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 2

Version 1

Contents – Part 2

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.

Exemplar 7 – Set C (High)

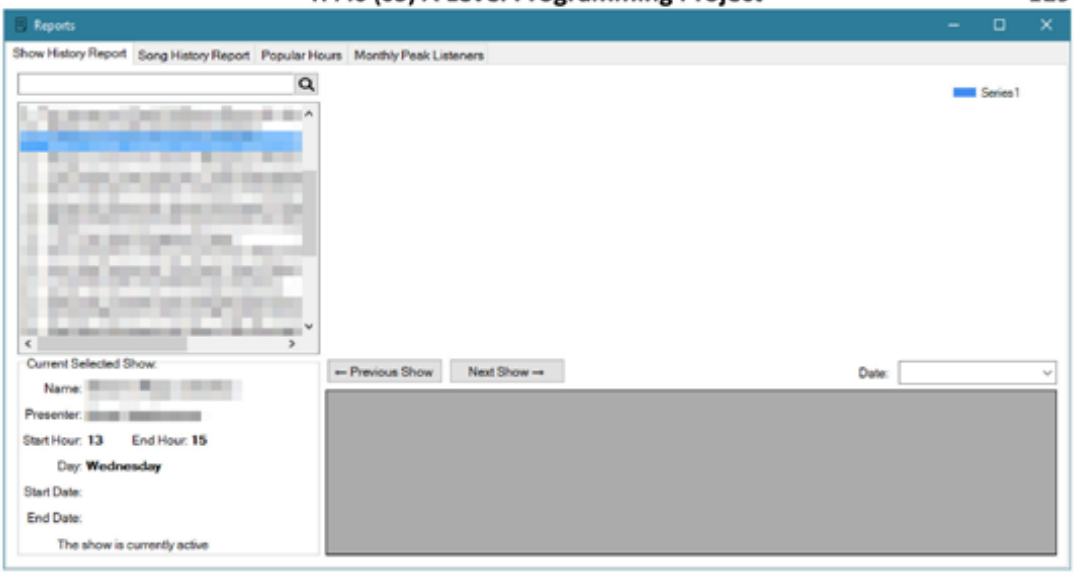
Programming project (non exam assessment)

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

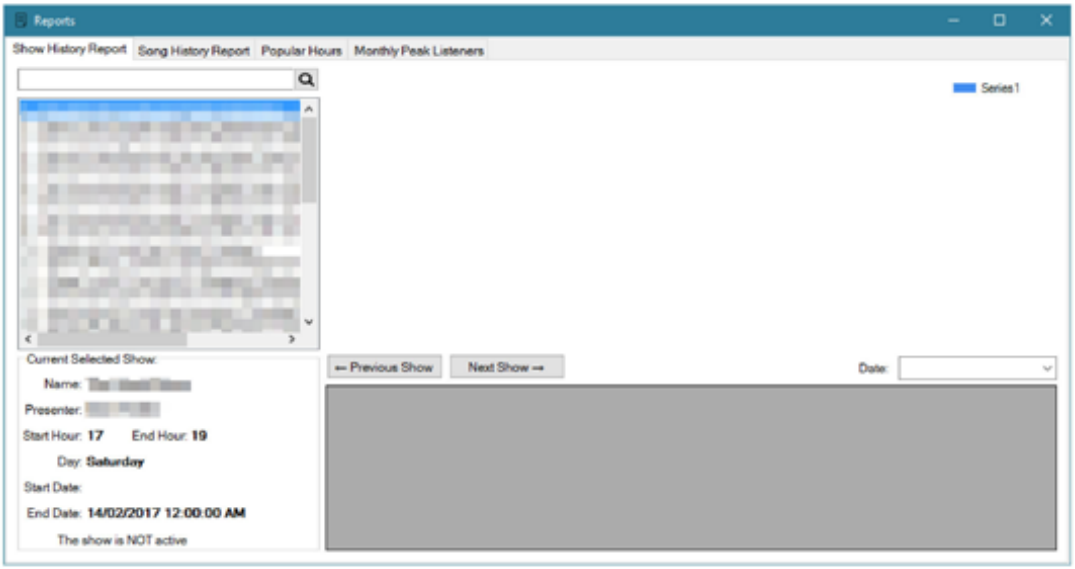
H446 (03) A Level Programming Project

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



For an inactive show:



Both of these tests have passed, as the message is now displaying more relevant text. Finally I will now also change the Start Date and End Date to not include the time on the end. This can be done by adjusting the SQL statement from the following:

```
SELECT * FROM shows WHERE Show_ID = {showID.ToString()}
```

To the following

```
SELECT Show_ID, Show_Name, Show_Presenter, Show_Day, Show_Starttime, Show_Endtime,
DATE(Show_StartDate) AS 'Show_StartDate', DATE(Show_EndDate) AS 'Show_EndDate',
Show_Active FROM shows WHERE Show_ID = {showID.ToString()}
```

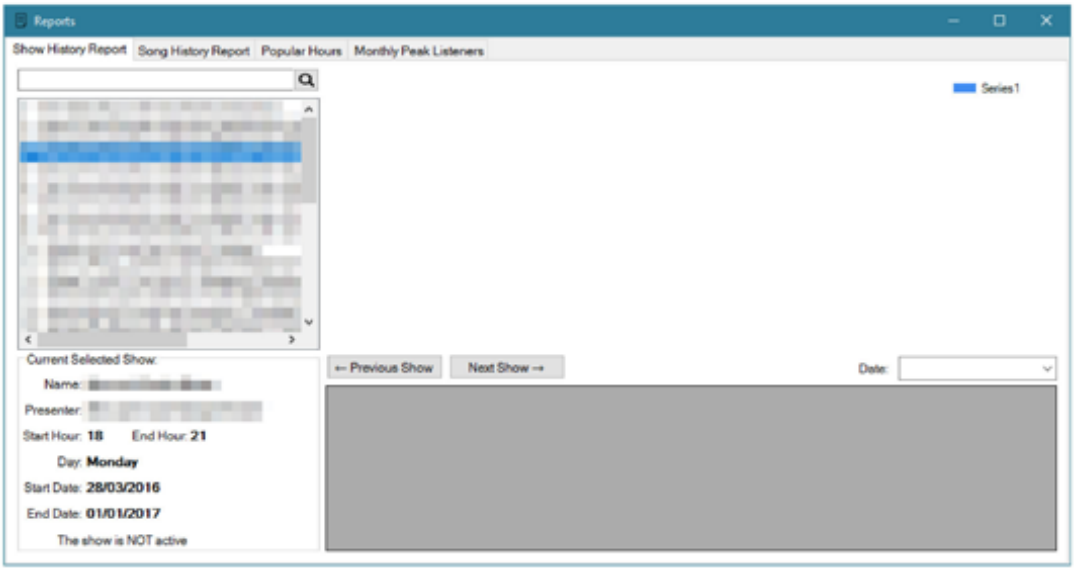
And the following code that assigns the text values to the labels:

```
lblStartDate.Text = reader["Show_StartDate"].ToString();
lblEndDate.Text = reader["Show_EndDate"].ToString();
```

To the following:

```
// Split the date to just display the date and drop the time off the end
lblStartDate.Text = reader["Show_StartDate"].ToString().Split(' ')[0];
lblEndDate.Text = reader["Show_EndDate"].ToString().Split(' ')[0];
```

This will make sure the date is in the correct format, and as you can see below, it looks a lot better as both of the tests have now passed.



Now I can move onto the code that will plot the graph with the relevant show data obtained from the database. This will show the average listener count per minute, throughout the duration of the show

The SQL statement to do this is as follows:

```
SELECT TIME(Log_DateTime) AS 'Time', ROUND(AVG(Log_ListenerCount), 0) AS
'Count', Log_CurrentSong FROM log WHERE Log_ShowID = 26 AND Log_DateTime
between "2017-03-27 18:00:00" and "2017-03-27 18:59:59" GROUP BY
MINUTE(log.Log_DateTime) ORDER BY Log_DateTime ASC;
```

This will load the following data for a show with an ID of 26, and between the times of 18:00:00 and 19:00:00. It will return the following data:

Time	Count	Log_CurrentSong
18:00:22	2	Cat Amongst The Pigeons by Bros
18:04:22	2	White Rabbit by Jefferson Airplane
18:06:52	2	Cool For Cats by Squeeze
18:07:22	1	Cool For Cats by Squeeze
18:10:22	1	Dark Horse by Katy Perry Fe. Juicy J
18:13:52	1	Cat's In The Cradle by Ugly Kid Joe
18:17:52	1	Your Station. Your Way. Your Community.
18:20:22	1	The Lion Sleeps Tonight by Tight Fit
18:23:52	1	Hound Dog by Elvis Presley
18:25:52	1	Hot Dog by Shakin' Stevens
18:28:22	1	Your Station. Your Way. Your Community.
18:29:22	2	Your Station. Your Way. Your Community.
18:32:22	2	Hungry Like The Wolf by Duran Duran
18:33:22	2	Lions by Skillet
18:34:52	1	Lions by Skillet
18:36:22	1	Songbird by Oasis
18:38:22	2	Your Station. Your Way. Your Community.
18:40:22	2	Lovebird by Leona Lewis
18:43:22	2	See You Later Alligator by Bill Haley And ...
18:45:22	2	Guns And Horses by Ellie Goulding
18:48:52	2	See You Later Alligator by Bill Haley And ...
18:51:22	2	Crazy Horses by Osmonds
18:53:52	2	Your Station. Your Way. Your Community.
18:54:22	2	Only The Horses by Scissor Sisters
18:56:52	3	Only The Horses by Scissor Sisters
18:57:22	2	Only The Horses by Scissor Sisters

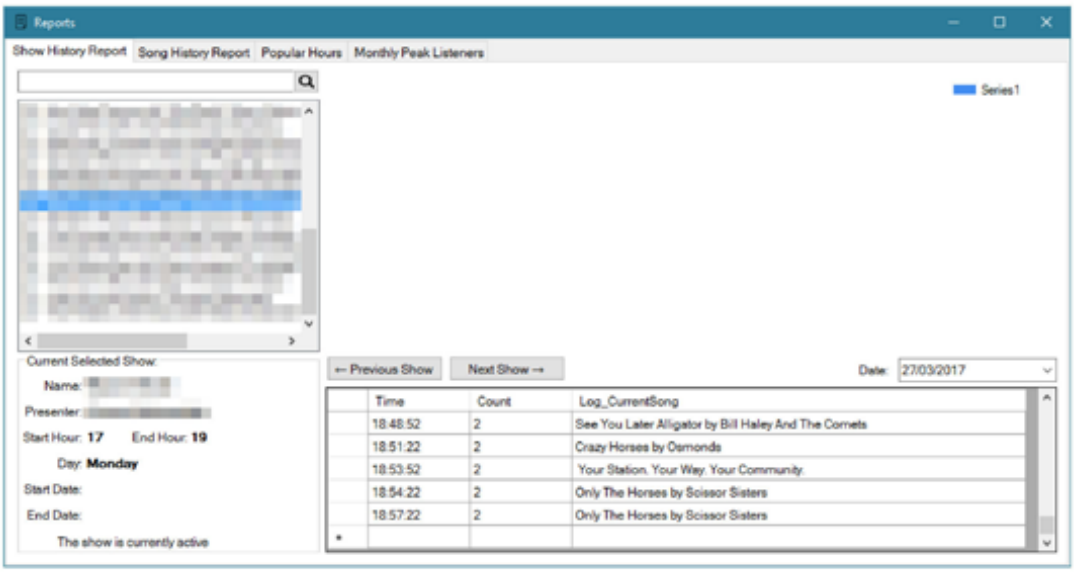
This data can then be added to the table to show the data raw, and it can also be plotted on the graph as well. I will first add the method that will load the data obtained into the data grid view. This code is as follows:

```
private void obtainShowStats(int showID, int startHour, int endHour)
        {
            // Create new query to obtain show history data
            MySqlCommand command = mySqlConn.CreateCommand();

            // Convert date to SQL form
            String date = DateTime.Parse(comboBoxDate.Text).ToString("yyyy-MM-dd");

            // Create command to query database
            command.CommandText = $"SELECT TIME(Log_DateTime) AS 'Time',
ROUND(AVG(Log_ListenerCount), 0) AS 'Count', Log_CurrentSong FROM log WHERE Log_ShowID =
{showID} AND Log_DateTime between \"{date} {startHour}:00:00\" and \"{date} {endHour -
1}:59:59\" GROUP BY MINUTE(log.Log_DateTime) ORDER BY Log_DateTime ASC;";

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

            // Create a data table to hold the data so it can be displayed in the data
grid view
            DataTable dt = new DataTable();
            dt.Load(reader);

            // Assign data source to table
            dataGridView1.DataSource = dt;

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

The date has had to be converted from the usual form of "DD/MM/YYYY" to SQL's date time representation of "YYYY-MM-DD". I will also need to add a call to this method on the event that is ran when the selected index of the combo box is changed:

```
private void comboBoxDate_SelectedIndexChanged(object sender, EventArgs e)
        {
            // Obtain show stats for current show
            obtainShowStats(int.Parse(listBoxShows.Text.Split('-')[0].Trim(' ')),
int.Parse(lblStartHour.Text), int.Parse(lblEndHour.Text));
        }
```

This can now be tested by selecting a show and checking to see if the data matches what should be returned from the database:



This matches the data stored for the show in the database. There are two improvements I would like to make on this view though. Firstly, I would like the 'Log_CurrentSong' to say 'Current Song', and I would like the data grid view to be made read only, so that data cannot be modified.

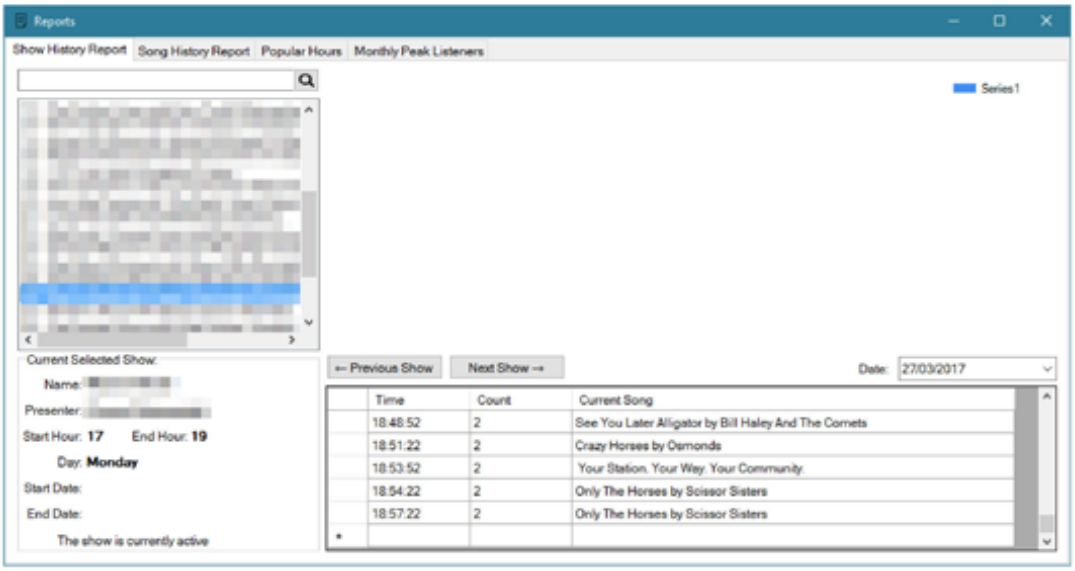
Firstly I will adjust the SQL query from:

```
SELECT TIME(Log_DateTime) AS 'Time', ROUND(AVG(Log_ListenerCount), 0) AS
'Count', Log_CurrentSong FROM log WHERE Log_ShowID = 26 AND Log_DateTime
between "2017-03-27 18:00:00" and "2017-03-27 18:59:59" GROUP BY
MINUTE(log.Log_DateTime) ORDER BY Log_DateTime ASC;
```

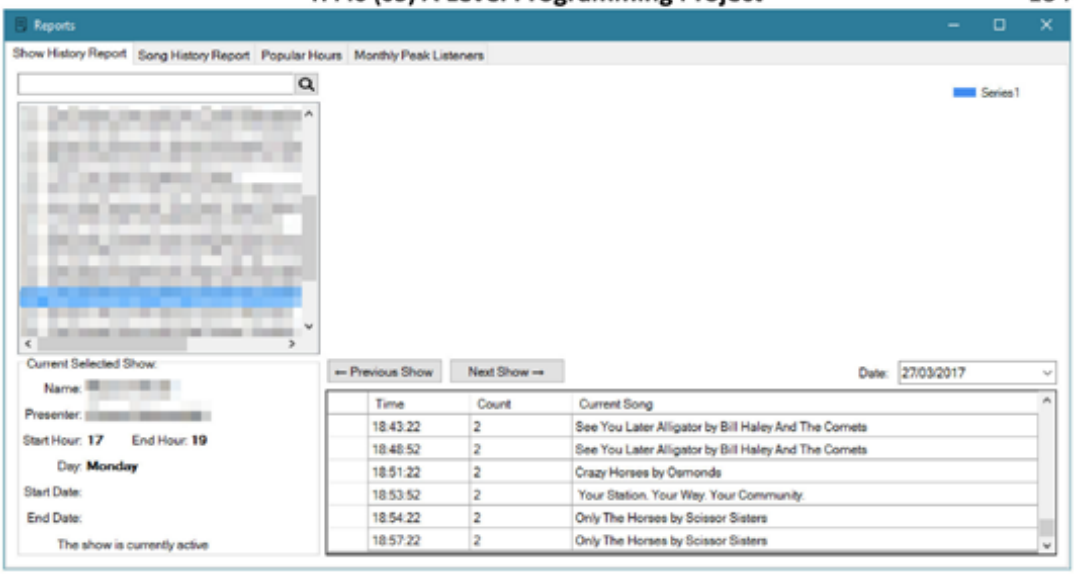
To

```
SELECT TIME(Log_DateTime) AS 'Time', ROUND(AVG(Log_ListenerCount), 0) AS
'Count', Log_CurrentSong AS 'Current Song' FROM log WHERE Log_ShowID = 26 AND
Log_DateTime between "2017-03-27 18:00:00" and "2017-03-27 18:59:59" GROUP BY
MINUTE(log.Log_DateTime) ORDER BY Log_DateTime ASC;
```

As you can see from the screenshot below, this has now fixed that issue:



The only thing left to fix is that data can be edited in the data grid view, and new data can be added. I will change this by setting the 'Read Only' property of the data grid view to 'True'. I will also set 'AllowUserToAddRows' and 'AllowUserToDeleteRows' to false. As you can see from the screenshot below, this now prevents the table from being edited or changed.



The only other issue is that when you click on a show with no data then the data grid view will not clear itself. I can fix the issue by adding the following line to the method that runs when the selected index of the listbox changes (that currently also calls the obtain show data method to update the form's information to the selected show).

```
// Wipe current data source data
dataGridView1.DataSource = null;
```

Now the data table is working I can create the graph to hold the data.

I will need to import a system reference to the top of the class:

```
using System.Windows.Forms.DataVisualization.Charting;
```

This will allow me to setup the chart in code and modify settings on it.

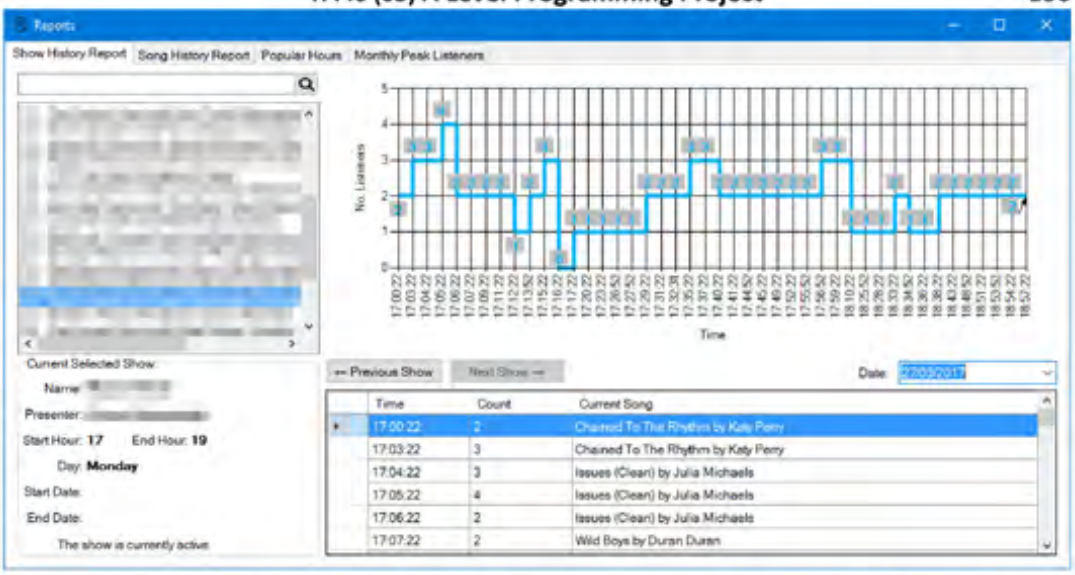
I will create a setup void that will setup the chart with its relevant settings, this is below:

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

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

            // Setup Y Axis
            chartShowHistory.ChartAreas[0].Axes[1].Title = "No. Listeners";
            chartShowHistory.ChartAreas[0].Axes[1].IsLabelAutoFit = true;
            chartShowHistory.ChartAreas[0].Axes[1].LabelAutoFitStyle =
LabelAutoFitStyles.LabelsAngleStep30;
            chartShowHistory.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
            chartShowHistory.Series[0].ChartType = SeriesChartType.StepLine;
            chartShowHistory.Series[0].LegendText = "Listener Count";
            chartShowHistory.Series[0].BorderWidth = 4;
            chartShowHistory.Series[0].Color = Settings.colours.Blue;
            chartShowHistory.Series[0].LabelForeColor = Settings.colours.Blue;
        }
```

This will now allow for the data to be plotted to the graph. I will need to modify the obtainShowStats method to also take the reader and add the values to a graph:

```
private void obtainShowStats(int showID, int startHour, int endHour)
        {
            // Create new query to obtain show history data
            MySqlCommand command = mySqlConn.CreateCommand();

            // Convert date to SQL form
            String date = DateTime.Parse(comboBoxDate.Text).ToString("yyyy-MM-dd");

            // Create command to query database
            command.CommandText = $"SELECT TIME(Log_DateTime) AS 'Time',
ROUND(AVG(Log_ListenerCount), 0) AS 'Count', Log_CurrentSong AS 'Current Song' FROM log
WHERE Log_ShowID = {showID} AND Log_DateTime between \"{date} {startHour}:00:00\" and
\"{date} {endHour - 1}:59:59\" GROUP BY MINUTE(log.Log_DateTime) ORDER BY Log_DateTime
ASC;";

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

            // Create a data table to hold the data so it can be displayed in the data
grid view
            DataTable dt = new DataTable();
            dt.Load(reader);

            // Assign data source to table
            dataGridView1.DataSource = dt;

            // Set 3rd column to expand
            dataGridView1.Columns[2].AutoSizeMode = DataGridViewAutoSizeColumnMode.Fill;

            //Clear graph so data can be plotted
            chartShowHistory.Series[0].Points.Clear();

            // Create dictionary average to hold data to plot onto graph
            Dictionary<string, int> dictionaryAverage = new Dictionary<string, int>();

            // Data reader from data table
            reader = dt.CreateDataReader();

            // Loop through all results returned and add them to the dictionary
            while (reader.Read())
            {
                dictionaryAverage.Add(reader["Time"].ToString(),
int.Parse(reader["Count"].ToString()));
            }

            // Cycle through each item in the dictionary and add it to the graph
            foreach (KeyValuePair<string, int> point in dictionaryAverage)
            {
                chartShowHistory.Series[0].Points.AddXY(point.Key, point.Value);
            }

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

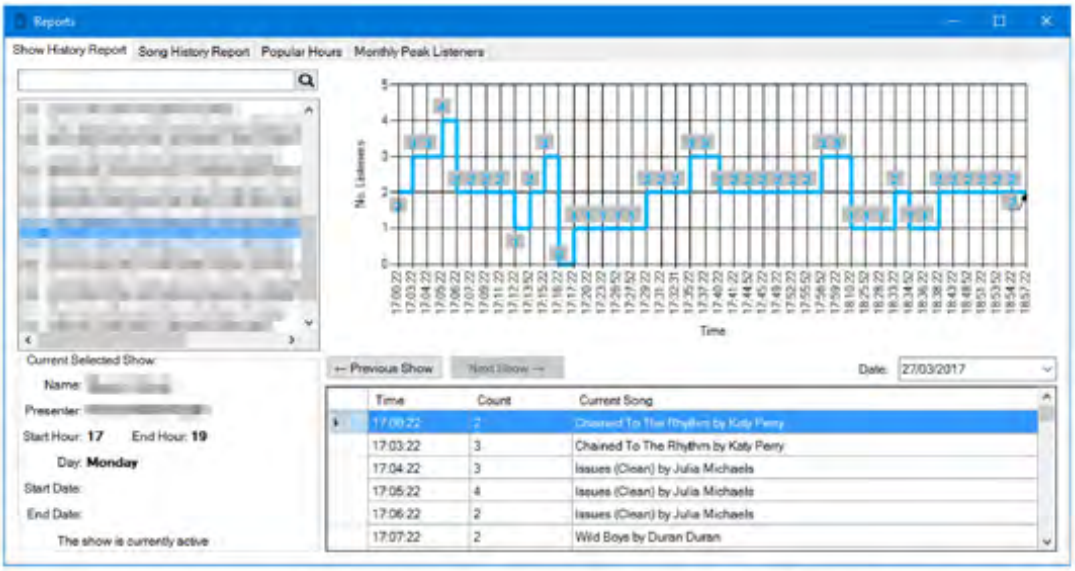
I can now test this method when it is ran by changing the index of the combo box:



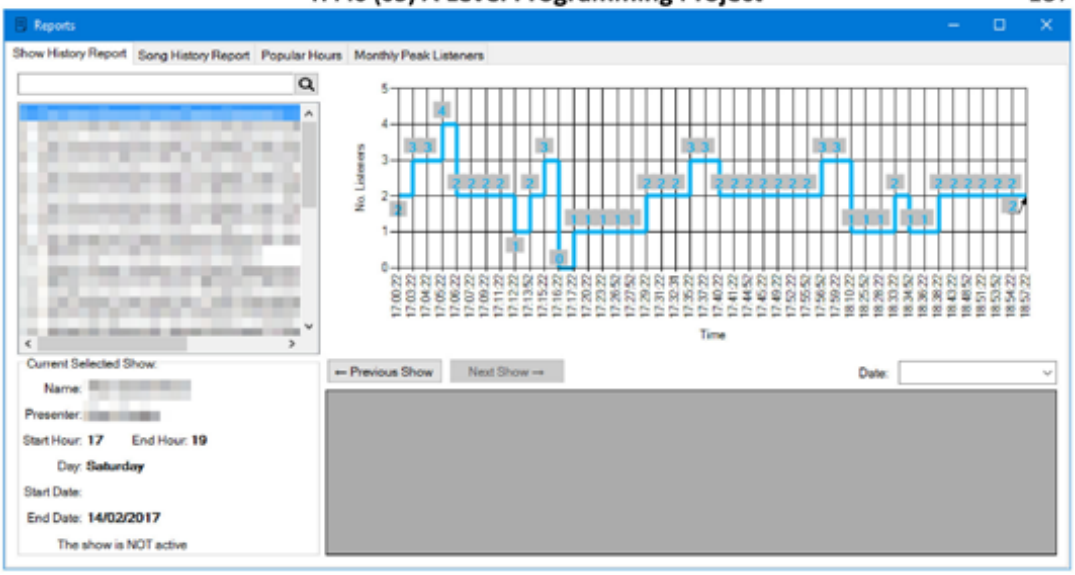
This now works great. The only issue is that when a show is first clicked on, the graph is not updated with the values until a date is selected from the combo box. It would be much nicer if that when the show is clicked on, the combo box will have the last value in it selected (as that will be the most recent show). This can be done by adding the following line of code to the ObtainShowStats method:

```
// Select the last item in the combo box
comboBoxDate.SelectedIndex = comboBoxDate.Items.Count - 1;
```

Now I can test this again, and when a show is selected, the graph is automatically created for the most recent show:



The only other issue is that if a show is selected with no data recorded the following is displayed:



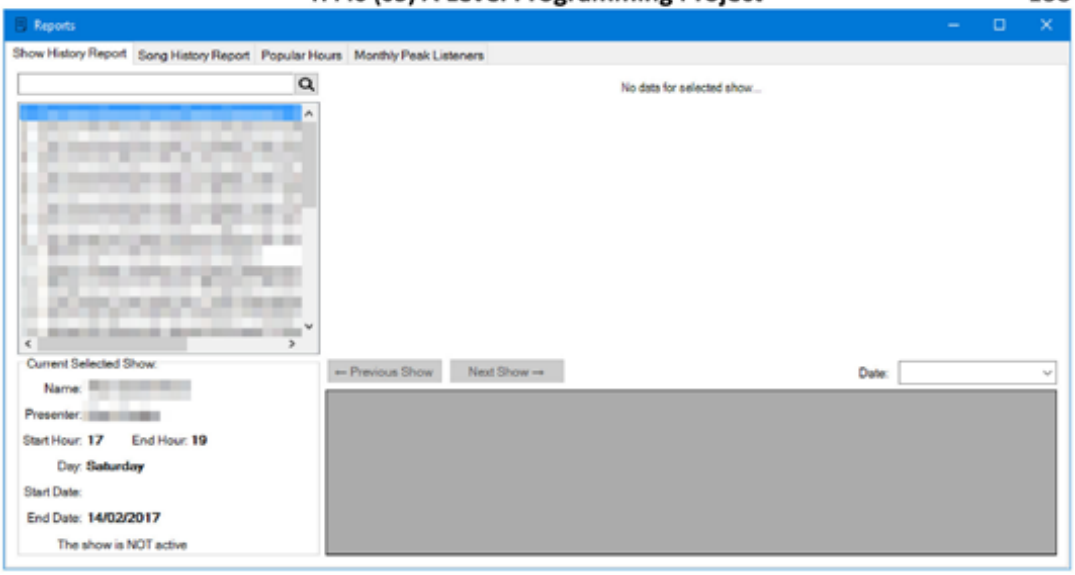
While the data and date fields are blank, the graph still holds it values, so I will need to add a check to see if a show is updated, and if there are no items in the combo box (then there has been no data recorded for the show), I can set the graph to clear all points, and I will set the title to "No data for selected show...". To do this I will create a title for the chart in the setupShowHistoryGraph method by adding the following lines to it, making sure to set its visibility to 'False' so it is not displayed on all graphs:

```
// Add title for no data
chartShowHistory.Titles.Add("No data for selected show...");
chartShowHistory.Titles[0].Visible = false;
```

Now I can add the following code to the obtainShowData method:

```
// Check to see if there are any values in the combo box
            if (comboBoxDate.Items.Count == 0)
            {
                //Clear graph so data can be plotted
                chartShowHistory.Series[0].Points.Clear();

                // Set title to say no data could be loaded
                chartShowHistory.Titles[0].Visible = true;

