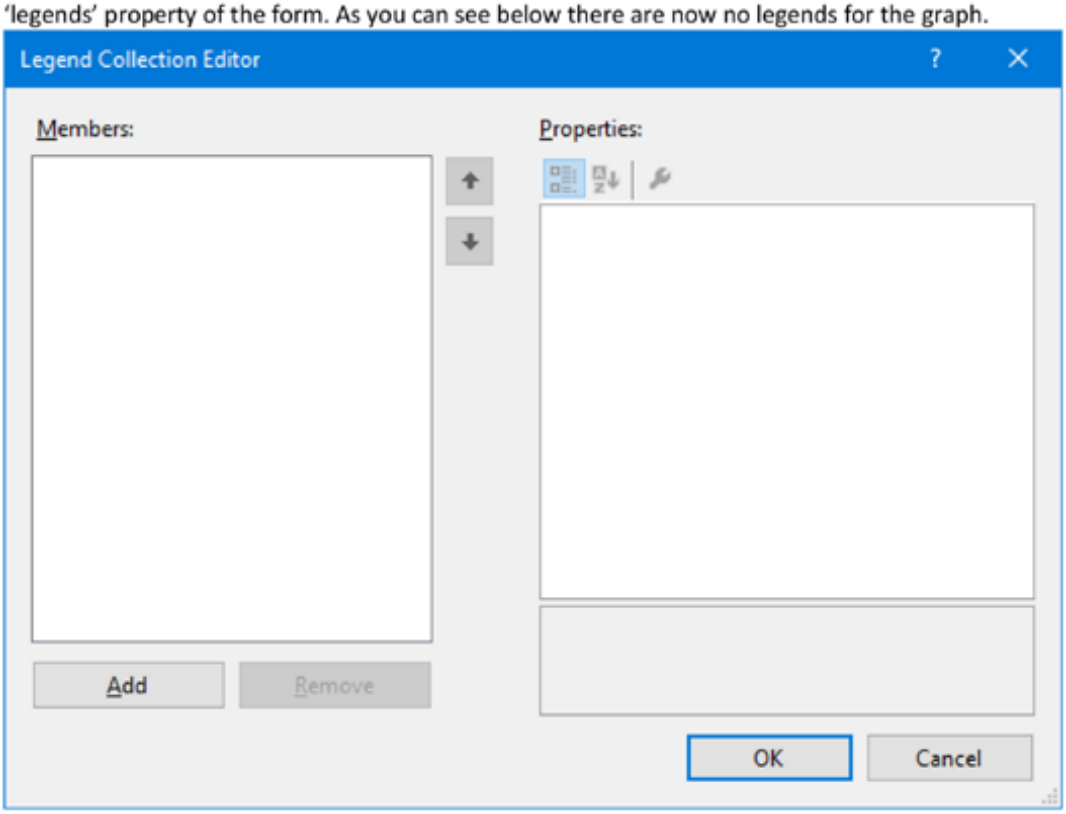

As you can see from comparing the data shown in the table on the left to the graph on the right, all of that data matches successfully, so the graph is working fine. The graph will automatically update the ranges of the X and Y axis as needed so that all of the data will fit on the graph. For example, if I edit one of the records in the database on the 5/2/2017 and set it to having 47 listeners, then the peak value for that day will increase to 47. When the graph update button is pressed the following data is obtained and plotted on the graph:

| Date | Average | Peak |
|---|---|---|
| 31/1/2017 | 4 | 6 |
| 1/2/2017 | 4 | 7 |
| 2/2/2017 | 3 | 4 |
| 3/2/2017 | 3 | 4 |
| 5/2/2017 | 5 | 47 |
| 7/2/2017 | 5 | 5 |



Last 60 minutes of listener history:

Now the weekly overview graph has been made, I can now add the main graph on the dashboard which will display the last 60 minutes of listener history. This will be setup the same as the graph before, this time difference being it is only using one series (rather than the previous graph which used two).
Like the other chart I will be creating a 'setupchart' method to define various axis settings, but first I will edit some of the chart's properties directly.
Firstly I will make sure that the legend is not visible on the graph, as there will only be one series, so a legend will not be needed. This can be done by deleting the legend item from the list of items under the 'legends' property of the form. As you can see below there are now no legends for the graph.




Next I need to turn on value labels for the points plotted, so that the values come up. This can be done by going to the series that is on the chart. By opening the series collection under the chart's series property. Now under the properties for the only series on the chart 'series1' I can set the option of 'isValueShownAsLabel' under the labels subcategory, to true. Which will now display labels for the data added to the chart. I have also set the label back colour to be a dark grey. So that the label will stand out for each value.
Now I can create the 'setupHourGraph' method, which will setup the graph with all of the correct settings, and will be ran as part of the form's load event.

```
private void setupHourGraph()
{
// Setup X Axis
hourGraph.ChartAreas[0].Axes[0].Title = "Time";
hourGraph.ChartAreas[0].Axes[0].IsLabelAutoFit = true;
hourGraph.ChartAreas[0].Axes[0].Interval = 1;

hourGraph.ChartAreas[0].Axes[0].LabelAutoFitStyle = LabelAutoFitStyles.LabelsAngleStep30;
hourGraph.ChartAreas[0].Axes[0].LabelStyle.Enabled = true;

// Setup Y Axis
hourGraph.ChartAreas[0].Axes[1].Title = "No. Listeners";
hourGraph.ChartAreas[0].Axes[1].IsLabelAutoFit = true;
hourGraph.ChartAreas[0].Axes[1].LabelAutoFitStyle = LabelAutoFitStyles.LabelsAngleStep30;
hourGraph.ChartAreas[0].Axes[1].LabelStyle.Enabled = true;

// Setup the series to hold the average listener count as a normal line graph in blue
with a bold line with markers
```

```
hourGraph.Series[0].ChartType = SeriesChartType.StepLine;
hourGraph.Series[0].LegendText = "Listener Count";
hourGraph.Series[0].BorderWidth = 4;
hourGraph.Series[0].Color = Settings.colours.Blue;
hourGraph.Series[0].LabelForeColor = Settings.colours.Blue;

}
```

I will now add this method to the form's load event, so that the graph has its settings defined when the form loads:

```
private void Dashboard_Load(object sender, EventArgs e)
{
        ////////////////////////////////
        // Other code from before
        ////////////////////////////////

        // Setup hour graph
        setupHourGraph();

}
```

Now I can write the code to update the graph with the listener values from the past 60 minutes. This will be almost identical to the previous graph, but only needed to obtain one set of data. The following SQL query will be ran which will select the time and the average listener count for that minute, based of the past 60 minutes of listener statistics.

```
SELECT TIME(Log_DateTime) AS 'Time', ROUND(AVG(Log_ListenerCount), 0) AS
'Count' FROM log WHERE Log_DateTime between AddDate(NOW(), INTERVAL -1 HOUR)
and NOW() GROUP BY MINUTE(log.Log_DateTime) ORDER BY Log_DateTime ASC;
```

The query returns the following results:

| Time | Count |
|---|---|
| 16:01:15 | 3 |
| 16:04:45 | 3 |
| 16:07:45 | 3 |
| 16:09:45 | 3 |
| 16:12:15 | 3 |
| 16:14:15 | 2 |
| 16:17:06 | 2 |
| 16:20:45 | 3 |
| 16:21:15 | 2 |
| 16:22:15 | 2 |
| 16:25:45 | 2 |
| 16:28:45 | 2 |
| 16:29:15 | 3 |
| 16:32:46 | 2 |
| 16:36:45 | 2 |
| 16:37:15 | 3 |
| 16:38:15 | 4 |
| 16:39:15 | 3 |
| 16:40:15 | 4 |
| 16:42:45 | 3 |
| 16:45:45 | 4 |
| 16:46:15 | 3 |
| 16:51:15 | 3 |

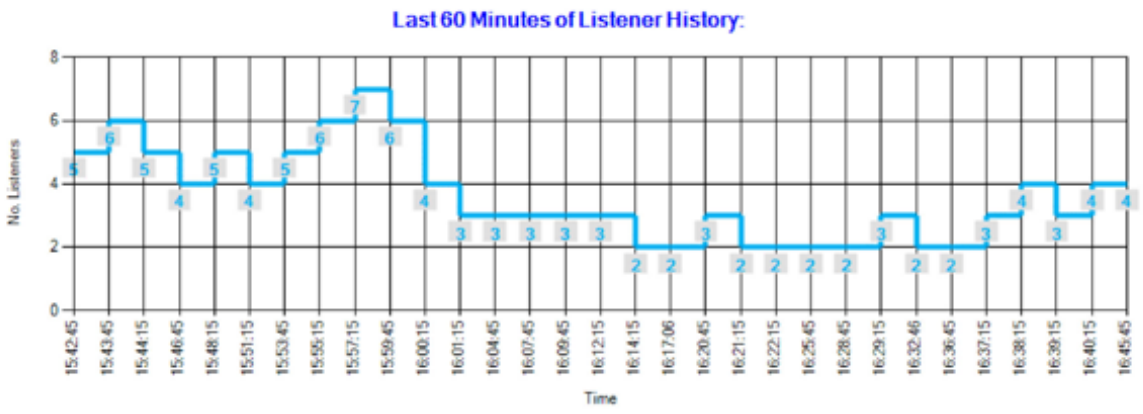
This can now be plotted on the graph. We can plot these results as they are, with the time on the X axis, and the count on the Y axis. They do not need to be reversed, as it will be running from left to right chronologically so the order on the graph will match up with the time order.

Now I can create a method called 'updateWeekGraph'. This will run the query, obtain the results from it, and then plot all of the points on the graph, by assigning them to a dictionary, with the key of the time and the value of the listener count.

```
private void updateHourGraph()
{
        // Clear points on graph for the listener count
        hourGraph.Series[0].Points.Clear();

        // Run query against database to obtain points to plot
        MySqlCommand command = mySqlConn.CreateCommand();
        command.CommandText = "SELECT TIME(Log_DateTime) AS 'Time',
        ROUND(AVG(Log_ListenerCount), 0) AS 'Count' FROM log WHERE Log_DateTime between
        AddDate(NOW(), INTERVAL -1 HOUR) and NOW() GROUP BY MINUTE(log.Log_DateTime) ORDER
        BY Log_DateTime ASC;";

        // Run the query, and obtain the data back as a DataReader
        IDataReader reader = command.ExecuteReader();

        // Create a dictionary to hold the listener count
        Dictionary<string, int> dictionaryCount = new Dictionary<string, int>();

        // Cycle through each record returned by the query
        while (reader.Read())
        {
                // Add the time and the listener value to the dictionary to then plot on
                the graph
                dictionaryCount.Add(reader["Time"].ToString(),
                int.Parse(reader["Count"].ToString()));
        }

        // Cycle through each value in the dictionary, then plot each value on the graph
        foreach (KeyValuePair<string, int> point in dictionaryCount)
        {
                hourGraph.Series[0].Points.AddXY(point.Key, point.Value);
        }

        // Close the reader so more data can be queried from the database
        reader.Close();
}
```

When the code is ran the following graph is displayed:

| Time | Count |
|---|---|
| 15:42:45 | 5 |
| 15:43:45 | 6 |
| 15:44:15 | 5 |
| 15:46:45 | 4 |
| 15:48:15 | 5 |
| 15:51:15 | 4 |
| 15:53:45 | 5 |
| 15:55:15 | 6 |
| 15:57:15 | 7 |
| 15:59:45 | 6 |
| 16:00:15 | 4 |
| 16:01:15 | 3 |
| 16:04:45 | 3 |
| 16:07:45 | 3 |
| 16:09:45 | 3 |
| 16:12:15 | 3 |
| 16:14:15 | 2 |
| 16:17:06 | 2 |
| 16:20:45 | 3 |
| 16:21:15 | 2 |
| 16:22:15 | 2 |
| 16:25:45 | 2 |
| 16:28:45 | 2 |
| 16:29:15 | 3 |
| 16:32:46 | 2 |
| 16:36:45 | 2 |
| 16:37:15 | 3 |
| 16:38:15 | 4 |
| 16:39:15 | 3 |
| 16:40:15 | 4 |
| 16:45:45 | 4 |




As you can see, the points on the graph match up with the results returned by the query. You may also notice that the graph now touches the edges of each side of the axis. This is due to changing the margin on each side of the chart area to 0, so that the graph line will fill from on end to the other, as it makes sense due to the fact it is continuous data, and makes it easier to read, as a line that ends before the end of the graph can look rather confusing.

The only other values left to populate on the form are the two labels, one which holds the current listener count, and another that holds the peak listener count for the day (as you can see below):

To do this I will create another method called 'updateCurrentAndPeakListeners' this will run two queries, the first one will obtain the most recent record from the database, and take the listener count, and set that to the value of the label called 'currentListenerCount'. The second query will select all data from 00:00:00 to 23:59:59 on the current day, and obtain the maximum value record for the listener count, this will then be set as the text on the label 'peakListenerCount'.

The first query (to obtain the latest record) is as follows:

```
SELECT Log_ListenerCount FROM log ORDER BY Log_ID DESC LIMIT 1
```

This returns the following:

| Log_ListenerCount |
|---|
| 6 |

This matches with the latest record having 6 listeners, so this query is working.

The second query (to obtain the maximum peak for that day is as follows:

```
SELECT MAX(Log_ListenerCount) AS Peak FROM log WHERE DATE(Log_DateTime) =
DATE(NOW())
```

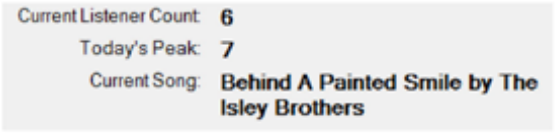
This returns the following:

| Peak |
|---|
| 7 |

This matches the peak of 7 listeners today so this query is also working. Now all that is left it to combine both of them into the method and test it:

```
// Update current and peak listener count
private void updateCurrentAndPeakListeners()
{
        // Return latest record from database
        MySqlCommand command = mySqlConn.CreateCommand();
        command.CommandText = "SELECT Log_ListenerCount FROM log ORDER BY Log_ID DESC
        LIMIT 1";

        // Create new reader to hold results of query
        IDataReader reader = command.ExecuteReader();

        // Check to make sure values have been returned
        if (reader.Read())
        {
                lblCurrentListenerCount.Text = reader["Log_ListenerCount"].ToString();
        }

        // Close the reader so more data can be queried from the database
        reader.Close();

        // Calculate the peak value from today via separate query
        command.CommandText = "SELECT MAX(Log_ListenerCount) AS Peak FROM log WHERE
        DATE(Log_DateTime) = DATE(NOW())";

        // Create new reader to hold results of query
        reader = command.ExecuteReader();

        // Check to make sure values have been returned
        if (reader.Read())
        {
                lblPeakListenerCount.Text = reader["Peak"].ToString();
        }

        // Close the reader so more data can be queried from the database
        reader.Close();
}
```

I have also added this method to the method that is ran when the help button is pressed in the toolbar for testing purposes. When the help button is pressed, now all of the form is successfully updating, and the two labels are now displaying the correct information.




The only thing left in the dashboard is to define a global update loop, this will call all of the functions that I have created to update the dashboard. This will be ran every 60 seconds.

I will firstly create a method called 'updateDashboard' which will call all of the methods we have just made:

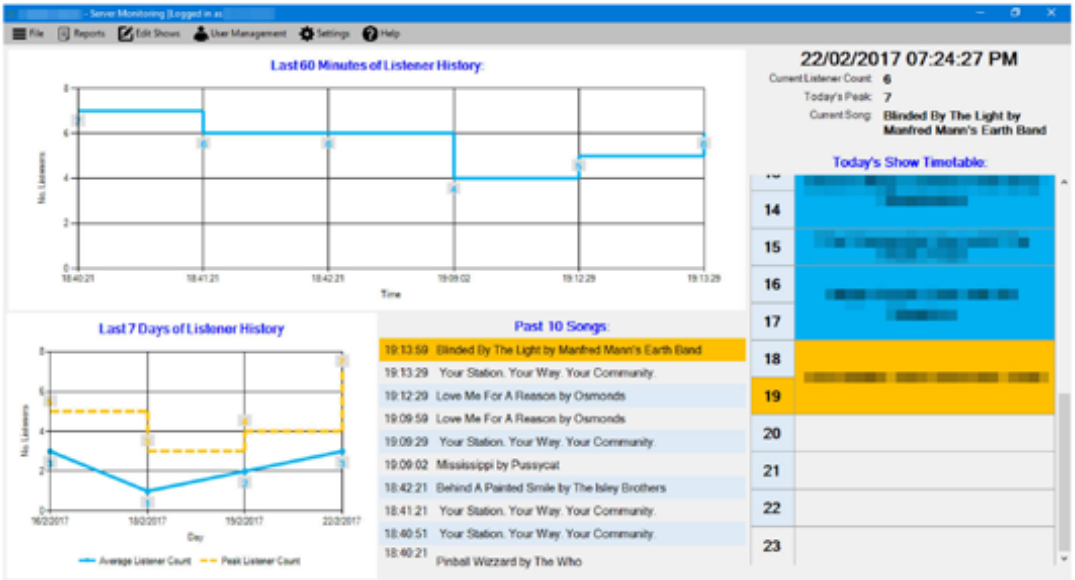
```
private void udpateDashboard()
{
        // This will update the dashboard
        updateTimetable();
        updatePastSongs();
        updateWeekGraph();
        updateHourGraph();
        updateCurrentAndPeakListeners();
}
```

Now I can create a timer called 'globalUpdateLoopTimer'. This timer's tick event will be set to call the updateDashboard method. And its interval will be set to 6000 mili-seconds, which is 1 minute. It will be set to 'enabled' so it will start running then the form opens.

```
private void globalUpdateLoopTimer_Tick(object sender, EventArgs e)
{
        // Update the dashboard
        updateDashboard();
}
```

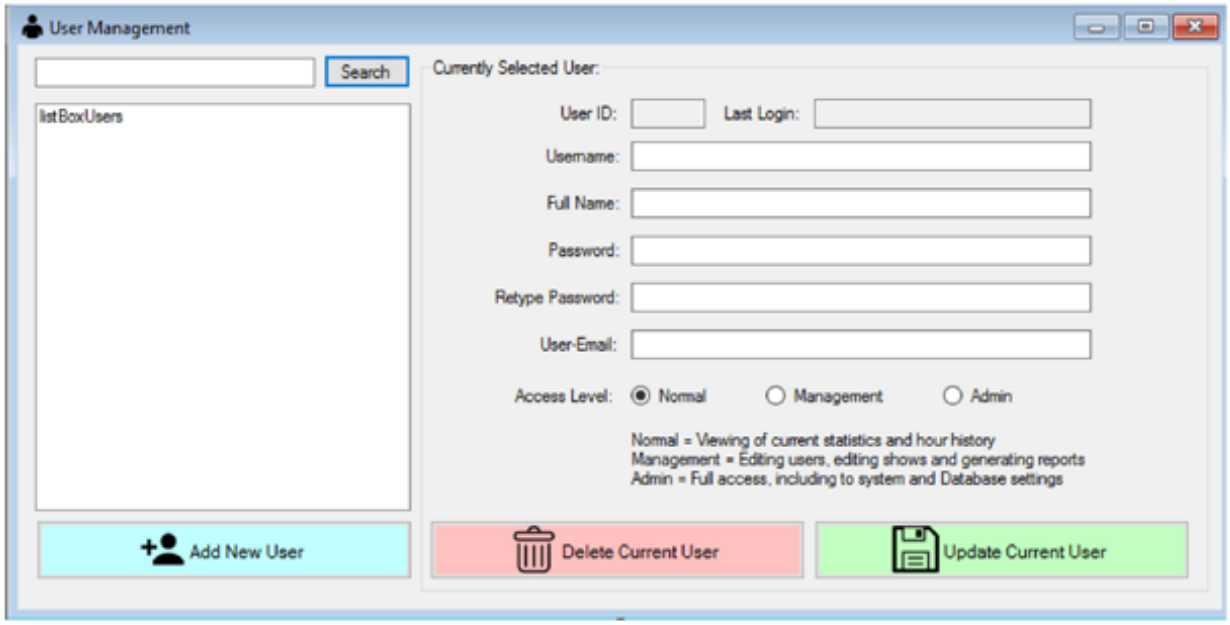
I will also add this to the end of the form load method, so that the dashboard is populated with data when it first opens.