                // Disable the show navigation buttons
                btnNextShow.Enabled = false;
                btnPreviousShow.Enabled = false;
            }
            else
            {
                // Hide the title on the graph as it is not needed
                chartShowHistory.Titles[0].Visible = false;
            }
```

I have also started to add some code to enable the functionality of the Next and Previous show buttons, these will both have to be disabled as if there is no data you don't want the user to be able to switch to shows that don't exist.



This works great, and even when changing between different shows the graph is cleared for all shows with no data.

Now I can work on adding the functionality for the next show and previous show buttons.

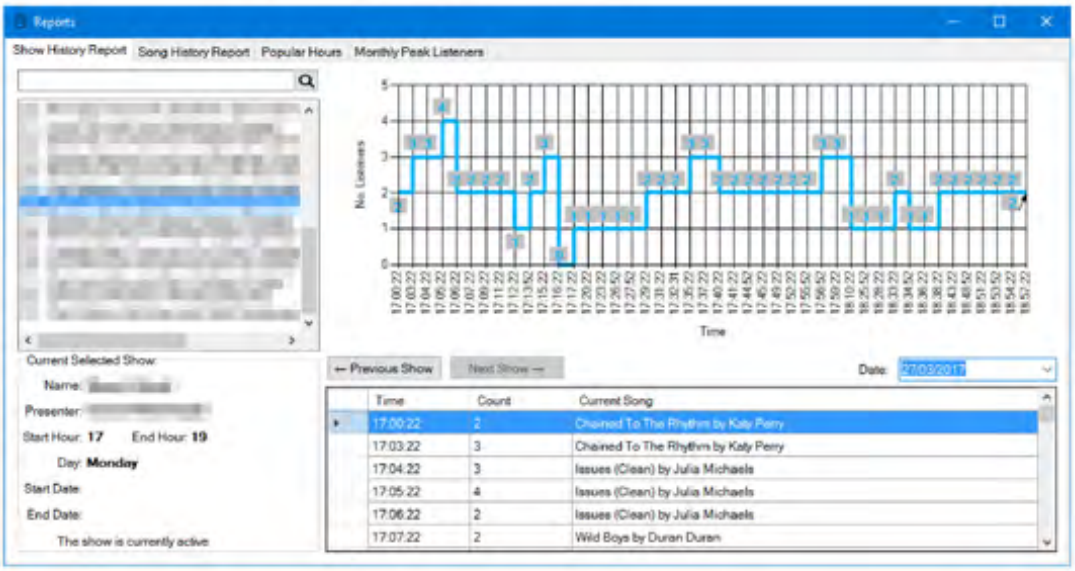
```
private void btnPreviousShow_Click(object sender, EventArgs e)
{
       // Move backwards one show (change the index of the combo box)
       comboBoxDate.SelectedIndex = comboBoxDate.SelectedIndex - 1;
}

private void btnNextShow_Click(object sender, EventArgs e)
{
       // Move forwards one show (change the index of the combo box)
       comboBoxDate.SelectedIndex = comboBoxDate.SelectedIndex + 1;
}
```

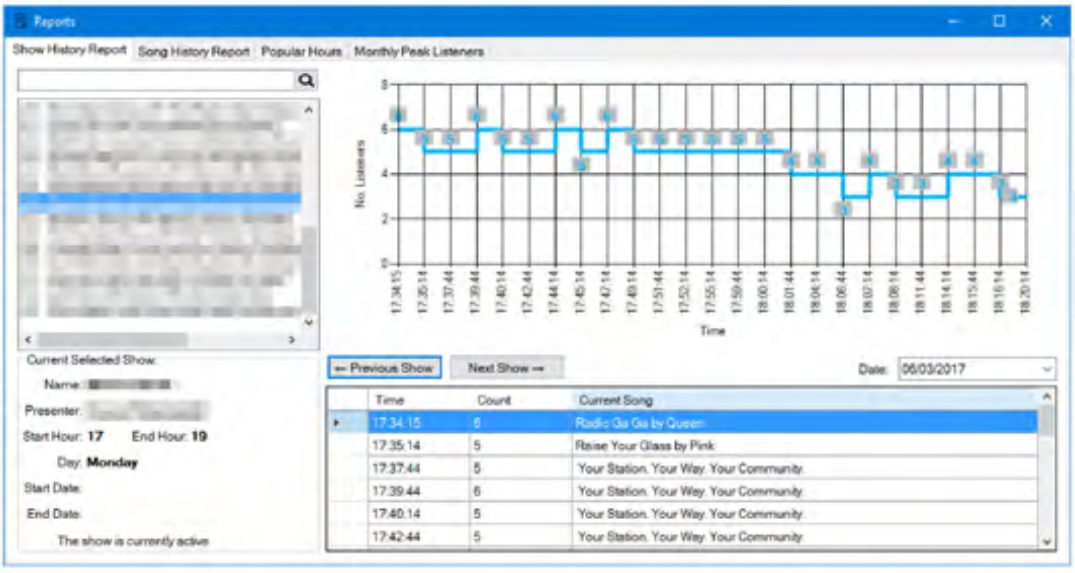
Now I will need to add some code to the method that is ran when the selected index of the combo box is changed, so that I can work out if there are any more shows to navigate to.

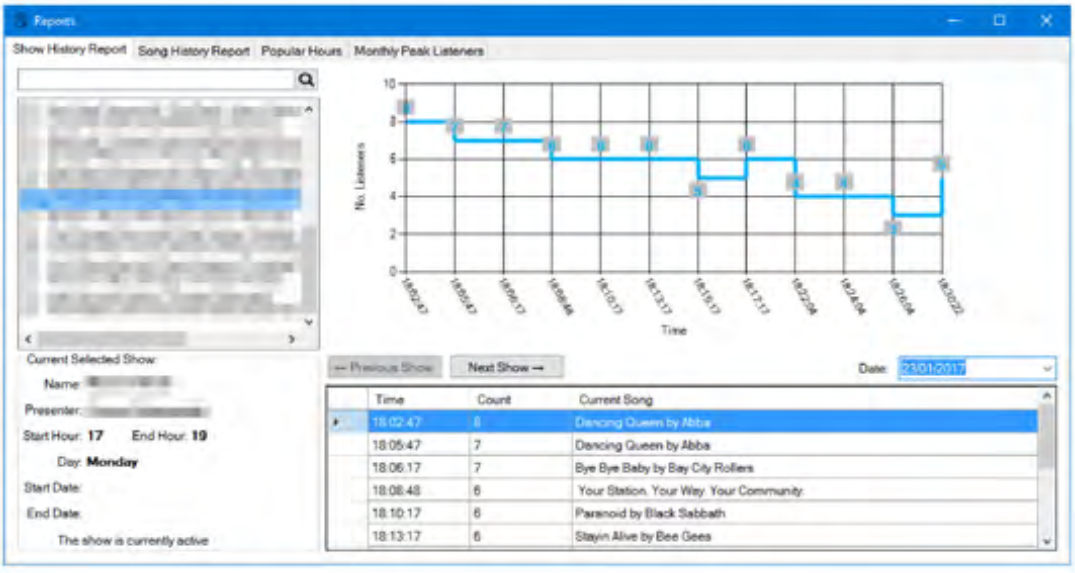
```
// Check to see if there are any shows prior or after to enable the respective buttons
            if(comboBoxDate.Items.Count < 2)
            {
                // If there are 0 or 1 show, then there is nowhere to navigate to, so
disable both buttons
                btnNextShow.Enabled = false;
                btnPreviousShow.Enabled = false;
            }
            else if (comboBoxDate.SelectedIndex == comboBoxDate.Items.Count - 1)
            {
                // If the last item in the listbox is selected, disable the next show
button and enable the previous button
                btnNextShow.Enabled = false;
                btnPreviousShow.Enabled = true;
            }
            else if(comboBoxDate.SelectedIndex == 0)
            {
                // If the first index is selected enable the next show button, but
disable the previous show button
                btnNextShow.Enabled = true;
                btnPreviousShow.Enabled = false;
            }
            else
            {
                // Either direction is fine as there is a show above and below the
current point so enable both buttons
                btnNextShow.Enabled = true;
```

```
            btnPreviousShow.Enabled = true;
        }
```

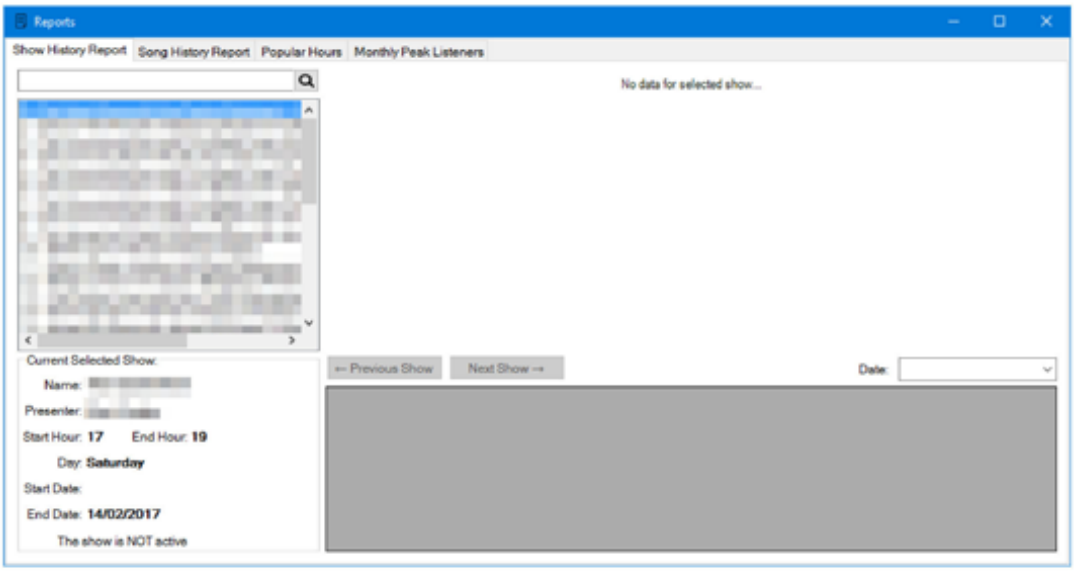
Now this can be tested by scrolling through the hours in a show:



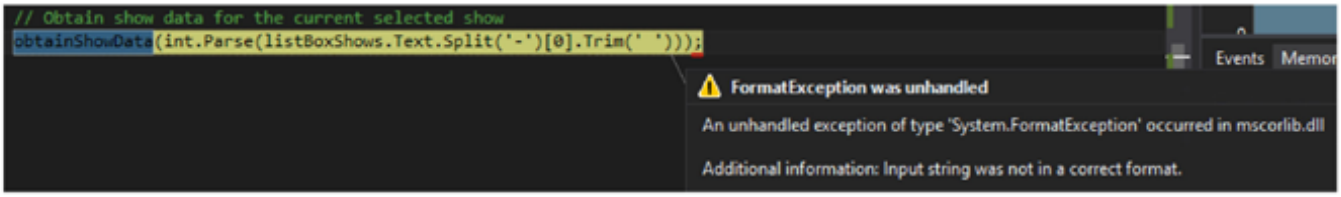




As you can see from the screenshots above, the buttons are working great, and are enabling and being disabled at the correct points. Now the only thing left to test is that both of the buttons will disable themselves when a show with no data is selected:



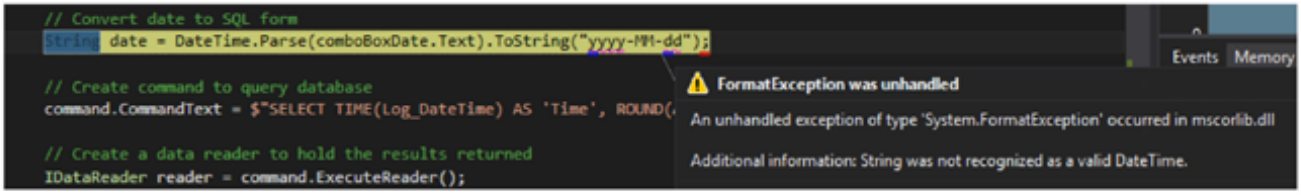
This works great. And now the show reporting and selection is working great. During further testing I have noticed there are 2 errors. The first one is caused by if the list box is clicked with no show being selected the following error occurs as there was no show to query:



This can be fixed by changing the code to the following:

```
int showid;

// Check to make sure an index is selected
if (int.TryParse(listBoxShows.Text.Split('-')[0].Trim(' '), out showid))
{
        // Obtain show data for the current selected show
        obtainShowData(showid);
}
```

This now works, and will prevent the application from crashing if something is searched for, and the list box is clicked on without any item being selected. The second issue is that a user can type any value into the combo box, and press the enter key, or the tab key, and the application will error out, as the string they have entered is not a date, as you can see below:



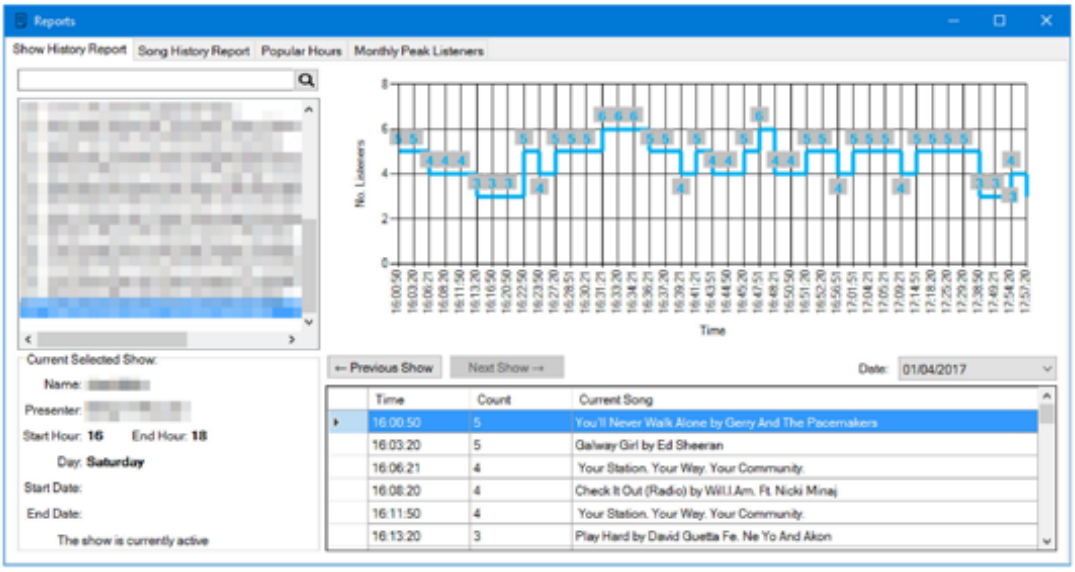
This can be fixed by setting the following properties of the combo box:

AutoCompleteMode = SuggestAppend

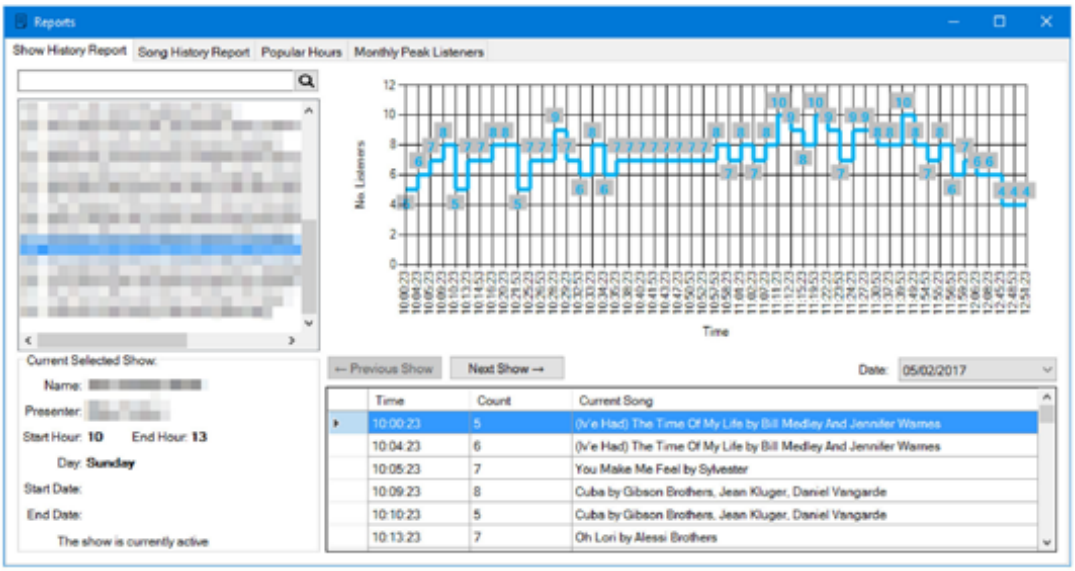
AutoCompleteSource = ListItems

DropDownStyle = DropDownList

Now as you can see from the screenshot below, the data can't be edited in the combo box, and only one of the set values can be selected:



After looking back at show data recorded for various shows, shows that are 3 hours or more, and all displayed on one graph, and the data can start to become very clumped together, as you can see from the screenshot below:



For this 3 hour show the data can become hard to read. To help solve these I recently came across a feature of charts called XCursor and YCursor. These allow the user to select points or a range of data and zoom into certain areas of the graph. I am going to use the YCursor to plot a dotted red line at the maximum listener value throughout the show as a point of reference, and I will use the XCursor to enable zooming into certain sections of a graph. These cursors can be enabled by adding the following code to the chart's setup method that I called 'setupShowHistoryGraph'

```
//Add selection and peak line setup
chartShowHistory.ChartAreas[0].CursorX.IsUserSelectionEnabled = true;
chartShowHistory.ChartAreas[0].CursorY.LineWidth = 2;
chartShowHistory.ChartAreas[0].CursorY.LineDashStyle = ChartDashStyle.Dash;
```

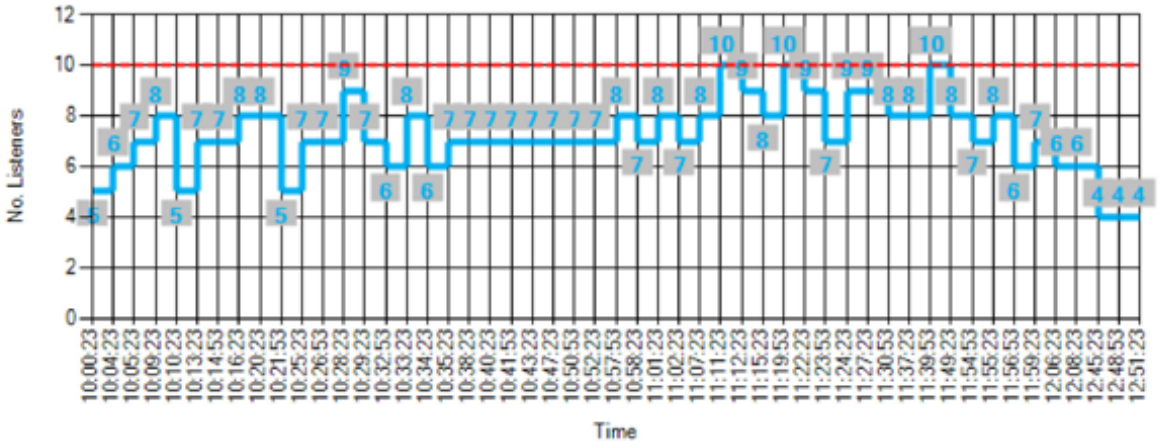
I will also need to calculate the peak listener count during the show, and set this to be the value of the YCursor. To do this I will update the code in the obtainShowStats method. I will change the following code:

```
foreach (KeyValuePair<string, int> point in dictionaryAverage)
{
        chartShowHistory.Series[0].Points.AddXY(point.Key, point.Value);
}
```

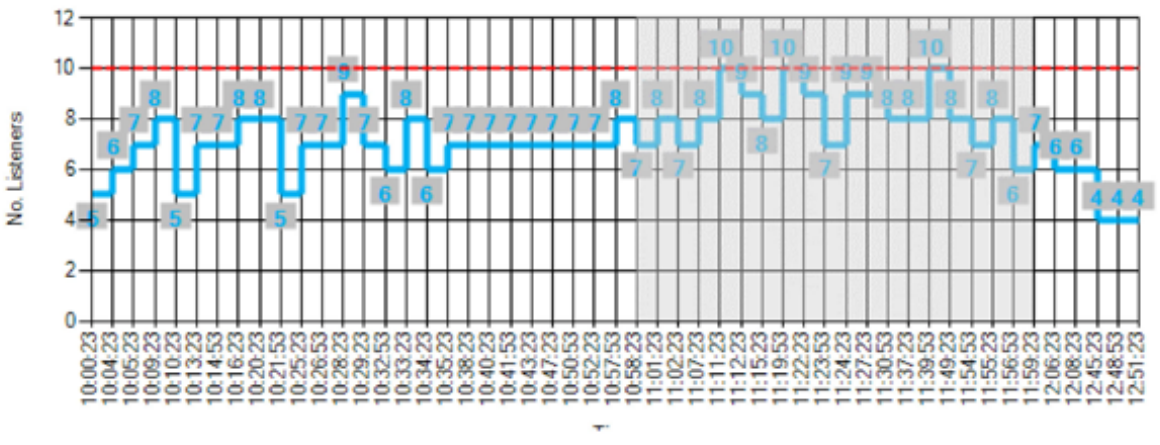
To the following:

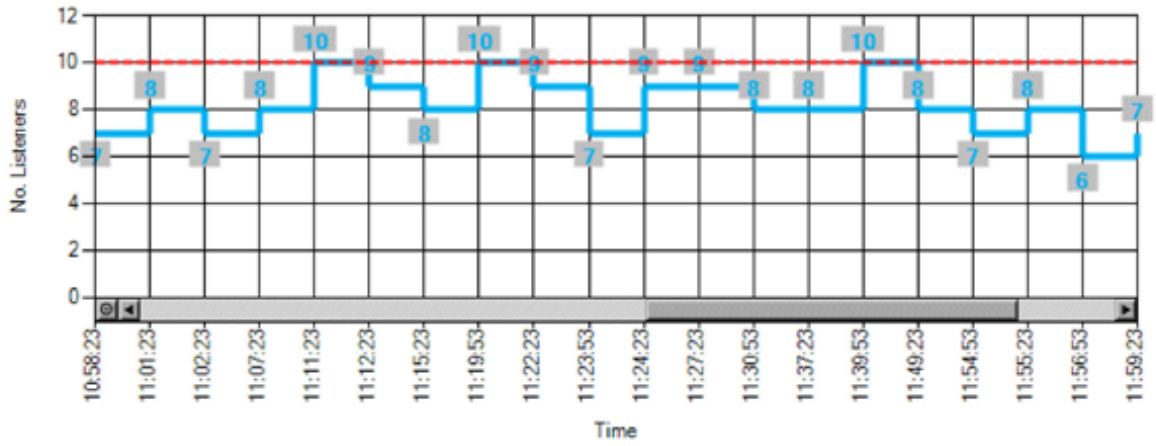
```
// Create a MAX integer to hold the peak count for the show
int max = 0;

// Cycle through each item in the dictionary and add it to the graph
foreach (KeyValuePair<string, int> point in dictionaryAverage)
{
        chartShowHistory.Series[0].Points.AddXY(point.Key, point.Value);

        // update the max count so it holds the peak value so far
        max = Math.Max(max, point.Value);
}

// Set this value as the YCursor position on the graph
chartShowHistory.ChartAreas[0].CursorY.Position = max;
```

Now when looking at the show graph, you can see the red max line plotted on it:

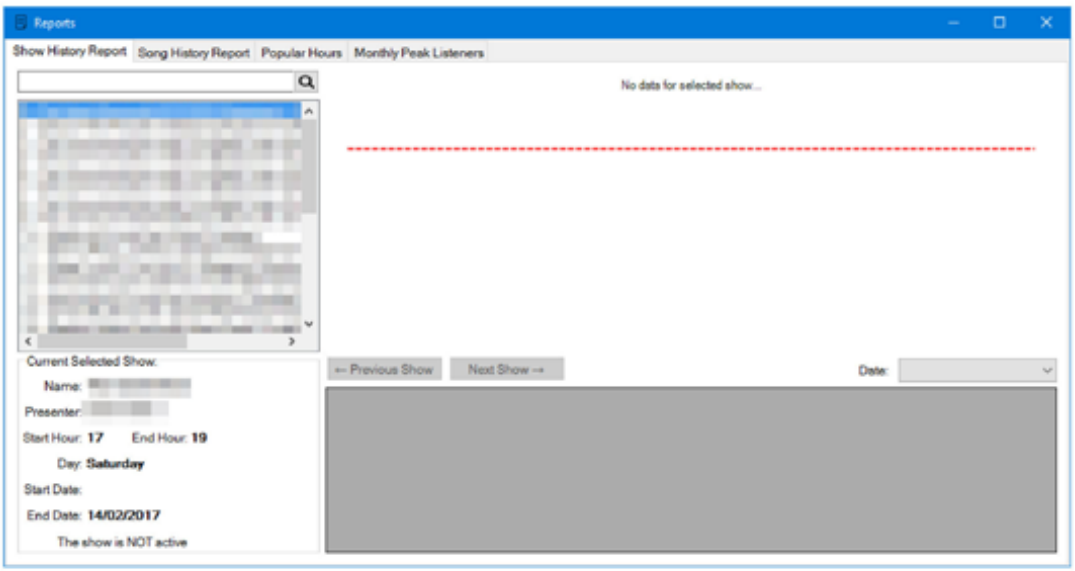


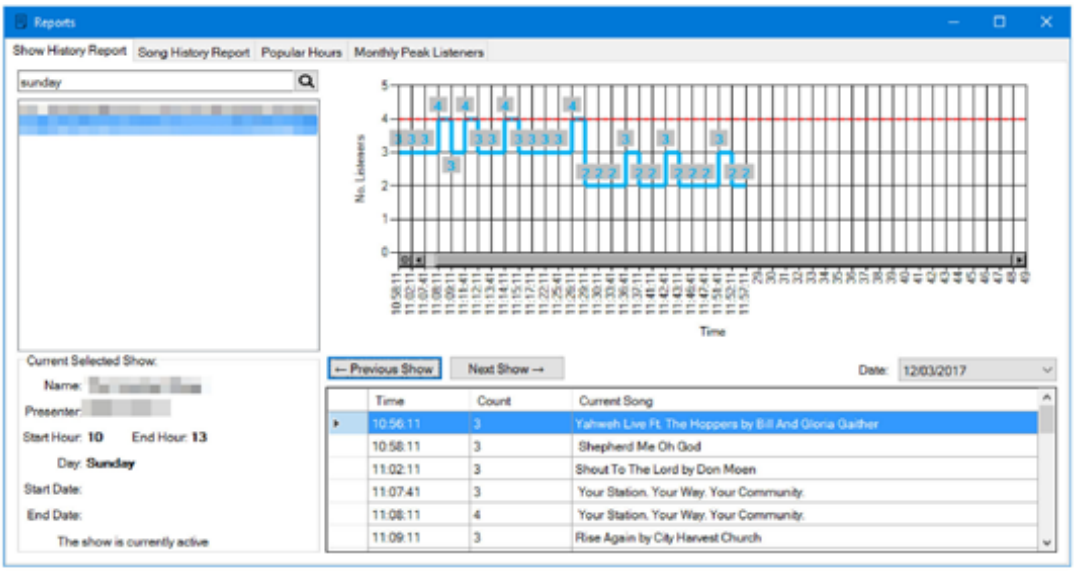
You can also drag and select a certain portion of the graph to zoom in on it. After selecting a certain area, and releasing the selection the graph will zoom in and a scroll bar will appear on the bottom allowing you to scroll to a different section of the graph to see in closer detail the data plotted on it, as you can see from the screenshot below:





This works great, and allows for a much better view of shows that have a lot of data. The only issue with this is that if you are on a show, and are zoomed into a section, then when you switch to a show with no data the cursors are still on the graph. And if you switch to a show with data outside the dataset, then the application will either crash, or the graph will display incorrectly, as you can see below:





This can be fixed by hiding the cursors on the graph when a show with no data is loaded, and by resetting the zoom level on the graph when another show is selected.

This can be achieved by adding the following code to the end of the method that is ran when the selected index of the list box is changed:

```
// Reset the zoom level on the chart, needs a while loop
// as there is no way of knowing how many times the chart was zoomed in
while (chartShowHistory.ChartAreas[0].AxisX.ScaleView.IsZoomed)
{
        chartShowHistory.ChartAreas[0].AxisX.ScaleView.ZoomReset();
}
```

And change the following as well:

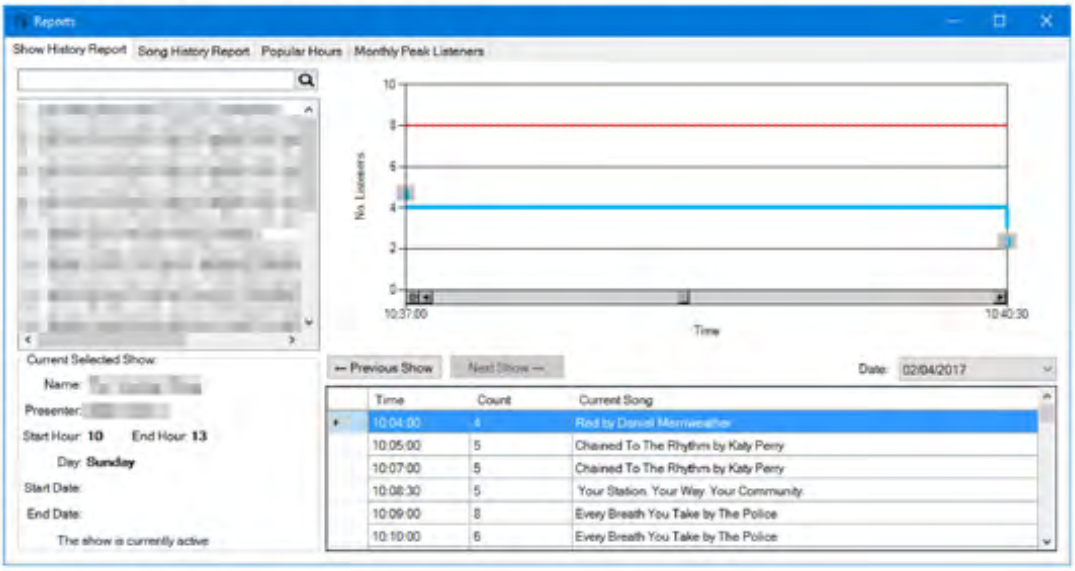
```
// check to see if there are any values in the combo box
            if (comboBoxDate.Items.Count == 0)
            {
                //Clear graph so data can be plotted
                chartShowHistory.Series[0].Points.Clear();

                // Set title to say no data could be loaded
                chartShowHistory.Titles[0].Visible = true;

                // disable the show navigation buttons
                btnNextShow.Enabled = false;
                btnPreviousShow.Enabled = false;
            }
            else
            {
                // Hide the title on the graph as it is not needed
                chartShowHistory.Titles[0].Visible = false;


            }
```

To the following:

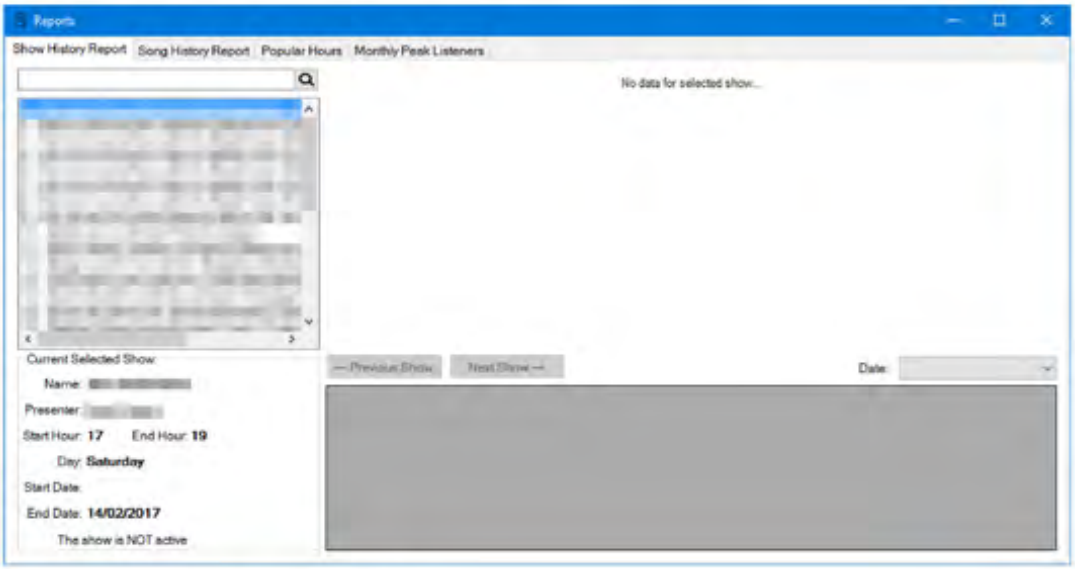
```
// check to see if there are any values in the combo box
            if (comboBoxDate.Items.Count == 0)
            {
                //Clear graph so data can be plotted
                chartShowHistory.Series[0].Points.Clear();

                // Set title to say no data could be loaded
                chartShowHistory.Titles[0].Visible = true;

                // hide the cursor
                chartShowHistory.ChartAreas[0].CursorX.IsUserSelectionEnabled = false;
                chartShowHistory.ChartAreas[0].CursorY.LineWidth = 0;

                // disable the show navigation buttons
                btnNextShow.Enabled = false;
                btnPreviousShow.Enabled = false;
            }
            else
            {
                // Hide the title on the graph as it is not needed
                chartShowHistory.Titles[0].Visible = false;

                // Enable the cursors again
                chartShowHistory.ChartAreas[0].CursorX.IsUserSelectionEnabled = true;
                chartShowHistory.ChartAreas[0].CursorY.LineWidth = 2;
            }
```

Now when the application is tested:

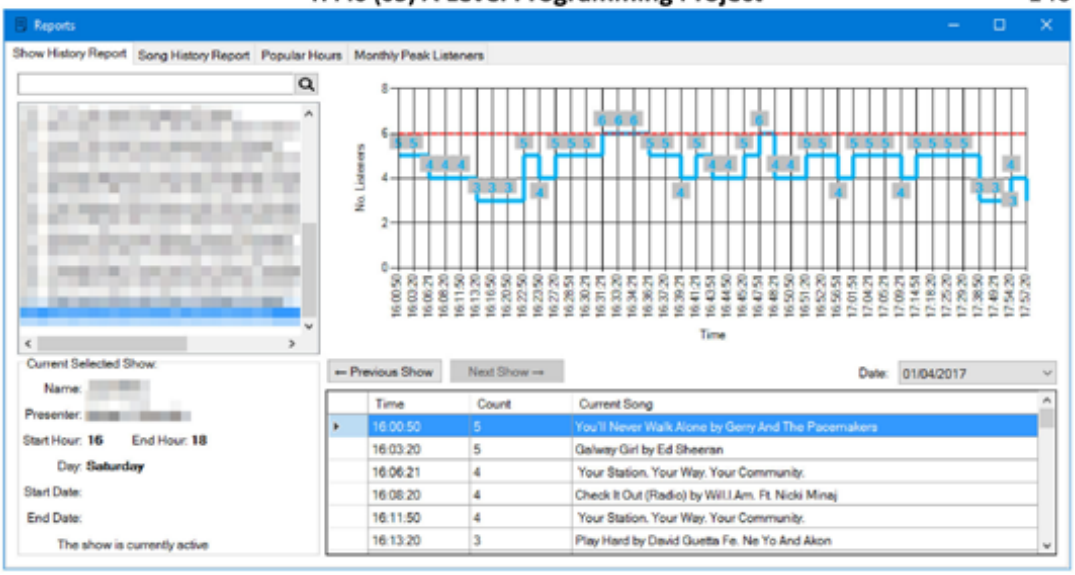
Selecting a show with data and zooming in



Then selecting a show with no data



Then selecting a show with data again



As you can see from the above screenshots, the graph is now updating properly, and is working great when zoomed in, then being automatically set back to the default zoom level when another show is selected.

As all of the tests have passed, and any issues have been corrected, I know the Show History Report section is working great, now I can move onto developing the Song History Report section.

The song history report will allow a user to search for a certain song, during a certain time period between two dates and times, and see when the song was played, and what show that was during.

The SQL query that will be ran is the following:

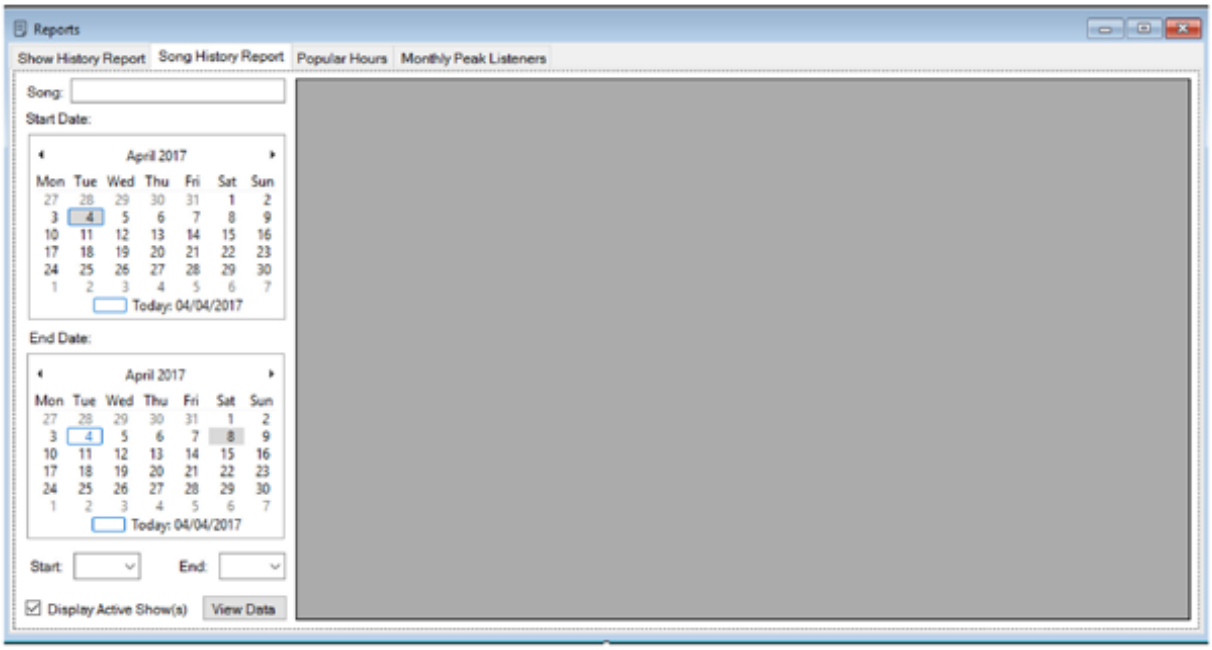
```
SELECT Log_DateTime AS 'Date/Time', Log_CurrentSong AS 'Song
Name',Log_ListenerCount AS 'Listeners', (SELECT CONCAT(Show_Name, ' with
',Show_Presenter) FROM shows WHERE Show_ID=Log_ShowID) AS 'Current Show',
Log_ShowID AS 'Show ID' FROM log WHERE Log_DateTime between "STARTDATE" and
"ENDDATE" AND Log_CurrentSong LIKE "%%" ORDER BY Log_DateTime ASC;
```

For example, if the query was data between the dates of 02/04/2017 and 04/04/2017 with the song query of 'Dan' the following data would be returned

Date/Time	Song Name	Listeners	Current Show	Show ID
2017-04-02 10:04:00	Red by Daniel Merriweather	5	The Sunday Show with Mike Parker	28
2017-04-02 13:53:00	Out On The Dancefloor by Dobie Gray	1	(NULL)	(NULL)
2017-04-02 15:10:31	Dancing With Lilies by Anna Shannan	1	Local Folk with John MacKenzie	21
2017-04-02 18:30:00	Strictly Come Dancing by Sacre	1	Mad About Musicals with Steve Little	24
2017-04-02 19:17:00	Jennifer Cody-Sutton Foster-Daniel Break...	2	Mad About Musicals with Steve Little	24
2017-04-02 20:03:30	You Make Me Feel Like Dancing by Leo S...	3	The Weekend Wind-down with Simon H...	25
2017-04-02 20:07:30	I'm In The Mood For Dancing by The Nol...	3	The Weekend Wind-down with Simon H...	25
2017-04-02 20:09:00	I'm In The Mood For Dancing by The Nol...	4	The Weekend Wind-down with Simon H...	25
2017-04-02 20:21:30	Dance With My Father by Luther Vandross	5	The Weekend Wind-down with Simon H...	25

Now I can add components to the form based of the design in the design section.

I can now define the properties for the various components on the form, Firstly both of the month calendars need to have their 'MaxSelectionCount' set to 1, so that only one date can be selected. The two hour selection combo boxes will need to have the numbers from 0 to 23 for their item selection, they will also have their 'AutoCompleteMode' to 'SuggestAppend', 'AutoCompleteSource' to 'ListItems' and their 'DropDownStyle' to 'DropDownList'. I will also set the start hour's default value to 00, and the end hour's default value to 23. This will prevent a user from entering a value that is outside of the range of the 23 hours. The 23rd hour will actually be at midnight, as the start hour will be 00:00:00 and the end hour will be 23:59:59, with the end ':00:00' and ':59:59' on the start and end. The display active shows checkbox will be set to checked.



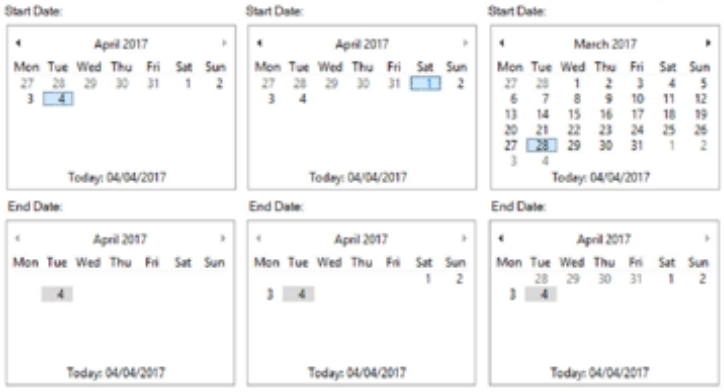
Next I will add the code that when the selected date is changed on the Start date that will be set as the minimum date on the end date. This will prevent a user from selecting a date range that wouldn't exist.

```
private void monthCalendarStartDate_DateChanged(object sender, DateRangeEventArgs e)
        {
            // Set the minimum date on the end hour to the current selected date
            monthCalendarEndDate.MinDate = monthCalendarStartDate.SelectionStart;
        }
```

I will also add the following code to the form's load event, to set the maximum value of both graphs to be today's date, this will prevent a user from selecting any data that doesn't exist.

```
// Setup the max and min date ranges on the Song History report
            monthCalendarStartDate.MaxDate = DateTime.Now;
            monthCalendarEndDate.MaxDate = DateTime.Now;
```

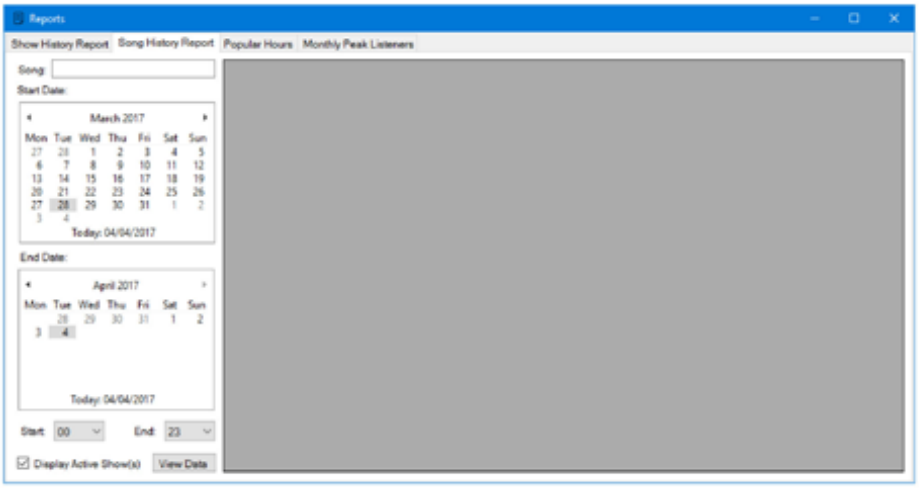
This can now be tested by selecting different date ranges on the calendars:



As you can see from the above screenshots the start date is always before the end date, and neither of them can be set any later than today's date. This has successfully passed the test, so I can now move onto making sure when the form opens, both of the hour combo boxes are set to their default values. I can do this by adding the following code to the bottom of the form's load method:

```
// Set the default values for the combo boxes for start and end hours
            cbStartHour.SelectedIndex = 0;
            cbEndHour.SelectedIndex = 23;
```

Now when the form open's you can see that the two hour selection combo boxes are set to the correct values, along with the month views displaying correctly.



Now I can move onto changing the properties of the data grid view to the following:

AllowUserToAddRows = False

AllowUserToDeleteRows = False

ReadOnly = True

SelectionMode = FullRowSelect

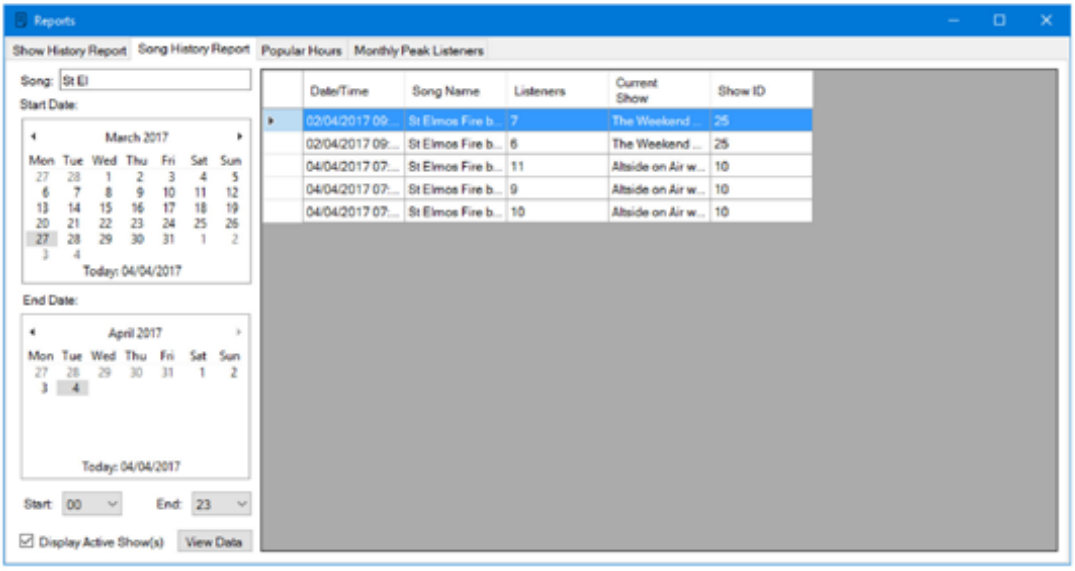
ShowEditingIcon = False

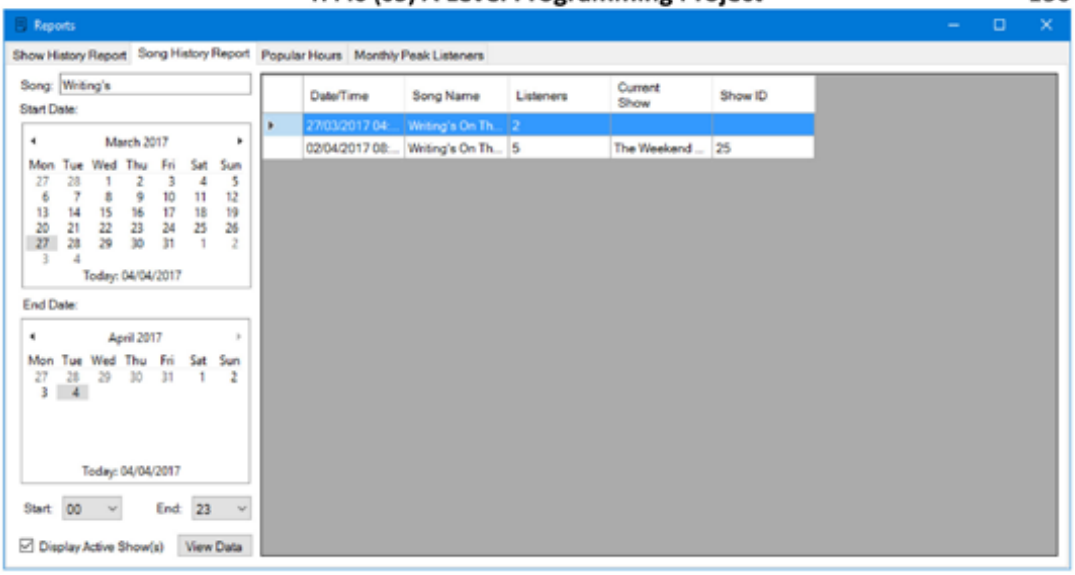
This will prevent data from being changed or manipulated by the user, as it is just for reference for them to read. I can now work on the method that will be ran when the View Data button is pressed, this will have the following code within the click method:

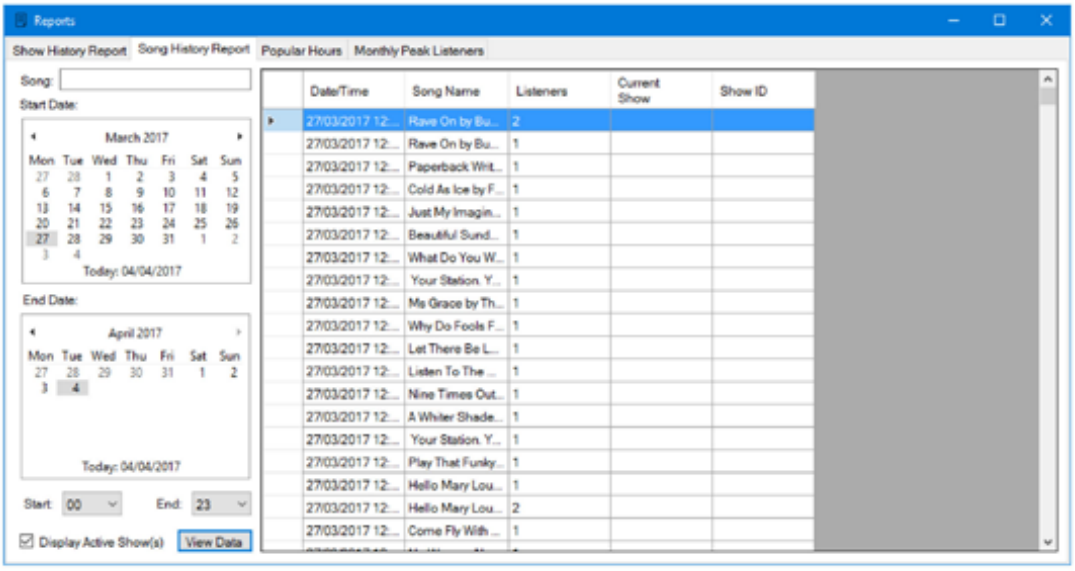
```
private void btnViewData_Click(object sender, EventArgs e)
        {
            // Create new query to obtain song history data
            MySqlCommand command = mySqlConn.CreateCommand();
```

```
            // Create command to query database
            command.CommandText = $"SELECT Log_DateTime AS 'Date/Time', Log_CurrentSong
AS 'Song Name',Log_ListenerCount AS 'Listeners', (SELECT CONCAT(Show_Name, ' with
',Show_Presenter) FROM shows WHERE Show_ID=Log_ShowID) AS 'Current Show', Log_ShowID AS
'Show ID' FROM log WHERE Log_DateTime between
\"{monthCalendarStartDate.SelectionStart.ToString("yyyy-MM-dd")}
{cbStartHour.Text.ToString()}:00:00\" and
\"{monthCalendarEndDate.SelectionStart.ToString("yyyy-MM-dd")}
{int.Parse(cbEndHour.Text.ToString()) - 1}:59:59\" AND Log_CurrentSong LIKE @query ORDER
BY Log_DateTime ASC;";
            command.Parameters.AddWithValue("@query",
$"%{txtSongQuery.Text.ToString()}%");

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


            // Create a data table to hold the data so it can be displayed in the data
grid view
            DataTable dt = new DataTable();
            dt.Load(reader);

            // Assign data source to table
            dataGridViewSongHistory.DataSource = dt;

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

        }
```

I can now test this by seeing what data it returns when searching for a certain song over the past few days:



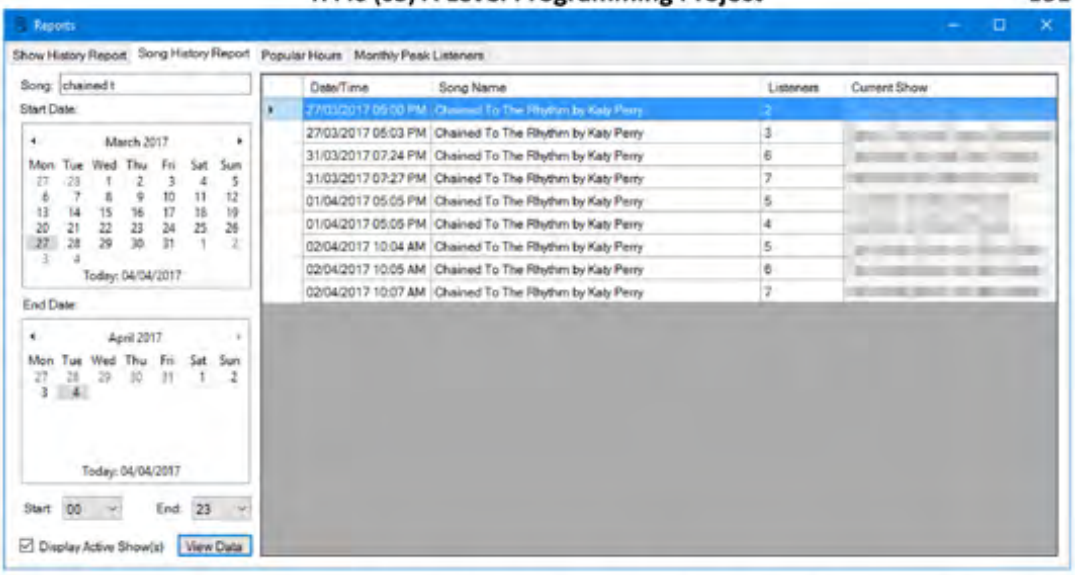




As you can see from the above screenshots the data is being returned correctly, and the system even accepts special characters due to the fact the search query was passed in as a parameter rather than using string interpolation or concatenation. If no search query is specified, then all data between the two dates is displayed. One issue with the view is that the Song Name column is very small, and the Show ID column is not needed. I can resolve this by adding the following code to the end of the View Data button's click method:

```
// Adjust data grid view columns
            dataGridViewSongHistory.Columns[0].AutoSizeMode =
DataGridViewAutoSizeColumnMode.AllCells;
            dataGridViewSongHistory.Columns[1].AutoSizeMode =
DataGridViewAutoSizeColumnMode.Fill;
            dataGridViewSongHistory.Columns[2].AutoSizeMode =
DataGridViewAutoSizeColumnMode.AllCells;
            dataGridViewSongHistory.Columns[3].AutoSizeMode =
DataGridViewAutoSizeColumnMode.AllCells;
            dataGridViewSongHistory.Columns[4].Visible = false;
```

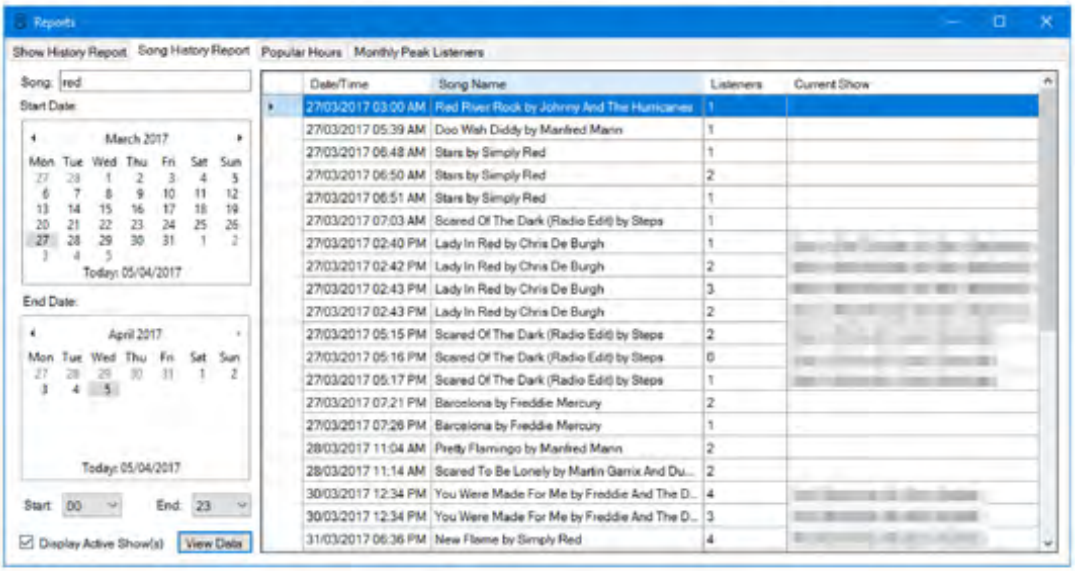
As you can see from the screenshot below, this has fixed the issue, and now the data is much easier to read. Any data that is cut off due to the column being too small can be easily viewed by hovering your mouse over the text, and a tooltip will appear displaying the full value

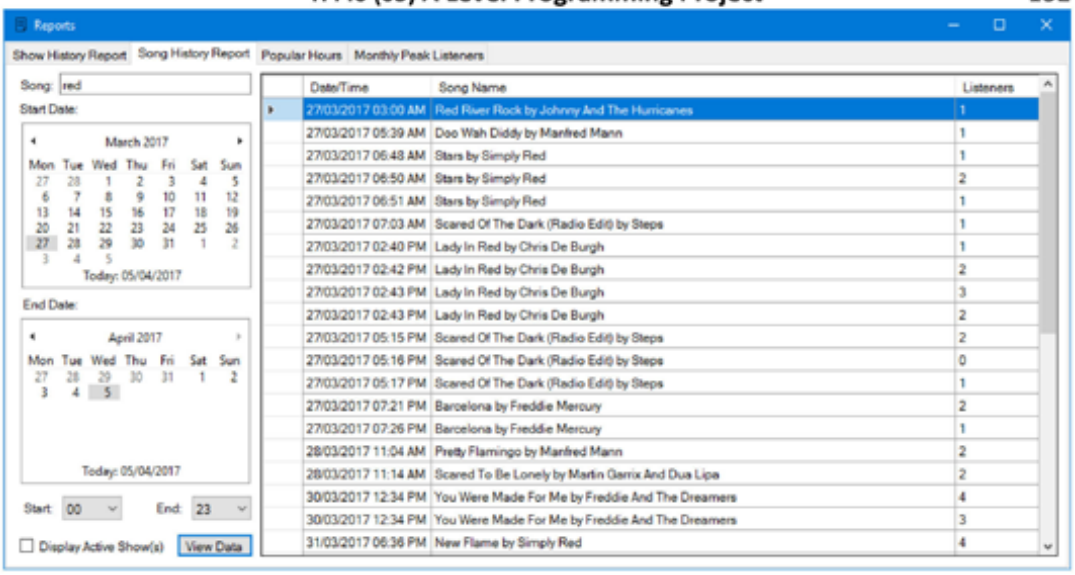


The only thing left to implement is the checkbox that will hide the current show, this will show or hide the column that contains the show name. I can do this by adding some extra code to the end of the view data button click method:

```
// Check to see if the current show needs to be displayed
            if (!checkBoxDisplayShows.Checked)
            {
                dataGridViewSongHistory.Columns[3].Visible = false;
            }
            else
            {
                dataGridViewSongHistory.Columns[3].Visible = true;
            }
```

Now this can be tested by viewing some data with the checkbox checked, and without it checked:





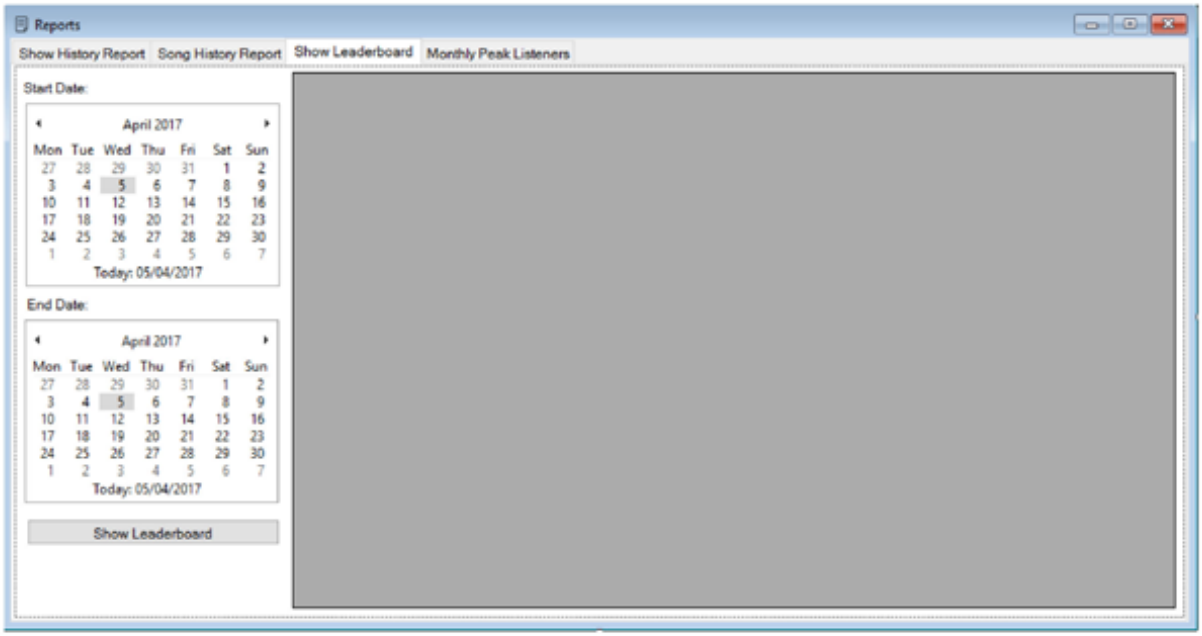
As you can see from the screenshots above, the checkbox is now working, and the current show is successfully hidden when the checkbox is not checked.

Now the Song History Report is complete, I can move onto the popular hour report, after looking into the design of the report, I realised that it was actually not required, as it was just a combination of data you could easily obtain from the show history report, and the song history report. The report itself would cause confusion, as it wouldn't be the easiest thing to read a massive selection of numbers, instead I have decided to make a show leader board, that will show the name of the program with the largest peak listener count over the past week. For this I will change Popular Hours to Show Leader Board. The components on this form are below:

```
SELECT MAX(Log_ListenerCount) AS 'Peak', (SELECT CONCAT(Show_Name, ' with
',Show_Presenter) FROM shows WHERE Show_ID=Log_ShowID) AS 'Show' FROM log
WHERE DATE(Log_DateTime) BETWEEN "STARTDATE" and "ENDDATE" AND Log_ShowID IS
NOT NULL GROUP BY Log_ShowID ORDER BY Peak DESC
```

For example, if I was to select the date ranges of 27/03/2017 02/04/2017, that would be last week's overview of top performing shows and would return the following data:

This data can then be added to a grid layout view so it can be easily read to see the most popular shows during the week. Below is the form layout with all of the components on it.



Both the month selection views have their maximum selection count set to 1 so only one date can be selected, and they have had their 'ShowTodayCircle' set to False, as this can look confusing to have another date highlighted in blue. The data grid view also has the following properties set:

'AllowUserToAddRows' = False

'AllowUserToDeleteRows' = False

'ReadOnly' = True

'SelectionMode' = FullRowSelect

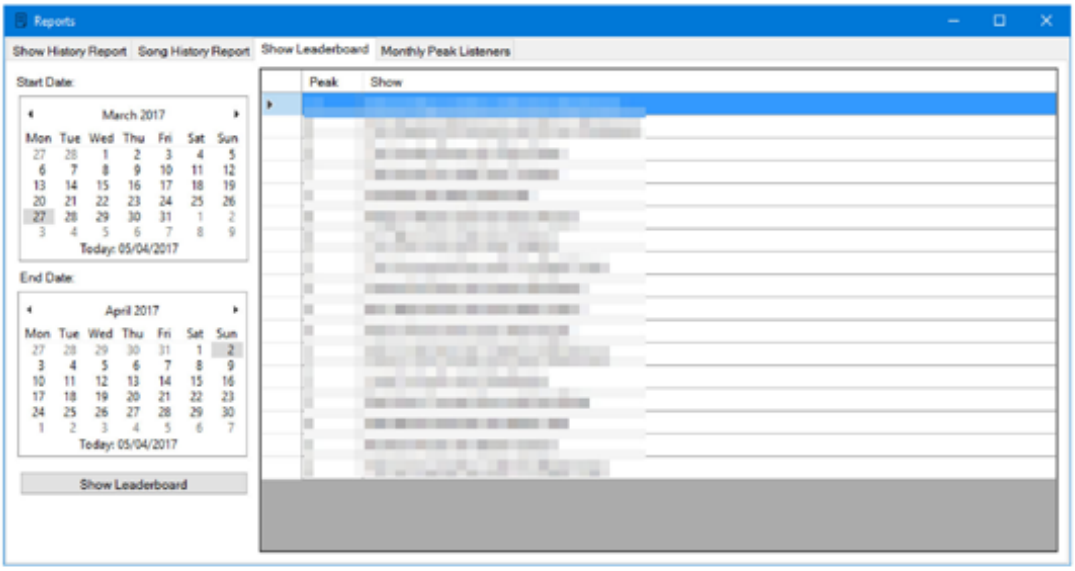
This will prevent the user from editing any of the data, adding new rows, or deleting any of them, it will also prevent them selecting single cells, and will make sure they only ever focous on one row, this will prevent the data from being changed, as it is just a reporting view only.

Now I can work on the code that will be ran when the Show Leaderbord button is pressed, this will run the query to obtain the data, assign it to a datatable, and then display it in the datagridview.

```
private void btnShowLeaderBoard_Click(object sender, EventArgs e)
        {
            // Create new query to obtain song history data
            MySqlCommand command = mySqlConn.CreateCommand();

            // Create command to query database
            // {monthCalendarEndDate.SelectionStart.ToString("yyyy-MM-dd")}
            //{monthCalendarStartDate.SelectionStart.ToString("yyyy-MM-dd")}
            command.CommandText = $"SELECT MAX(Log_ListenerCount) AS 'Peak', (SELECT
CONCAT(Show_Name, ' with ',Show_Presenter) FROM shows WHERE Show_ID=Log_ShowID) AS 'Show'
FROM log WHERE DATE(Log_DateTime) BETWEEN
\"{monthCalendarShowLeaderboardStart.SelectionStart.ToString("yyyy-MM-dd")}\" and
\"{monthCalendarShowLeaderboardEnd.SelectionStart.ToString("yyyy-MM-dd")}\" AND
Log_ShowID IS NOT NULL GROUP BY Log_ShowID ORDER BY Peak DESC";

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


            // Create a data table to hold the data so it can be displayed in the data
grid view
            DataTable dt = new DataTable();
```

```
            dt.Load(reader);

            // Assign data source to table
            dataGridViewShowLeaderboard.DataSource = dt;

            // Adjust data grid view columns so the show name fills all of the space,
after the listener count has been set as small as possible
            dataGridViewShowLeaderboard.Columns[0].AutoSizeMode =
DataGridViewAutoSizeColumnMode.AllCells;
            dataGridViewShowLeaderboard.Columns[1].AutoSizeMode =
DataGridViewAutoSizeColumnMode.Fill;

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

Now I can test the code to see if it returns the same data as shown in the database above:



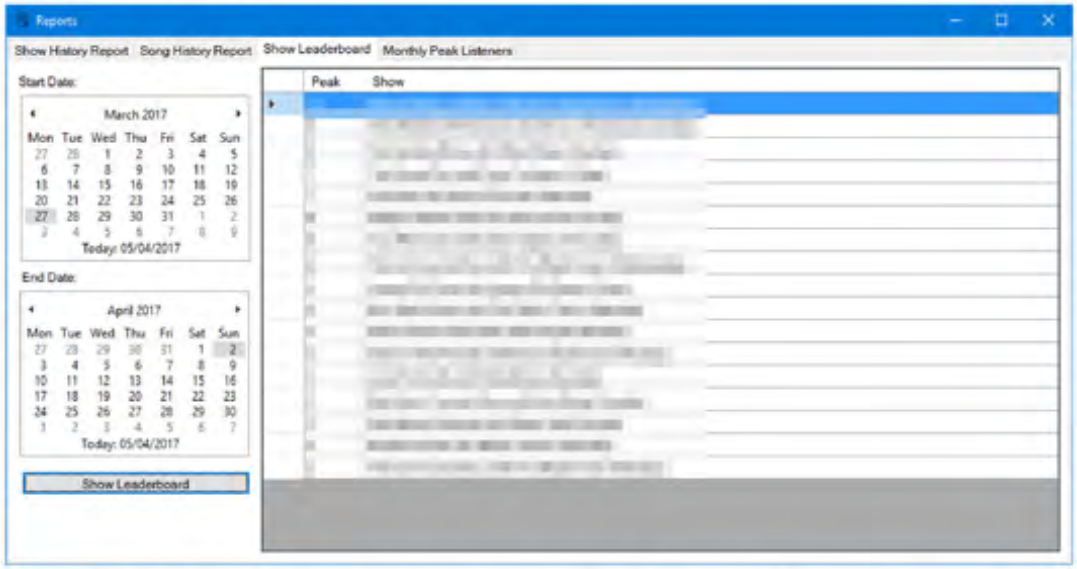
As you can see, this data matches up with the data shown before, so this report is working successfully.