Now when the dashboard is loaded it is automatically populated with the correct data, and after waiting for a few minutes, the dashboard updated every 60 seconds with the latest data.



User Management

Now the dashboard form has been completed. Now I need to develop the other forms that span off the dashboard. Firstly I will create the User Management form. This will allow users to be added, edited and removed.




This form has been setup according to the design earlier in the documentation. It has its FormBorderStyle set to FixedDialog and its StartPosition to CenterScreen so that it will appear in the middle of the screen when it is opened.

The accept button of the form has been changed to the Search button, so that when the user presses the enter key it will run a search query for users in the database matching that text. The buttons have a background colour set to them to make them stand out, along with an icon image on them matching what their functions are.

The User ID and Last Login text boxes are set to be read-only, as that information is not to be changed by the user, and only by the database when a user account is created or when a user logs in. The Password and repeat password text boxes are also have their 'UseSystemPasswordChar' property set to true, so that black dots will appear for each character of the user's entered password.

Finally the 3 radio boxes, are all assigned to the same group, and as they are radio boxes only one of them can be checked at a time (unlike checkboxes where more than one can be checked). There is also a label beneath them which explains what each of the access levels mean, so the user knows when creating an account.

Now the form has been setup, as this form will be talking to the database, like all other forms before it, I need to import the MysqlClient library, by adding the following line to the top of the form's code file:

```
using MySql.Data.MySqlClient;
```

I will also define the SQL connection string in the public section of the form's code as well.

```
namespace Server_Listener_Statistics
{
  public partial class UserManagement : Form
  {
       // Define MySQL Connection details
       MySqlConnection mySqlConn = new MySqlConnection(Settings.dbConnectionString);

  }

}
```

The first thing that is needed is for the list box to be populated with usernames when the form first loads. I will add code to the form's load method that will run when it is first opened.

The SQL statement that will be ran is as follows:

```
SELECT * FROM `users` ORDER BY `User_FullName` ASC
```

And it returns the following data:

| | User_Ful... | Use... | User_Email | User_Password | | User_LastLogin |
|---|---|---|---|---|---|---|
| 1 | [redacted] | admin | [redacted] | $2a$12$Ggz7G9taHmwft2qeCVaje6dKVVfz7zLjcVLPqotKY4iGxBg/56va | 3 | 2017-02-25 17:38:35 |

I will then cycle through each of the user's returned, and add them to the list box.

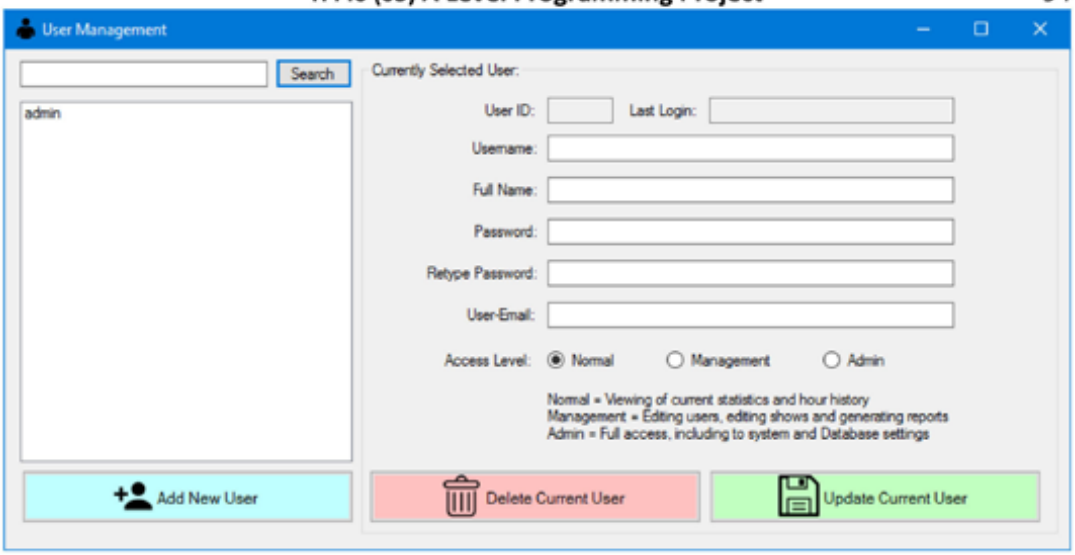
```
private void UserManagement_Load(object sender, EventArgs e)
{
        // Open connection to DB
        mySqlConn.Open();

        // Create SQL command to list all of the users
        MySqlCommand command = mySqlConn.CreateCommand();
        command.CommandText = "SELECT * FROM `users` ORDER BY `User_FullName` ASC";

        // Create a data reader to hold the returned data
        IDataReader reader = command.ExecuteReader();

        // Cycle through each of the users returned
        while (reader.Read())
        {
                // Add the username to the listbox as a new item
                listBoxUsers.Items.Add(reader["User_LoginName"].ToString());
        }

        // Close the reader so it can be used again with other queries
        reader.Close();
}
```

I will also need to add the form to the dashboard, so that when the User Management menu option is pressed, the form will open as a dialogue.

This can be done by adding the following code to the userManagementToolStripMenuItem click event on the dashboard form

```
private void userManagementToolStripMenuItem_Click(object sender, EventArgs e)
{
        // Create a new User Management form object
        // and open a new form as a Dialogue window
        UserManagement userManagement = new UserManagement();
        userManagement.ShowDialog();
}
```

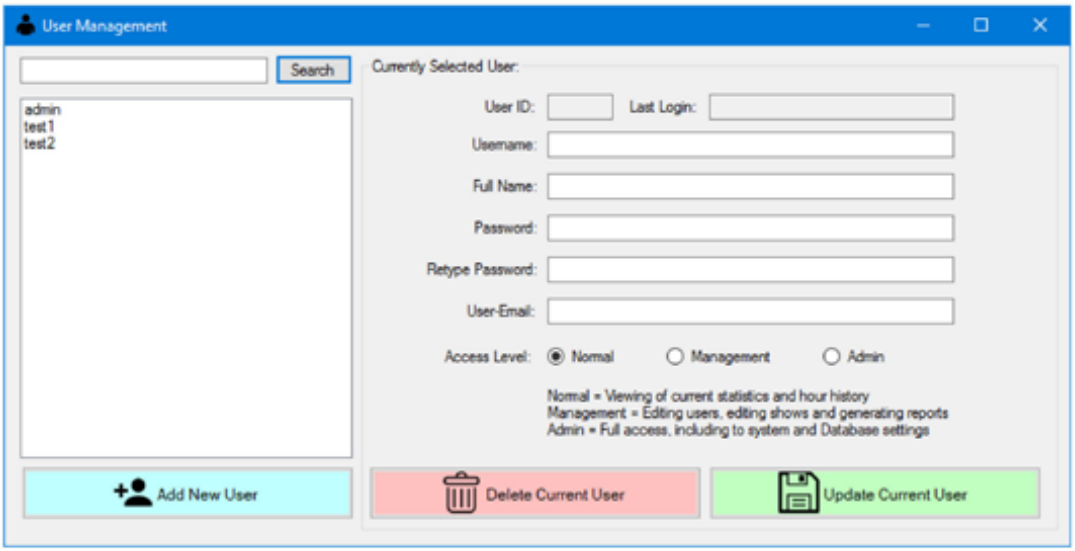
Now when the User Management option is pressed on the dashboard menu bar the following is displayed:




This has taken all of the users from the database, for testing purposes I am going to add some more test users to the users table. Test User 1 and Test User 2.

| | User_Full... | Use... | User_Email | User_Password | | User_LastLogin |
|---|---|---|---|---|---|---|
| 1 | | admin | | | 3 | 2017-02-25 18:44:02 |
| 16 | Test User 1 | test1 | test1@gmail.com | | 3 | 2017-02-25 18:46:27 |
| 17 | Test User 2 | test2 | test2@gmail.com | | 3 | 2017-02-25 18:46:41 |

Now when the form is opened again the following information is displayed:




Now I can add an option so that when the user's name is clicked on their relevant information will be populated on the form.

This code will be added to the selectedIndexChange event on the listbox, so that whenever the focus of the selected item is changed then the data on the form will be updated to reflect that user account.

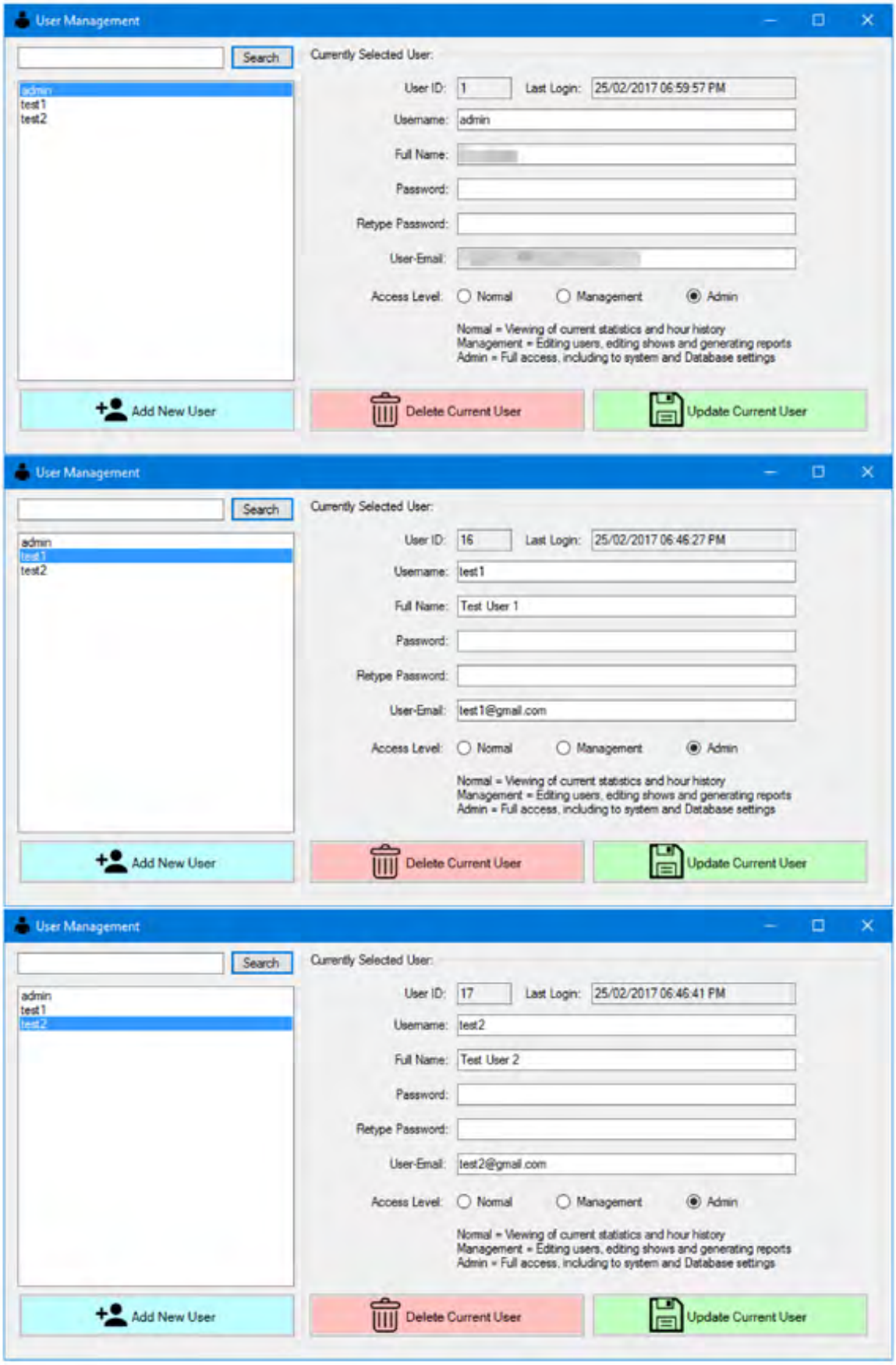
The SQL command that will be executed for this is as follows:

```
SELECT * FROM `users` WHERE `User_LoginName` = 'USERNAME'
```

This will return all of the fields for the user with the specified username entered. There is no need for SQL escaping on this field, as the user will not be typing in any values themselves, data will be coming out of the database, and then that exact data will then be queried to obtain other records.

```
private void listBoxUsers_SelectedIndexChanged(object sender, EventArgs e)
{
        // This code needs to be surrounded in a try catch loop, as the first time it is
        ran it will fail due to the fact the list box will not be populated yet or have a
        value selected
        try
        {
                // Run query to obtain selected user's details from the database
                MySqlCommand command = mySqlConn.CreateCommand();
                command.CommandText = $"SELECT * FROM `users` WHERE `User_LoginName` =
                '{listBoxUsers.SelectedItem.ToString()}'";

                // Create a data reader to hold the results returned from the database
                IDataReader reader = command.ExecuteReader();

                // Check to make sure that the reader has returned some data
                if (reader.Read())
                {
                        // Assign the values returned from the query to the appropriate text
                        boxes
                        txtUserID.Text = reader["User_ID"].ToString();
                        txtFullName.Text = reader["User_FullName"].ToString();
                        txtUsername.Text = reader["User_LoginName"].ToString();
                        txtUsername.Tag = reader["User_LoginName"].ToString();
                        txtEmail.Text = reader["User_Email"].ToString();
                        txtLastLogin.Text = reader["User_LastLogin"].ToString();
                        txtPassword.Text = "";
                        txtRepeatPassword.Text = "";

                        // Check the correct radio box dependent on the user's access level
                        switch (int.Parse(reader["User_AccessLevel"].ToString()))
                        {
                                case 1:
                                        rb1.Checked = true;
                                        rb2.Checked = false;
                                        rb3.Checked = false;
                                        break;
                                case 2:
                                        rb1.Checked = false;
                                        rb2.Checked = true;
                                        rb3.Checked = false;
                                        break;
                                case 3:
                                        rb1.Checked = false;
                                        rb2.Checked = false;
                                        rb3.Checked = true;
                                        break;
                        }

                }
                // Close the reader so that it can be used again with other queries against
                the database
                reader.Close();
        }
        catch (Exception)
        {

                // Do nothing as mentioned before, the code will fail on the first time as
                the listbox will not have been populated yet or have a selected index
        }
}
```

When the index is changed on the listbox the following information is displayed, I have also put the information stored in the database as well below them:


User Management
Search
admin
test1
test2
Currently Selected User:
User ID: 1
Last Login: 25/02/2017 06:59:57 PM
Username: admin
Full Name:
Password:
Retype Password:
User-Email:
Access Level: Normal Management Admin
Normal = Viewing of current statistics and hour history
Management = Editing users, editing shows and generating reports
Admin = Full access, including to system and Database settings
Add New User
Delete Current User
Update Current User
User Management
Search
admin
test1
test2
Currently Selected User:
User ID: 16
Last Login: 25/02/2017 06:46:27 PM
Username: test1
Full Name: Test User 1
Password:
Retype Password:
User-Email: test1@gmail.com
Access Level: Normal Management Admin
Normal = Viewing of current statistics and hour history
Management = Editing users, editing shows and generating reports
Admin = Full access, including to system and Database settings
Add New User
Delete Current User
Update Current User
User Management
Search
admin
test1
test2
Currently Selected User:
User ID: 17
Last Login: 25/02/2017 06:46:41 PM
Username: test2
Full Name: Test User 2
Password:
Retype Password:
User-Email: test2@gmail.com
Access Level: Normal Management Admin
Normal = Viewing of current statistics and hour history
Management = Editing users, editing shows and generating reports
Admin = Full access, including to system and Database settings
Add New User
Delete Current User
Update Current User

| | User_Ful... | Use... | User_Email | User_Password | | User_LastLogin |
|---|---|---|---|---|---|---|
| 1 | | admin | | | 3 | 2017-02-25 18:44:02 |
| 16 | Test User 1 | test1 | test1@gmail.com | | 3 | 2017-02-25 18:46:27 |
| 17 | Test User 2 | test2 | test2@gmail.com | | 3 | 2017-02-25 18:46:41 |

As you can see, this matches all of the information stored about the user in the database.

The password fields are both blank, as they will only be updated by the user if they want to change a user's password, they can do so by entering a new password in there.

Now I can create the code that will allow searching of users in the listbox. This will clear all values currently in the listbox, then it will run a query against the database and will obtain all users where their username or full name contains the search query the user has entered in. These results will then be added to the listbox.

The SQL query to perform this operation is as follows:

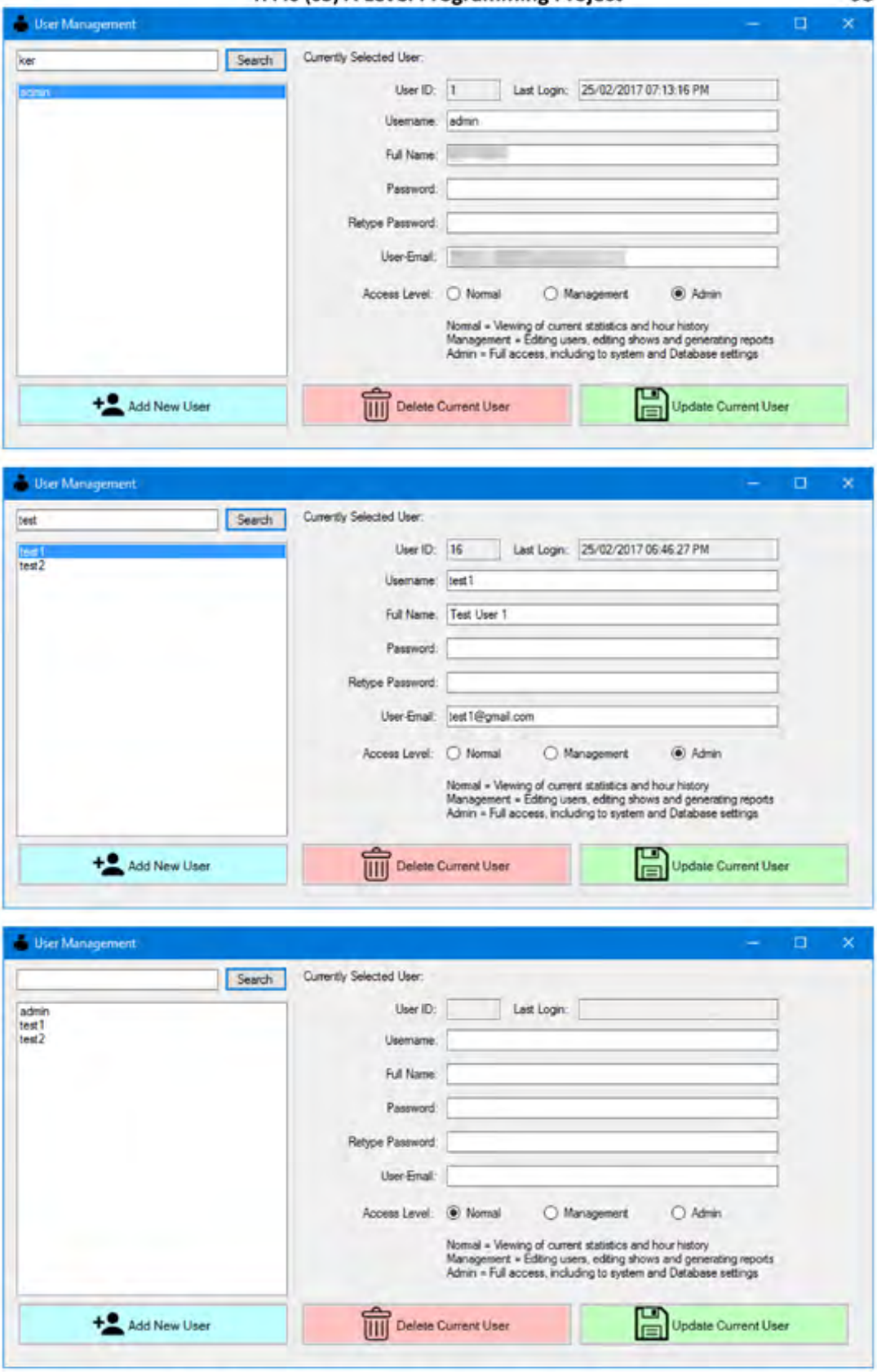
```
SELECT * FROM `Users` WHERE `User_LoginName` LIKE '%QUERY%' OR
`User_FullName` LIKE '%QUERY%' ORDER BY `User_LoginName` ASC
```

This will look for the string the user has entered (called query) and return the results the user has entered. The '%' symbol denotes to search for the occurrence anywhere in the string. So for example the search query of 'ker' would return [redacted] as that string contains the string 'ker'. The Ascending ordering by username on the end of the query makes sure that the results are returned in alphabetical order to make them easier to read.

The code will be added in the search button's click event.

```
private void btnSearch_Click(object sender, EventArgs e)
{
        // Clear all of the items in the listbox
        listBoxUsers.Items.Clear();

        // Perform query to obtain a list of all users who contain the entered search
        query as part of their username or their full name
        MySqlCommand command = mySqlConn.CreateCommand();
        command.CommandText = $"SELECT * FROM `Users` WHERE `User_LoginName` LIKE
        '%{txtQuery.Text}%' OR `User_FullName` LIKE '%{txtQuery.Text}%' ORDER BY
        `User_LoginName` ASC";

        // Create a data reader to hold the results returned
        IDataReader reader = command.ExecuteReader();

        // Cycle through each of the records that were returned
        while (reader.Read())
        {
                // Add the user's username as an item to the listbox
                listBoxUsers.Items.Add(reader["User_LoginName"].ToString());
        }

        // Close the data reader so that it can be used in other queries
        reader.Close();
}
```

I can now test this piece of code by searching for parts of the test user's information and see what results are returned:




You can also see that when an empty search is performed, all of the users in the database are returned.

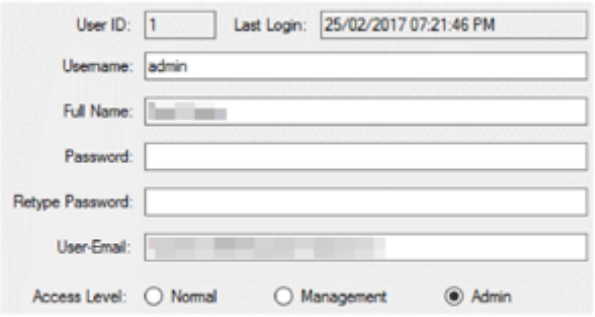
Now the searching functionality is working, I can now move onto adding new users.

This will simply update the current user's id to be the string value of 'NEW' this will then allow me to check in the code later when the update current user button is pressed to create a new account rather

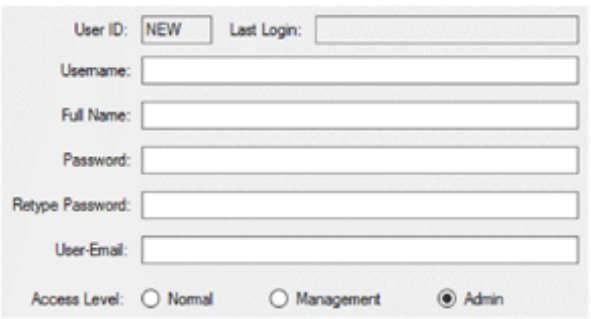
than updating an account that already exists. It is also beneficial to not define a new user ID for a new account as the database will automatically do that, as the field has auto increment enabled, so this will prevent duplicate primary keys for multiple users.

```
private void btnAddNewUser_Click(object sender, EventArgs e)
{
        // Change current id to 'NEW'
        txtUserID.Text = "NEW";

        // wipe all other fields
        txtEmail.Text = "";
        txtFullName.Text = "";
        txtLastLogin.Text = "";
        txtPassword.Text = "";
        txtRepeatPassword.Text = "";
        txtUsername.Text = "";
}
```

Testing this code produces the following:




*Before pressing the button*




*After pressing the button*

This works successfully, so I can now move onto creating the code to update the current user, or insert a new user if the user's ID is set to 'NEW'.

If the user's ID is not set to NEW, then firstly it will check to see if the user's username has changed. This is based off the fact the username text was set to the user's username, along with the tag property of the text field. If the user has changed the text in the text box, then both of these properties will be different. The new username will then be checked to see if it available in the database, if it is free it will move onto the next validation step, if not it will not proceed any further and display an error message. Next it will see if any values have been entered for the password, if it has then it will check to make sure the password and retyped password are the same, if they are it will proceed to check that all other fields are filled in, and then will run the query to update the user's information. If they are not, then an error message will be displayed, and it will not proceed any further.

If the user's ID was set to 'NEW' then the same process will be followed, other than the password will be required for the user. And rather than updating the user's record, it will insert a new one into the database.

The update code is as follows:

```
        // Code to update the current user
        private void btnUpdateCurrentUser_Click(object sender, EventArgs e)
        {
            // Create a flag which will be changed to false if any verification steps
fail
            bool pass = true;

            // Calculate user's access level (with a default of 1 in case none is
checked)
            int accesslevel = 1;
```

```
            if (rb1.Checked)
            {
                accesslevel = 1;
            }
            else if (rb2.Checked)
            {
                accesslevel = 2;
            }
            else if (rb3.Checked)
            {
                accesslevel = 3;
            }

            // Check to see if all of the other fields have been filled in
            if (txtFullName.Text == "" || txtEmail.Text == "" || txtUsername.Text == "")
            {
                MessageBox.Show("Please fill in all fields before trying to update a user
account");
                pass = false;
            }

            // Check to see if the username has been changed, if so check to see if the
username already exists
            else if (txtUserID.Text != "NEW")
            {
                if(txtUsername.Text != txtUsername.Tag.ToString()) {
                // The username has been changed  so create a query to check to see if
the new username already exists
                MySqlCommand command = mySqlConn.CreateCommand();
                command.CommandText = $"SELECT Count(`User_LoginName`) As Count from
users WHERE `User_LoginName` = @username";
                command.Parameters.AddWithValue("@username", txtUsername.Text);

                // Create a data reader to hold the value of the results returned
                IDataReader reader = command.ExecuteReader();

                // Read data from query
                reader.Read();

                // Check to see if the username exists
                if(reader[0].ToString() != "0")
                {
                    pass = false;
                    // The username already exists
                    MessageBox.Show("The username already exists in the database. Please
try again with a different username");

                }
                reader.Close();
                }
            }

            // Check to see that both of the passwords match, if the user has entered a
password
            if(txtPassword.Text != "" && txtUserID.Text != "NEW")
            {
                // User has updated a password
                if (txtPassword.Text != txtRepeatPassword.Text)
                {
                    // The repeated passwords does not match the first entered password
                    MessageBox.Show("The two passwords you have entered do not match.");
                    pass = false;
                }
                else
                {
                    // update the user if the passwords match
                    if (pass)
                    {
                        // Update user details with hashed password
```

```
                    string hashedpassword = Hash.hashPassword(txtPassword.Text);

                    // Create a query to update the user's details
                    MySqlCommand command = mySqlConn.CreateCommand();
                    command.CommandText = $"UPDATE users SET User_LoginName =
@username, User_Email = @email, User_FullName = @fullname, User_AccessLevel =
{accesslevel.ToString()}, User_Password = @password WHERE User_ID = {txtUserID.Text}";
                    command.Parameters.AddWithValue("@username", txtUsername.Text);
                    command.Parameters.AddWithValue("@email", txtEmail.Text);
                    command.Parameters.AddWithValue("@fullname", txtFullName.Text);
                    command.Parameters.AddWithValue("@password", hashedpassword);

                    // Run the query without returning any results from it
                    command.ExecuteNonQuery();

                    // Display a message to the user to tell them their account has
been updated
                    MessageBox.Show("User has been updated successfully");

                    // Refresh the listbox to update the user's information and
select the user
                    int selectedindex = listBoxUsers.SelectedIndex;
                    btnSearch.PerformClick();
                    listBoxUsers.SelectedIndex = selectedindex;
                }
            }
        }
        else if (pass && txtUserID.Text != "NEW")
        {
            // update user with details excluding password

            MySqlCommand command = mySqlConn.CreateCommand();
            command.CommandText = $"UPDATE users SET User_LoginName = @username,
User_Email = @email, User_FullName = @fullname, User_AccessLevel =
{accesslevel.ToString()} WHERE User_ID = {txtUserID.Text}";
            command.Parameters.AddWithValue("@username", txtUsername.Text);
            command.Parameters.AddWithValue("@email", txtEmail.Text);
            command.Parameters.AddWithValue("@fullname", txtFullName.Text);

            // Run the query without returning any results from it
            command.ExecuteNonQuery();

            // Display a message to the user to tell them their account has been
updated
            MessageBox.Show("User has been updated successfully");

            // Refresh the listbox to update the user's information and select the
user
            int selectedindex = listBoxUsers.SelectedIndex;
            btnSearch.PerformClick();
            listBoxUsers.SelectedIndex = selectedindex;
        }
        else if(pass && txtUserID.Text == "NEW")
        {
            // Check username is available by counting the number of times a username
exists in the table
            MySqlCommand command = mySqlConn.CreateCommand();
            command.CommandText = $"SELECT Count(`User_LoginName`) As Count from
users WHERE `User_LoginName` = @username";
            command.Parameters.AddWithValue("@username", txtUsername.Text);

            // Create a reader to hold the result returned
            IDataReader reader = command.ExecuteReader();

            // Read data from query
            reader.Read();

            // Check to see if the username exists, by seeing if the record returned
was not equal to 0
```

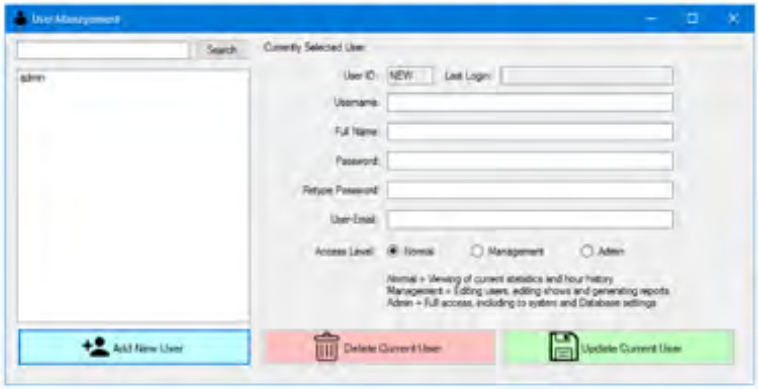
```
                if (reader[0].ToString() != "0")
                {
                    // The username already exists so set the pass flag to false
                    pass = false;
                    MessageBox.Show("The username already exists in the database. Please
try again with a different username");

                }

                // Close the reader so that it can be used again and more data can be
queried
                reader.Close();

                // Check to see if passwords match, and the global pass flag is true
                if(txtPassword.Text != "" && pass == true)
                {
                    if(txtPassword.Text == txtRepeatPassword.Text)
                    {
                        // If both of the passwords match, update the user with the new
password
                        // Hash the password via the Hash function
                        string hashedpassword = Hash.hashPassword(txtPassword.Text);

                        // Create a new SQL command to insert the new record into the
database
                        MySqlCommand command2 = mySqlConn.CreateCommand();
                        command2.CommandText = $"INSERT INTO users
(User_FullName,User_LoginName,User_Email,User_Password,User_AccessLevel) VALUES
(@fullname, @username, @email, @password, {accesslevel.ToString()})";
                        command2.Parameters.AddWithValue("@fullname", txtFullName.Text);
                        command2.Parameters.AddWithValue("@username", txtUsername.Text);
                        command2.Parameters.AddWithValue("@email", txtEmail.Text);
                        command2.Parameters.AddWithValue("@password", hashedpassword);

                        // Execute the query without returning any results
                        command2.ExecuteNonQuery();

                        // Show a message box to the user to let them know the account
has been created successfully
                        MessageBox.Show($"User '{txtFullName.Text}' has been created
successfully");

                        // Refresh the list of users in the database
                        btnSearch.PerformClick();


                    } else {
                        // As both of the passwords entered do not match, show and error
message
                        MessageBox.Show("The passwords you have entered do not match,
please try again");
                    }

                }
                else if(txtPassword.Text == "")
                {
                    // As there was no password entered for the user display an error
message
                    MessageBox.Show("Please enter a password for the user account");
                }

            }
```

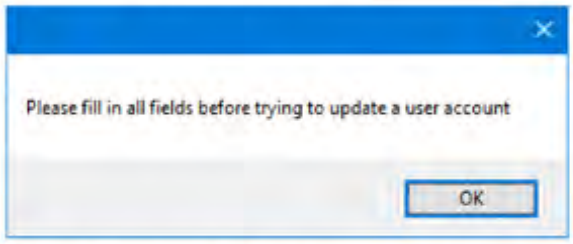
Now I need to test the code.