The only issue is that for shows with the same name (such as the community hour shown above) you don't know which day it is referring to. For this I can adjust the SQL query to the following:

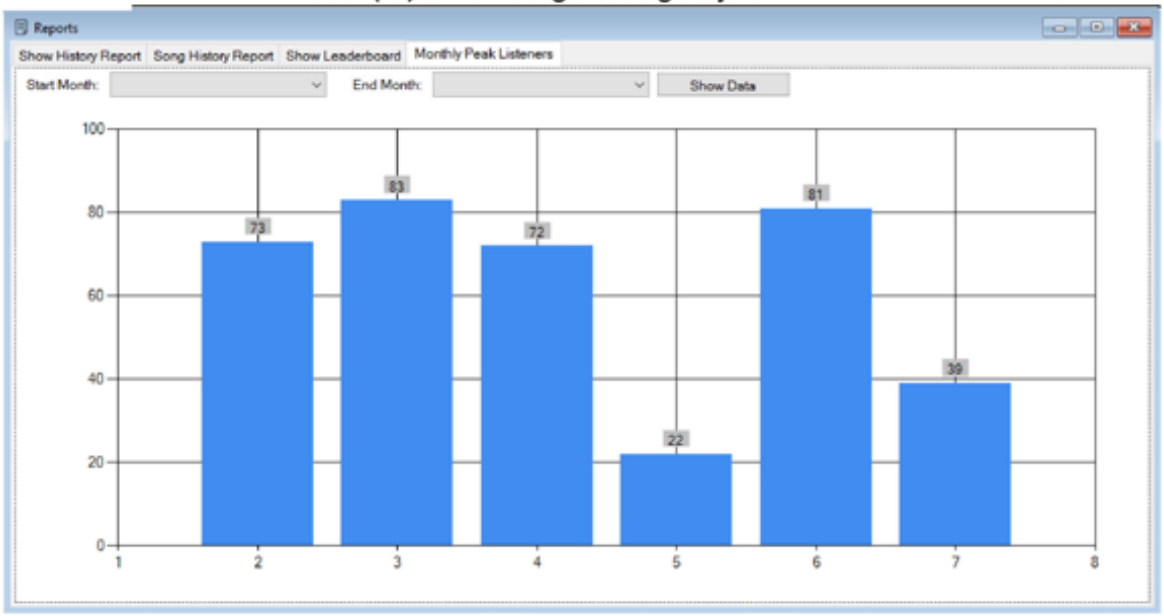
```
SELECT MAX(Log_ListenerCount) AS 'Peak', (SELECT CONCAT(Show_Name, ' with
',Show_Presenter,' (',Show_Day,')') FROM shows WHERE Show_ID=Log_ShowID) AS
'Show' FROM log WHERE DATE(Log_DateTime) BETWEEN "STARTDATE" and "ENDDATE"
AND Log_ShowID IS NOT NULL GROUP BY Log_ShowID ORDER BY Peak DESC
```

This would then return the following data between the two dates above:

This will look a lot better, as you can now tell the day each show was on. I have also tested this within the application as you can see below:



This works a lot better, now I can move onto the Monthly Peak Listeners Report. This report will give an overview of the monthly peak figure, for each month between the two ranges specified. This will then be plotted onto a line graph. The form will have the following controls:



Both of the combo boxes have the following properties set:

'AutoCompleteMode' = 'SuggestAppend'

'AutoCompleteSource' = 'ListItems'

'DropDownStyle' = 'DropDownList'

This will prevent the user from typing in any values into the combo boxes, and means they can only select values that are within the dropdown box.

The chart also has the legend and title turned off, as they are not needed, as I will be putting axis labels on each side of the chart. Also, on the default series, the option of show axis labels has been set to 'True' the rest of the chart settings can be configured in a setup method as shown below:

```
private void setupMonthlyPeakGraph()
        {
            // Setup X Axis
            chartMonthlyPeak.ChartAreas[0].Axes[0].Title = "Month";
            chartMonthlyPeak.ChartAreas[0].Axes[0].IsLabelAutoFit = true;
            chartMonthlyPeak.ChartAreas[0].Axes[0].Interval = 1;
            chartMonthlyPeak.ChartAreas[0].Axes[0].IsMarginVisible = false;

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

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

            // Setup the series to hold the monthly listener peak count
            chartMonthlyPeak.Series[0].ChartType = SeriesChartType.Line;
            chartMonthlyPeak.Series[0].LegendText = "Listener Count";
            chartMonthlyPeak.Series[0].BorderWidth = 4;
            chartMonthlyPeak.Series[0].Color = Settings.colours.Blue;
            chartMonthlyPeak.Series[0].LabelForeColor = Settings.colours.Blue;

        }
```

I will also add this to the end of the form's load method. So that it sets up the graph when the form loads. Now I need to create a SQL query that will list all of the available months in the database that the user can select from. This is shown below:

```
SELECT CONCAT(MONTHNAME(Log_DateTime), ' ', YEAR(Log_DateTime)) AS 'Months'
FROM log GROUP BY MONTH(Log_DateTime) ORDER BY Log_DateTime ASC
```

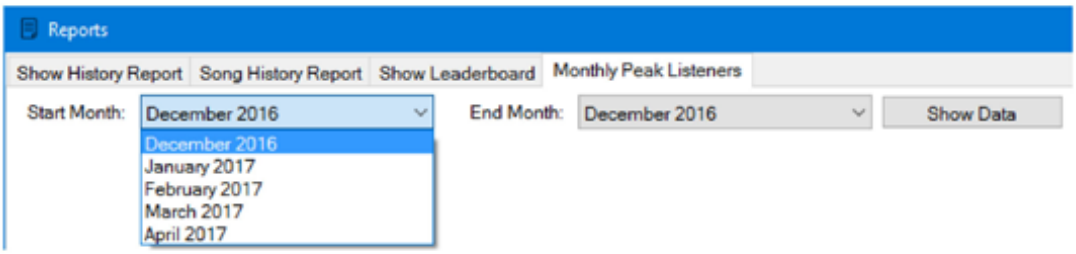
This will return the following data

Months
December 2016
January 2017
February 2017
March 2017
April 2017

This can then be used in the code to add this to the list of selectable items in the combo box, I will create a method called updatePeakMonthRanges:

```
private void updatePeakMonthRanges()
        {
            // Update the peak ranges
            // Clear the values currently in the combo boxes
            comboBoxStartMonth.Items.Clear();
            comboBoxEndMonth.Items.Clear();

            // Create new query to obtain the ranges of dates available
            MySqlCommand command = mySqlConn.CreateCommand();
            command.CommandText = $"SELECT CONCAT(MONTHNAME(Log_DateTime), ' ',
YEAR(Log_DateTime)) AS 'Months' FROM log GROUP BY MONTH(Log_DateTime) ORDER BY
Log_DateTime ASC";

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

            // Loop through each result and add them to the combo boxes
            while (reader.Read())
            {
                comboBoxStartMonth.Items.Add(reader[0].ToString());
                comboBoxEndMonth.Items.Add(reader[0].ToString());
            }

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

        }
```

This has also been added to the form's load event, to make sure there is a selectable set of dates to use when the form opens. I can now test this by loading the form, and making sure that the combo boxes contain the same sets of dates as shown above:



This now works great, by default I will setup the code so that the first item is automatically selected for the start month, and the last item selected for the end month. This can be done by adding the following code to the end of the updatePeakMonthRanges method:

```
// Select the first item in the list for the start month, and the last one for the end
month
comboBoxStartMonth.SelectedIndex = 0;
comboBoxEndMonth.SelectedIndex = comboBoxEndMonth.Items.Count - 1;
```

Now I can work on creating the SQL query that will return the data used to plot on the graph, this will have to take each month between the two dates specified, and return the peak listener count for each month. This is shown below:

```
SELECT CONCAT(MONTHNAME(Log_DateTime), ' ', YEAR(Log_DateTime)) AS 'Months',
MAX(Log_ListenerCount) AS 'Peak' FROM log WHERE Log_DateTime BETWEEN
"STARTDATE" and "ENDDATE" GROUP BY MONTH(Log_DateTime) ORDER BY Log_DateTime
ASC
```

For example, if I obtain the data from this query between the dates of "2016-01-01" and "2017-05-01" it returns the following data:

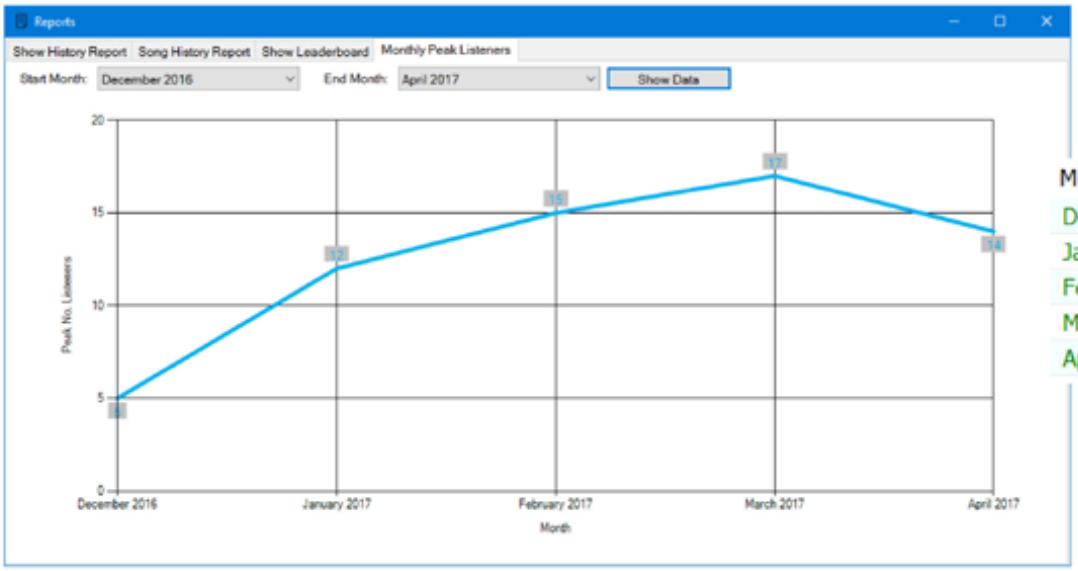
Months	Peak
December 2016	5
January 2017	12
February 2017	15
March 2017	17
April 2017	14

This data can now be used to plot a graph. I will do this by adding the following code to the click method called when the Show Data button is pressed:

```
private void btnMonthlyPeakListenersShow_Click(object sender, EventArgs e)
        {
            // Create variables to hold the start and end dates
            string startdate = "";
            string enddate = "";

            // Get start and end dates
            startdate = comboBoxStartMonth.SelectedItem.ToString().Split(' ')[1] + "-" +
(DateTimeFormatInfo.CurrentInfo.MonthNames.ToList().IndexOf(comboBoxStartMonth.SelectedIt
em.ToString().Split(' ')[0]) + 1).ToString() + "-1";
            enddate = comboBoxEndMonth.SelectedItem.ToString().Split(' ')[1] + "-" +
(DateTimeFormatInfo.CurrentInfo.MonthNames.ToList().IndexOf(comboBoxEndMonth.SelectedItem
.ToString().Split(' ')[0]) + 2).ToString() + "-1";

            // Create query
            MySqlCommand command = mySqlConn.CreateCommand();
            command.CommandText = $"SELECT CONCAT(MONTHNAME(Log_DateTime), ' ',
YEAR(Log_DateTime)) AS 'Months', MAX(Log_ListenerCount) AS 'Peak' FROM log WHERE
Log_DateTime BETWEEN \"{startdate}\" and \"{enddate}\" GROUP BY MONTH(Log_DateTime) ORDER
BY Log_DateTime ASC";

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

            // Create dictionary to hold data
            Dictionary<string, int> dictionaryMonthPeak = new Dictionary<string, int>();

            // Loop through each of the results returned
            while (reader.Read())
            {
                dictionaryMonthPeak.Add(reader["Months"].ToString(),
int.Parse(reader["Peak"].ToString()));
            }

            // Clear data currently on graph
            chartMonthlyPeak.Series[0].Points.Clear();
```

```
            // Cycle through each item in the dictionary and add it to the graph
            foreach (KeyValuePair<string, int> point in dictionaryMonthPeak)
            {
                chartMonthlyPeak.Series[0].Points.AddXY(point.Key, point.Value);
            }

            // Close reader
            reader.Close();

        }
```

I can now test this code by running a report for the dates I used above, and checking to make sure that the graph matches up with the data returned from the database:



Months	Peak
December 2016	5
January 2017	12
February 2017	15
March 2017	17
April 2017	14

As you can see from above, this matches with the data from the database, so the final report in the reporting section is working great. Now reporting is done I can move onto finishing off the rest of the dashboard features.

User's are restricted based on an access level, the higher the level the more access they have. This is shown in the diagram towards the end of the design section, showing what forms certain users will have access to. I can now add this functionality into the system by checking when the dashboard form loads to see what buttons need to be hidden to prevent them from being used. I will do this by creating a new method called updateAccessRestrictions, I will also add this to the end of the dashboard form's load event, so it is ran when the form is opened.

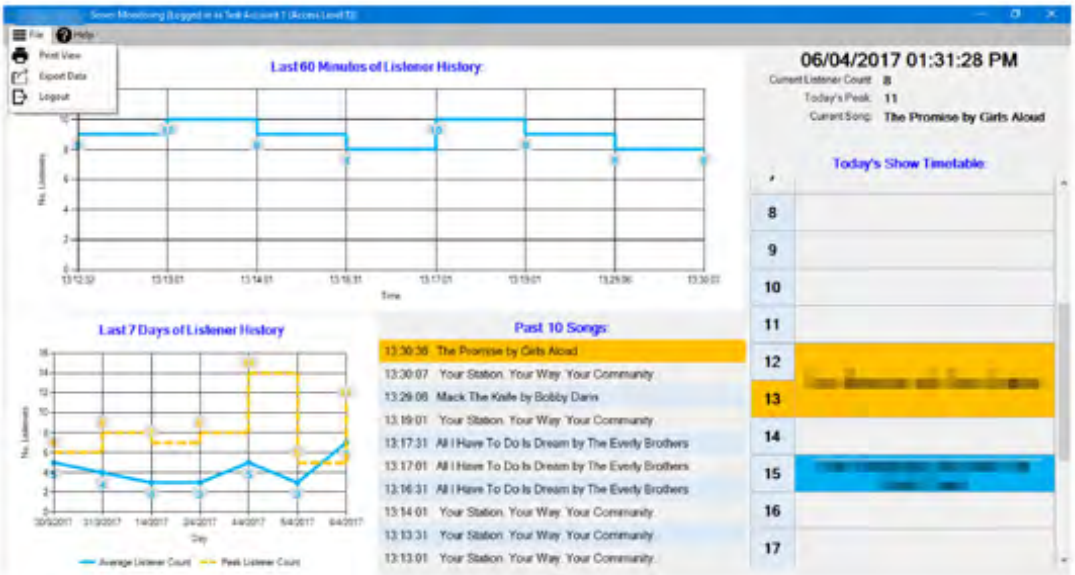
```
private void updateAccessRestrictions()
        {
            // Limit certain functionalities of the system
            if (Settings.accessLevel == 3)
            {
                exitToolStripMenuItem.Visible = true;
                reportsToolStripMenuItem.Visible = true;
                userManagementToolStripMenuItem.Visible = true;
                settingsToolStripMenuItem.Visible = true;
            }
            else if(Settings.accessLevel == 2)
            {
                exitToolStripMenuItem.Visible = false;
                reportsToolStripMenuItem.Visible = true;
                userManagementToolStripMenuItem.Visible = false;
                settingsToolStripMenuItem.Visible = false;
            }
            else
            {
                exitToolStripMenuItem.Visible = false;
```

```
            reportsToolStripMenuItem.Visible = false;
            userManagementToolStripMenuItem.Visible = false;
            settingsToolStripMenuItem.Visible = false;
        }

    }
```

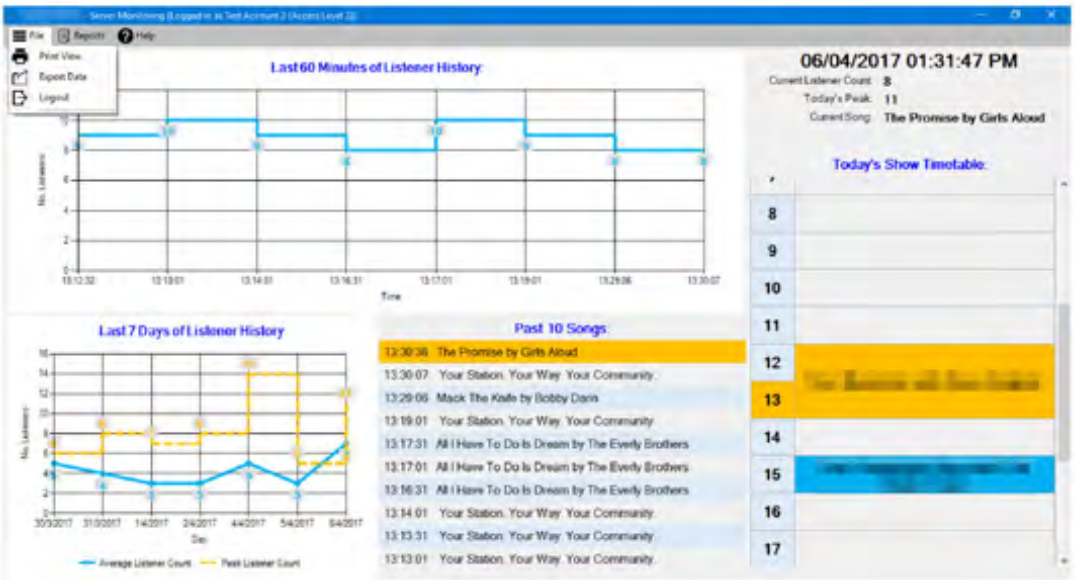
I can test this by creating 3 test accounts, one with each of the 3 permission levels, then test to see if the features are limited correctly.

	User_FullName	Use...	User_Email	User_Password		User_Las
21		admin	admin@gmail.com		3	2017-04-
22	Test Account 1 (Access Level 1)	test1	test1@gmail.com		1	(NULL)
23	Test Account 2 (Access Level 2)	test2	test2@gmail.com		2	(NULL)
24	Test Account 3 (Access Level 3)	test3	test3@gmail.com		3	(NULL)

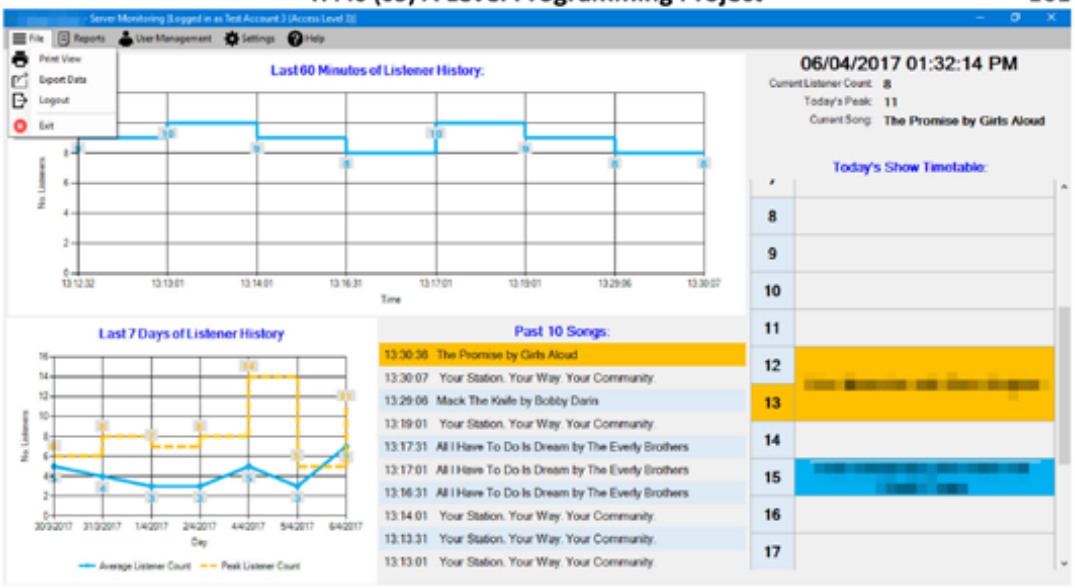
Test Account 1:



Test Account 2:



Test Account 3:



The access restrictions are working great, and all 3 users were given the correct access to certain modules of the system, with the admin user (Test 3) only being given access to quit the application altogether. Now I can move onto finishing off the last few features of the dashboard, this is the print view, export data, logout, and exit buttons. The print view button will be the same as the export data button, but will automatically open it in print preview mode. This report will contain both of the graphs on the screen and the past 10 songs, along with the current listener counts, song and the current time.

This will be done by creating a HTML document with this information in it, then run this as a process. The HTML layout will be the following:

```
<html>
<head>
<script>window.print();</script>
<title>Server Listener Statistics Report</title>
</head>
<body style='font-family: Arial;'>
<center>
<h1>Server Listener Statistics Report</h1>
<hr/>
<h2>06/04/2017 01:32:14 PM</h2>
<p>Current Listener Count: <b>8</b></p>
<p>Today's Peak: <b>11</b></p>
<p>Current Song: <b>The Promise by Girls Aloud</b></p>
<hr/>
<img src="./past10songs.png"></img><br/><br/>
<img src="./last7days.png"></img><br/><br/>
<img src="./last60mins.png"></img><br/><br/>
<hr>
</center>
</body>
</html>
```

The script tag containing "window.print();" will open the print dialogue as soon as the webpage is loaded in a users' web browser. The rest of the layout will take images of graphs on the dashboard form.

I can now create the code that will be ran when the user clicks on the Print View button, I will create a method called 'generateReport' that will take in a Boolean of whether to include the automatic opening of the print preview dialogue, this code is as follows:

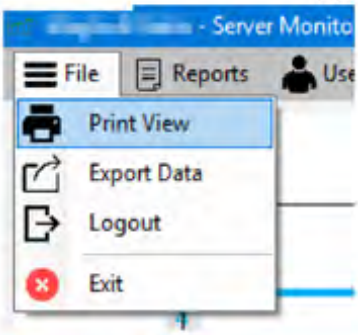
```
private void generateReport(bool printDocument)
        {
            // Create images of both charts
            weekGraph.SaveImage(@"last7days.png", ImageFormat.Png);
            hourGraph.SaveImage(@"last60mins.png", ImageFormat.Png);

            // Create a new bitmap image, and add the contents of the tableLayoutPanel
containing the past 10 songs to it
            using(Bitmap printImage = new Bitmap(tableLayoutPanelPast10Songs.Width,
tableLayoutPanelPast10Songs.Height))
            {
                tableLayoutPanelPast10Songs.DrawToBitmap(printImage, new Rectangle(0, 0,
printImage.Width, printImage.Height));
                // Save the image to a file
                printImage.Save(@"past10songs.png");
            }

            // Create a string to hold the JavaScript if the user wants a print preview
dialogue
            string printJavascript = "";

            // Check to see if the user wants to have an automatic print dialogue
            if (printDocument)
            {
                printJavascript = "<script>window.print();</script>";
            }

            // Create html file and save it to 'report.html'
            StreamWriter sw = new StreamWriter(@"report.html");
            sw.WriteLine($"<html> <head> {printJavascript} <title>Server Listener
Statistics Report</title> </head> <body style='font-family: Arial;'> <center> <h1>Server
Listener Statistics Report</h1> <hr/> <h2>{lblCurrentTime.Text}</h2> <p>Current Listener
Count: <b>{lblCurrentListenerCount.Text}</b></p> <p>Today's Peak:
<b>{lblPeakListenerCount.Text}</b></p> <p>Current Song: <b>{lblCurrentSong.Text}</b></p>
<hr/> <img src=\"./past10songs.png\"></img><br/><br/> <img
src=\"./last7days.png\"></img><br/><br/> <img src=\"./last60mins.png\"></img><br/><br/>
<hr> </center> </body> </html>");
            sw.Close();

            // Open the report in the user's default web-browser
            string path = System.Environment.CurrentDirectory + "/report.html";
            Process.Start(path);

        }
```

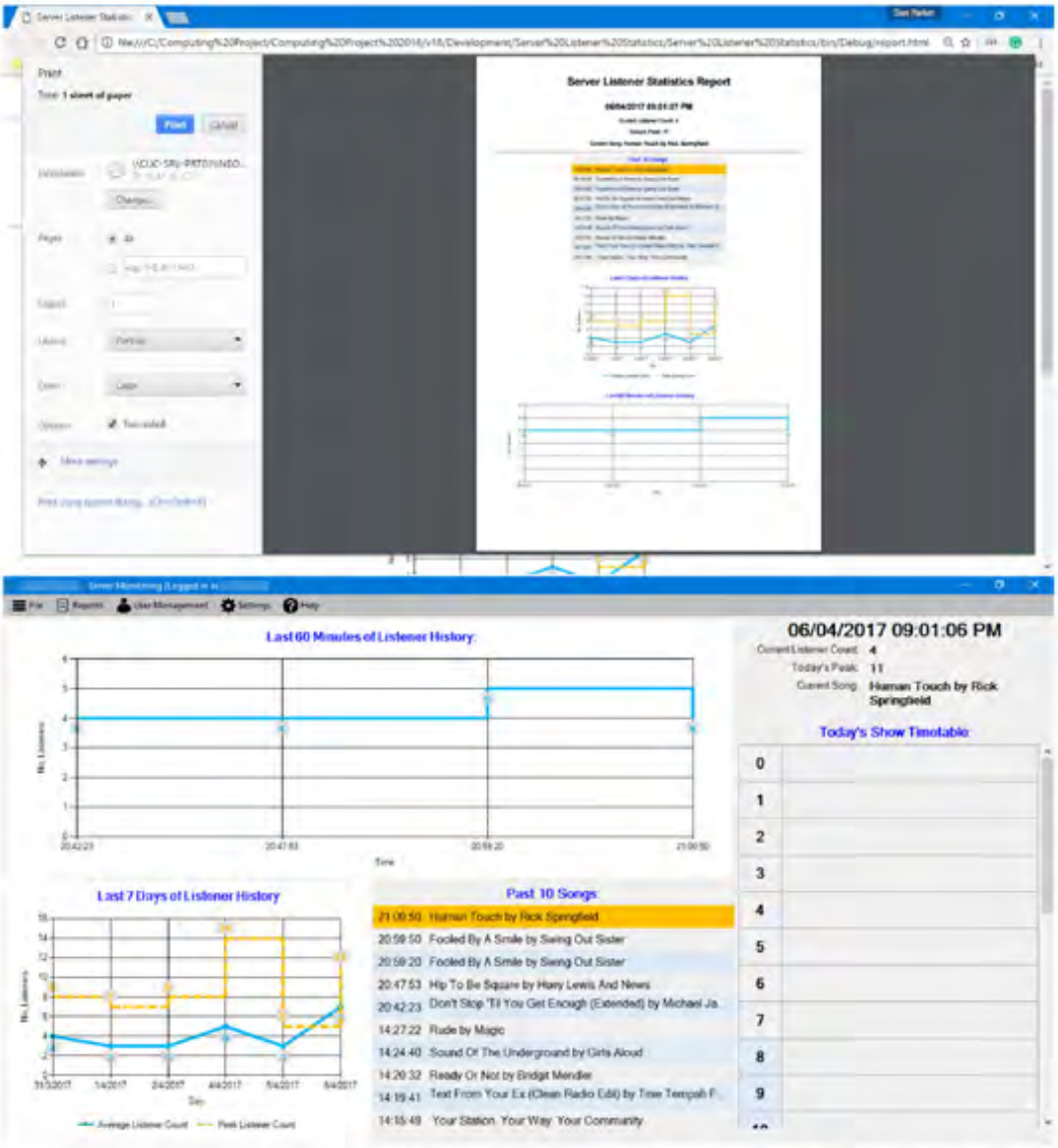
This will create all of the required images for the report, then it will generate the HTML file, then open it in the user's default web browser.

I will also need to add a call to this method to the method that is ran when the user clicks on the Print View menu option:

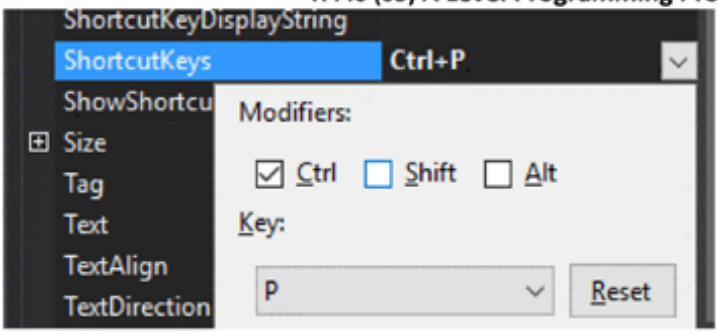
```
        private void printViewToolStripMenuItem_Click(object sender, EventArgs e)
        {
            // Generate a report and show the print preview dialogue
            generateReport(true);
        }
```

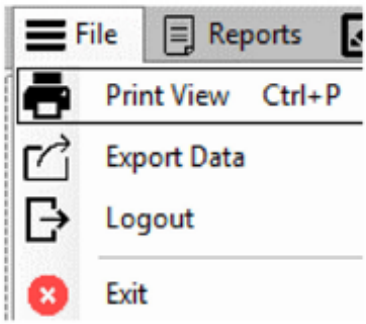
This code can now be tested to see if it works by pressing the button on the menu bar:





As you can see from the screenshots above, the report has generated successfully, and opened in the default web browser, it has also opened the print preview dialogue as well. It also matches the data on the dashboard so it has passed testing and this function works. One extra feature would be to add a keyboard shortcut of Ctrl + P to perform this action quickly, this can be easily setup by changing the following setting of the PrintViewToolStripMenuItem:





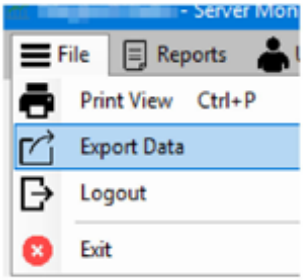
Now when the menu is clicked on, you can see the keyboard shortcut that you can press as it is shown automatically next to the name of the menu item:

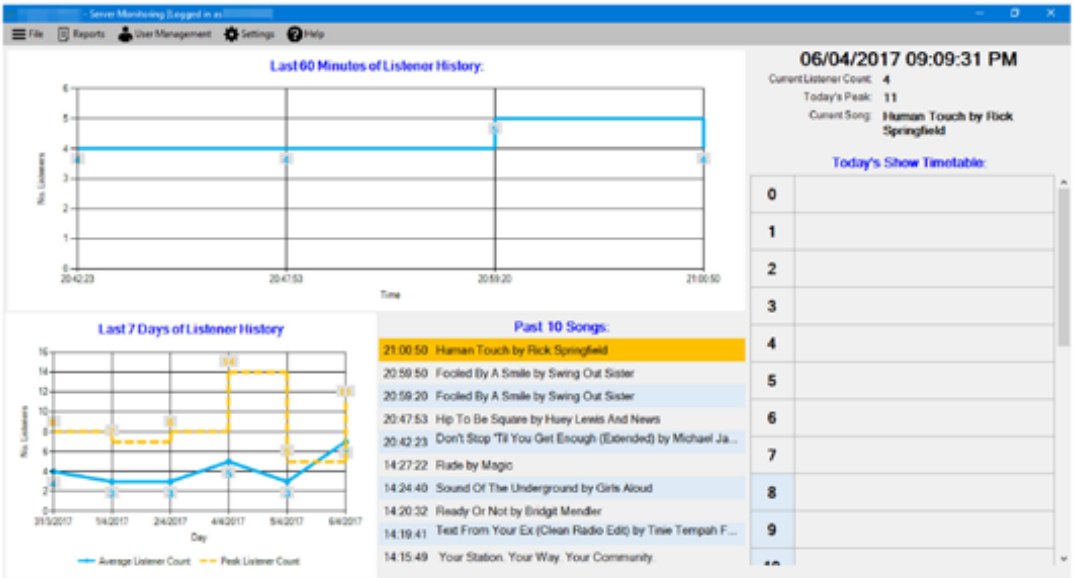
I have also tested this by pressing Ctrl + P when on the dashboard form, and it performs exactly the same as pressing the Print View button.