I will first of all check to see what happens if no values are entered and the update button is pressed after the new user button:

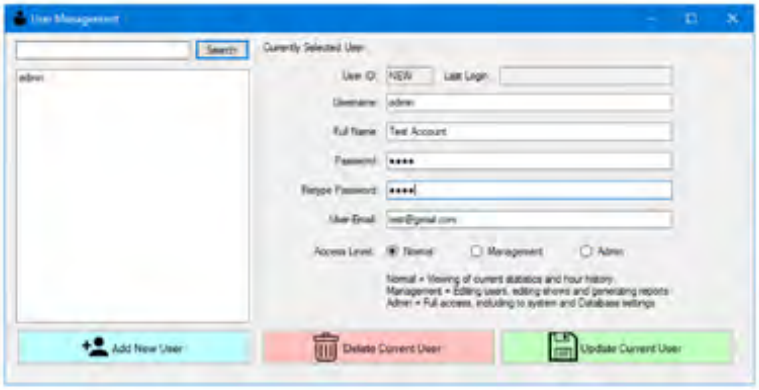
**H446 (03) A Level Programming Project** 103

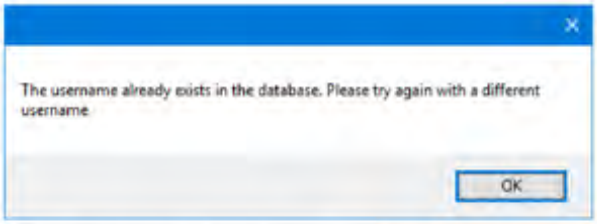
Form before pressing Update button: Message Displayed:



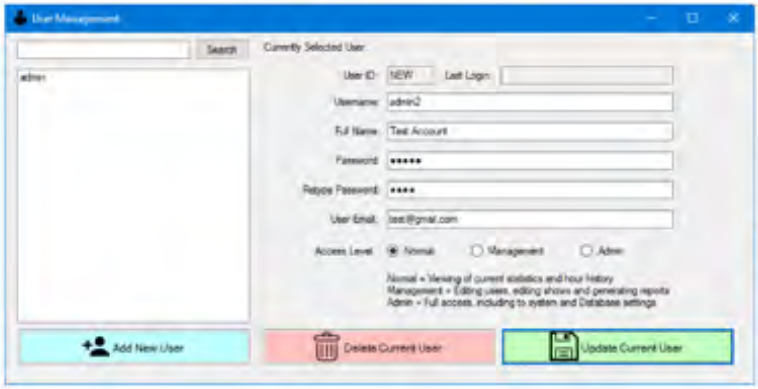


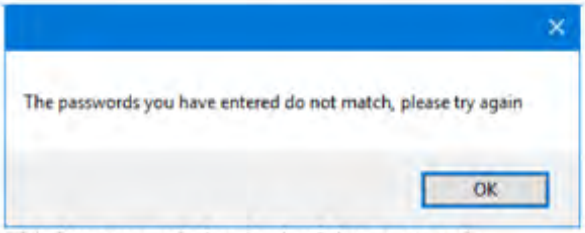

This has succeeded as it has not updated the user and notified them they need to fill in all fields.



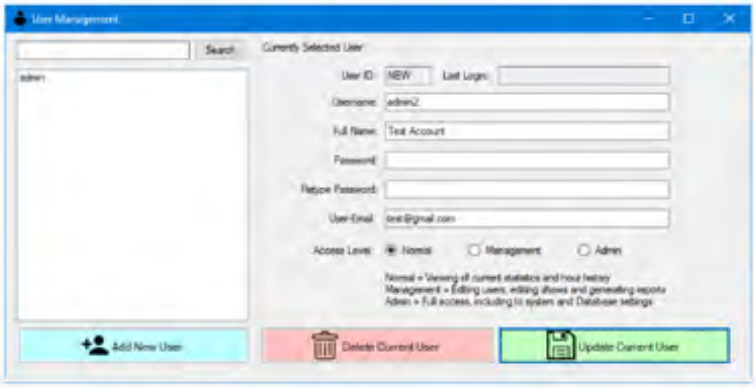


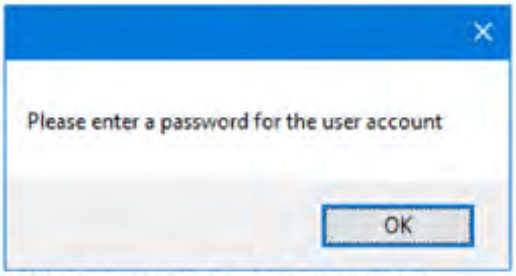

This has succeeded as it has not updated the user as the username 'admin' already exists, so an error message has been displayed.



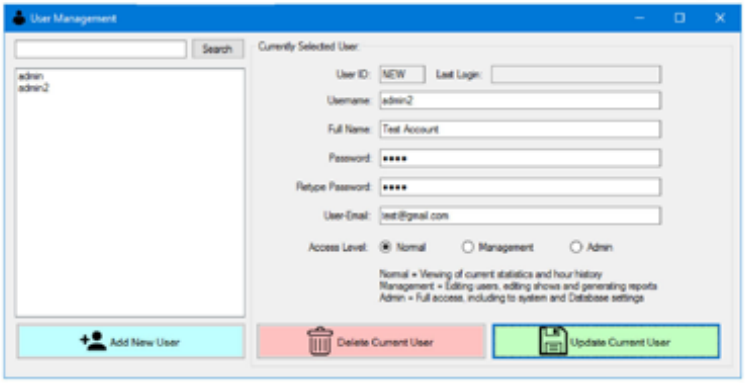


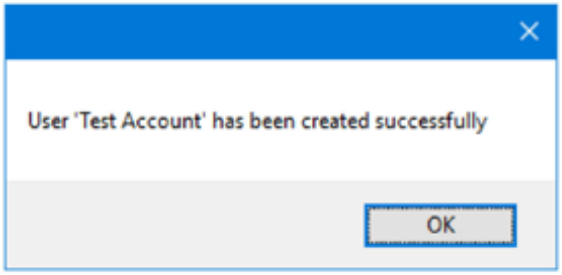

This has succeeded as both of the passwords were different, so it has not updated the user and an error message has been displayed.






This has succeeded as no password was entered, so the record was not updated and an error message was displayed.




This has succeeded as the account has been added, and a success message has been displayed. You can see below the record that was inserted into the database:

| | User_FullN... | User... | User_Email | User_Password | | User_LastLogin |
|---|---|---|---|---|---|---|
| 21 | | admin | admin@gmail.com | | 3 | 2017-03-08 10:03:00 |
| 22 | Test Account | admin2 | test@gmail.com | | 1 | (NULL) |

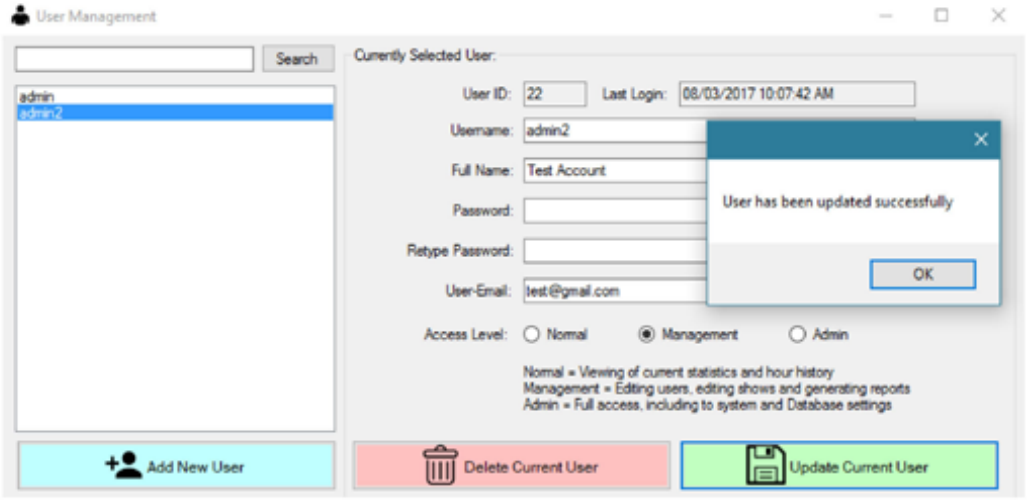
As you can see, the test account data matches the data entered on the form, along with the user level of '1'.

Now the record has been inserted I can now test the user update function to make sure that the information of a pre-existing user that already exists is updated correctly.

I will not need to retest data entry validation, as that was tested prior in the insertion of a new user. All I will need to test is if the data is updated, and if a new password is entered, if that is updated as well.

Below is the data before I run an update on the test account:

| User_ID | User_FullName | User_LoginName | User_Email | User_Password | User_AccessLevel | User_LastLogin |
|---|---|---|---|---|---|---|
| 21 | | admin | admin@gmail.com | | 3 | 2017-03-08 13:48:42 |
| 22 | Test Account | admin2 | test@gmail.com | | 3 | 2017-03-08 10:07:42 |




After updating the value from 'admin' to 'management' that should have changed the user's access level in the database from '3' to '2', and should have not changed any other parts of the user's record.

As you can see from the updated data below. The user's account has been updated successfully:

| User_ID | User_FullName | User_LoginName | User_Email | User_Password | User_AccessLevel | User_LastLogin |
|---|---|---|---|---|---|---|
| 21 | | admin | admin@gmail.com | | 3 | 2017-03-08 13:48:42 |
| 22 | Test Account | admin2 | test@gmail.com | | 2 | 2017-03-08 10:07:42 |

This also works for setting the account to a 'Normal' user, by setting the accounts user access level to 1.

Now I can test the change password functionality. This should change the user's password hash in the database if there is a value in the password textbox, and if it matches the value in the repeat password textbox.

This can been seen below:

| User_ID | User_FullName | User_LoginName | User_Email | User_Password | User_AccessLevel | User_LastLogin |
|---|---|---|---|---|---|---|
| 21 | [redacted] | admin | admin@gmail.com | [redacted] | 3 | 2017-03-08 13:48:42 |
| 22 | Test Account | admin2 | test@gmail.com | [redacted] | 2 | 2017-03-08 10:07:42 |

After update:

| User_ID | User_FullName | User_LoginName | User_Email | User_Password | User_AccessLevel | User_LastLogin |
|---|---|---|---|---|---|---|
| 21 | [redacted] | admin | admin@gmail.com | [redacted] | 3 | 2017-03-08 13:48:42 |
| 22 | Test Account | admin2 | test@gmail.com | [redacted] | 2 | 2017-03-08 10:07:42 |

This has also successfully passed, as the user's password hash has updated correctly, and after testing on the login screen, the user can login with their new password.

Now the update and new user functions are working, I can now finally add the delete user function that will delete the currently selected user.

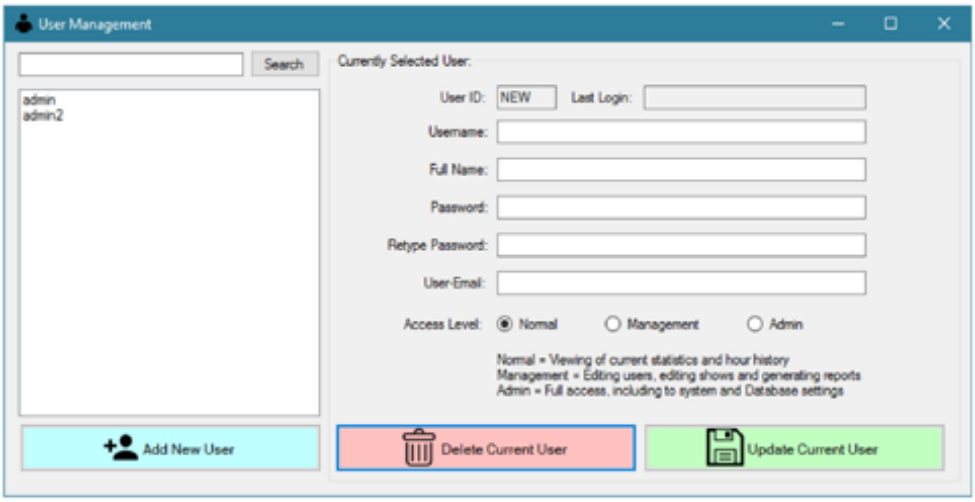
This code will firstly check to make sure that the user ID does not equal 'NEW' as the user account will not exist yet. Then it will show a confirmation dialogue box, asking the user if they wish to delete the user account, if they press yes it will then delete the user's account and update the list of users. If they pressed 'No' it will not delete the user's account.

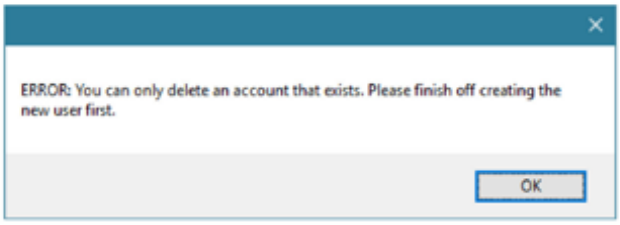
The code is as follows:

```
private void btnDeleteCurrentUser_Click(object sender, EventArgs e)
        {
            // Delete the current user

            // Check to make sure that the user actually exists and has an ID for us to
use
            if (txtUserID.Text != "NEW")
            {
                // Ask user whether they want to delete the current user
                DialogResult result = MessageBox.Show($"Are you sure you want to delete
{txtFullName.Text}'s account?", "Delete Account?", MessageBoxButtons.YesNo,
MessageBoxIcon.Warning, MessageBoxDefaultButton.Button2);

                // Check to see if they pressed yes
                if(result == DialogResult.Yes)
                {
                    // Delete the user's account
                    MySqlCommand command = mySqlConn.CreateCommand();
                    command.CommandText = $"DELETE FROM `Users` WHERE `User_ID` =
{txtUserID.Text}";

                    // Run the query
                    command.ExecuteNonQuery();

                    // Update the user's listbox
                    btnSearch.PerformClick();

                    // Remove values of current user
                    txtFullName.Text = "";
                    txtEmail.Text = "";
                    txtLastLogin.Text = "";
                    txtPassword.Text = "";
                    txtRepeatPassword.Text = "";
                    txtUserID.Text = "";
                    txtUsername.Text = "";
                    rb1.Checked = false;
                    rb2.Checked = false;
                    rb3.Checked = false;

                    // Display a success message to the user
                    MessageBox.Show("The account has been successfully deleted!");
```

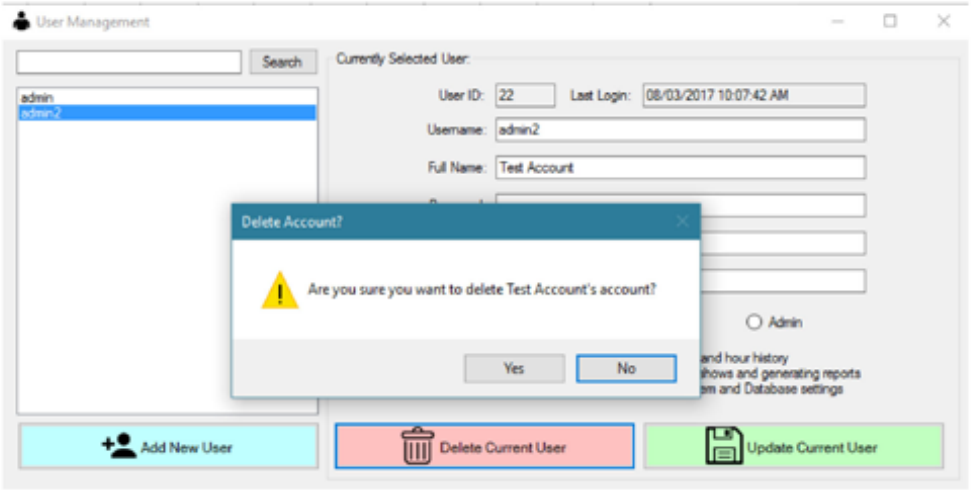
```
            }
        }
        else
        {
            // Display and error message to stop a user from deleting an account that
doesn't already exist
            MessageBox.Show("ERROR: You can only delete an account that exists.
Please finish off creating the new user first.");
        }
    }
```

Now I can test this by firstly trying to delete a user after pressing the new user button, and secondly delete a user whose account actually exists.

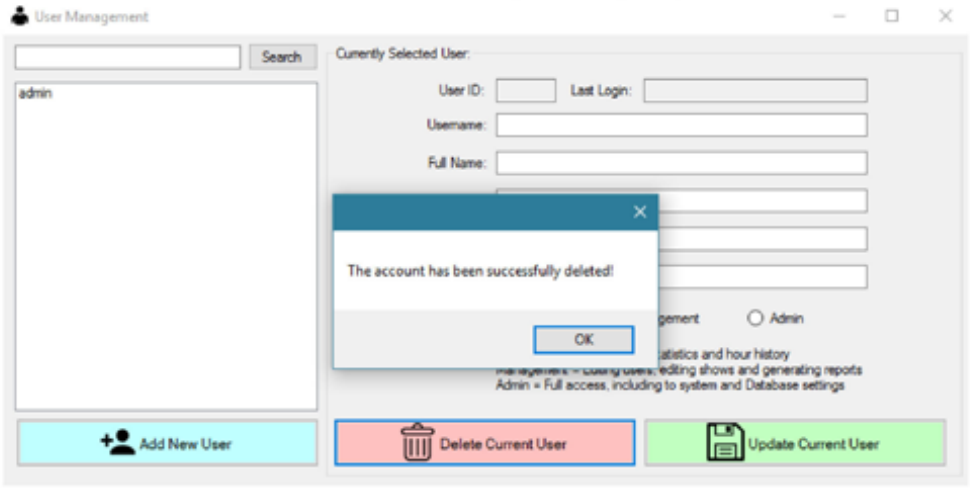



As you can see from the error box to the left, when the delete user button was pressed while creating a new user, the correct error message was display, and no account was removed. This test has succeeded.




When there is an account and the delete button is pressed, the dialogue box is displayed to confirm whether to delete the account or not. When No is pressed, then the account is not deleted, but when yes is pressed the account is deleted, and a confirmation message is shown (as below)




| User_ID | User_FullName | User_LoginName | User_Email | User_Password | User_AccessLevel | User_LastLogin |
|---|---|---|---|---|---|---|
| 21 | | admin | admin@gmail.com | | 3 | 2017-03-08 14:36:36 |

You can also see from the database view above of the user's table, that the record has indeed been deleted successfully.

An issue with this is that if you press the delete button when the form is first opened, no user will have been selected, so it will produce an error as it has no ID to find.

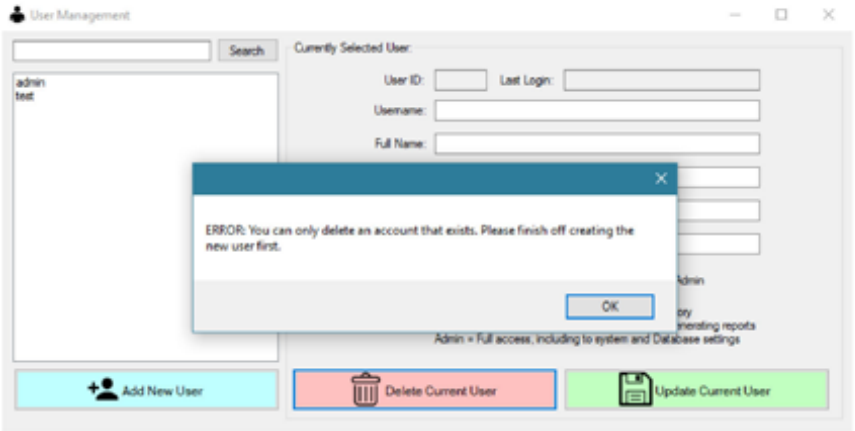
To fix this I will change the if statement to also check to see if the text box is not empty:

```
if (txtUserID.Text != "NEW")
```

Changed to:

```
if (txtUserID.Text != "NEW" && txtUserID.Text != "")
```

Now I can test this code:




As you can see from the screenshot above, the code has worked and now you can't delete a user if you haven't selected one first.

The other issue identified with this is that you can delete all users in the database, and then you will not have any access to the system.

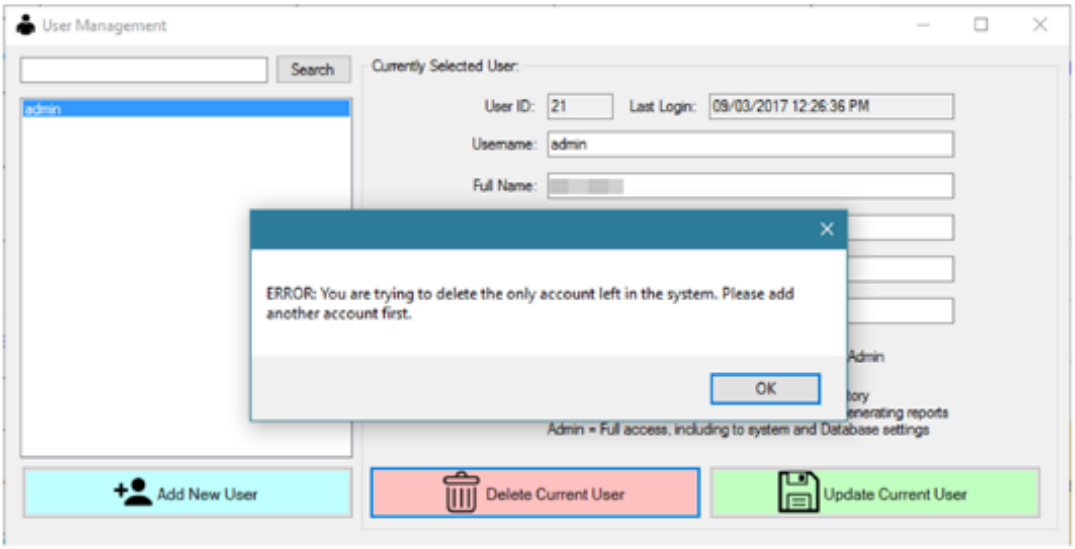
To prevent this I will change the delete user code, so that it will only run if there is more than 1 item in the list box.

```
// Check to make sure that there is more than 1 user in the listbox
if (listBoxUsers.Items.Count > 1)
{
        // Earlier code //
```

```
}
else
{
        // There is only 1 user listed in the listbox, so don't delete it
        MessageBox.Show("ERROR: You are trying to delete the only account left in the
        system. Please add another account first.");
}
```

The earlier code is the code from before, the if statement will wrap around it to perform the check.

As you can see from the example below this code is now working and will prevent deletion of the only account left:



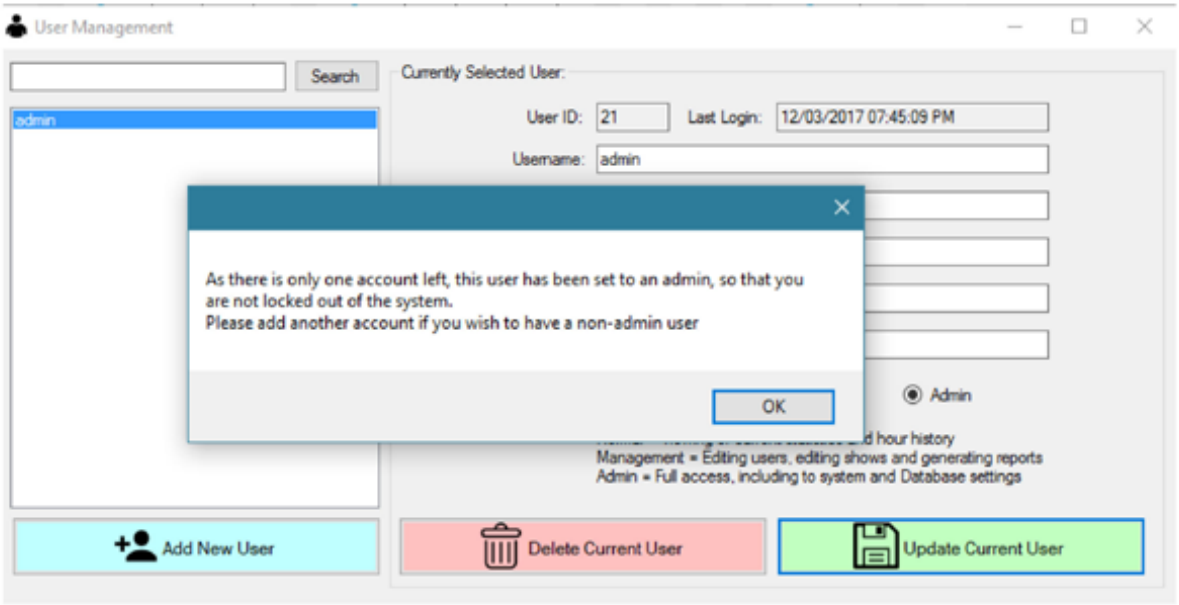

The only issue with this, is that if there is only one account left, you could update their user privileges to 'Normal' so that they would not have any access to changing user details. And would lock themselves out of creating any other accounts or changing any settings.

I all add a check to make sure if it is the last user then the only user access level allowed is 'Admin'.

The following code is added to the top of the method that is ran when the update/save button is pressed:

```
// Check to see if there is only one account left
if (listBoxUsers.Items.Count == 1 && txtUserID.Text != "NEW" && rb3.Checked == false)
{
        // Set access level to Admin
        accesslevel = 3;
        rb3.Checked = true;

        // Display message to user
        MessageBox.Show("As there is only one account left, this user has been set to an
        admin, so that you are not locked out of the system.\nPlease add another account
        if you wish to have a non-admin user");
}
```

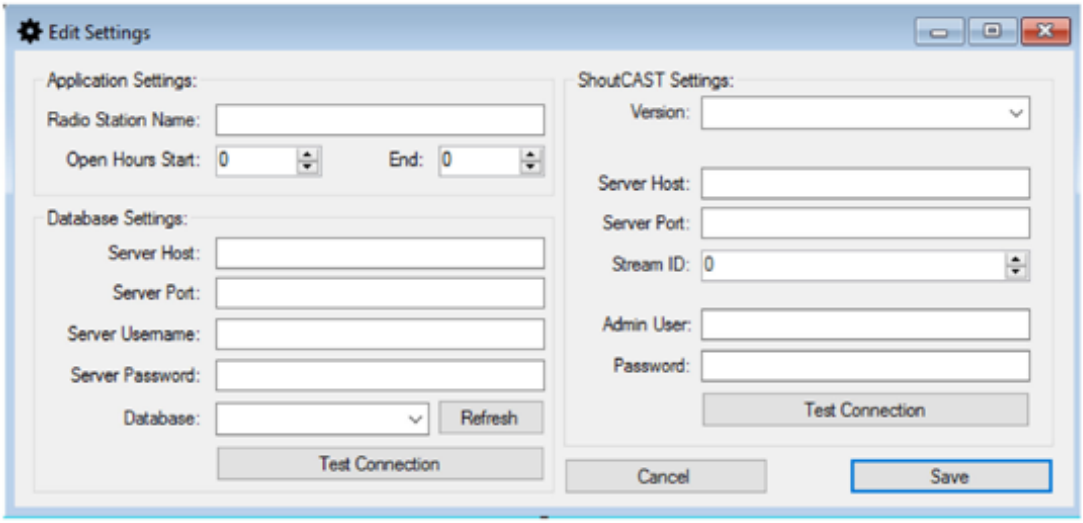
As you can see below, the code is now working and the last user left can't be deleted or changed to anything below the access level of admin (this was after trying to update the admin account to a 'Normal' user).




Now the user creation and management section of the system is working. I can now move onto the settings form; I have created this like any other form, and have given it a title and a relevant settings icon.

The settings form will allow for all of the settings to be changed for the application, this will cover General Radio station settings (Radio Name, Start Hour and End Hour), Radio Server Settings (The settings for the radio streaming server) and the Database settings (The connection string to the database).

I will add components to the form based off the design mentioned earlier, as you can see below, all of the controls have been added. They are all textboxes apart from numeric up-down boxes for port numbers, and combo boxes for the database selection dropdown. Also, for the password text fields, the "Use System Password Char" has been set to true, so that the password is blocked out with black dots as it is typed. Finally, the accept button on the form has been set to the Save button, so that when the enter key is pressed it will save the settings.




When the form opens it will need to load the settings from the Settings class, as they are the current ones.

This will be performed by creating a new method called 'updateSettingsFromClass', this will set the values off all of the form controls to the current values that have been loaded for the settings.

```
private void updateSettingsFromClass()
{

    // Obtain settings from settings class
    txtRadioName.Text = Settings.radioName;
    numHoursStart.Value = Settings.radioStartHour;
    numHoursEnd.Value = Settings.radioEndHour;

    // Take the connection string and split it up to each of the individual values
    txtDatabaseServerHost.Text = Settings.dbConnectionString.Split(';')[0].Split('=')[1];
    txtDatabaseServerPort.Text = Settings.dbConnectionString.Split(';')[1].Split('=')[1];
    txtDatabaseServerUsername.Text = Settings.dbConnectionString.Split(';')[2].Split('=')[1];
    txtDatabaseServerPassword.Text = Settings.dbConnectionString.Split(';')[3].Split('=')[1];
    cbDatabase.Text = Settings.dbConnectionString.Split(';')[4].Split('=')[1];

    // Obtain radio server settings
    cbServerVersion.Text = Settings.serverVersion;
    txtServerHost.Text = Settings.serverHost;
    txtServerPort.Text = Settings.serverPort;
    numServerStreamID.Value = int.Parse(Settings.serverID);
    txtServerAdminUser.Text = Settings.serverUsername;
    txtServerAdminPassword.Text = Settings.serverPassword;
}
```

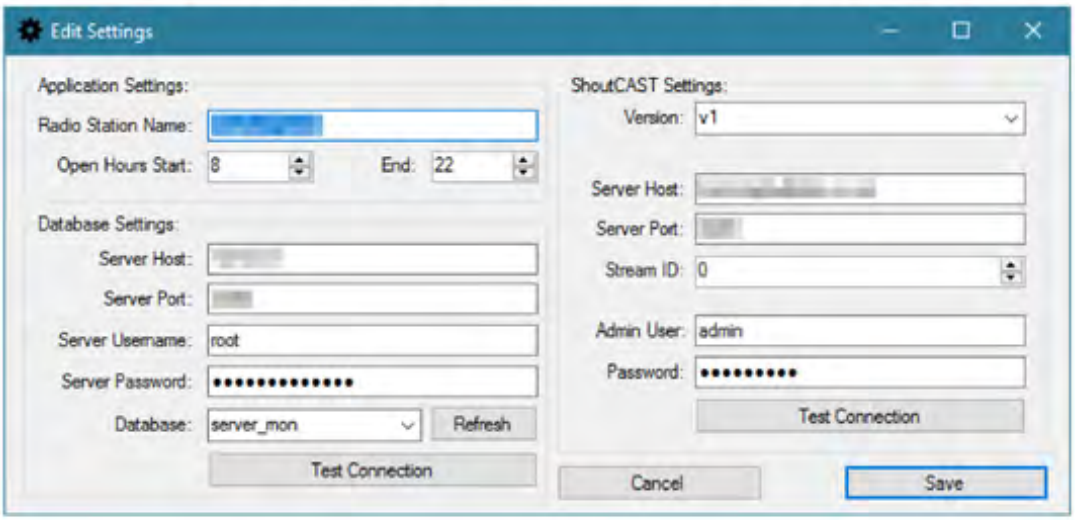
I will also add a call to run this method on the form's load event.

```
private void EditSettings_Load(object sender, EventArgs e)
{

    // Update settings from the settings class
    updateSettingsFromClass();

}
```

Before I test this I will need to add code to the click event on the 'Edit Settings' menu item on the dashboard form. This is so that when the Edit Settings item is clicked on, the settings form will open as a dialogue box.

```
private void settingsToolStripMenuItem_Click(object sender, EventArgs e)
{
    // Create a edit settings dialogue and open it
    EditSettings editSettings = new EditSettings();
    editSettings.ShowDialog();

}
```

Now when the application is ran, and the settings button is clicked, the following is displayed:



This test has succeeded, as all of the settings have been loaded correctly from the settings class.

Now I need to have a method that will test to see if the connection to the database is successful.

```
//Test SQL server
private bool testSqlServer(bool returnMessageOnTrue)
{
        // Define MySQL Connection details
        MySqlConnection mySqlConn = new
        MySqlConnection($"Server={txtDatabaseServerHost.Text};Port={txtDatabaseServerPort.
        Text};User
        ID={txtDatabaseServerUsername.Text};Password={txtDatabaseServerPassword.Text}");

        // Try to open connection
        try
        {
                mySqlConn.Open();

                // Connection succeeded close connection
                mySqlConn.Close();

                if (returnMessageOnTrue) { MessageBox.Show("Connection to the database was
                successful"); }
                return true;

        }
        catch (Exception e)
        {
                MessageBox.Show($"There was an error connecting to the SQL Server.
                \n\n{e.Message}");
                return false;
        }
}
```

There is also a conditional variable that is passed in, to determine whether to display a message on a successful connection or not.

This will be built into another method that will be used to return a list of databases on the SQL server. This code will first check to make sure the SQL server connection is valid, and will then run the following query to return the active databases on the server:

```
SHOW DATABASES
```

When this query is ran the following data is returned:

| Database |
|---|
| information_schema |
| cust0522 |
| darp_mag_ng |
| darp_mag_ng-backup |
| flights |
| helpdeskz |
| magmondb |
| mysql |
| openbiblio |
| osticket |
| performance_schema |
| piwigo |
| sakila |
| samdb |
| server_mon |
| server_mon_mag2 |
| sys |
| tpcc |
| world |

*There are a lot of databases on the server, as it is used to run a lot of other systems and not just this one.*

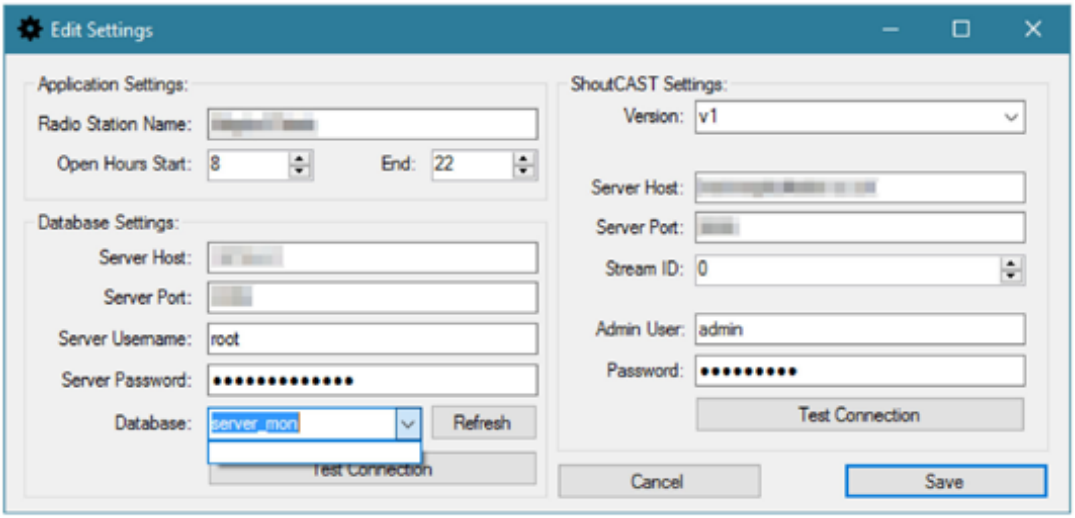
Now these items will be added to a list box that will then allow the user to select a database to use to store the data in on the server.