Now I can add the code for the Export Data button, this will simply open the same report in a web browser, but will not open the print view dialogue. This is because more accurate data can be exported from the reporting form by simply using copy and paste, and transferring the selected information into another document manually.

```
private void exportDataToolStripMenuItem_Click(object sender, EventArgs e)
{
    // Generate a report, but do not show print preview dialogue
    generateReport(false);
}
```

This can now be tested by clicking the export data button in the toolbar:





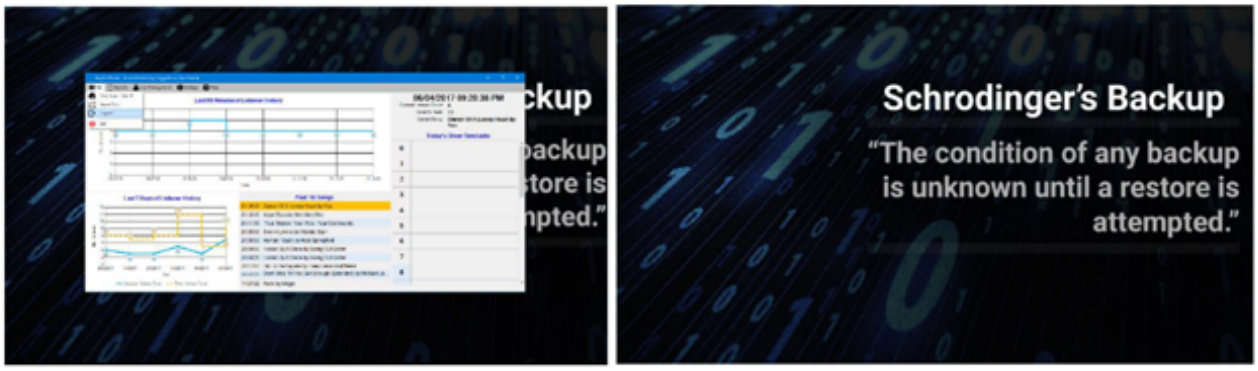
As you can see, this opens the report in the user's default web browser but does not display the print preview dialogue. This is now working great, and will allow for a user to quickly save the current information on the dashboard, or print it off. Now I can move onto the final two menu options. Logging out and exiting the application.

The logging out button will close the current form, and the exit button will close the current application (this will only be available to users who have an access level of 3 as it will completely close the application and will not record any more statistics until it is opened again).

The logout menu item will have the following code:

```
private void logoutToolStripMenuItem_Click(object sender, EventArgs e)
{
    // Close the current form
    this.Close();
}
```

When the menu item is clicked the form closes as you can see below:

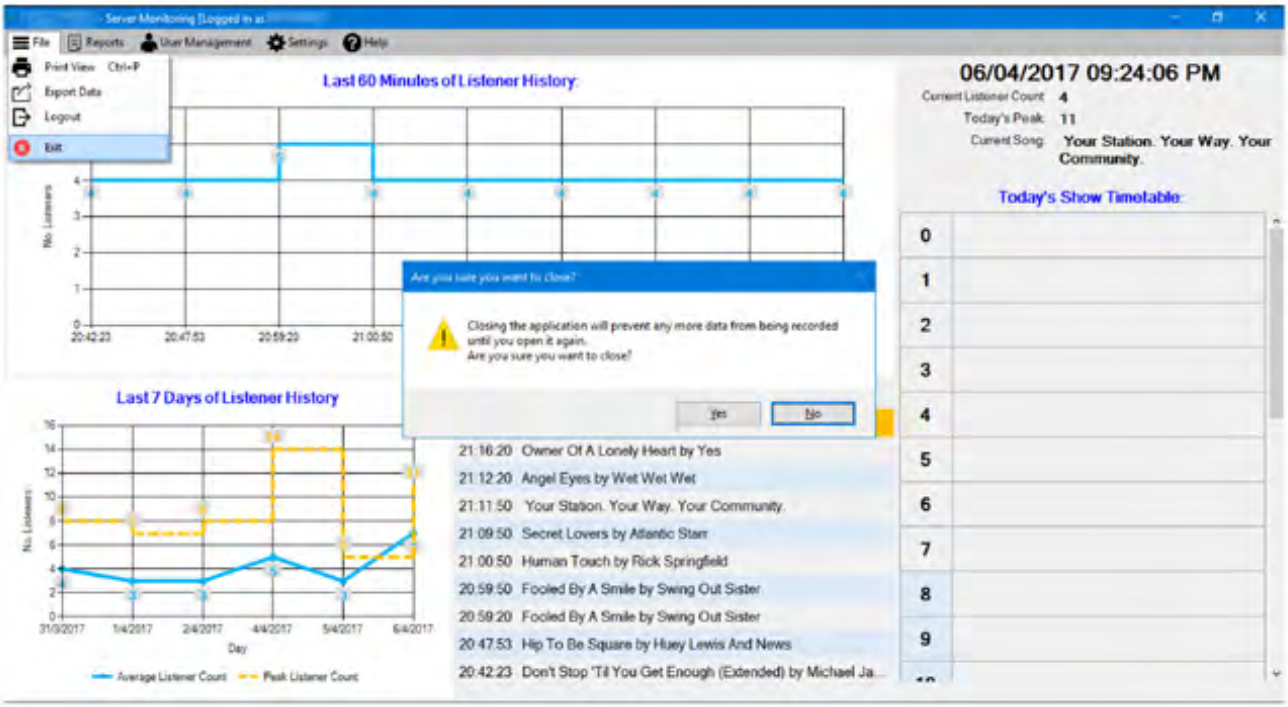


This is now working successfully. And will work for all users, as the menu item is always visible regardless of access level. Now I can move onto the code to quit the application. This will need to display a message box to confirm with the user that they want to quit the application, and it will be defaulted to the option of "No".

The code for this is as follows:

```
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            // Check to see if user really wants to close the application
            DialogResult result = MessageBox.Show("Closing the application will prevent
any more data from being recorded until you open it again.\nAre you sure you want to
close? ", "Are you sure you want to close?", MessageBoxButtons.YesNo,
MessageBoxIcon.Warning, MessageBoxDefaultButton.Button2);

            // Close the application if yes was pressed
            if (result == DialogResult.Yes)
            {
                Application.Exit();
            }
        }
```

This can now be tested by logging in as an administrator and checking to see if both options on the message box work:

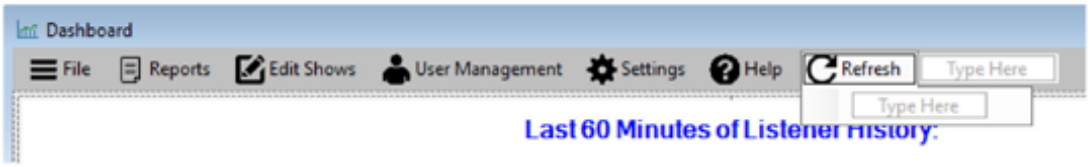


If No is pressed:

If Yes is pressed:

The application notification icon has disappeared, and the application has completely closed.

As you can see from the above screenshots, the test has worked and the application will only close if yes is pressed. One extra feature that would be nice is a Refresh button that the user can click on, so that they can manually refresh the dashboard to update it, save having to wait every 60 seconds. I can create another menu item, and I will give it a refresh logo for its icon image:

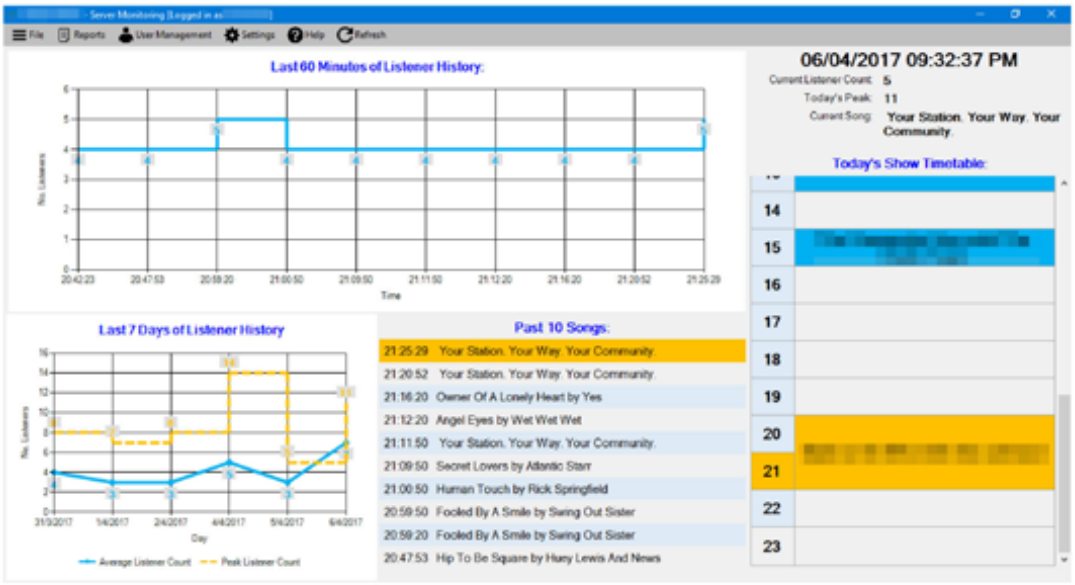


This button will have the same code that the globalUpdateLoop has to update the form:

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

This can now be tested to make sure it works:

Before pressing button:



After pressing button:

As you can see, the button updated all of the data on the form to the most recent data, which is a really useful extra feature for the toolbar to contain.

The help button will open a user guide with some troubleshooting tips, this will not be imperative for the operation of the system though, as I will be giving training to first time users on how to use it, as the target audience will only be 4 managers within the radio station.

Another consideration would be that if the application was to be started up with two or more instances running on separate machines they may conflict. I was going to build a feature into each install to either have it as a reporting and recording system, or just a reporting system. However after realising the setup of how the update function works, having multiple instances would not cause conflict as they both check for the most recent record in the database, so they will both be checking the same information. While this may add extra load to the database, it is negligible and both instances would be querying the database every 30 seconds regardless whether they were saving data or not.

One final issue left to develop is what happens if somebody opens the application for the first time, this can be checked as a config file will not exist, this can be handled by the handle form's load event.

This code is as follows:

```
private void Handle_Load(object sender, EventArgs e)
{
    // Check to see if settings file exists
    if (File.Exists(@"config.conf"))
    {
        // Obtain the settings from the config file
        Settings.obtainSettings();
    }
    else
    {
        // Hide the notification icon
        notifyIcon.Visible = false;

        // Disable the update timer
        timerObtainStats.Enabled = false;

        // Show a new settings
        EditSettings editSettings = new EditSettings();
        editSettings.ShowDialog();

    }

}
```

I will also add the following code to the edit settings form, so that it will not automatically load the settings into the class if they do not exist:

```
private void EditSettings_Load(object sender, EventArgs e)
        {
            // Check to see if it is an initial start-up
            if (File.Exists(@"config.conf"))
            {
                // Update settings from the settings class
                updateSettingsFromClass();
            }
            else
            {
                // Display a message to let the use know it is a first time setup
                MessageBox.Show("As this is a first time setup. Please enter the settings
to setup the application.");
            }

        }
```

I will also adjust the closing event of the form, so if nothing was entered it warns the user that the same settings dialogue will appear when they open the application again. I will also set it to restart the application once settings have been saved.

I will change the following code:

```
// Display confirmation message
MessageBox.Show("Settings have been saved successfully!");
```

To:

```
// Display confirmation message
MessageBox.Show("Settings have been saved successfully! The application will now
restart.");
Application.Restart();
```

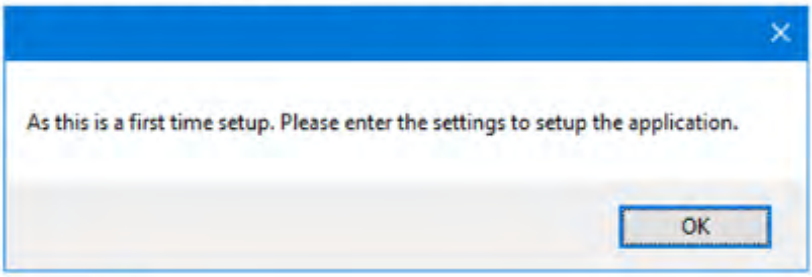
Now I need to adjust the closing event of the settings form, to quit the application if it is a first time setup, and no settings were saved:

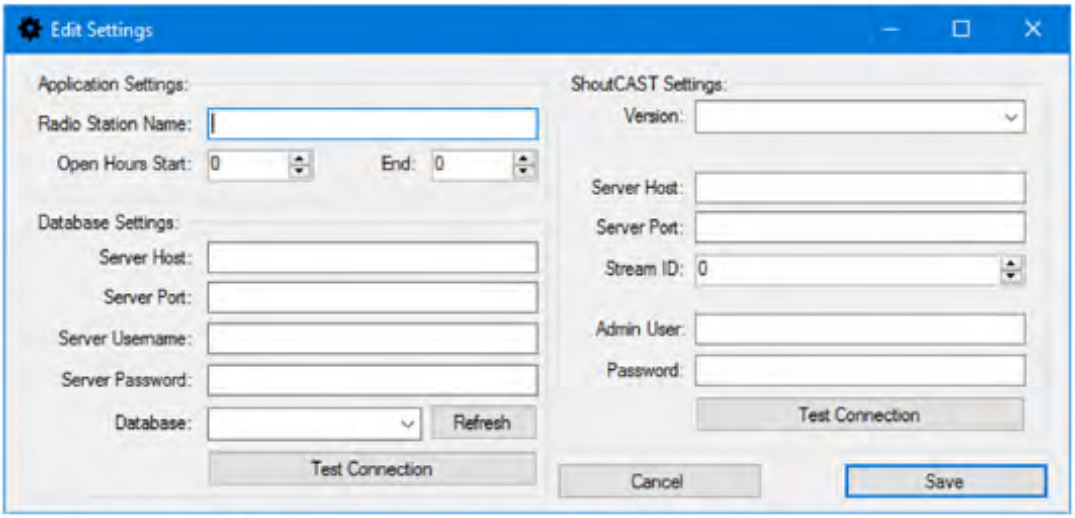
```
private void EditSettings_FormClosing(object sender, FormClosingEventArgs e)
        {
            // Check to see if it's a first time setup
            if (File.Exists("config.conf") == false)
            {
                // First time setup so warn user of closing
                DialogResult result = MessageBox.Show("As you have not saved valid
settings the next time you open the application this initial settings dialogue will open
again. Are you sure you wish to close?", "Closing without valid settings?",
MessageBoxButtons.YesNo, MessageBoxIcon.Warning, MessageBoxDefaultButton.Button2);

                // If the user didn't want to close, then cancel the closing event
                if (result == DialogResult.No)
                {
                    e.Cancel = true;
                }
                else
                {
                    // If the user did want to close, forcibly quit the application
                    Environment.Exit(0);
                }

            }
        }
```

I can now test this to see if it successfully works by deleting the config file from the debug directory:

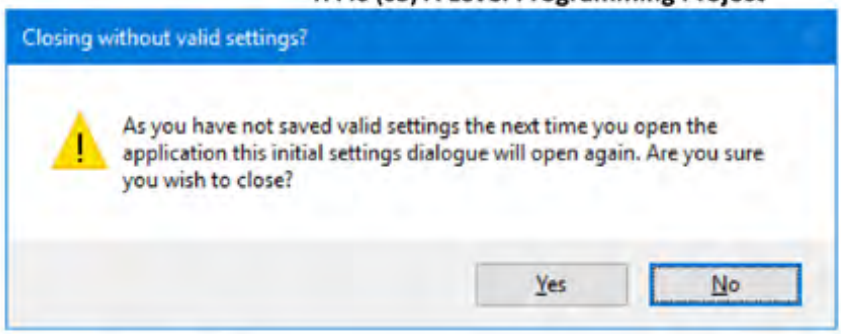
Name	Date modified	Type	Size
BCrypt.Net.dll	21/11/2013 04:53 ...	Application extens...	15 KB
BCrypt.Net.pdb	21/11/2013 04:53 ...	PDB File	24 KB
BCrypt.Net.xml	21/11/2013 04:53 ...	XML Document	12 KB
chart.png	24/02/2017 06:56 ...	PNG File	13 KB
last7days.png	06/04/2017 09:09 ...	PNG File	12 KB
last60mins.png	06/04/2017 09:09 ...	PNG File	8 KB
MySql.Data.dll	17/06/2016 06:29 ...	Application extens...	415 KB
past10songs.png	06/04/2017 09:09 ...	PNG File	17 KB
report.html	06/04/2017 09:09 ...	Chrome HTML Do...	1 KB
Server Listener Statistics.exe	07/04/2017 11:08 ...	Application	1,186 KB
Server Listener Statistics.exe.config	21/12/2016 03:59 ...	CONFIG File	1 KB
Server Listener Statistics.pdb	07/04/2017 11:08 ...	PDB File	126 KB
Server Listener Statistics.vshost.exe	07/04/2017 11:08 ...	Application	23 KB
Server Listener Statistics.vshost.exe.config	21/12/2016 03:59 ...	CONFIG File	1 KB
Server Listener Statistics.vshost.exe.manif...	16/07/2016 12:44 ...	MANIFEST File	1 KB

Now I can test what happens when I run the project for the first time:

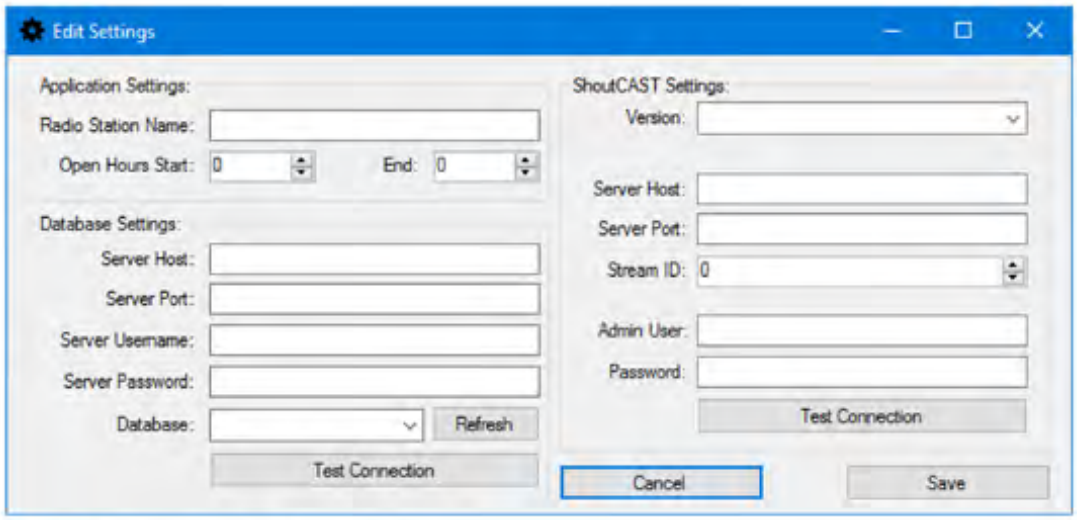




The settings validation as tested previously will work fine, so there is no need to re-test validation as none of that code has been changed. I can now test what happens when a user tries to close the form when no data has been saved:



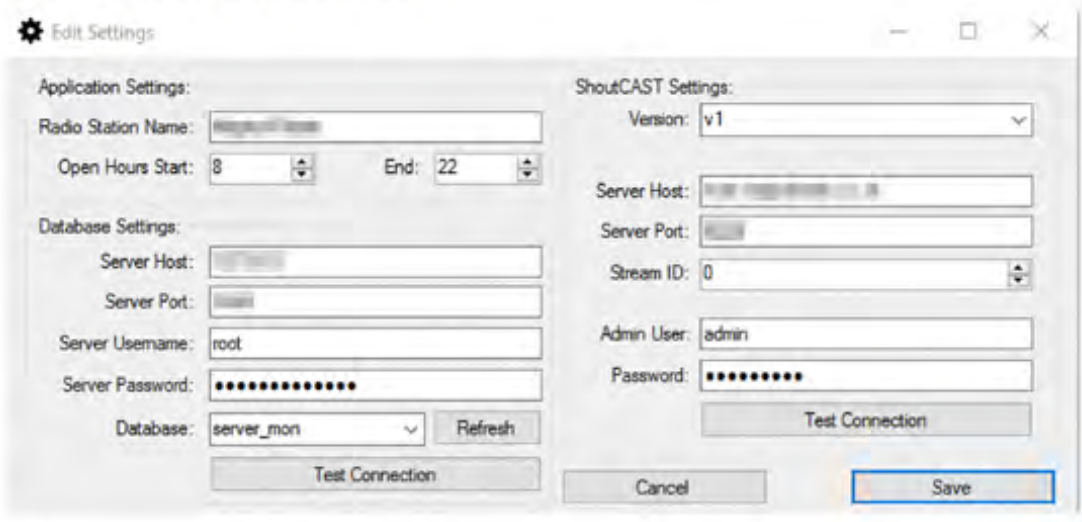
When no is pressed:

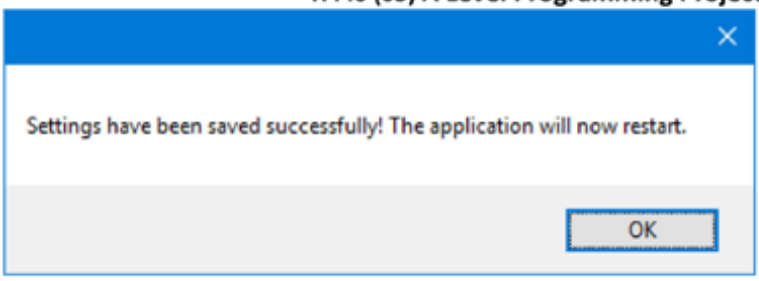


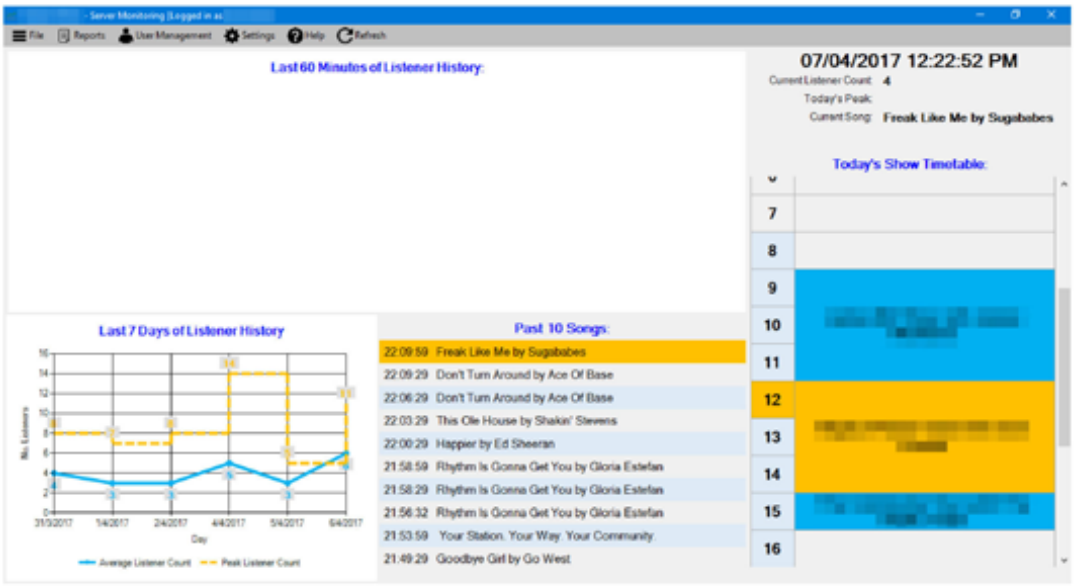
This works as it returns back to the form

When yes is pressed the form is closed and the application quits successfully (the icon was already removed from the notification icon dock when it started up without a valid config):

Now I can test to make sure that new settings entered work correctly, I will re-enter the settings that the application used to use, then make sure that the settings work correctly by displaying all of the data that the application should still contain:







As you can see from the above view the data has been loaded correctly from the settings that have been entered (the reason no data is in the past 60 minutes is because the system has just been loaded for today so not data has been recorded as of yet). This concludes all of the development for the system. Now I can move onto final testing.

Testing

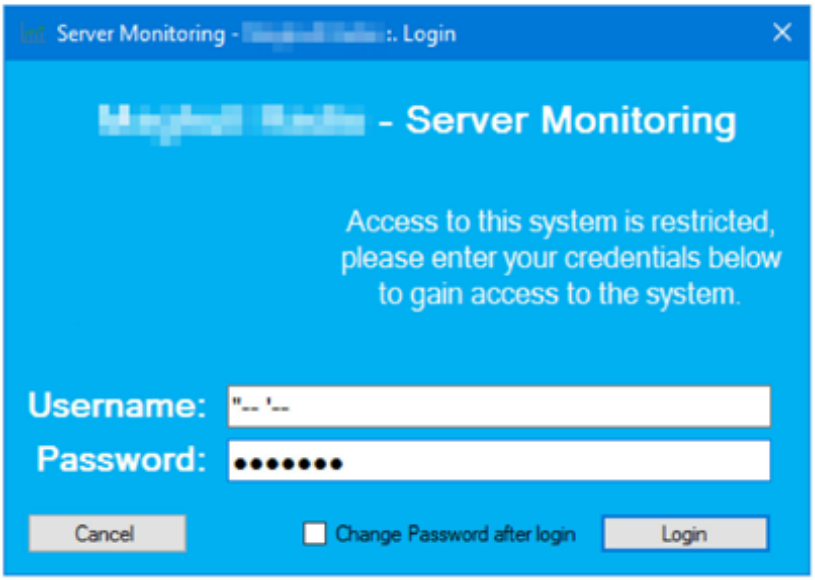
Throughout Development I have used iterative testing to test each module in depth and then if all modules work and opening each module also works, then the entire system will work together as a whole. However, I still need to do Destructive Testing to make sure I can't break the system. I will plan a test plan to record these tests:

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

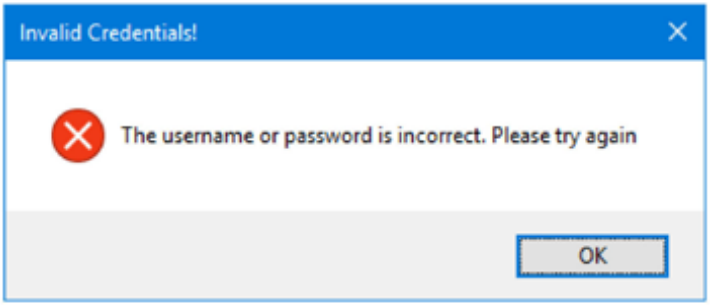
6	SQL Connection dropout		It should try again within 30 seconds
7	Leave application running for 48 hours		It should run fine and store data for 48 hours
8	SQL Injection On Song Name	Use a packet injector to change the name of the song to the SQL test escape string of "-- '--	It should insert the record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

Test 1: SQL Injection

User Login form:

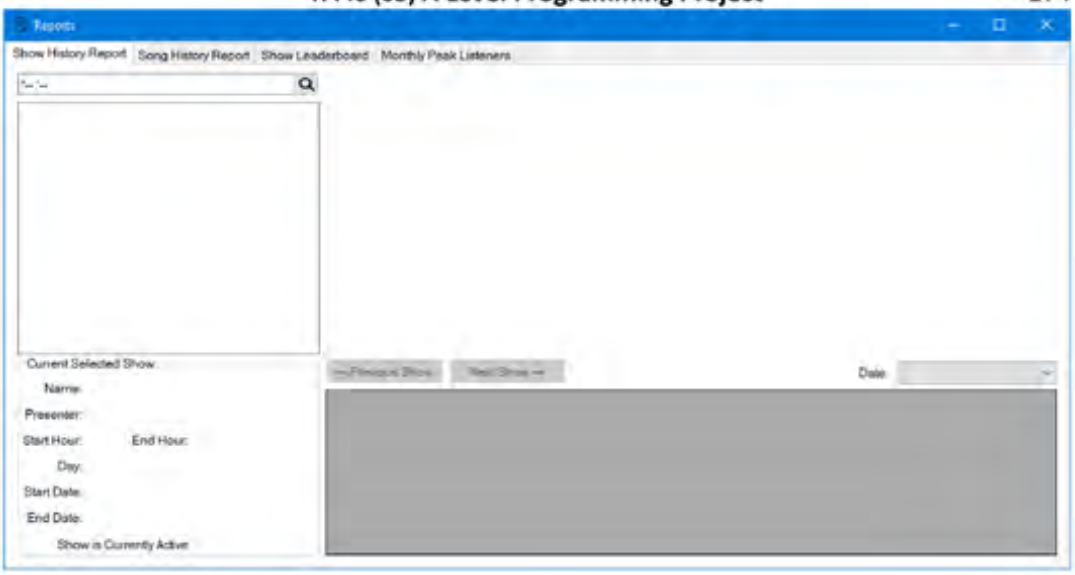


Result:

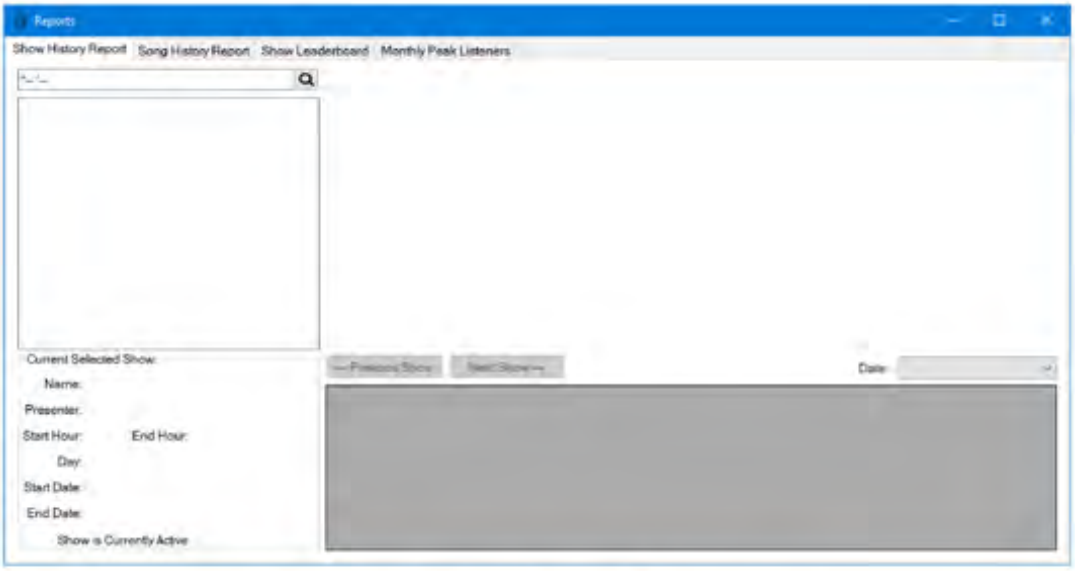


Test passed application did not crash.