The code that the refresh button calls is below:

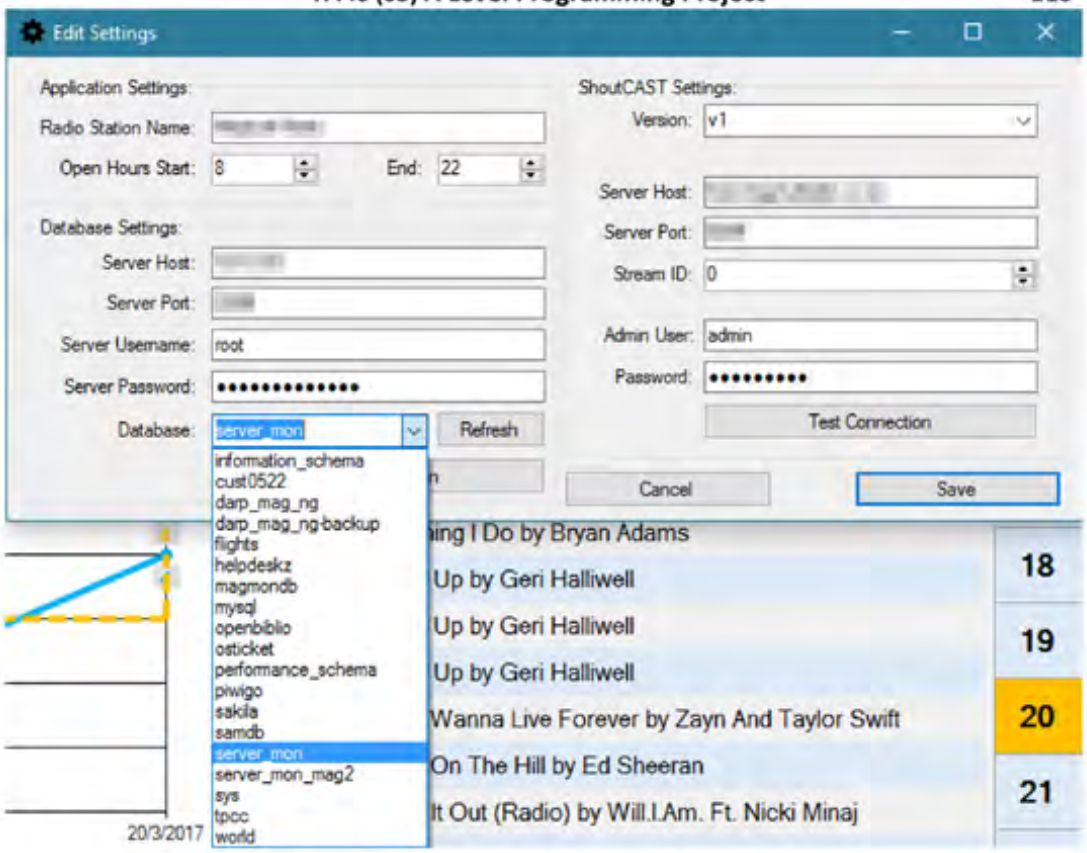
```
private void btnRefreshDatabases_Click(object sender, EventArgs e)
{
        if (testSqlServer(false))
        {

        // Define MySQL Connection details
        MySqlConnection mySqlConn = new
        MySqlConnection($"Server={txtDatabaseServerHost.Text};Port={txtDatabaseServerPort.
        Text};User
        ID={txtDatabaseServerUsername.Text};Password={txtDatabaseServerPassword.Text}");

        // open connection
        mySqlConn.Open();

        MySqlCommand command = mySqlConn.CreateCommand();
        command.CommandText = "SHOW DATABASES";


        // Clear items in database combo box
        cbDatabase.Items.Clear();

        // Create a reader to hold the results returned from the query
        IDataReader reader = command.ExecuteReader();

        // Cycle through each of the returned results and add them to the listbox
        while (reader.Read())
        {
                cbDatabase.Items.Add(reader[0].ToString());
        }

        // Close the reader so it can be used again
        reader.Close();

        // Close the connection to the sql server
        mySqlConn.Close();

        }
}
```

I can now test this code to make sure it is returning correct values and adding them to the list box:

As you can see, when you open the combo box, no databases are listed, other than the current one you are connected to.



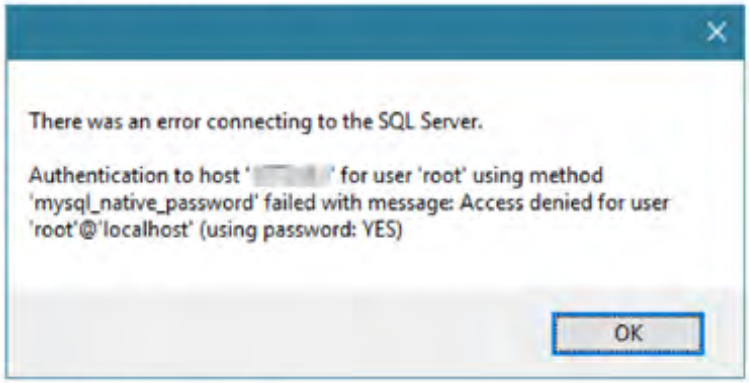

When the refresh button is pressed:



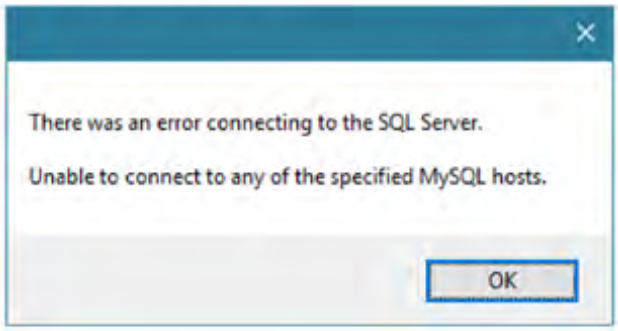

This successfully displays the databases that are associated with the database.

If there is an issue with the SQL connection string, then the error will be returned to the user. For example:

If the password is incorrect:




If the host doesn't exist:



Now the refresh database function is working, I can now make sure that the database contains the required tables. This will be part of the Test Database button that will check the database structure, to make sure all of the required tables exist. If they do not, it will ask the user whether they want to create them, this will then create the required tables in the database. The code to create the tables has been exported from HeidiSQL, so it will run and create the needed tables with all of the correct columns and relationships.

The code is below:

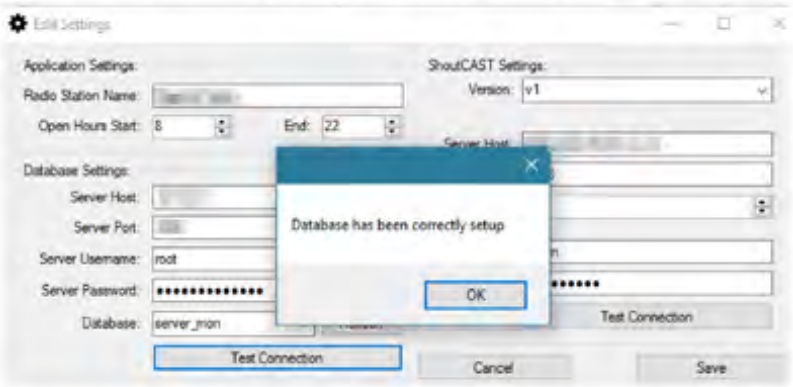
```
private bool checkSqlStructure(bool showMessageOnTrue)
        {
            // Test to make sure server connection is valid, don't display a message if
the connection is successfull
            if (testSqlServer(false))
            {
                // Make sure database contains the correct tables
                // Define MySQL Connection details
                MySqlConnection mySqlConn = new
MySqlConnection($"Server={txtDatabaseServerHost.Text};Port={txtDatabaseServerPort.Text};U
ser
ID={txtDatabaseServerUsername.Text};Password={txtDatabaseServerPassword.Text};Database={c
bDatabase.Text}");

                // Open database connection
                mySqlConn.Open();

                // Query database for a list of tables in the database
                MySqlCommand command = mySqlConn.CreateCommand();
                command.CommandText = "SHOW TABLES";

                // Create a reader to hold the information retrieved from the database
                IDataReader reader = command.ExecuteReader();

                // Create a counter to count the number of tables that match in the
database
                int requiredTables = 0;

                // Loop through each record, if it matches add one to the counter
                while (reader.Read())
                {
                    switch (reader[0].ToString())
                    {
                        case "log":
                            requiredTables++;
                            break;
                        case "users":
                            requiredTables++;
                            break;
                        case "shows":
                            requiredTables++;
                            break;
                    }
                }

                // Close the reader so it can be used again
                reader.Close();

                // Check to see if the 3 tables exists
                if (requiredTables == 3)
                {
                    // Database is correctly setup
                    if (showMessageOnTrue) { MessageBox.Show("Database has been correctly
setup"); }
                    return true;
                }
                else
                {
                    // Database is not setup
```

```
                    DialogResult result = MessageBox.Show("The required tables for the
database were not found. Would you like to create these now? ", "Required tables not
found", MessageBoxButtons.YesNo, MessageBoxIcon.Error, MessageBoxDefaultButton.Button1);

                    if (result == DialogResult.Yes)
                    {
                        // Create tables in database (this code is exported from HeidiSQL)
                        command = mySqlConn.CreateCommand();
                        command.CommandText = "/*!40101 SET
@OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */; /*!40101 SET NAMES utf8 */; /*!50503
SET NAMES utf8mb4 */; /*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */; /*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */; CREATE TABLE IF NOT EXISTS `log` ( `Log_ID` int(11)
NOT NULL AUTO_INCREMENT, `Log_DateTime` datetime DEFAULT NULL, `Log_ListenerCount`
int(11) DEFAULT NULL, `Log_CurrentSong` varchar(250) DEFAULT NULL, `Log_ShowID` int(11)
DEFAULT NULL, PRIMARY KEY (`Log_ID`), KEY `ShowIDFK` (`Log_ShowID`), CONSTRAINT
`ShowIDFK` FOREIGN KEY (`Log_ShowID`) REFERENCES `shows` (`Show_ID`) ON DELETE NO ACTION
ON UPDATE NO ACTION ) ENGINE=InnoDB DEFAULT CHARSET=utf8; CREATE TABLE IF NOT EXISTS
`shows` ( `Show_ID` int(11) NOT NULL AUTO_INCREMENT, `Show_Name` varchar(500) DEFAULT
NULL, `Show_Presenter` varchar(500) DEFAULT NULL, `Show_Day` varchar(250) DEFAULT NULL,
`Show_Starttime` int(11) DEFAULT NULL, `Show_Endtime` int(11) DEFAULT NULL,
`Show_StartDate` date DEFAULT NULL, `Show_EndDate` date DEFAULT NULL, `Show_Active`
tinyint(1) DEFAULT '1', PRIMARY KEY (`Show_ID`) ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE IF NOT EXISTS `users` ( `User_ID` int(11) NOT NULL AUTO_INCREMENT,
`User_FullName` varchar(250) DEFAULT NULL, `User_LoginName` varchar(250) DEFAULT NULL,
`User_Email` varchar(250) DEFAULT NULL, `User_Password` varchar(1000) DEFAULT NULL,
`User_AccessLevel` int(10) DEFAULT NULL, `User_LastLogin` datetime DEFAULT NULL, PRIMARY
KEY (`User_ID`) ) ENGINE=InnoDB DEFAULT CHARSET=utf8; /*!40101 SET
SQL_MODE=IFNULL(@OLD_SQL_MODE, '') */; /*!40014 SET
FOREIGN_KEY_CHECKS=IF(@OLD_FOREIGN_KEY_CHECKS IS NULL, 1, @OLD_FOREIGN_KEY_CHECKS) */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;";

                        // Execute query to create tables in database
                        command.ExecuteNonQuery();

                        // Show success message
                        MessageBox.Show("Tables have been created");

                    }
                    // The user doesn't want tables to be created so return false
                    return false;
                }

            }
            else
            {
                // Couldn't connect to database so return false
                return false;
            }
        }
```

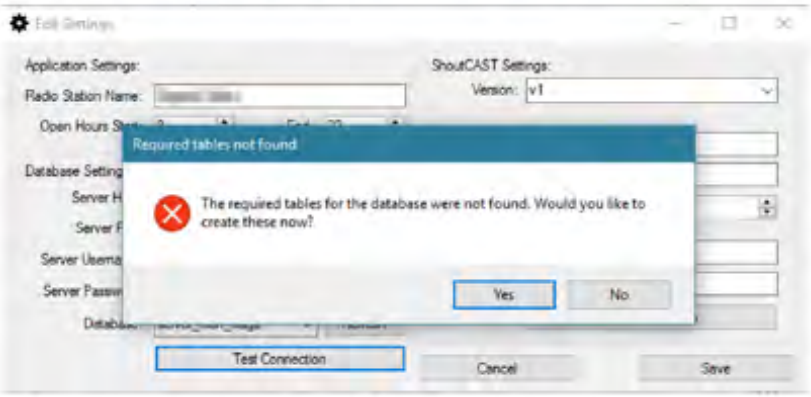
Now I can test the code by adding this call to the 'Test Button' click method

```
        private void btnDatabaseServerTestConnection_Click(object sender, EventArgs e)
        {
            // Run the code to test the sql structure
            checkSqlStructure(true);
        }
```

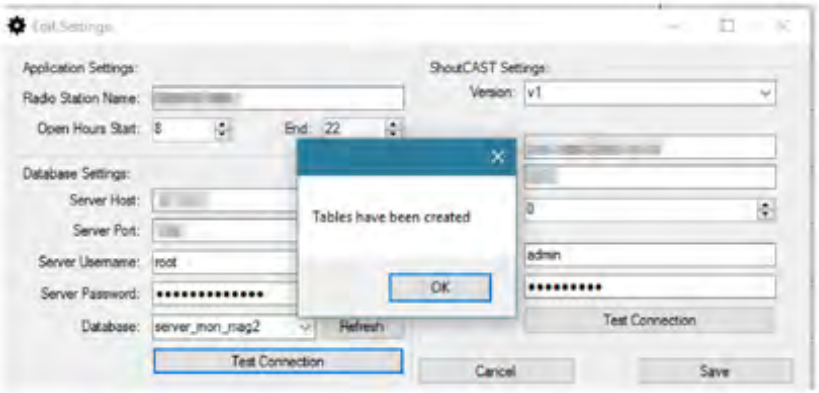
Now when the button is pressed with valid database settings:




If the button is pressed and the database does not contain the required tables:




If the yes button is pressed the tables will be created, then a success message will be displayed:




Before

| Name | Rows | Size |
|---|---|---|

After

| Name | Rows | Size |
|---|---|---|
| log | 0 | 32.0 KiB |
| shows | 0 | 16.0 KiB |
| users | 0 | 16.0 KiB |

Now the tables have been successfully created. Then all of the database settings are valid. The test button will also display any errors with the SQL settings, as it runs the same error checking method – 'testSqlServer' – so the testing from above will be the same in this example.

Now the SQL portion of the settings form is complete, I can now add the methods for the radio server settings.

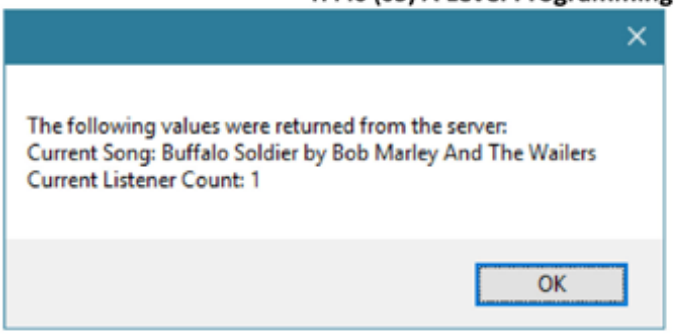
This will try to connect to the server using the credentials provided, and return the current playing song and listener count to the user. To do this I will create a new function called 'testServerConnection' which will return a Boolean value of true or false. The code for this is as follows:

```
private bool testServerConnection(bool showDetailsFromServer)
        {
            try
            {
                // Check to see if values are returned when connecting to the server
                WebClient client = new WebClient();

                // Set the web headers user agent to the Mozilla Firefox browser (as all
others appear to not work) It's the default internet headers on a computer, so it will
connect to the server without any issues
                client.Headers["User-Agent"] = "Mozilla/4.0 (Compatible; Windows NT 5.1;
MSIE 6.0) " + "(compatible; MSIE 6.0; Windows NT 5.1; " + ".NET CLR 1.1.4322; .NET CLR
2.0.50727)";

                // Store the entire XML result in a variable called result
                string result =
client.DownloadString($"http://{txtServerHost.Text}:{txtServerPort.Text}/admin.cgi?mode=v
iewxml&pass={txtServerAdminPassword.Text}");

                // Now split to obtain listener count and current song
                // The current song is between two tags of '<SONGTITLE>' and
'</SONGTITLE>' as it's C# each split needs an array of characters, so it will split a
string by a string delimiter
                string currentSong = result.Split(new string[] { "<SONGTITLE>" },
StringSplitOptions.None)[1].Split(new string[] { "</SONGTITLE>" },
StringSplitOptions.None)[0].Replace("&#x27;", "'");

                // Now do the same for the current listener count between
'<CURRENTLISTENERS>' and '</CURRENTLISTENERS>' also parse it to an integer, so it's
easier to handle later on
                int currentListenerCount = int.Parse(result.Split(new string[] {
"<CURRENTLISTENERS>" }, StringSplitOptions.None)[1].Split(new string[] {
"</CURRENTLISTENERS>" }, StringSplitOptions.None)[0]);

                if (showDetailsFromServer) { MessageBox.Show($"The following values were
returned from the server:\nCurrent Song: {currentSong}\nCurrent Listener Count:
{currentListenerCount.ToString()}"); }

                // Return true as the connection was successful
                return true;
            }
            catch (Exception e)
            {
                // If there was an error, display a message stating the contents of the
error message
                MessageBox.Show($"There was an error connecting to the Radio Server.
\n\n{e.Message}");
                return false;
            }
        }
```

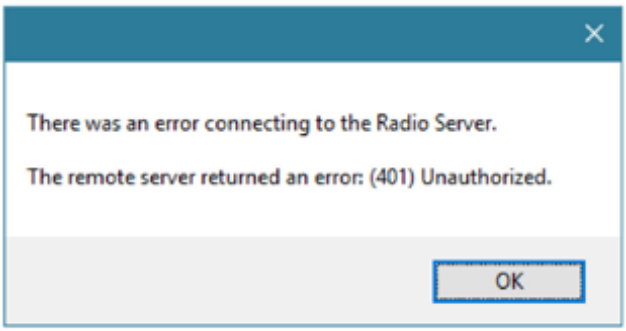
This code can now be tested by adding a call to run it from the click method of the 'Test Connection' button.

```
private void btnServerTestConnection_Click(object sender, EventArgs e)
        {
            testServerConnection(true);
        }
```

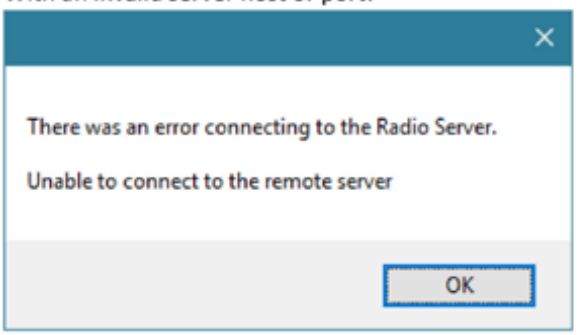
With a successful connection to the server:




With an invalid username or password:




With an invalid server host or port:




This will be the same for any errors, and will display the error content from the server. This will then allow the problem to be rectified.

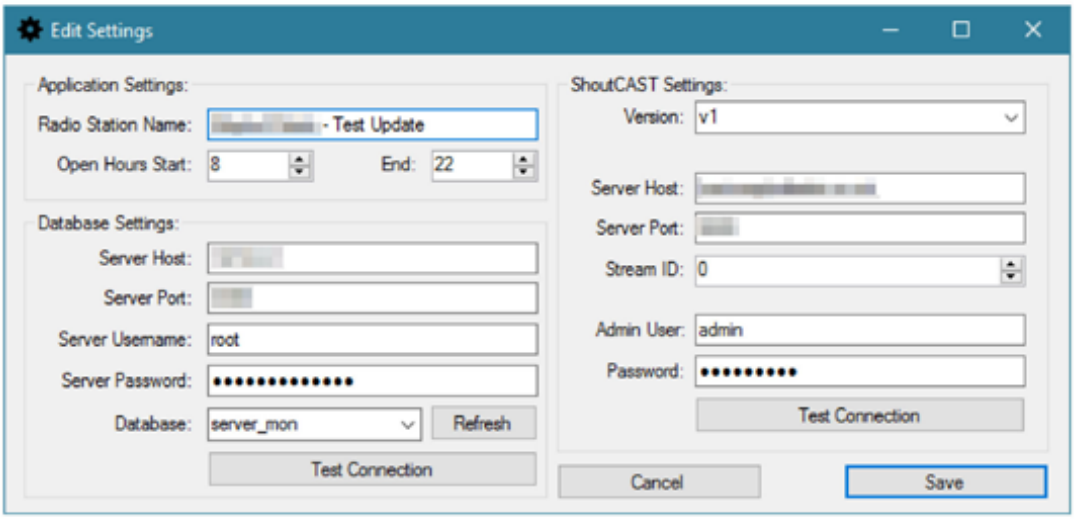
The only thing left is to now create the save settings button. This will need to check to make sure the SQL connection is valid; the SQL structure is valid; and that the radio server is returning data. It will then check to make sure there is a file for it to write to, if not it will create one, otherwise it will be updated with the new settings.

The code is as follows:

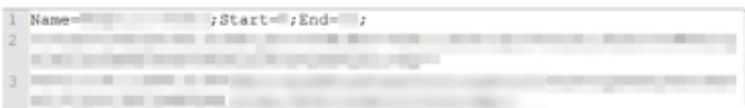
```
private void saveSettings()
        {
            // Check to see if config file exists
            if (File.Exists("config.conf"))
            {
                // Check to see if all settings are valid
                if (checkSqlStructure(false) && testServerConnection(false) &&
txtRadioName.Text != "")
                {
                        // Save settings to file via a stream writer
                        StreamWriter sw = new StreamWriter("config.conf");

sw.WriteLine($"Name={txtRadioName.Text};Start={numHoursStart.Value.ToString()};End={numHo
ursEnd.Value.ToString()};");

sw.WriteLine(Settings.EncryptToBase64($"Server={txtDatabaseServerHost.Text};Port={txtData
```

```
baseServerPort.Text};User
ID={txtDatabaseServerUsername.Text};Password={txtDatabaseServerPassword.Text};Database={c
bDatabase.Text}"));

sw.Write(Settings.EncryptToBase64($"Version={cbServerVersion.Text};Host={txtServerHost.Te
xt};Port={txtServerPort.Text};ID={numServerStreamID.Value.ToString()};Username={txtServer
AdminUser.Text};Password={txtServerAdminPassword.Text}"));

                    // Close file so it can written to again
                    sw.Close();

                    // update settings in class
                    Settings.obtainSettings();

                    // Display confirmation message
                    MessageBox.Show("Settings have been saved successfully!");
                }
                else
                {
                    // The user hasn't entered a name for the radio station
                    MessageBox.Show("Please enter a radio station name");
                }
            }
            else
            {
                // Create a new config file and save the settings again
                File.Create("config.conf");
                saveSettings();
            }
        }
```

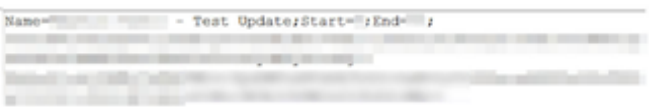
Now I can test this code by changing some of the settings and checking to see if the file updates:




Before



After

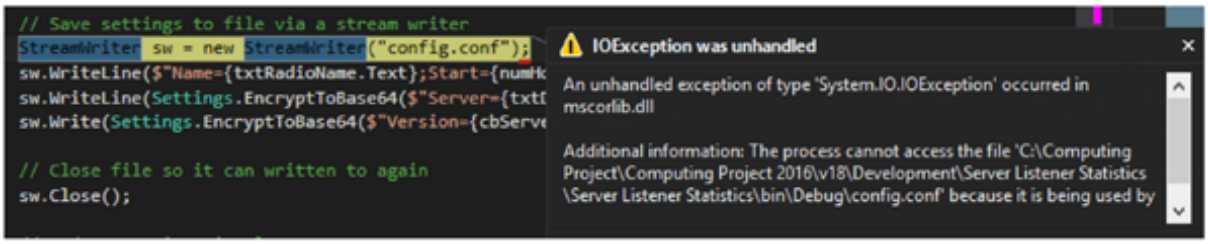


*Part of the files have had to be blurred out to protect the passwords from the actual radio server for security reasons.*

As you can see this test has succeeded, and the data has been saved successfully. The two base 64 encoded lines are identical, meaning the settings have all been successfully written to the file.

Now I will test what happens if there isn't a Config file saved already (this will be the case if it's the applications first time start-up, so a new Config file can be written). I will do this by opening the settings form, changing some settings. Then I will delete the config file and press save to see what will happen.

This caused an error shown below:




This was due to the fact the file had been created but was still being used by the process that created it. After looking to the StreamWriter, I realised that if you try to write to a file that doesn't exist, it will create it for you. This will save the headache of having to check if the file exists or not.

I can now amend the code to remove and checking to see if the file exists beforehand.

The Save Settings method now looks like the code below:

```
        // Save the user's settings
        private void saveSettings()
        {
                // Check to see if all settings are valid
                if (checkSqlStructure(false) && testServerConnection(false) &&
txtRadioName.Text != "")
                {
                        // Save settings to file via a stream writer
                        StreamWriter sw = new StreamWriter("config.conf");

sw.WriteLine($"Name={txtRadioName.Text};Start={numHoursStart.Value.ToString()};End={numHo
ursEnd.Value.ToString()};");

sw.WriteLine(Settings.EncryptToBase64($"Server={txtDatabaseServerHost.Text};Port={txtData
baseServerPort.Text};User
ID={txtDatabaseServerUsername.Text};Password={txtDatabaseServerPassword.Text};Database={c
bDatabase.Text}"));

sw.Write(Settings.EncryptToBase64($"Version={cbServerVersion.Text};Host={txtServerHost.Te
xt};Port={txtServerPort.Text};ID={numServerStreamID.Value.ToString()};Username={txtServer
AdminUser.Text};Password={txtServerAdminPassword.Text}"));

                        // Close file so it can written to again
                        sw.Close();

                        // update settings in class
                        Settings.obtainSettings();

                        // Display confirmation message
                        MessageBox.Show("Settings have been saved successfully!");
                }
                else
                {
                    // The user hasn't entered a name for the radio station
                    MessageBox.Show("Please enter a radio station name");
                }

        }
```

Now I can run the same test again, and this is successful and creates a Config file for the application to use.

config.conf

```
1 Name=[redacted] - Test Update;Start=[redacted];End=[redacted];
2 [redacted]
3 [redacted]
```

This also allowed for live swapping of databases while the application was running, this identified an issue where the system would not update any data or record any new records. This was due to the fact the system takes the current values, and checks them against the latest record in the database to see if they have changed. As the database was newly created, then there were no records for it to compare against, so it didn't log or record any data. To fix this I will add an example record to the log table, set back with a date of 1900, so it will not come up in any of the reports. I will also add an example admin user, so that they can login to the system.

This means I will run an extra query to add two new records, one to the log table and one to the users table. This is as follows:

```
INSERT INTO log (Log_DateTime,Log_ListenerCount,Log_CurrentSong) VALUES
("1900-01-01 00:0:00", "0","No Songs Yet Logged..."); INSERT INTO users
(User_FullName,User_LoginName,User_Email,User_Password,User_AccessLevel)
VALUES ("Administrator", "admin", "admin@example.com",
"$2a$12$TsunmkI9HPDBKl26XNdqGOQcPVC2RgHWo1YwM5nPpgNBgOhrMTjQq", 3);
```

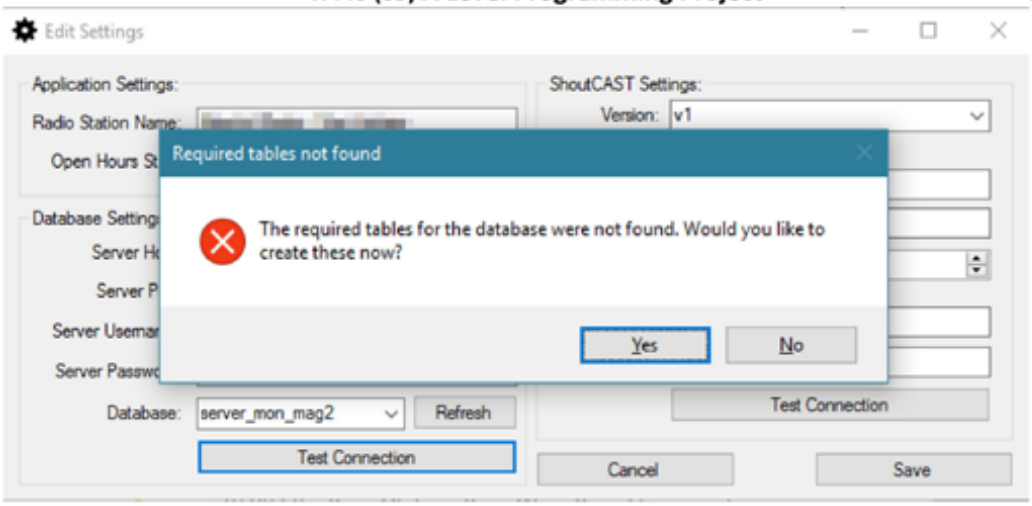
This admin account will have the highest access level, and will have a username of 'admin' and a password of 'admin', a name of 'Administrator' and an email of 'admin@example.com'.

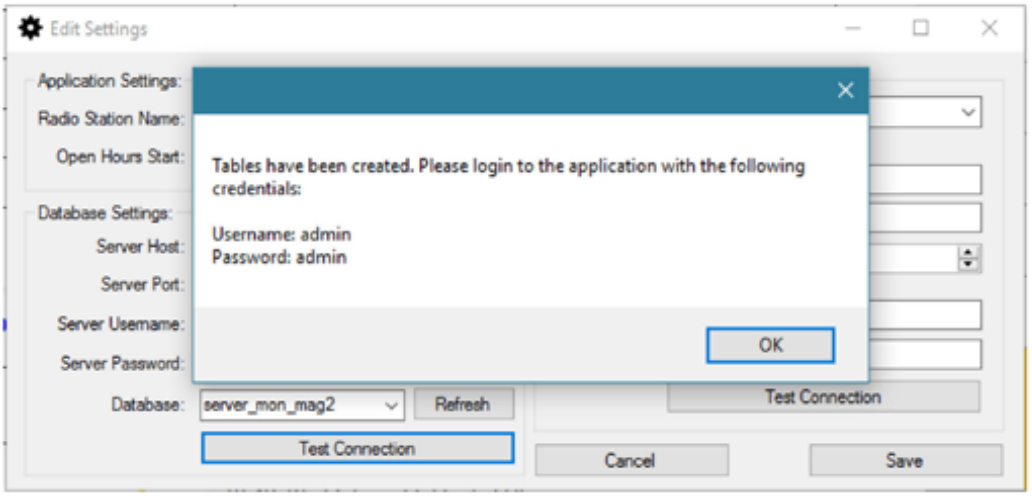
These details will be displayed in a message box once the tables have been created and the data has been added.

This query will be ran after the tables have been created in the new database, you can see the updated code below this will be inserted into the checkSqlStructure Boolean after the creation of the tables:

```
// Create user account, and add a starting log record
                        command.CommandText = "INSERT INTO log
(Log_DateTime,Log_ListenerCount,Log_CurrentSong) VALUES (\"1900-01-01 00:0:00\",
\"0\",\"No Songs Yet Logged...\"); INSERT INTO users
(User_FullName,User_LoginName,User_Email,User_Password,User_AccessLevel) VALUES
(\"Administrator\", \"admin\", \"admin@example.com\",
\"$2a$12$TsunmkI9HPDBKl26XNdqGOQcPVC2RgHWo1YwM5nPpgNBgOhrMTjQq\", 3);";
                        // Execute query to add data to database
                        command.ExecuteNonQuery();

                        // Show success message
                        MessageBox.Show("Tables have been created. Please login to the
application with the following credentials:\n\nUsername: admin\nPassword: admin");
```

I can now test this new code to see if it is working by trying to setup a new database called 'server_mon_mag2':




Before:

| Name | Rows |
|---|---|

After:

| Name | Rows |
|---|---|
| log | 1 |
| shows | 0 |
| users | 1 |

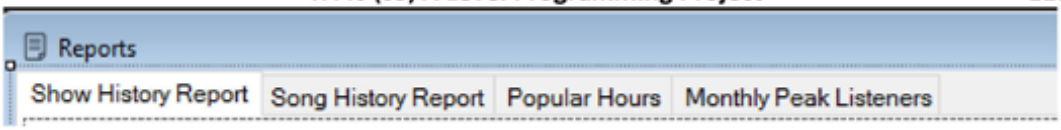
| Log_ID | Log_DateTime | Log_ListenerCount | Log_CurrentSong | Log_ShowID |
|---|---|---|---|---|
| 1 | 1900-01-01 00:00:00 | 0 | No Songs Yet Logged... | (NULL) |

| User_ID | User_FullName | User_LoginName | User_Email | User_Password | User_AccessLevel | User_LastLogin |
|---|---|---|---|---|---|---|
| 1 | Administrator | admin | admin@example.com | [redacted] | 3 | (NULL) |

As you can see from the data above, all of the tables have been created and the default values have been inserted and the data is now being logged to the new database.

Now settings have been added, I can now focus on adding the reporting section of the application. This will be a reporting form that will allow 4 different reports to be selected and then ran. These are "Show History Report", "Song History Report", "Popular Hours" and "Monthly Peak Listeners".

I will work through each report at a time, testing its functionality throughout the process. Firstly I will need to design the form that will have 4 tabs to allow selection of the different reports. I will use a tab control for this, and I will label each of the 4 tabs with their report names as you can see below:

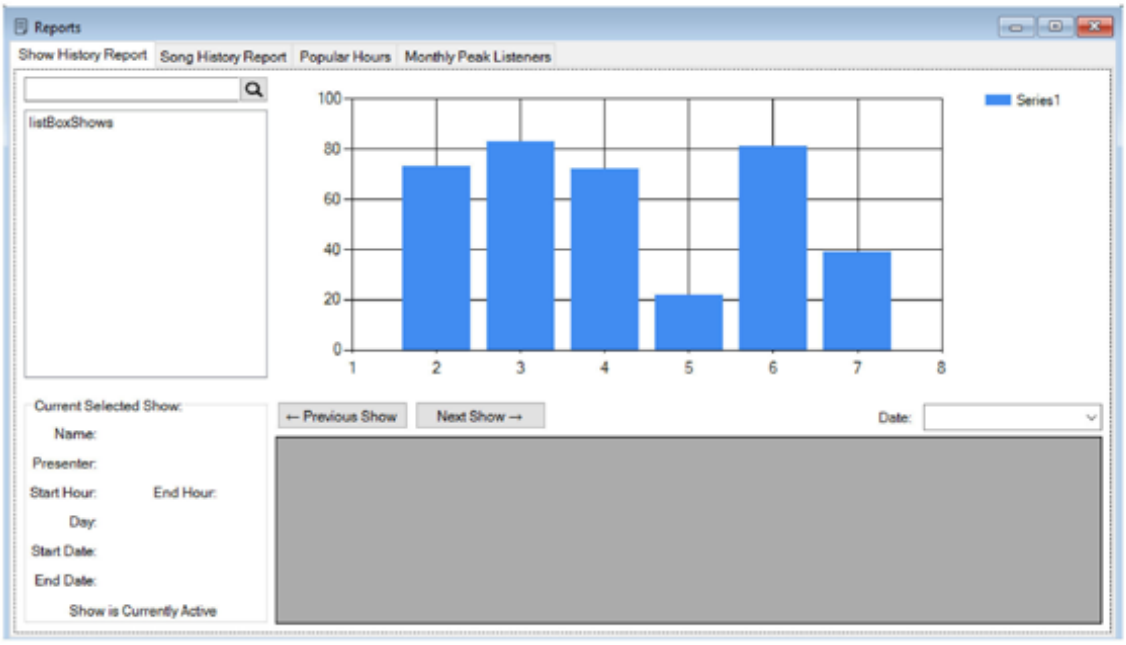



I have also set the text of the form to be 'Reports' along with setting to start in the centre of the screen, as well as having an icon that matches the one on the report button on the dashboard.

I will also add the following code to the dashboard form so that when the reports button is pressed the form will open:

```
private void reportsToolStripMenuItem_Click(object sender, EventArgs e)
        {
            // Create a new reports form object, and open it as a Dialogue window
            Reports reports = new Reports();
            reports.ShowDialog();

        }
```

Now I can start adding components to the form, the first report will be the Show History report. I have decided to make a slight design change, due to issues with trying to plot multiple shows on the one graph, it is much easier to do it show by show, so the form will search for a certain show, and will then list dates of that show, allowing you to see the average and peak listeners throughout the show. It will also give a raw data table for reference as well.




I will need to make a method that will search for shows based off their name or presenter this will then be populated in the list box for shows.

The query I will use to list show data is as follows:

```
SELECT * FROM shows WHERE Show_Name LIKE "%QUERY%" OR Show_Presenter LIKE
"%QUERY%"
```

The 'QUERY' is the user's query, and it is surrounded with the '%' character, so it will return results that contain the query as any part of them. For example, the query 'ker' would return '[redacted]' as a valid show.

| Show_ID | Show_Name | Show_Presenter | Show_Day | Show_Starttime | Show_Endtime | Show_StartDate | Show_EndDate | Show_Active |
|---|---|---|---|---|---|---|---|---|
| 1 | [redacted] | [redacted] | Saturday | 17 | 19 | (NULL) | 2017-02-14 | 0 |
| 28 | [redacted] | [redacted] | Sunday | 10 | 12 | (NULL) | (NULL) | 1 |

To prevent against SQL injection, I will be passing the query in as a parameter, rather than normally via string concatenation. This caused a lot of issues when getting the SQL syntax right to pass a parameter into a string.

Firstly I tried the following method:

```
command.CommandText = $"SELECT * FROM shows WHERE Show_Name LIKE \"%@query%\" OR
Show_Presenter LIKE \"%@query%\"";
command.Parameters.AddWithValue("@query", query);
```

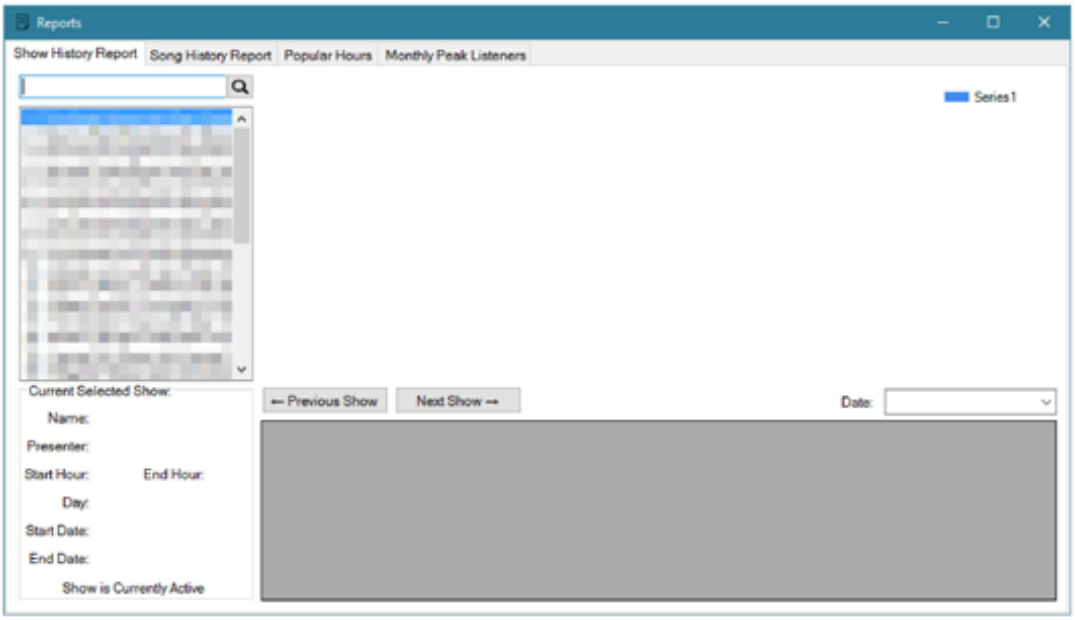
But this caused a lot of SQL syntax errors, as the '@query' parameter was part of a string. To get around this, I changed the code to the following, which concatenates the '%' character around the initial query input, then passes all of it in as a command.

```
command.CommandText = $"SELECT * FROM shows WHERE Show_Name LIKE @query OR Show_Presenter
LIKE @query";
command.Parameters.AddWithValue("@query", $"%{query}%");
```

This method worked much better as the code would actually run.

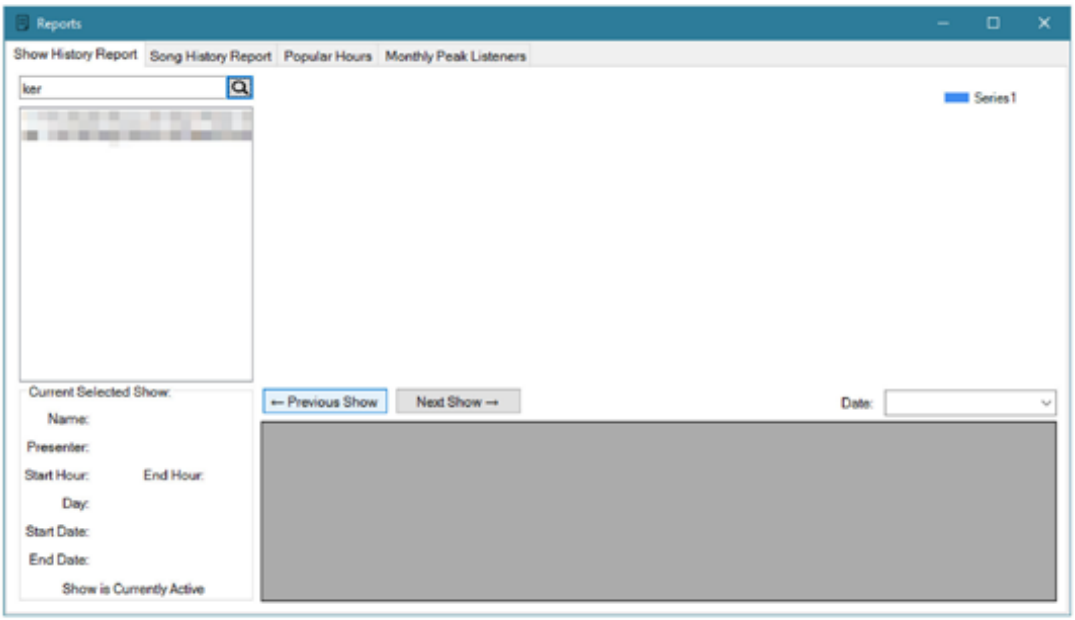
```
// List shows from DB
        private void listShows(string query)
        {
            // Create a new command to list all shows that have the query string as part
of their Show Name or Presenter Name
            MySqlCommand command = mySqlConn.CreateCommand();
            command.CommandText = $"SELECT * FROM shows WHERE Show_Name LIKE @query OR
Show_Presenter LIKE @query";
            command.Parameters.AddWithValue("@query", $"%{query}%");

            // Create a data reader to hold the results returned
            IDataReader reader = command.ExecuteReader();

            //Clear the listbox so new values can be added
            listBoxShows.Items.Clear();

            // Loop through each returned value and add it to the listbox
            while (reader.Read())
            {
                listBoxShows.Items.Add($"{reader["Show_ID"]} - {reader["Show_Name"]} with
{reader["Show_Presenter"]} ({reader["Show_Day"]})");
            }

            // Close the reader so it can be used again
            reader.Close();
        }
```

Now I can test the code in the form, by trying to search for some shows:



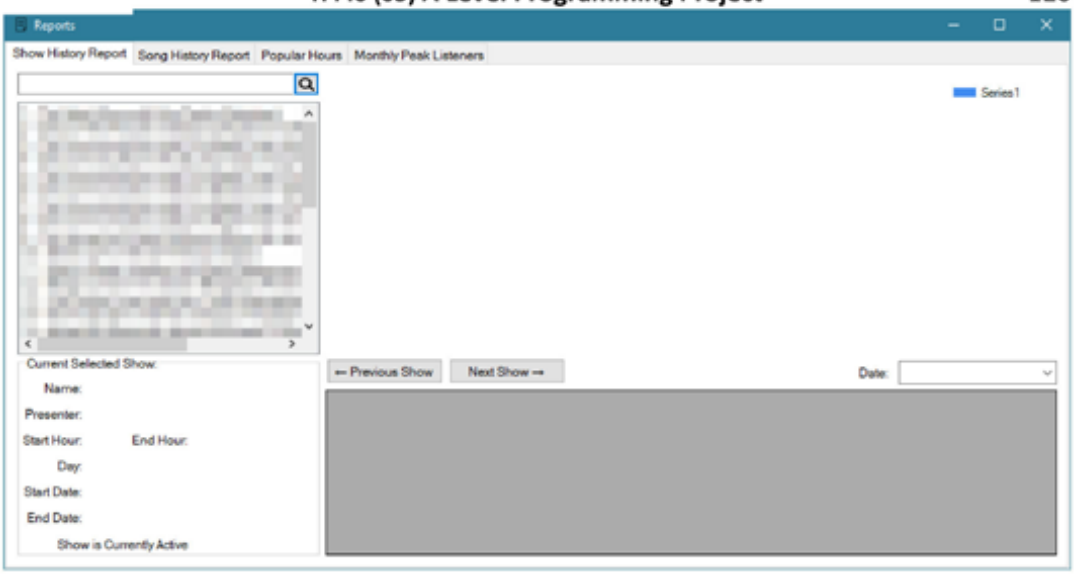

With nothing in the query box, all of the shows are returned successfully.

Now if I search with the query 'ker':




It returns two shows as both of the presenters have the characters 'ker' in their surnames.

One issue with this is it's hard to read the full name of the shows as they can be easily cut off with the size of the listbox. To fix this issue I am going to enable the horizontal scroll bar, so the list can also be scrolled left to right, so that the full name of the shows can be read. I will also make the list box a bit wider, so more text can be displayed as well. After making these changes you can see the improvements from the screenshot below:




Finally, another change I will make is so when the return key is pressed on the keyboard, the shows will be searched through, instead of having to manually pressing the search button every time. This can be done by adding the following code to the text box's KeyDown event:

```
private void txtQuery_KeyDown(object sender, KeyEventArgs e)
        {
            // If the enter key is pressed, search for the text entered
            if (e.KeyCode == Keys.Enter)
            {
                btnSearch.PerformClick();
            }
        }
```

After testing this it is working great. And makes searching for shows feel a lot more natural.

Now I can move onto creating the method to obtain show details, this will take a show ID, then return relevant information about the show, and setting this to the values of the labels on the form. It will also obtain a list of all of the dates the show has occurred on. These dates will be added to the combo box, so a certain date can be selected.

The first SQL statement will be used to obtain the information about a show in particular:

```
SELECT * FROM shows WHERE Show_ID = ID
```

For example, if the show ID was '1' it would return the following information:

| Show_ID | Show_Name | Show_Presenter | Show_Day | Show_Starttime | Show_Endtime | Show_StartDate | Show_EndDate | Show_Active |
|---|---|---|---|---|---|---|---|---|
| 1 | [redacted] | [redacted] | Saturday | 17 | 19 | (NULL) | 2017-02-14 | 0 |

The second SQL statement will list a set of dates a show has aired according to the log, this query is as follows:

```
SELECT DATE(Log_DateTime) AS 'Date' FROM log WHERE Log_ShowID = ID GROUP BY
DAY(Log_DateTime) ORDER BY Log_DateTime ASC
```

For example, if the show ID was '11' it would return the following information:

| Date |
|---|
| 2017-02-01 |
| 2017-02-22 |
| 2017-03-08 |
| 2017-03-15 |

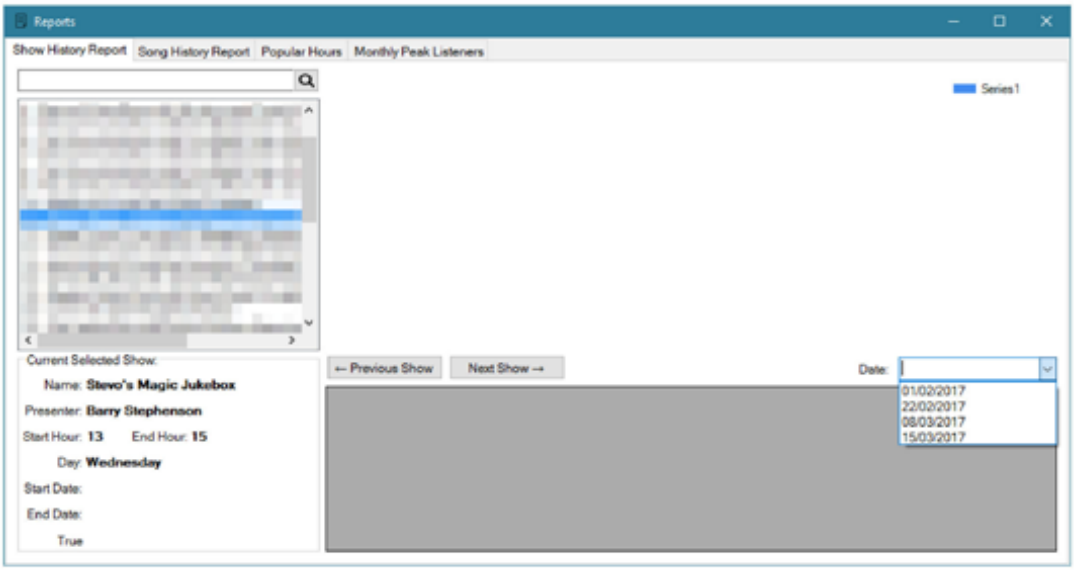
Now this data can be added to the relevant combo boxes and labels. The show name obtained from the database will need to have a Regular Expression applied to it. This is because when the name of the show is read from the database it has a newline character appended to it. This will have to be removed otherwise the name will not be displayed.

```
        // Obtain show data
        private void obtainShowData(int showID)
        {
            // Create a command to obtain show data
            MySqlCommand command = mySqlConn.CreateCommand();
            command.CommandText = $"SELECT * FROM shows WHERE Show_ID =
{showID.ToString()}";

            // Create a data reader to hold the results returned
            IDataReader reader = command.ExecuteReader();

            if (reader.Read())
            {
                // Clear current values in labels
                lblShowName.Text = "";
                lblShowPresenter.Text = "";
                lblStartHour.Text = "";
                lblEndHour.Text = "";
                lblDay.Text = "";
                lblStartDate.Text = "";
                lblEndDate.Text = "";
                lblActive.Text = "";

                // Update values in labels
                lblShowName.Text = Regex.Replace(reader["Show_Name"].ToString(),
@"\t|\n|\r", ""); ;
                lblShowPresenter.Text = reader["Show_Presenter"].ToString();
                lblStartHour.Text = reader["Show_Starttime"].ToString();
                lblEndHour.Text = reader["Show_Endtime"].ToString();
                lblDay.Text = reader["Show_Day"].ToString();
                lblStartDate.Text = reader["Show_StartDate"].ToString();
                lblEndDate.Text = reader["Show_EndDate"].ToString();
                lblActive.Text = reader["Show_Active"].ToString();
            }

            // Close the reader so it can be used again
            reader.Close();

            // Find the past dates of shows
            command.CommandText = $"SELECT DATE(Log_DateTime) AS 'Date' FROM log WHERE
Log_ShowID = {showID.ToString()} GROUP BY DAY(Log_DateTime) ORDER BY Log_DateTime ASC";

            // Create a data reader to hold the results returned
            reader = command.ExecuteReader();

            // Clear all of the current dates in the combo box
            comboBoxDate.Items.Clear();

            // Clear the current selected value in the combo box
            comboBoxDate.Text = "";

            // Loop through all of the values returned
            while (reader.Read())
            {
                // Add each item to the combination box
                comboBoxDate.Items.Add(reader["Date"]);
            }

            // Close the data reader so it can be used again
            reader.Close();

            }
```

Before this can be tested, I will need to add the code to the event ran when the selected item is changed, this will then update the information on the form when the item is clicked on:

```
private void listBoxShows_SelectedIndexChanged(object sender, EventArgs e)
        {
            // Obtain show data for the current selected show
            obtainShowData(int.Parse(listBoxShows.Text.Split('-')[0].Trim(' ')));
        }
```

Now I can test this by checking various shows to see if the correct show information is displayed, and if the show's air dates are displayed in the combo box correctly.




This test has succeeded, as the dates are displayed in the combo box to the right of the screen, and these match the dates obtained from the database. This also works for switching between other shows, and will display their dates and correct information as well. Two improvements I will make on this is to change the value of 'True' or 'False' at the bottom of the show details to give a more accurate description of what 'True' (the show is active) or 'False' (the show is not active) means. I will also change the start and end dates from the format of "DD/MM/YY HH:MM:SS" to just "DD/MM/YY". This will make it a lot easier to read, and will not make it look as confusing.

To change the code to make the label display more useful information about a shows active status, I will change the following code:

```
lblActive.Text = reader["Show_Active"].ToString();
```

To the following:

```
// Check to see if the show is active or not, and display an appropriate message
                if (bool.Parse(reader["Show_Active"].ToString()))
                {
                    lblActive.Text = "The show is currently active";
                }
                else
                {
                    lblActive.Text = "The show is NOT active";
                }
```

This can now be tested

For an active show:

The small print

We'd like to know your view on the resources we produce. By clicking on the 'Like' or 'Dislike' button you can help us to ensure that our resources work for you. When the email template pops up please add additional comments if you wish and then just click 'Send'. Thank you.

Whether you already offer OCR qualifications, are new to OCR, or are considering switching from your current provider/awarding organisation, you can request more information by completing the Expression of Interest form which can be found here: www.ocr.org.uk/expression-of-interest



Please get in touch if you want to discuss the accessibility of resources we offer to support delivery of our qualifications: resources.feedback@ocr.org.uk

## Looking for a resource?

There is now a quick and easy search tool to help find **free** resources for your qualification:

www.ocr.org.uk/i-want-to/find-resources/

**www.ocr.org.uk/alevelreform**

OCR Customer Contact Centre

**General qualifications**

Telephone 01223 553998
Facsimile 01223 552627
Email general.qualifications@ocr.org.uk

OCR is part of Cambridge Assessment, a department of the University of Cambridge. *For staff training purposes and as part of our quality assurance programme your call may be recorded or monitored.*



Cambridge Assessment

LLOYD'S REGISTER LRQA
ISO 9001
UKAS MANAGEMENT SYSTEMS
001