Reports: Show History Report



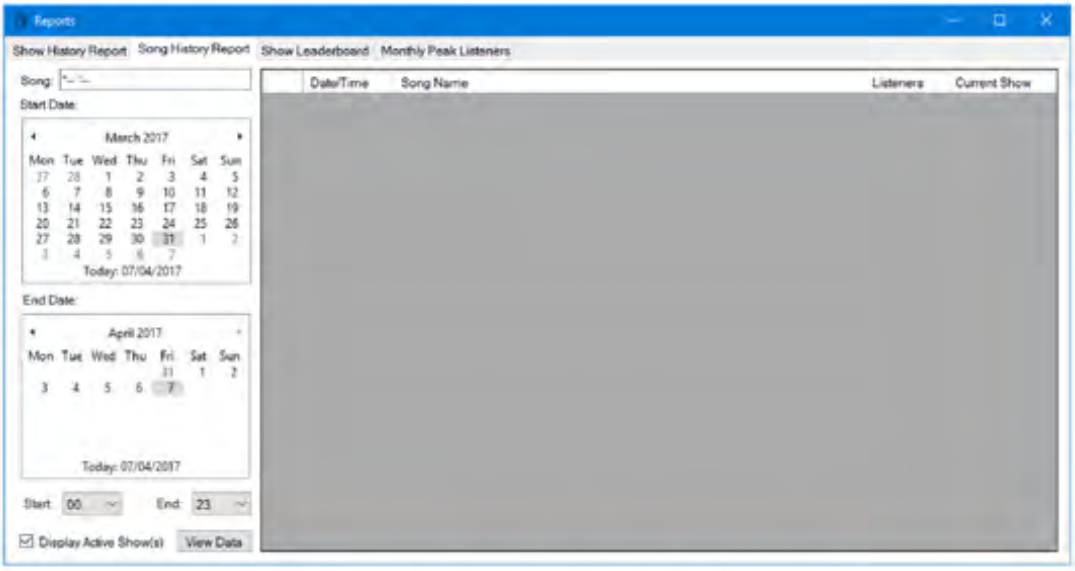
Result:



Nothing was returned, and application did not crash, so the test has passed:

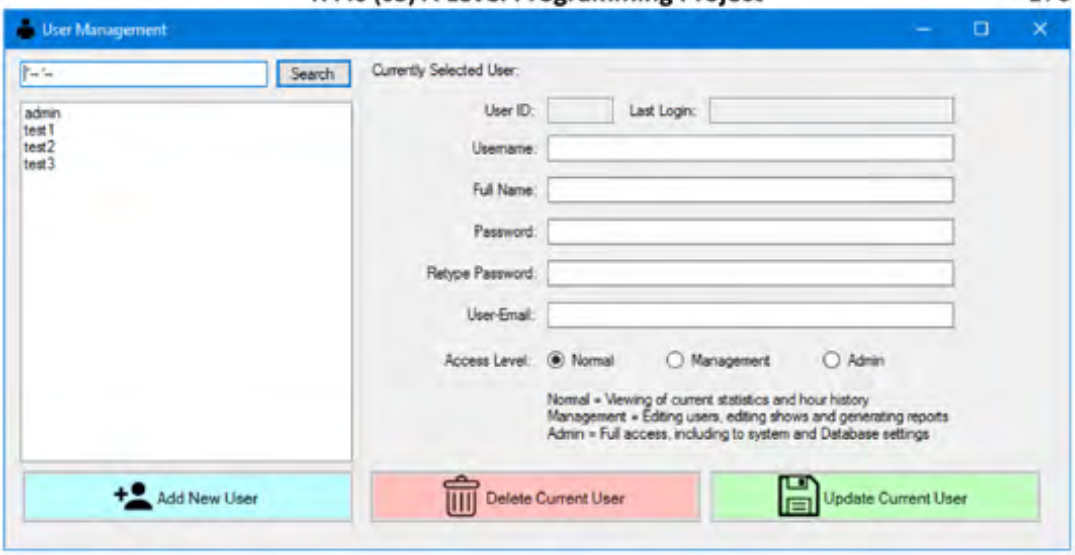
Reports: Song History Report

Result:

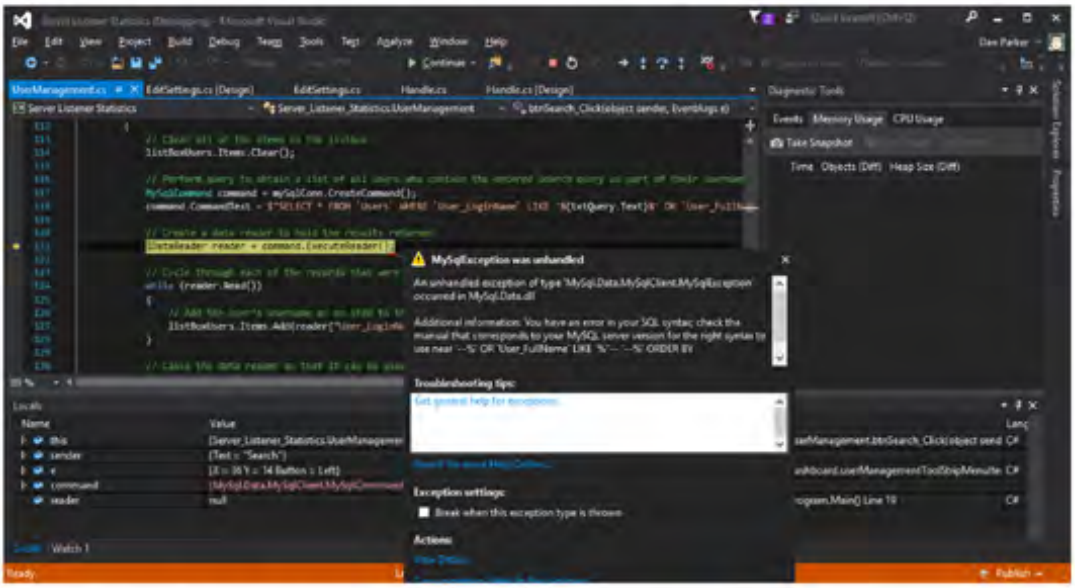


No data was returned, and the application did not crash so this test has passed.

Searching for a user:



After pressing search the application crashed with the following error:



This can be rectified by adjusting the following code so it passes in the search parameter by a command rather then as a string:

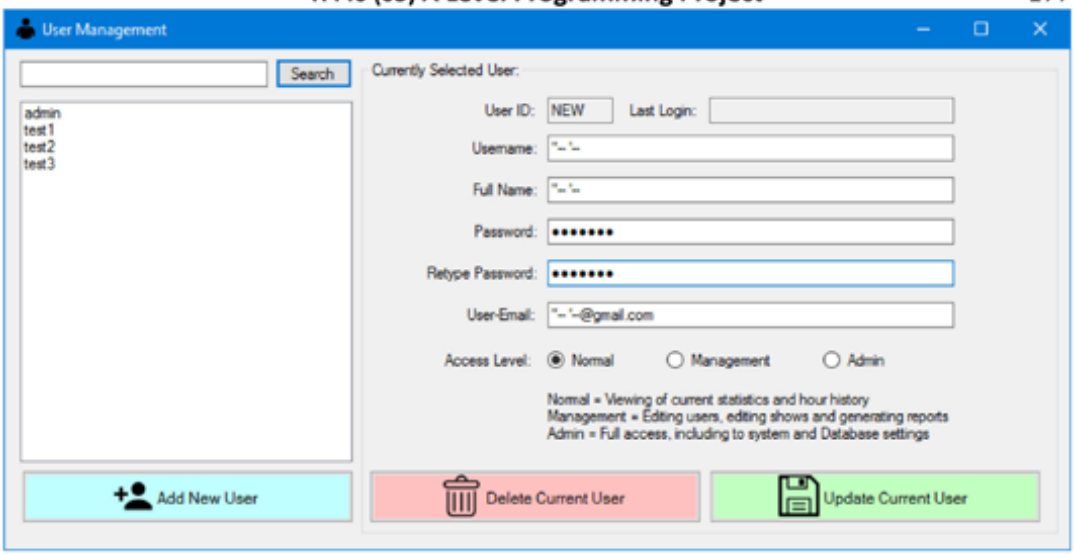
```
// Perform query to obtain a list of all users who contain the entered search query as
part of their username or their full name
            MySqlCommand command = mySqlConn.CreateCommand();
            command.CommandText = $"SELECT * FROM `Users` WHERE `User_LoginName` LIKE
'%{txtQuery.Text}%'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

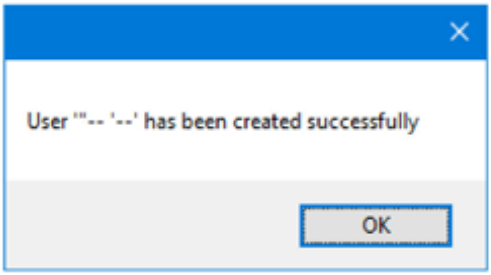
With the following:

```
// 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
@query OR `User_FullName` LIKE @query ORDER BY `User_LoginName` ASC";
            command.Parameters.AddWithValue("@query", $"%{txtQuery.Text}%");
```

This now prevents any SQL injection as it is passed in as a command rather then using string interpolation.

Adding a New user:





	User_FullName	Use...	User_Email	User_Password		User_LastLogin
21		admin	admin@gmail.com		3	2017-04-07 15:12:3
22	Test Account 1 (Access Level 1)	test1	test1@gmail.com		1	2017-04-06 14:24:2
23	Test Account 2 (Access Level 2)	test2	test2@gmail.com		2	2017-04-07 12:36:3
24	Test Account 3 (Access Level 3)	test3	test3@gmail.com		3	2017-04-06 13:32:0
25	'-- '--	'-- '--	'-- '--@gmail.com		1	(NULL)

As you can see from the above screenshot, the test has passed and the expected result occurred, with the user being successfully added to the system.

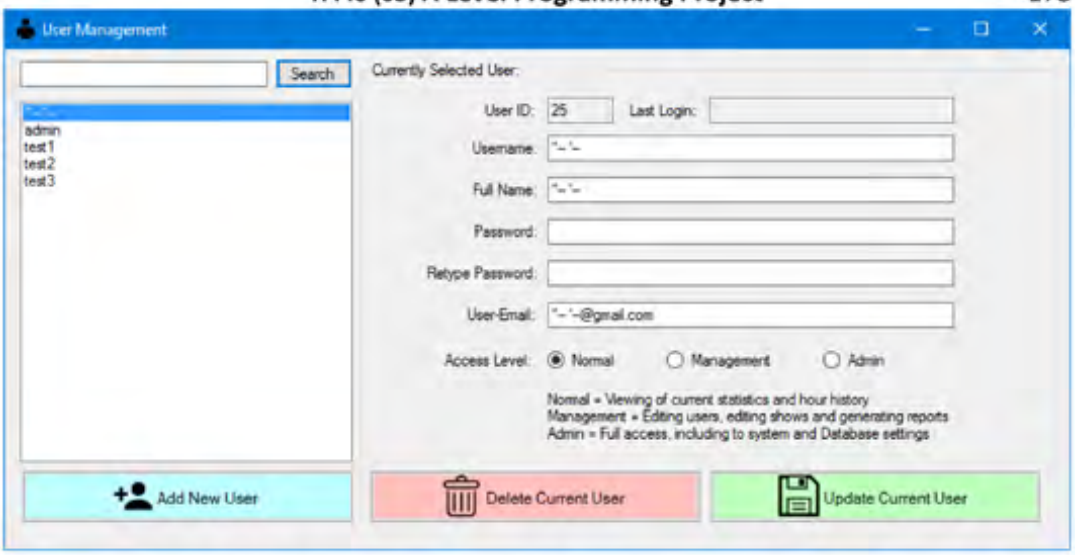
The only issue is that when the user is selected, their data will not appear as the SQL query will produce an error when loading the user's data. To correct this I will need to change the following code to also use a command parameter rather than string interpolation:

```
// 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()}'";
```

To the following:

```
// Run query to obtain selected user's details from the database
                MySqlCommand command = mySqlConn.CreateCommand();
                command.CommandText = $"SELECT * FROM `users` WHERE `User_LoginName` =
@user";
                command.Parameters.AddWithValue("@user",
listBoxUsers.SelectedItem.ToString());
```

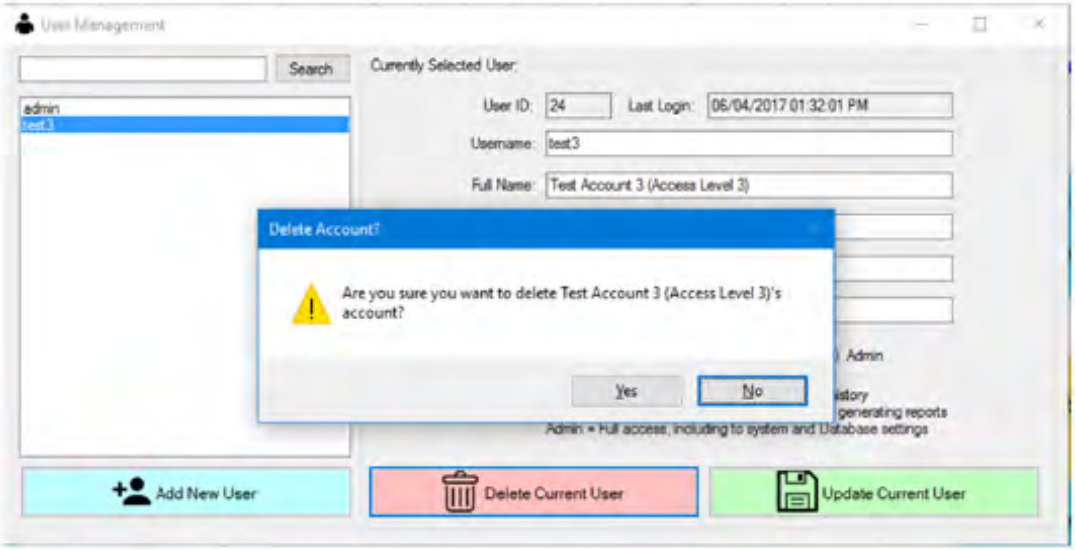
I can now test to make sure that this works:

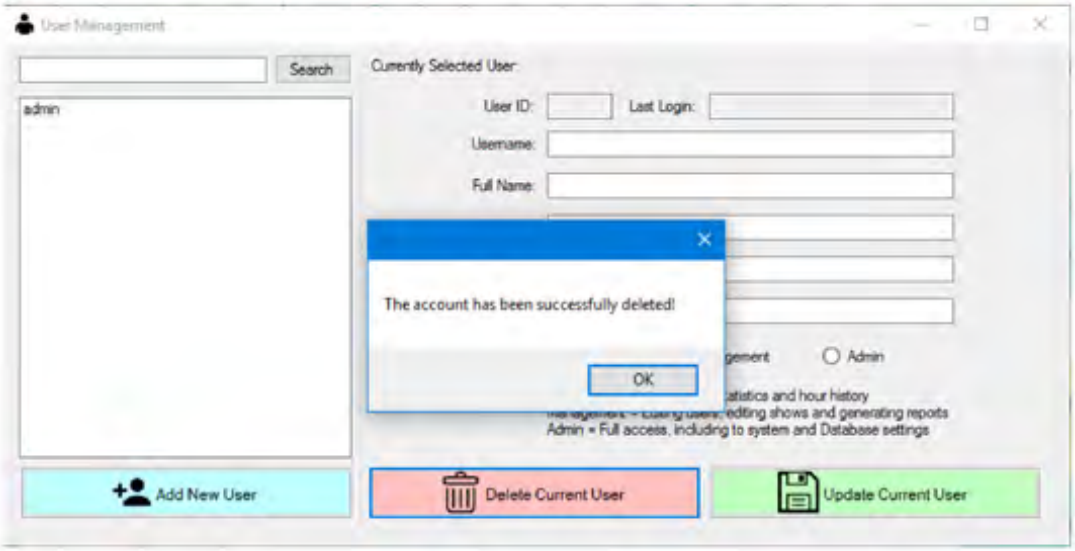


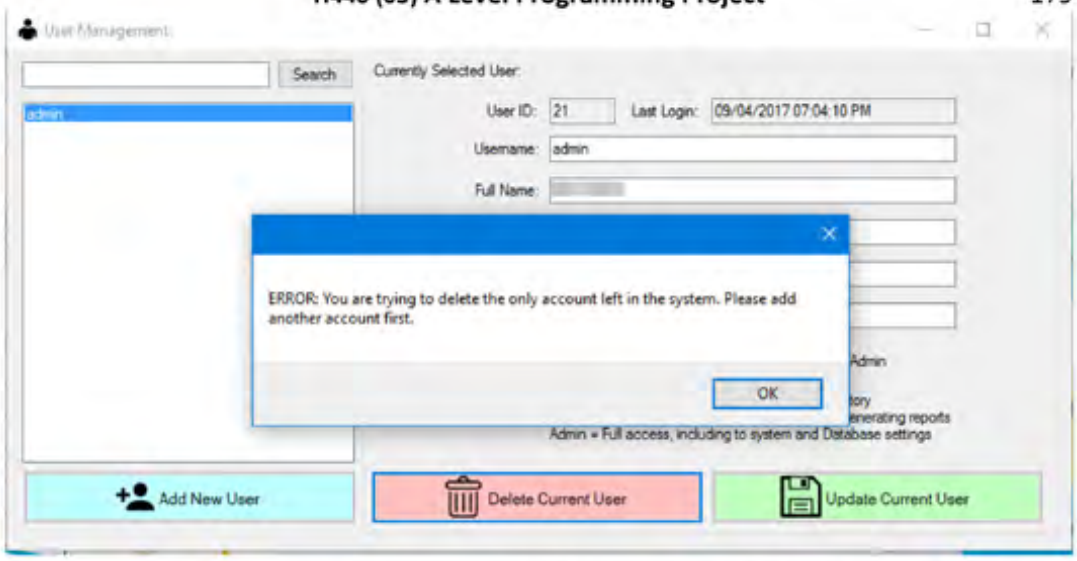
This now works, so the User Management form is no longer suseptable to SQL injection, or errors caused with special characters.

Test 2: Deleting all users

This test will try to delete all users, and make sure the user can not lock themselves out of the system.







As you can see from the screenshots above, the Test 3 account could be deleted successfully, but the admin account would not be deleted as it is the only one left in the database. This test has passed.

Test 3: Running multiple instances of the one application on the same PC

I will test this by opening two copies of the application and logging in as the same user to make sure that the dashboard is working correctly. This will also mean that the rest of the application will work if the dashboard works and can be logged into:

As you can see from the screenshot above, two instances are successfully running on the same computer with no problems, they are both also displaying the same data, so this test has passed aswell.

Test 4: Running multiple instances of the one application across the network

I can test this by installing the application on site in the radio station, and remotely connecting to both computers to show that it is working:

As you can see from the above screenshot, this shows that the application can successfully work on multiple computers within the radio station. This test has successfully passed.

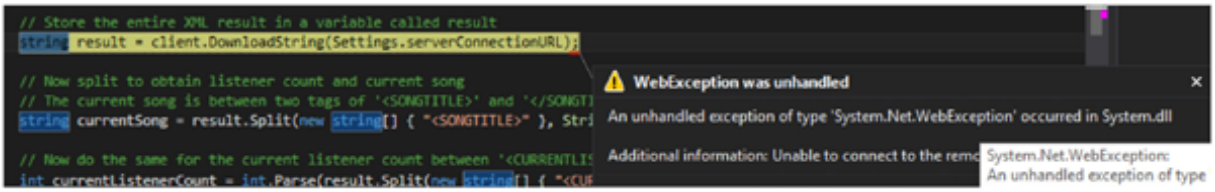
Test 5: Internet connection dropout

To perform this test I will run a local copy of the application, and turn off the internet connection and see what happens if there is an internet dropout, or if the server can't be reached.

With internet connection:

When wifi is turned off:

This produces an error as the application can't download the required information from the server:



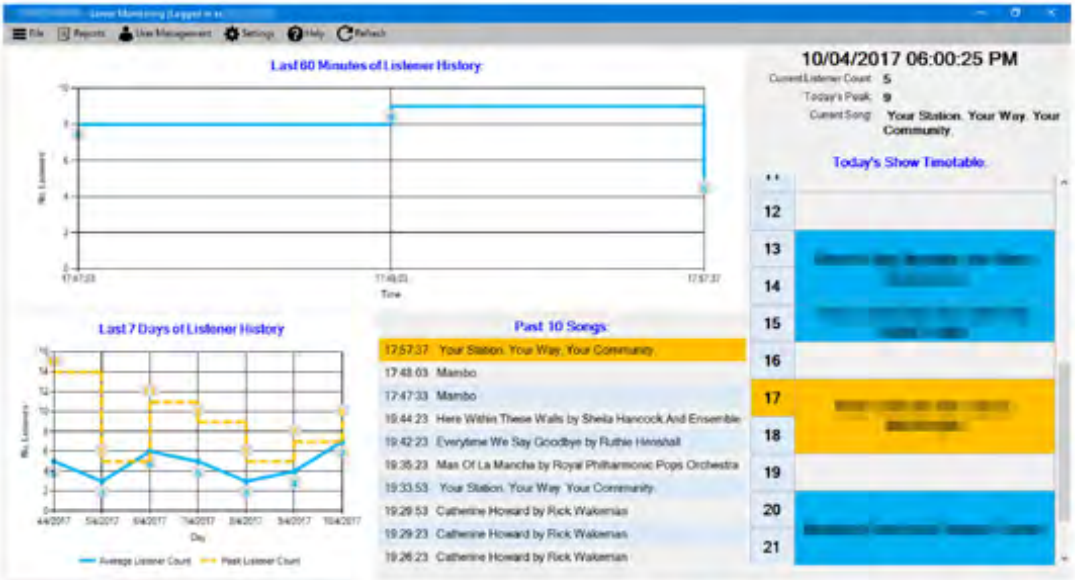
This could happen from time to time, as you can't always guarantee if there will be a stable internet connection. To fix this I will surround it in a try catch loop, so if there is an error produced, it will ignore it and try again in 30 seconds next time it runs. It will also log the error to a log file, so that it can be opened to see what the issue was for debugging purposes. This can be seen in the modified timerObtainStats_Tick method below:

```
        private void timerObtainStats_Tick(object sender, EventArgs e)
        {
            // Create a try loop so if the code errors out the application will not crash
            try
            {
                // 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 charachters, so it will split a
string by a string delimeter
                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]);


                // 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
```

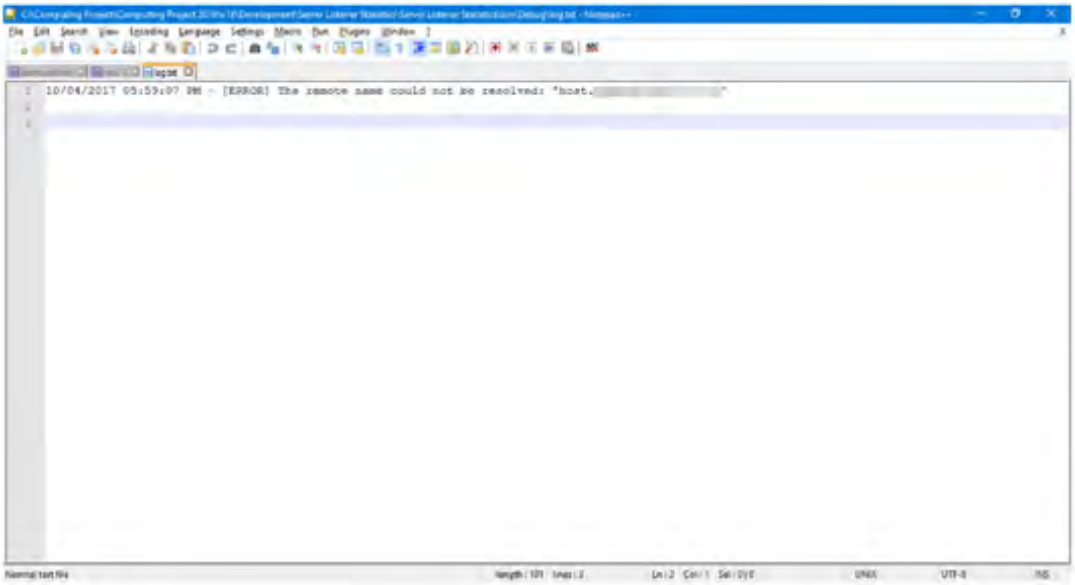
```
                IDataReader reader = command.ExecuteReader();

                // 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();

                        //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 characters long
                        if (notifyText.Length > 63)
                        {
                            // The text is too long, we need the first 60 characters, and
add on an ellipsis 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;

                    }

                }

                // 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();
            }
```

```
            catch (Exception ex)
            {
                // No handling needed, as loop will run again in 30 seconds
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();
            }

    }
```

I can now test this again and turn the internet connection off while the application is running.



As you can see from the screenshot above the application did not crash and could still be used, however the error was written to the log file as you can see below:

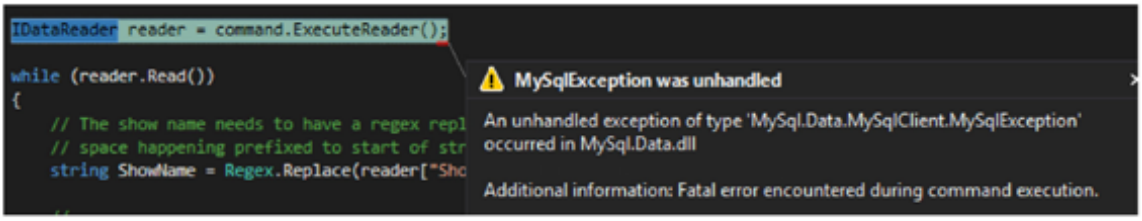


This is a much better way of handling errors, as the system is designed to be unobtrusive, however if somebody is using the computer and the internet connection drops, then the application would display the JIT (Just In Time) Error Handling message which would interrupt what the user is doing. This new way will allow for the errors to still be viewed to see what the issues were, but they will not be annoying for the end user.

Test 6: SQL Connection dropout

I can now turn off the SQL server while the application is running, and I can see what will happen.

This produces the following exception:



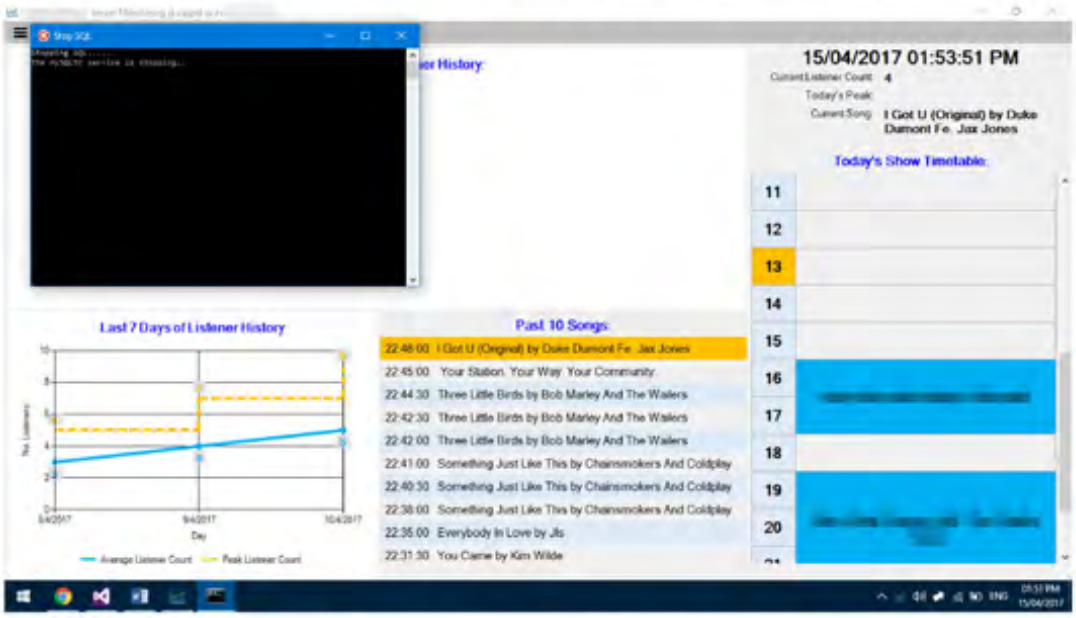
The issue with this is that it will happen at whatever is the next SQL query to be executed. And this will vary depending on what the user is doing. The 30 second update will cause an error, but if the user is trying to open the login form and the SQL server is not on, then as soon as they try to open the form it will produce an error. This means I will have to surround all SQL connections and queries with a try catch loop. If the error occurs on the update loop that inserts the current statistics into the database then it will be written to the log file, and will try again the next time the loop runs. However if it is part of any other form, it will be written to the log file, the error will also be displayed in a message box, and the application will then exit. The user will then have to manually restart the application afterwards.

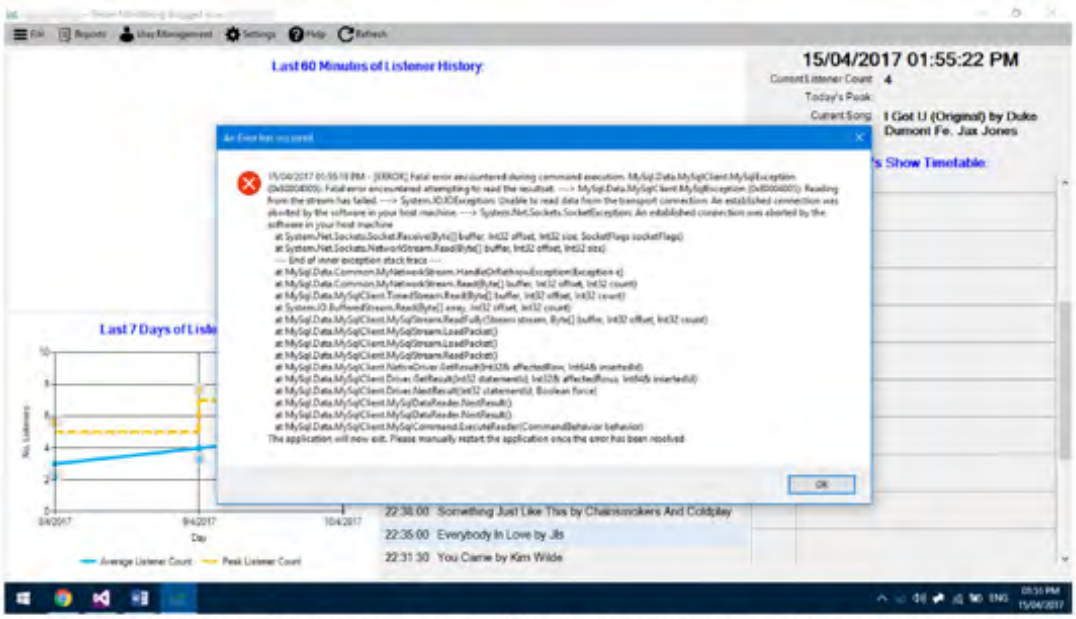
The update loop will not have to be changed, as the SQL error will be caught with the current code used to originally catch the internet connection dropout error, as no error type was defined it will catch all errors. I will need to surround all other instances of SQL connections with the following code across all forms:

```
try
            {
                // CODE TO CATCH
            }
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Environment.Exit(0);
            }
```

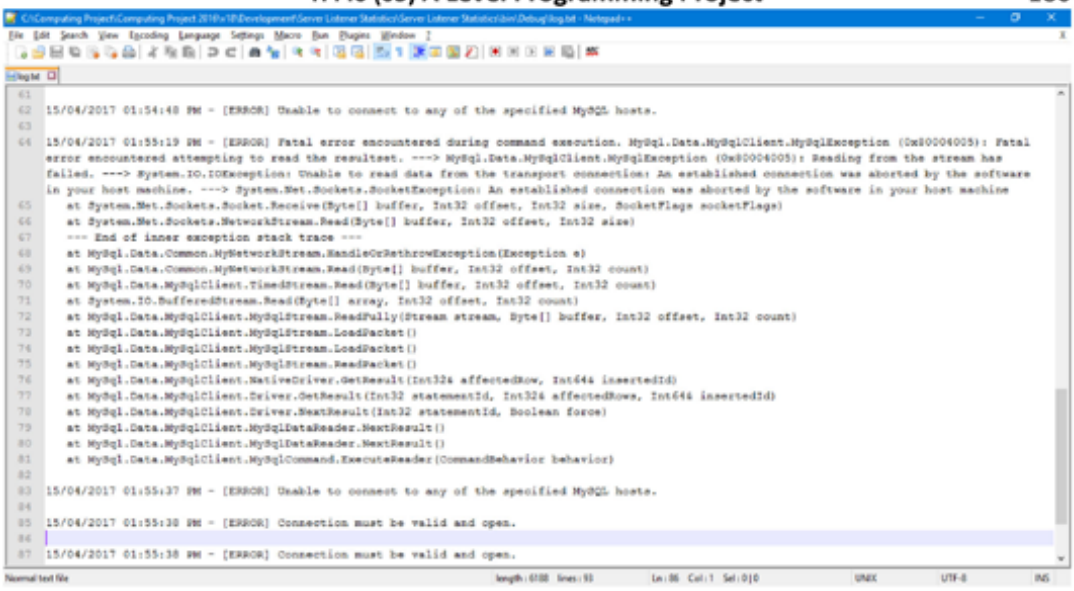
The comment of 'CODE TO CATCH' will be the code that is to be ran. It would take too much time and space to repeat this in the documentation for all methods of code, so it can be seen in the appendix.

This code can now be tested by turning off the SQL server when the application is running:





This works great. While the message is long it allows the user to read it and find out what the problem is, you can also see from the log text file below, that the error has been successfully logged in the application:



This has logged the error correctly, and would allow you to see that the SQL server was not active or running, the issue could then be fixed. This is much better than the application freezing and the Just in Time debugger opening. This test has passed.

Test 7: Leaving the application running for 48 hours

This test will make sure that the application runs successfully for 48 hours or more. This can be tested by leaving the application debugging in visual studio and see if it crashes after a long period of time. I started the application running at 1pm, and at 6pm the application crashed. This was due to Visual studio not releasing memory, as it proceeded to take up all of the 9GB of RAM by storing all of the web requests it made in case they needed to be inspected afterwards. I then re-ran the test, this time without the application in visual studio, by running the executable created in the debug directory. The system successfully reported on listener statistics and ran consistently for more than 48 hours, it ran for a full week without any problems. Unfortunately there are no screenshots I can provide to prove this, as it would take a massive dataset to show it consistently worked, but I can show two records that are far apart in the database, knowing the system functioned perfectly between both of those points, as no errors were logged:

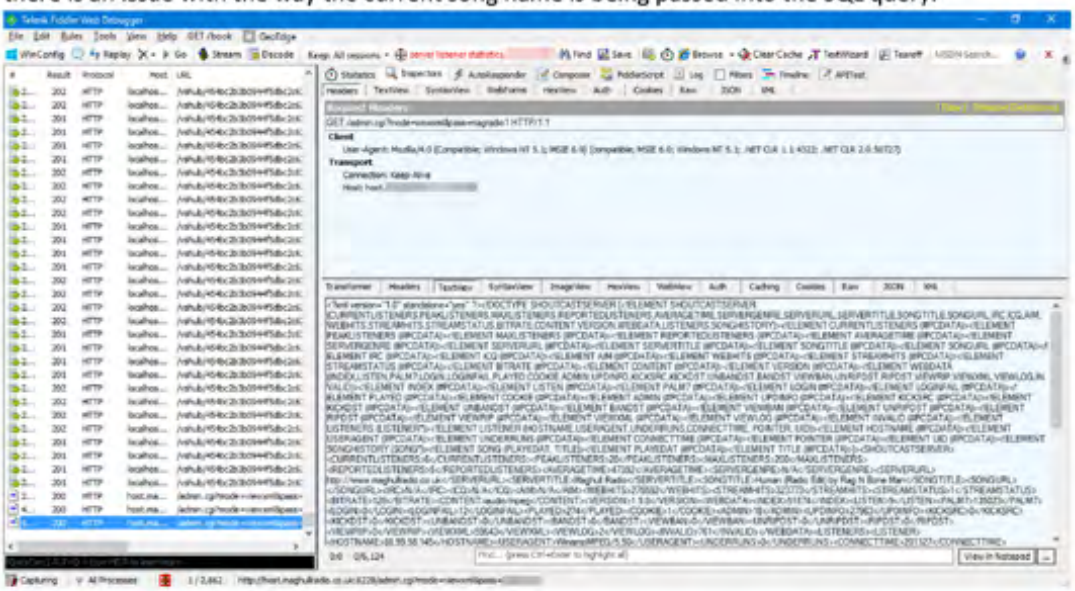
Log_ID	Log_DateTime	Log_ListenerCount	Log_CurrentSong	Log_ShowID
5,580	2017-04-07 15:15:55	3	Firework by Katy Perry	8
9,919	2017-04-15 14:36:06	3	The Hindu Times by Oasis	(NULL)

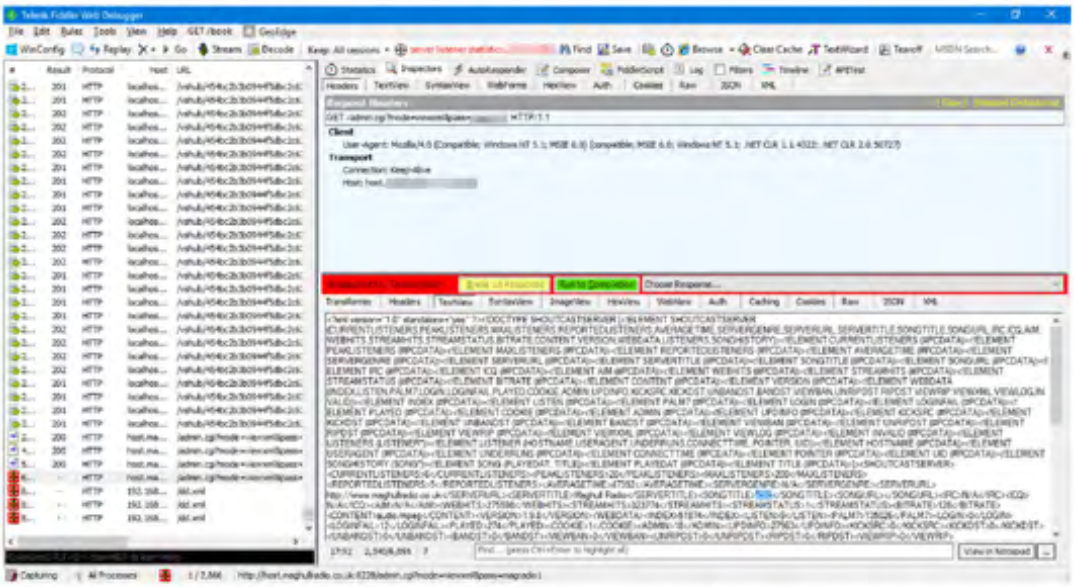
This shows that the system has been running continuously for 7 days, 23 hours without producing any errors. This test has now passed.

Test 8: SQL Injection on Song Name

I will now test to see if the current song name is vulnerable to the same SQL injection as the other user inputs. This will also check to make sure that the system can accept all special characters within the song name, as if the SQL test injection string can successfully be saved, then any other combination of characters in the string will also be saved correctly. I can use a program called Telerik fiddler 4, which will allow for the network data being sent to the application, to be modified before the program receives it. I will change the current song name in the XML file to "-- '-- this will then enable me to see if

there is an issue with the way the current song name is being passed into the SQL query.





This produces the following error saved to the log file, as errors to do with the handle are recorded in the log file and not displayed to the user:

```
[ERROR] You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use
near '' at line 1
```

This can be fixed by changing the SQL query, to pass in the name of the song as a command parameter, rather than a concatenated string. I will change the following line:

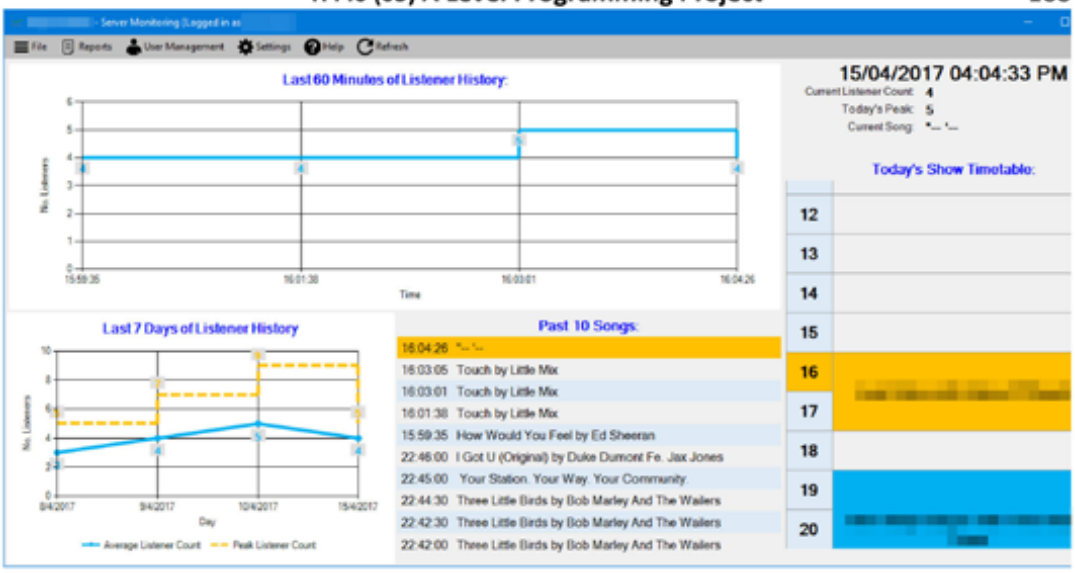
```
command.CommandText = $"INSERT INTO log(Log_DateTime, Log_ListenerCount, Log_CurrentSong,
Log_ShowID) VALUES(NOW(), {currentListenerCount}, \"{currentSong}\", {currentShowID});";
```

To the following:

```
command.CommandText = $"INSERT INTO log(Log_DateTime, Log_ListenerCount, Log_CurrentSong,
Log_ShowID) VALUES(NOW(), {currentListenerCount}, @currentsong, {currentShowID});";
command.Parameters.AddWithValue("@currentsong", currentSong);
```

This can now be tested again, using the same test data above. This time the song name was successfully added to the database, and it did not cause an SQL syntax error, as you can see from the screenshot below:

H446 (03) A Level Programming Project 188



Test No.	Test Description	Test Data	Expected Result	Actual Result	Action taken after test
1	SQL Injection	Pass the string "'-- '-- into every textbox	The system should handle this without error	Everything worked fine other then the user query search	Adjusted user query search to pass in as command with paramater
2	Deleting all users		The last user should not be removed	The last user could not be removed	None needed as test passed
3	Running multiple instances of the one application on the same PC		It should run fine with no errors	Ran fine with no errors	None needed as test passed
4	Running multiple instances of the one application across the network		It should run fine with no errors	Ran fine with no errors	None needed as test passed
5	Internet connection dropout		It should try again within 30 seconds	Application crashed	Surrounded code with try catch loop to log to file if error occoured and try again in 30 seconds
6	SQL Connection dropout		It should try again within 30 seconds	Application crashed	Surrounded all sql connections/quries with try catch loop to log to file, display error to user and close application
7	Leave application running for 48 hours		It should run fine and store data for 48 hours	Ran fine (outside of Visual Studio) for 7 days and 23 hours	None needed as test passed

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	Song was not inserted if name contained special character or SQL injection test string, error was logged to file	Adjusted song insertion query to pass in the song name as command with parameter rather then a string

Evaluation

I can now look back at my system goals I defined in the analysis section of my project and justify if my end result suitably meets this criteria. The criteria was split into two sections. Essential features; ones that will need to be developed for the application to function and serve its correct purpose; and Desirable features, these are things that would be nice to have, but are not imperative for the normal operation of the system.

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.

The system meets this criteria from the start of the database design, the log table will store all of the data from when the system was first setup. It will log data with a minimum interval of every 30 seconds, containing the song name, listener count and the current show that is on. This then allows for quick and efficient querying of data to return results within any time frame. You can see this from the screenshots below that show how data is logged into the database:

Log_ID	Log_DateTime	Log_ListenerCount	Log_CurrentSong	Log_ShowID
8,206	2017-04-12 21:52:32	2	Royals by Lorde	(NULL)
8,207	2017-04-12 21:55:32	2	She Wolf (Falling To Pieces) by David Guetta Ft. Sia	(NULL)
8,208	2017-04-12 21:59:02	3	Bonnie And Clyde by Jay Z Ft Beyonce Knowles	(NULL)
8,209	2017-04-12 21:59:32	2	Bonnie And Clyde by Jay Z Ft Beyonce Knowles	(NULL)
8,210	2017-04-12 22:00:32	2	Call On Me by Starley	(NULL)
8,211	2017-04-12 22:02:02	3	Call On Me by Starley	(NULL)
8,212	2017-04-12 22:02:32	2	Call On Me by Starley	(NULL)
8,213	2017-04-12 22:03:02	2	So Strong by Labi Siffre	(NULL)
8,214	2017-04-12 22:07:32	2	Regret by New Order	(NULL)
8,215	2017-04-12 22:11:32	2	Go Let It Out by Oasis	(NULL)
8,216	2017-04-12 22:16:02	2	Just The Way You Are by Bruno Mars	(NULL)
8,217	2017-04-12 22:19:32	2	Your Station. Your Way. Your Community.	(NULL)
8,218	2017-04-12 22:20:32	2	Earth Song(Radio) by Michael Jackson	(NULL)
8,219	2017-04-12 22:25:32	2	Text From Your Ex (Clean Radio Edit) by Tinie Tempah Fe...	(NULL)
8,220	2017-04-12 22:27:32	4	Text From Your Ex (Clean Radio Edit) by Tinie Tempah Fe...	(NULL)
8,221	2017-04-12 22:28:02	3	Text From Your Ex (Clean Radio Edit) by Tinie Tempah Fe...	(NULL)
8,222	2017-04-12 22:28:32	3	Love Action by The Human League	(NULL)
8,223	2017-04-12 22:30:32	2	Love Action by The Human League	(NULL)
8,224	2017-04-12 22:32:02	2	Shine by Take That	(NULL)
8,225	2017-04-12 22:35:32	2	Solo Dance (Radio Edit) by Martin Jensen	(NULL)
8,226	2017-04-12 22:38:32	2	Love Is In Control by Donna Summer	(NULL)
8,227	2017-04-12 22:42:32	2	Your Station. Your Way. Your Community.	(NULL)
8,228	2017-04-12 22:43:32	2	Unchained Melody by Robson And Jerome	(NULL)
8,229	2017-04-12 22:47:02	2	Turn Me On by David Guetta Ft Nicki Minaj	(NULL)
8,230	2017-04-12 22:50:02	3	Your Station. Your Way. Your Community.	(NULL)
8,231	2017-04-12 22:50:32	2	Spectrum by Florence And The Machine	(NULL)
8,232	2017-04-12 22:54:32	2	Clown by Emeli Sandé	(NULL)
8,233	2017-04-12 22:58:02	2	Jennifer Lopez by Do It Well	(NULL)
8,234	2017-04-12 23:00:32	2	Shout Out To My Ex by Little Mix	(NULL)
8,235	2017-04-12 23:04:02	2	No Regrets by Midge Ure	(NULL)
8,236	2017-04-12 23:07:32	3	No Regrets by Midge Ure	(NULL)
8,237	2017-04-12 23:08:02	2	Come As You Are by Nirvana	(NULL)
8,238	2017-04-12 23:11:32	2	That's Not My Name by Ting Tings	(NULL)

This is also tested during the iterative testing of the handle form, and setting the taskbar icon to contain the current playing song and listener count. This means the system fully meets this criteria.

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

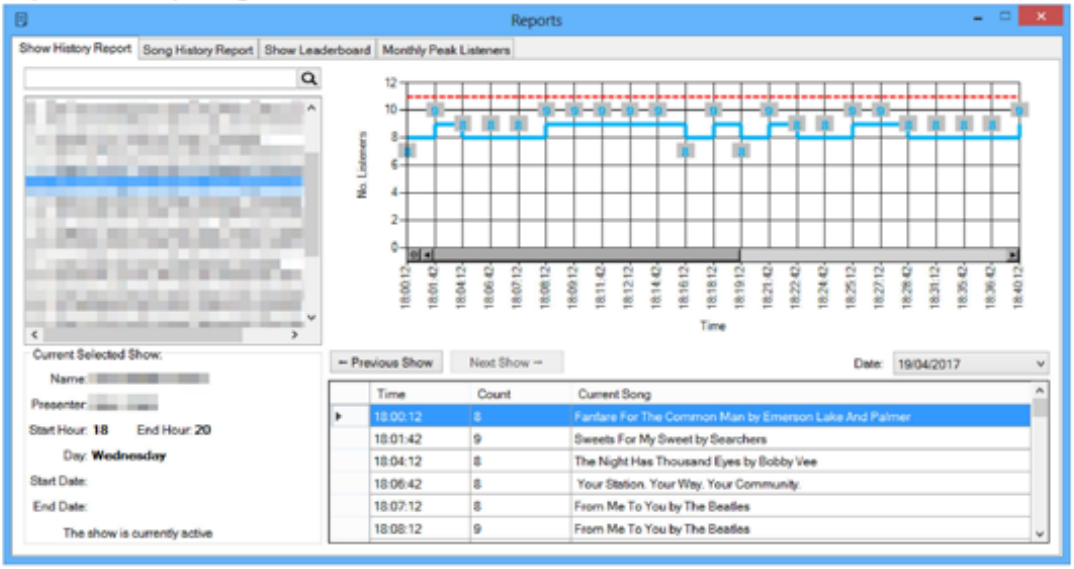
The database will have a structure enabling data to be stored into it about all of the past listener history, while being as efficient with data storage as possible.

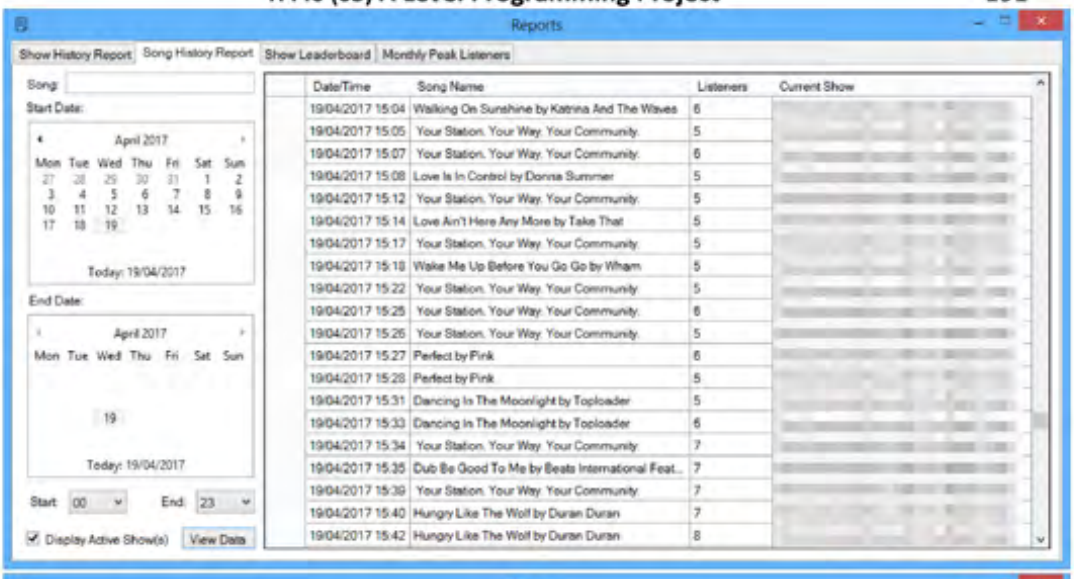
The database has a concise table layout, only storing key information. A new record is only inserted once data has changed, so duplicate data will not be stored until either the listener count, current song name, or show changes. This prevents on redundant data, and will reduce the total size of the database. The show details are stored separately in a shows table, the shows are only referenced by their ID. This allows for show names and times to be changed later on, but it also prevents duplicate data as the individual show details are not stored across many log records in the database. This will speed up querying time, as it has less data to sort through when obtaining records. It is also useful as you can use SQL commands to filter the time field by various options between two dates or times. You can also group this to view data by minutes, hours, days, weeks, or years. This will be really useful for reporting.
The system fully meets this criteria.

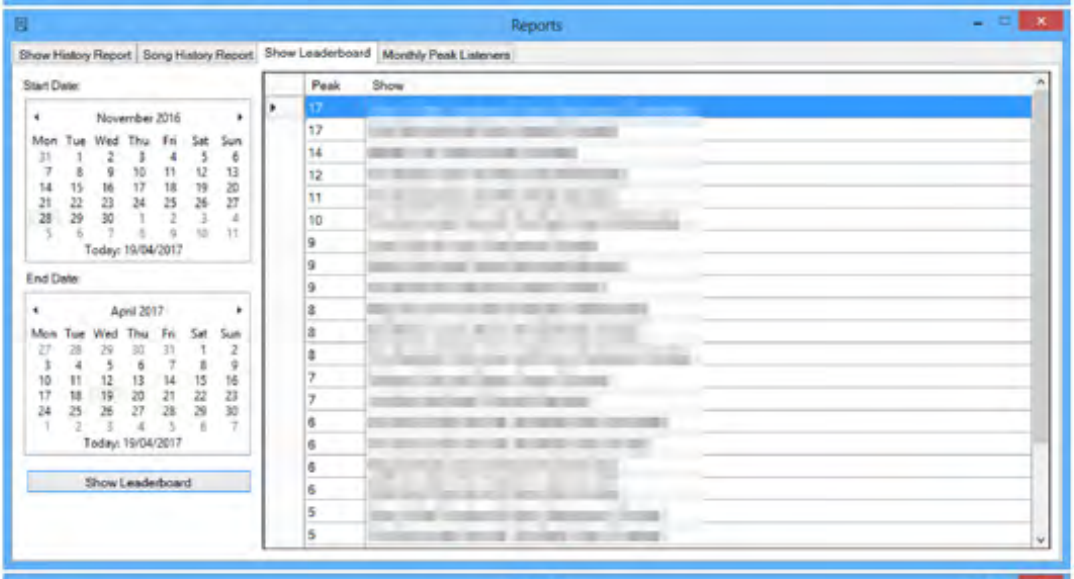
- **Different layouts/ways of viewing data (graphs or raw table reports)**

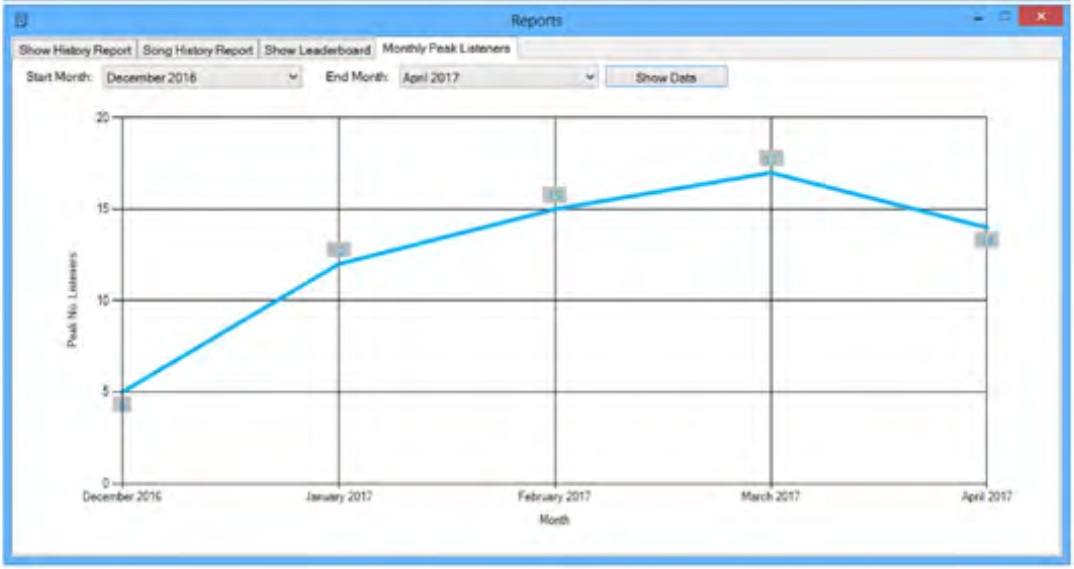
You will be able to pick from a selection of different reports you can run to obtain and view data from the database.

The reporting form contains both graphical and tabulated data from the database. This allows a wide selection of reports to be produced. This can be seen in screenshots below of the different tabs as part of the reporting form:





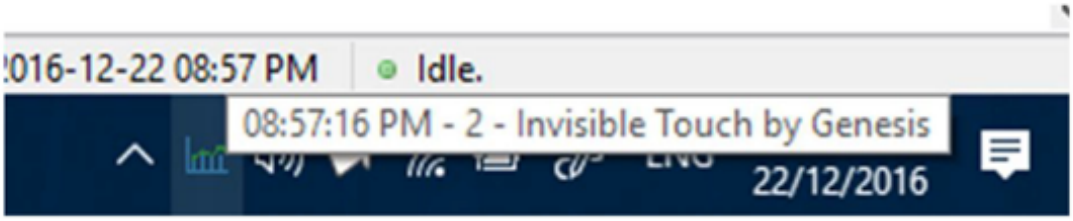




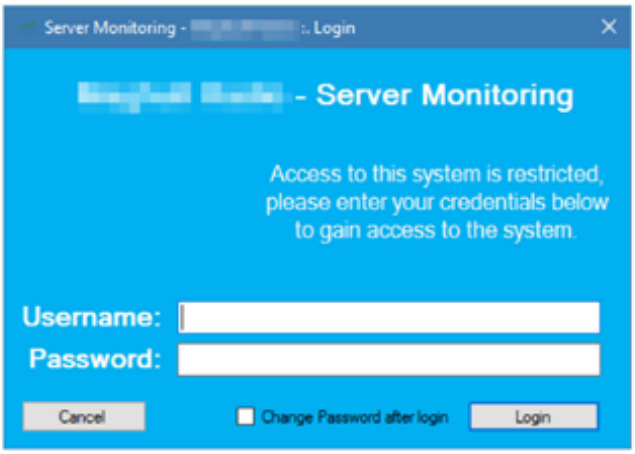
- **System can run in the background on a computer, without interfering with anything**

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. This is achieved by using a notification icon ran from the handle form that will sit in the user's toolbar next to the clock. This will then open the login form when the user double clicks on it. The notification icon also displays the current song and listener count when it is hovered over as a tooltip for quick reference. You can see this from the screenshots below of the notification icon:



This is unobtrusive as it will not prevent the user from normal computer use, and is not in the way or blocking any portion of the screen. When the icon is double clicked on the login screen will open as you can see below:

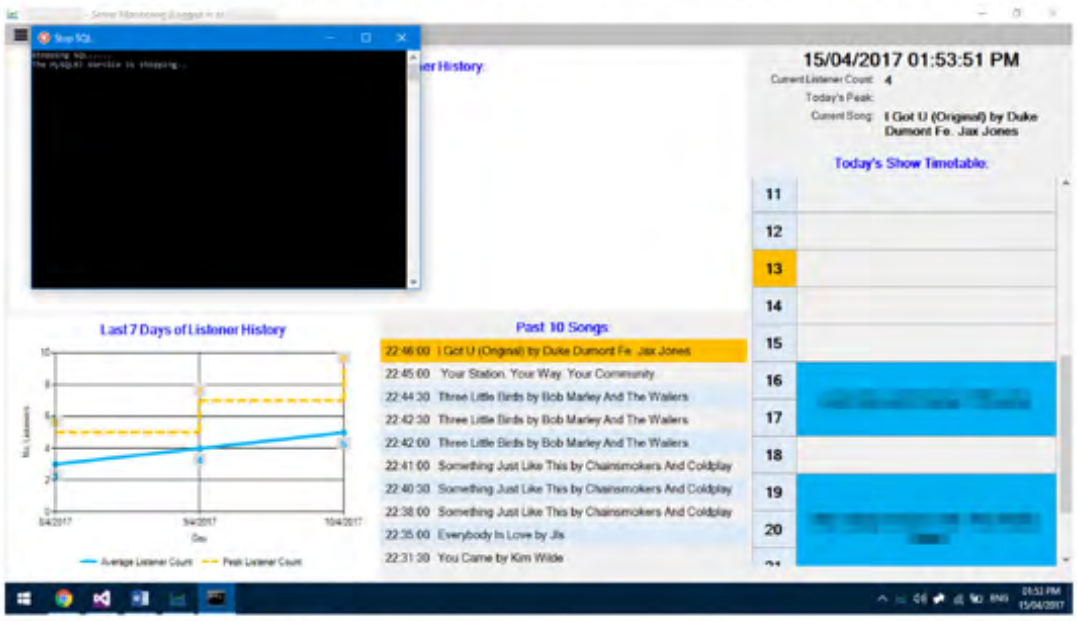


This will then allow the user to login to gain access based on their access level.
If the system produces any errors during its operation, they are logged to a file. If the error occurs on the handle form (the hidden form that runs code every 30 seconds to obtain new listener statistics) then it is logged to the file, and then it will try again in 30 seconds. However if the error occurs on the dashboard form when it is open, it is logged to the file, but also displayed as a message box to the user, and the application will also exit due to the error. As you can see from the screenshots below:
If the database is turned off while the application is running in the taskbar:

```
15/04/2017 01:55:19 PM - [ERROR] Fatal error encountered during command execution. MySql.Data.MySqlClient.MySqlException (0x80004005): Fatal
error encountered attempting to read the resultset. ---> MySql.Data.MySqlClient.MySqlException (0x80004005): Reading from the stream has
failed. ---> System.IO.IOException: Unable to read data from the transport connection: An established connection was aborted by the software
in your host machine. ---> System.Net.Sockets.SocketException: An established connection was aborted by the software in your host machine
   at System.Net.Sockets.Socket.Receive(Byte[] buffer, Int32 offset, Int32 size, SocketFlags socketFlags)
   at System.Net.Sockets.NetworkStream.Read(Byte[] buffer, Int32 offset, Int32 size)
   --- End of inner exception stack trace ---
   at MySql.Data.Common.MyNetworkStream.HandleOrRethrowException(Exception e)
   at MySql.Data.Common.MyNetworkStream.Read(Byte[] buffer, Int32 offset, Int32 count)
   at MySql.Data.MySqlClient.TimedStream.Read(Byte[] buffer, Int32 offset, Int32 count)
   at System.IO.BufferedStream.Read(Byte[] array, Int32 offset, Int32 count)
   at MySql.Data.MySqlClient.MySqlStream.ReadFully(Stream stream, Byte[] buffer, Int32 offset, Int32 count)
   at MySql.Data.MySqlClient.MySqlStream.LoadPacket()
   at MySql.Data.MySqlClient.MySqlStream.LoadPacket()
   at MySql.Data.MySqlClient.MySqlStream.ReadPacket()
   at MySql.Data.MySqlClient.NativeDriver.GetResult(Int32& affectedRow, Int64& insertedId)
   at MySql.Data.MySqlClient.Driver.GetResult(Int32 statementId, Int32& affectedRows, Int64& insertedId)
   at MySql.Data.MySqlClient.Driver.NextResult(Int32 statementId, Boolean force)
   at MySql.Data.MySqlClient.MySqlDataReader.NextResult()
   at MySql.Data.MySqlClient.MySqlDataReader.NextResult()
   at MySql.Data.MySqlClient.MySqlCommand.ExecuteReader(CommandBehavior behavior)
```

As you can see the error is just logged to the file shown above.

If the database server is turned off while the application is running on the dashboard:



The error message is also logged in the log file:

```
15/04/2017 01:55:19 PM - [ERROR] Fatal error encountered during command execution. MySql.Data.MySqlClient.MySqlException (0x80004005): Fatal
error encountered attempting to read the resultset. ---> MySql.Data.MySqlClient.MySqlException (0x80004005): Reading from the stream has
failed. ---> System.IO.IOException: Unable to read data from the transport connection: An established connection was aborted by the software
in your host machine. ---> System.Net.Sockets.SocketException: An established connection was aborted by the software in your host machine
   at System.Net.Sockets.Socket.Receive(Byte[] buffer, Int32 offset, Int32 size, SocketFlags socketFlags)
   at System.Net.Sockets.NetworkStream.Read(Byte[] buffer, Int32 offset, Int32 size)
   --- End of inner exception stack trace ---
   at MySql.Data.Common.MyNetworkStream.HandleOrRethrowException(Exception e)
   at MySql.Data.Common.MyNetworkStream.Read(Byte[] buffer, Int32 offset, Int32 count)
   at MySql.Data.MySqlClient.TimedStream.Read(Byte[] buffer, Int32 offset, Int32 count)
   at System.IO.BufferedStream.Read(Byte[] array, Int32 offset, Int32 count)
   at MySql.Data.MySqlClient.MySqlStream.ReadFully(Stream stream, Byte[] buffer, Int32 offset, Int32 count)
   at MySql.Data.MySqlClient.MySqlStream.LoadPacket()
   at MySql.Data.MySqlClient.MySqlStream.LoadPacket()
   at MySql.Data.MySqlClient.MySqlStream.ReadPacket()
   at MySql.Data.MySqlClient.NativeDriver.GetResult(Int32& affectedRow, Int64& insertedId)
   at MySql.Data.MySqlClient.Driver.GetResult(Int32 statementId, Int32& affectedRows, Int64& insertedId)
   at MySql.Data.MySqlClient.Driver.NextResult(Int32 statementId, Boolean force)
   at MySql.Data.MySqlClient.MySqlDataReader.NextResult()
   at MySql.Data.MySqlClient.MySqlDataReader.NextResult()
   at MySql.Data.MySqlClient.MySqlCommand.ExecuteReader(CommandBehavior behavior)
```

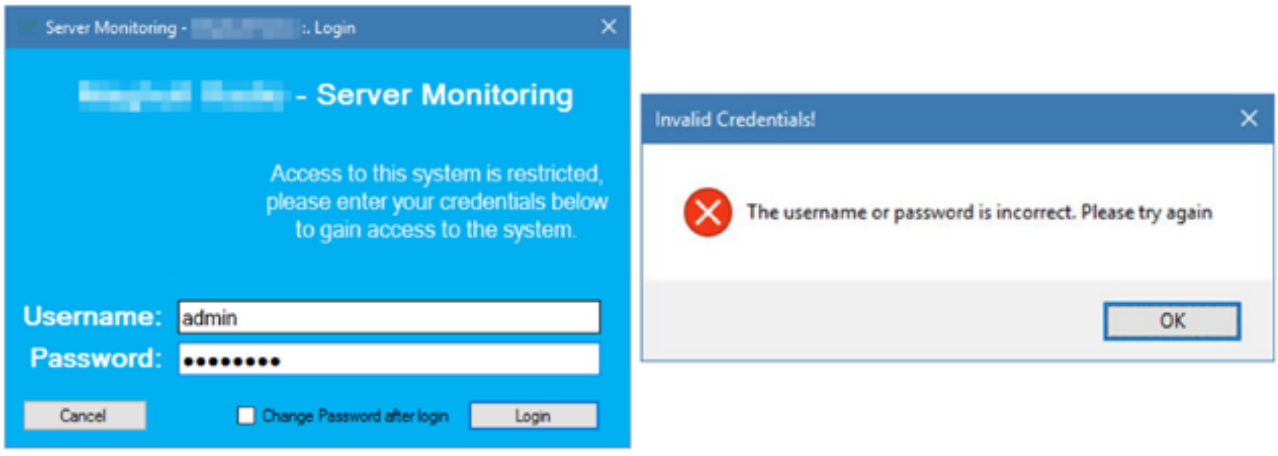
This allows for the system to fully meet this criteria, as the user will not be interrupted if errors occur that are related to the background process, and will only be informed about them when they are using the system to prevent it being intrusive.

Desirable:

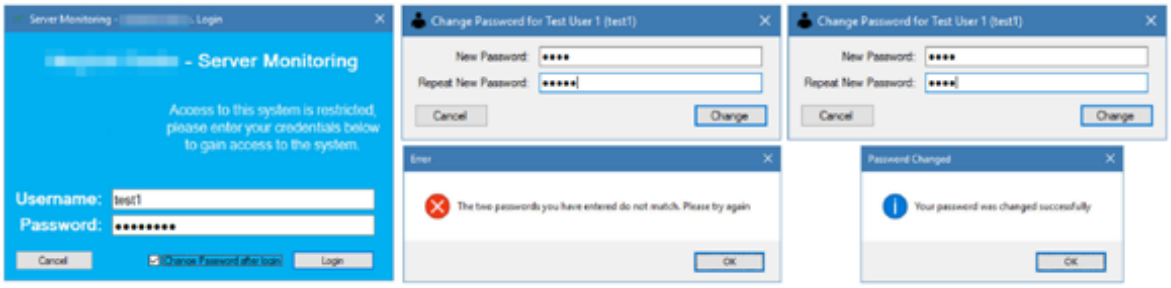
- **Login system to secure all of the data and history**

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

The system fills this criteria fully as it contains a fully functioning user system, where the user can change their password on the login screen, and can also add, edit and remove users from the system. This is shown below:
Logging in as an invalid user:

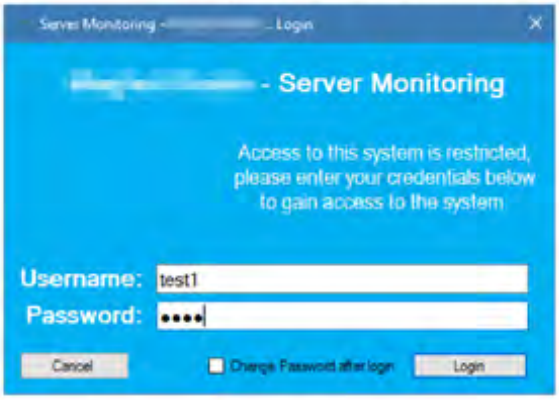


This shows that the system will only allow successful login attempts to the system.
You can also see below what happens on a valid login if the user wants to change their password:

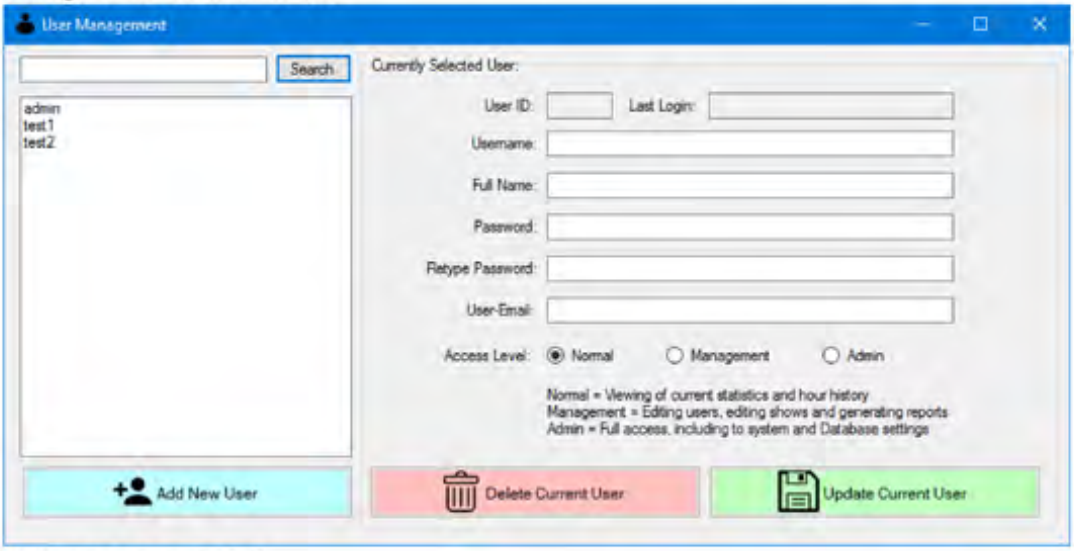


This shows that the user can change their password, and that the password is checked to make sure it has been typed correctly twice before it is updated in the database, to make sure the user has not made a typing error.

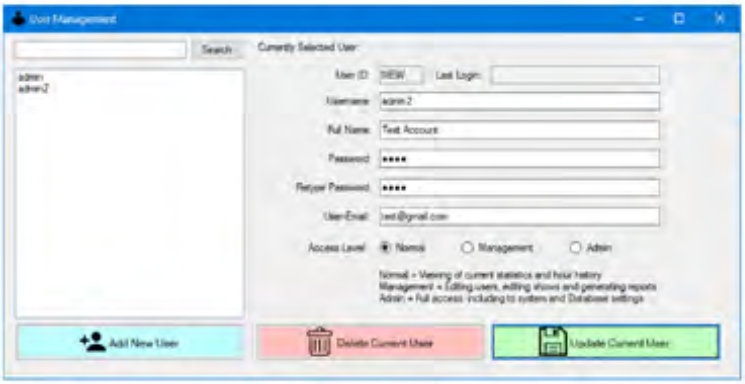
Finally, when the user successfully logs in the dashboard form will open:

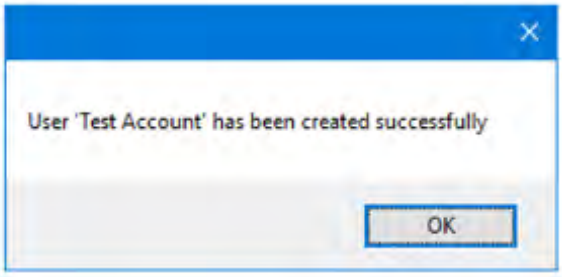


This meets the criteria of a secure system with user login, enabling users to change their password successfully. The system also meets the user management criteria as you can see from the user management form show below:



New users can be added:

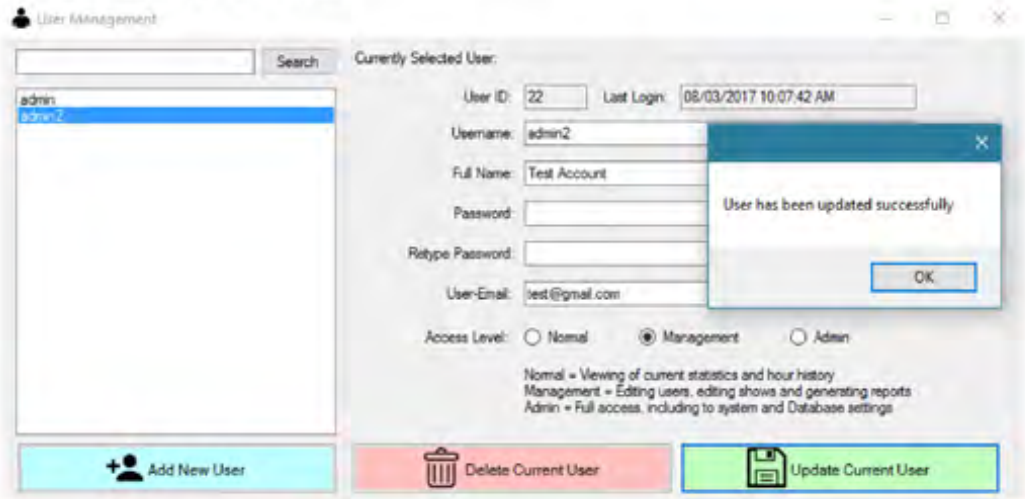




	User_FullN...	User...	User_Email	User_Password		User_LastLogin
21		admin	admin@gmail.com		3	2017-03-08 10:03:00
22	Test Account	admin2	test@gmail.com		1	(NULL)

And successfully added to the user table in the database.

Users can also be modified:



Before user was updated:

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 user was updated:

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

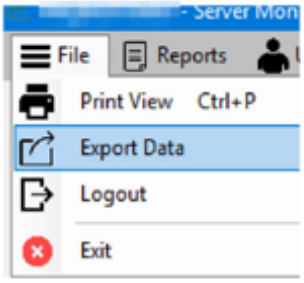
Finally user accounts can also be removed from the system as well:

This means that the system fully meets the user management criteria as well, allowing for the adding, editing and removing of user accounts. User accounts also have a individual access level, this determines what sections of the system they are allowed to access, and is used when the dashboard form first opens to disable certain features that the user will not be allowed to use.

The login system and user management systems have already been thoroughly tested during iterative Alpha testing in the development section earlier, the tests above are just to show that the system successfully meets the success criteria.

- **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.

Email's and text notifications are not a required feature, meaning that the system will still function and meet the essential criteria successfully. They were not developed due to time constraints, and while they would be nice, they would not add that much more value to the application. The data that email or text alerts would of sent can be manually obtained and sent instead, this can also be done via the 'Print View' option on the dashboard, then this file could be emailed on manually, as you can see from the screenshots below:



For emails I would of used the built in SmtpClient as part of C#, and for sending Text alerts I would of used the Twilio library that can be installed via Nuget Package Manager, using Twilio's API (located here: https://www.twilio.com/docs/libraries/csharp) to allow efficient delivery of text messages.

While the system does not meet this criteria, it wasn't imperative to the function of the system, and will be something I will work on in a future iteration of application development.

System Maintenance

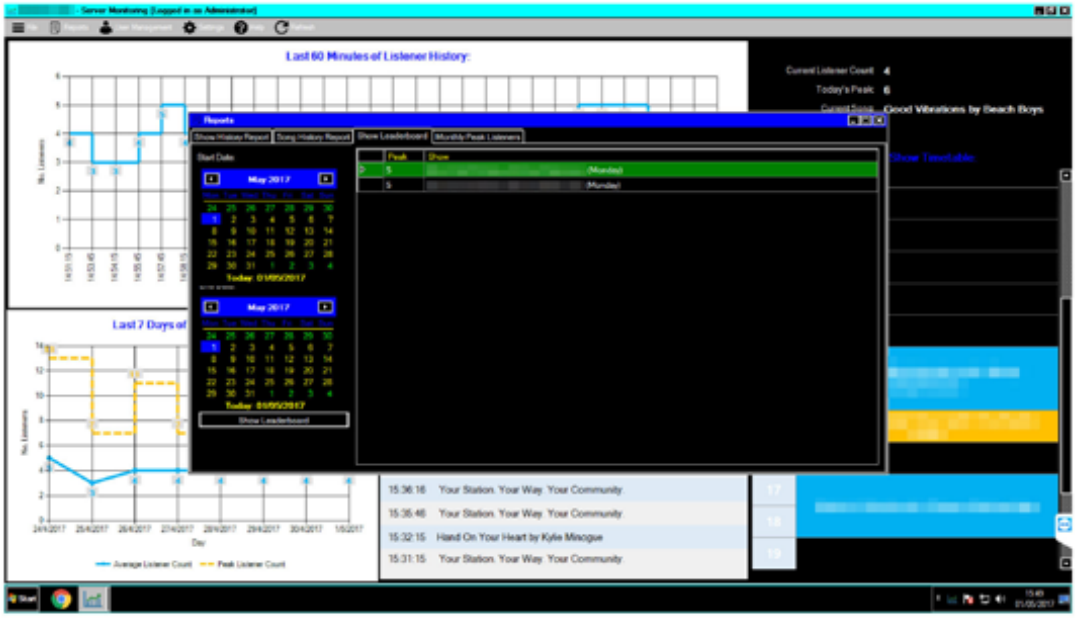
The system is very low maintenance, as it will easily run in the background on many computers without causing interference, or needing any changes. As data is constantly being recorded, with a new record being added at the most every 30 seconds. This means over a day approximately 2,880 records will be added. Currently it works out at approximately 100 bytes per record. So each day that will be 288 kilobytes per day, this will work out to 105.2 megabytes a year increase in database size. This is easily maintainable, as the average computer will have a 1 terabyte hard-drive. If 500gb are allocated to the application alone, then it could run for 4.7 thousand years, meaning there is absolutely no risk of the application running out of disk space.

Another maintenance issue could be caused by forgetting all of the usernames and passwords to the system, while the system protects against deleting all users, and making sure that there is always an account will full administrator access left on the system, the passwords could still be forgotten. There is only one way to fix the issue, firstly you will need to gain access to the database. These details would have been set when the SQL server has been setup. You would use a separate program to connect to the database then delete the users table. Then you would open the directory where the application was installed and delete the 'config.conf' file. This means that the next time the application starts it will ask for the initial configuration settings. You can fill in the same settings as before, then the system will leave the log and show tables alone, then it will create a user's table, and add an admin account with

the username and password of admin and admin. You can then login to the system normally and add user accounts again.

Accessibility

In case of users who are partially sighted, or need a high contrast colour scheme applied to the application this is possible by using the built in theme on many operating systems. The application will automatically use the default application colour sets on the PC, so when a high contrast theme is enabled, than the application will automatically use the colours from that. You can see this from the screenshots below:



The system will also support an on screen narrator, as any text on the form is part of a label. If the narrator is turned on and a label is hovered over, its content will be read out to the user. This will help when looking at results, or if a user can't read the data points on the graph, they can hover over them and they will be read out.

Overall Comments

The project as a whole has managed to meet all of the required criteria identified in the essential features section. It successfully fulfils its purpose of providing reports of listener statistics over a custom time period, and providing useful analysis on the data it holds. It is fully networkable, and scalable across multiple computers and devices. It can be really easily deployed to a working radio station environment with ease, and most importantly unobtrusively, only being visible when the user wants to run reports on certain data.

After installing the system successfully in the radio station and leaving it running for 2 months I asked for feedback from the Station Manager, Ian Porter, who I originally interviewed at the start of the project in the analysis section to identify system criteria along with essential and desirable features. He said the system was fantastic, and managed to fulfill all of his requirements that he had asked for. He said the reports it has given have proven very useful in justifying show statistics to advertisers, enabling him to show the target audience reach of adverts; along with providing useful information to use to analyse the performance of certain shows, and seeing if they are on in the right time slot. He said one of the best features is that is is hidden to normal presenters, but he can easily open it with two clicks, making it ideal for the environment of a radio station. He said he will be using the system for many years to come, and looks forward to see if there will be any features added to it in the future.

If I was to do the project again I would make sure to add email and text notifications to the system as mentioned earlier. I would also add a password reset feature via email, because currently if a user forgets their password another user with administrator or management permissions will need to login and reset their password for them. I could use the built in email SMTP Client in C# to send a temporary password that would enable the user to change theirs as soon as they login. This would prevent another user being needed to help reset a password, it would also help fix the issue if there is only one account and you forget the password, rather then going about the long reset method mentioned earlier (that took a fair bit of time, and was a complicated process). You could simply just type the temporary password in, enabling for a fast and efficient password reset, without the headache of having to setup the SQL databases again.

Finally I would also add a reporting printing feature for each individual show, this would allow for a user to print off the history view for a presenters past shows in a readable report, rather than using the current method that would be taking a screenshot of each individual graph per show, and then transferring them over to a word document. This would make it much easier to quickly get a historical overview of a certain show, and can be easily achieved with a slight modification to the current process used in generating the HTML report for the dashboard print-view report that can be generated by pressing the corresponding button in the file menu of the dashboard.

Code Appendix

Below is a copy of all of the code used in the solution pasted in a raw format.

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Server_Listener_Statistics
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Handle());
        }
    }
}
```

Handle.cs

```
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 System.Net;
using System.Diagnostics;
using MySql.Data.MySqlClient;
using System.IO;

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();
        }

        private void Handle_Load(object sender, EventArgs e)
        {
            // Check to see if settings file exists
            if (File.Exists(@"config.conf"))
            {
                // Obtain the settings from the config file
                Settings.obtainSettings();


                // Check to see if we are recording data or not
                if (!Settings.recordData)
                {
                    // We are not recording data so disable update loop
                    timerObtainStats.Enabled = false;
                }
            }
            else
            {
                // Hide the notification icon
                notifyIcon.Visible = false;

                // Disable the update timer
                timerObtainStats.Enabled = false;

                // Show a new settings
                EditSettings editSettings = new EditSettings();
                editSettings.ShowDialog();

            }

        }
```

```
        private void timerObtainStats_Tick(object sender, EventArgs e)
        {
            // Create a try loop so if the code errors out the application will not crash
            try
            {
                // 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 charachters, so it will split a
string by a string delimeter
                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]);


                // 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});";
                        command.Parameters.AddWithValue("@currentsong", currentSong);

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

                        //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 characters long
                        if (notifyText.Length > 63)
                        {
                            // The text is too long, we need the first 60 characters, and
add on an ellipsis 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;

                    }

                }

                // 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();
            }
            catch (Exception ex)
            {
                // No handling needed, as loop will run again in 30 seconds
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();
            }

    }

        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();
```

```
            }
        }
    }
```

Dashboard.cs

```
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;
using System.Text.RegularExpressions;
using System.Windows.Forms.DataVisualization.Charting;
using System.Drawing.Imaging;
using System.Diagnostics;
using System.IO;

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

        public Dashboard()
        {
            InitializeComponent();
        }

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

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

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

                // Add hours to timetable
                tableLayoutSchedule.SuspendLayout();
                addHourMarks();
                tableLayoutSchedule.ResumeLayout();

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

                // Update the labels
                colourSongLabels();

                // Setup week graph
                setupWeekGraph();

                // Setup hour graph
                setupHourGraph();

                // Update dashboard
                updateDashboard();

                // Enforce access restrictions
                updateAccessRestrictions();



            }
            catch (Exception ex)
            {
```

```
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Application.Exit();
            }


        }

private void flowLayoutPanelScheduleContainer_Resize(object sender, EventArgs e)
        {
            // When the container for the daily schedule is resized, update the width of
the schedule tableLayoutPanel
            tableLayoutSchedule.Width = flowLayoutPanelScheduleContainer.Width - 20;

        }

        private void timerUpdateTime_Tick(object sender, EventArgs e)
        {
            // Update the time label on the form with the current time
            lblCurrentTime.Text = DateTime.Now.ToString();
        }

        // Show timetable

        // 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 margin 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 Microsoft 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 column in
the row of the hour it represents
            tableLayoutSchedule.Controls.Add(lblTitle, 0, i);

        }
    }

    // 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 margin 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 Microsoft 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);

    }


// Update show timetable
    public void updateTimetable()
    {
        try
        {

            // 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", "");

                    //
                    string ShowPresenter = reader["Show_Presenter"].ToString();
                    int ShowStartTime = int.Parse(reader["Show_Starttime"].ToString());
                    int ShowEndTime = int.Parse(reader["Show_Endtime"].ToString());

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

                    if ((currentHour >= ShowStartTime) && (currentHour <= ShowEndTime -
1)) // 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
                    {
                        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();

            }
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
                // Close the application
                Application.Exit();
            }

        }

        // Update songs played
        public void updatePastSongs()
        {
            try
            {

                // RUN query "SELECT Log_DateTime, Log_CurrentSong FROM log GROUP BY
Log_CurrentSong ORDER BY Log_DateTime DESC LIMIT 10;"
                MySqlCommand command = mySqlConn.CreateCommand();
                command.CommandText = $"SELECT TIME(Log_DateTime) AS Log_Time,
Log_CurrentSong FROM log ORDER BY Log_DateTime DESC LIMIT 10;";

                IDataReader reader = command.ExecuteReader();

                // Cycle through each record returned
                if (reader.Read())
                {
                    // Set title of current song
                    lblCurrentSong.Text = reader["Log_CurrentSong"].ToString();

                    // Loop through all of the labels on the table and update them
                    for (int i = 0; i < 10; i++)
                    {
                        tableLayoutPanelPast10Songs.Controls.Find($"lblPastSongsTime{i}",
false)[0].Text = reader["Log_Time"].ToString();
                        tableLayoutPanelPast10Songs.Controls.Find($"lblPastSongName{i}",
false)[0].Text = reader["Log_CurrentSong"].ToString();
                        reader.Read();
                    }
                }

                reader.Close();

            }
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Application.Exit();
            }


        }
// Add labels to past songs
        private void colourSongLabels()
        {
            tableLayoutPanelPast10Songs.SuspendLayout();

            // Cycle through each show time
            for (int i = 0; i < 10; i++)
            {
```

```
                if (i == 0)
                {
                    tableLayoutPanelPast10Songs.Controls.Find($"lblPastSongsTime{i}",
false)[0].BackColor = Settings.colours.Orange;
                }
                else if(i % 2 == 0) // If the value is even colour in light blue
                {
                    tableLayoutPanelPast10Songs.Controls.Find($"lblPastSongsTime{i}",
false)[0].BackColor = Settings.colours.LightBlue;
                }
                else    // Otherwise if odd set the value to light grey
                {
                    tableLayoutPanelPast10Songs.Controls.Find($"lblPastSongsTime{i}",
false)[0].BackColor = Settings.colours.LightGrey;
                }
            }

            // Cycle through each show name
            for (int i = 0; i < 10; i++)
            {
                if (i == 0)
                {
                    tableLayoutPanelPast10Songs.Controls.Find($"lblPastSongName{i}",
false)[0].BackColor = Settings.colours.Orange;
                }
                else if (i % 2 == 0) // If the value is even colour in light blue
                {
                    tableLayoutPanelPast10Songs.Controls.Find($"lblPastSongName{i}",
false)[0].BackColor = Settings.colours.LightBlue;
                }
                else    // Otherwise if odd set the value to light grey
                {
                    tableLayoutPanelPast10Songs.Controls.Find($"lblPastSongName{i}",
false)[0].BackColor = Settings.colours.LightGrey;
                }
            }

            tableLayoutPanelPast10Songs.ResumeLayout();
        }

        // Update week graph
        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;

        }

    private void updateWeekGraph()
        {
            try
            {

                // 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();

            }
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
```

```
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Application.Exit();
            }

        }

        // Update Hour Graph
        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;

        }


  private void updateHourGraph()
        {
            try
            {

                // 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();

            }
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Application.Exit();
            }

        }

        // Update current and peak listener count
        private void updateCurrentAndPeakListeners()
        {
            try
            {

                // 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();

            }
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Application.Exit();
            }

        }


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

            // Scroll to current hour, if not it's not important (hence the try catch
loop)
            try
            {

flowLayoutPanelScheduleContainer.ScrollControlIntoView(tableLayoutSchedule.Controls.Find(
$"lblScheduleHour{DateTime.Now.Hour + 4}", false)[0]);
            }
            catch
            {
                // No need to correct anything
            }
        }

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

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

        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();
        }

        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();

        }

        private void updateAccessRestrictions()
        {
            // Limit certain functionalities of the system
            if (Settings.accessLevel == 3)
            {
                exitToolStripMenuItem.Visible = true;
                reportsToolStripMenuItem.Visible = true;
                userManagementToolStripMenuItem.Visible = true;
                settingsToolStripMenuItem.Visible = true;
            }
            else if(Settings.accessLevel == 2)
            {
                exitToolStripMenuItem.Visible = false;
                reportsToolStripMenuItem.Visible = true;
                userManagementToolStripMenuItem.Visible = false;
                settingsToolStripMenuItem.Visible = false;
            }
            else
            {
                exitToolStripMenuItem.Visible = false;
                reportsToolStripMenuItem.Visible = false;
                userManagementToolStripMenuItem.Visible = false;
                settingsToolStripMenuItem.Visible = false;
            }

        }

        private void printViewToolStripMenuItem_Click(object sender, EventArgs e)
        {
            // Generate a report and show the print preview dialogue
            generateReport(true);
        }

        private void generateReport(bool printDocument)
        {
            // Create images of both charts
            weekGraph.SaveImage(@"last7days.png", ImageFormat.Png);
            hourGraph.SaveImage(@"last60mins.png", ImageFormat.Png);

            // Create a new bitmap image, and add the contents of the tableLayoutPanel
containing the past 10 songs to it
            using(Bitmap printImage = new Bitmap(tableLayoutPanelPast10Songs.Width,
tableLayoutPanelPast10Songs.Height))
            {
                tableLayoutPanelPast10Songs.DrawToBitmap(printImage, new Rectangle(0, 0,
printImage.Width, printImage.Height));
                // Save the image to a file
                printImage.Save(@"past10songs.png");
            }

            // Create a string to hold the JavaScript if the user wants a print preview
dialogue
            string printJavascript = "";

            // Check to see if the user wants to have an automatic print dialogue
```

```
            if (printDocument)
            {
                printJavascript = "<script>window.print();</script>";
            }

            // Create html file and save it to 'report.html'
            StreamWriter sw = new StreamWriter(@"report.html");
            sw.WriteLine($"<html> <head> {printJavascript} <title>Server Listener
Statistics Report</title> </head> <body style='font-family: Arial;'> <center> <h1>Server
Listener Statistics Report</h1> <hr/> <h2>{lblCurrentTime.Text}</h2> <p>Current Listener
Count: <b>{lblCurrentListenerCount.Text}</b></p> <p>Today's Peak:
<b>{lblPeakListenerCount.Text}</b></p> <p>Current Song: <b>{lblCurrentSong.Text}</b></p>
<hr/> <img src=\"./past10songs.png\"></img><br/><br/> <img
src=\"./last7days.png\"></img><br/><br/> <img src=\"./last60mins.png\"></img><br/><br/>
<hr> </center> </body> </html>");
            sw.Close();

            // Open the report in the user's default web-browser
            string path = System.Environment.CurrentDirectory + "/report.html";
            Process.Start(path);

        }

        private void exportDataToolStripMenuItem_Click(object sender, EventArgs e)
        {
            // Generate a report, but do not show print preview dialogue
            generateReport(false);
        }

        private void logoutToolStripMenuItem_Click(object sender, EventArgs e)
        {
            // Close the current form
            this.Close();
        }

        private void exitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            // Check to see if user really wants to close the application
            DialogResult result = MessageBox.Show("Closing the application will prevent
any more data from being recorded until you open it again.\nAre you sure you want to
close? ", "Are you sure you want to close?", MessageBoxButtons.YesNo,
MessageBoxIcon.Warning, MessageBoxDefaultButton.Button2);

            // Close the application if yes was pressed
            if (result == DialogResult.Yes)
            {
                Application.Exit();
            }
        }

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

Settings.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Server_Listener_Statistics
{
    class Settings
    {
        // Define local private variables
        // Database Settings
        private static string dbConnectionStringLocal = "";

        // Radio Server Settings
        private static string serverHostLocal = "";
        private static string serverPortLocal = "";
        private static string serverVersionLocal = "";
        private static string serverIDLocal = "";
        private static string serverUsernameLocal = "";
        private static string serverPasswordLocal = "";

        // General Radio Settings
        private static string radioNameLocal = "";
        private static int radioStartHourLocal = 0;
        private static int radioEndHourLocal = 23;

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

        // Application data recording settings
        private static bool recordDataLocal = true;


        // Define global public read only variables
        // Database Settings
        public static string dbConnectionString { get { return dbConnectionStringLocal; }
}

        // Radio Server Settings
        public static string serverHost { get { return serverHostLocal; } }
        public static string serverPort { get { return serverPortLocal; } }
        public static string serverVersion { get { return serverVersionLocal; } }
        public static string serverID { get { return serverIDLocal; } }
        public static string serverUsername { get { return serverUsernameLocal; } }
        public static string serverPassword { get { return serverPasswordLocal; } }
        public static string serverConnectionURL { get { return
$"http://{serverHostLocal}:{serverPortLocal}/admin.cgi?mode=viewxml&pass={serverPasswordL
ocal}"; } }

        // General Radio Settings
        public static string radioName { get { return radioNameLocal; } }
        public static int radioStartHour { get { return radioStartHourLocal; } }
        public static int radioEndHour { get { return radioEndHourLocal; } }

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

        // 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);
        }

        // Application data recording settings
        public static bool recordData { get { return recordDataLocal; } }


        //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 variable 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];

                // Set data recording bool
                recordDataLocal = bool.Parse(sr.ReadLine());

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


        // 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;
        }

        // 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);
        }
    }
}
```

Hash.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Server_Listener_Statistics
{
    class Hash
    {
        // Hash password
        public static string hashPassword(string userPassword)
        {
            // This will return a hash (with a salt applied) which can then be stored
into a database
            return BCrypt.Net.BCrypt.HashPassword(userPassword,
BCrypt.Net.BCrypt.GenerateSalt(12));
        }


        // Verify password
        public static bool verifyPassword(string hash, string userPassword)
        {
            // This will verify the user's entered password by extrapolating the salt
from the hash,
            // and then hashing the user's password to make sure both of them match
            return BCrypt.Net.BCrypt.Verify(userPassword, hash);
        }

    }
}
```

LoginForm.cs

```
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;
using System.IO;

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)
        {
            try
            {

                // Open the connection to the database when the form loads
                mySqlConn.Open();

            }
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Application.Exit();
            }


            //txtUsername.Text = Hash.hashPassword("password");

        }

        private void btnLogin_Click(object sender, EventArgs e)
        {
            try
            {

                // 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` =
@username";
                command.Parameters.AddWithValue("@username", 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();

            }
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Application.Exit();
            }


        }

        private void btnExit_Click(object sender, EventArgs e)
        {
            // Close the current form
            this.Close();
        }
    }
}
```

ChangePassword.cs

```
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;
using System.IO;

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

        public ChangePassword()
        {
            InitializeComponent();
        }

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

            try
            {
                // Open connection to the database
                mySqlConn.Open();
            }
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Application.Exit();
            }


        }

        private void btnChange_Click(object sender, EventArgs e)
        {

            try
            {

                // Check to make sure the password the user entered is the same as the
repeat password
                if (txtNewPassword.Text == txtRepeatNewPassword.Text)
                {
                    // another comment
                    // Both of the passwords the user has entered 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);

                }

            }
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Application.Exit();
            }


        }

        private void ChangePassword_FormClosing(object sender, FormClosingEventArgs e)
        {
            // Close connection to the database
            mySqlConn.Close();
        }
    }
}
```

EditSettings.cs

```
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;
using System.Net;
using System.IO;

namespace Server_Listener_Statistics
{
    public partial class EditSettings : Form
    {
        public EditSettings()
        {
            InitializeComponent();
        }

        private void btnCancel_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void EditSettings_Load(object sender, EventArgs e)
        {
            // Check to see if it is an initial start-up
            if (File.Exists(@"config.conf"))
            {
                // Update settings from the settings class
                updateSettingsFromClass();
            }
            else
            {
                // Display a message to let the use know it is a first time setup
                MessageBox.Show("As this is a first time setup. Please enter the settings
to setup the application.");
            }

        }

        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;

            // Obtain record data details
            cbRecordData.Checked = Settings.recordData;

        }

        //Test SQL server
        private bool testSqlServer(bool returnMessageOnTrue)
        {
            // Define MySQL Connection details
            MySqlConnection mySqlConn = new
MySqlConnection($"Server={txtDatabaseServerHost.Text};Port={txtDatabaseServerPort.Text};U
ser 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;
            }
        }

        private void btnDatabaseServerTestConnection_Click(object sender, EventArgs e)
        {
            // Run the code to test the sql structure
            checkSqlStructure(true);
        }

        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};U
ser 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();
            }
        }


        private bool checkSqlStructure(bool showMessageOnTrue)
        {
            // Test to make sure server connection is valid, don't display a message if
the connection is successful
            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
                        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();

                        // 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");

                    }
                    // 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;
            }
        }

        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;
            }
        }


      private void btnServerTestConnection_Click(object sender, EventArgs e)
        {
            testServerConnection(true);
        }

        // 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.WriteLine(Settings.EncryptToBase64($"Version={cbServerVersion.Text};Host={txtServerHos
t.Text};Port={txtServerPort.Text};ID={numServerStreamID.Value.ToString()};Username={txtSe
rverAdminUser.Text};Password={txtServerAdminPassword.Text}"));
                        sw.Write(cbRecordData.Checked.ToString());

                        // 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! The
application will now restart.");
                        Application.Restart();
                }
                else
                {
                    // The user hasn't entered a name for the radio station
                    MessageBox.Show("Please enter a radio station name");
                }

        }

        private void btnSave_Click(object sender, EventArgs e)
        {
            saveSettings();
        }

        private void EditSettings_FormClosing(object sender, FormClosingEventArgs e)
        {
            // Check to see if it's a first time setup
            if (File.Exists("config.conf") == false)
            {
                // First time setup so warn user of closing
                DialogResult result = MessageBox.Show("As you have not saved valid
settings the next time you open the application this initial settings dialogue will open
again. Are you sure you wish to close?", "Closing without valid settings?",
MessageBoxButtons.YesNo, MessageBoxIcon.Warning, MessageBoxDefaultButton.Button2);

                // If the user didn't want to close, then cancel the closing event
                if (result == DialogResult.No)
                {
                    e.Cancel = true;
                }
                else
                {
                    // If the user did want to close, forcibly quit the application
                    Environment.Exit(0);
                }

            }
        }
    }
}
```

UserManagement.cs

```
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;
using System.IO;

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

        public UserManagement()
        {
            InitializeComponent();
        }

        private void UserManagement_Load(object sender, EventArgs e)
        {
            try
            {

                // 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();

            }
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Application.Exit();
            }
```

```
        }

        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` =
@user";
                command.Parameters.AddWithValue("@user",
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
            }
        }
```

```
        private void listBoxUsers_MouseClick(object sender, MouseEventArgs e)
        {


        }

        private void btnSearch_Click(object sender, EventArgs e)
        {
            try
            {

                // 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
@query OR `User_FullName` LIKE @query ORDER BY `User_LoginName` ASC";
                command.Parameters.AddWithValue("@query", $"%{txtQuery.Text}%");

                // 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();

            }
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Application.Exit();
            }

        }


        private void btnDeleteCurrentUser_Click(object sender, EventArgs e)
        {
            try
            {

                // 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" && txtUserID.Text != "")
                {
                    // Check to make sure that there is more than 1 user in the listbox
                    if (listBoxUsers.Items.Count > 1)
                    {
```

```
                        // There is more than one user, so go ahead and delete the
current user
                        // 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
                    {
                        // 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.");
                    }


                }
                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.");
                }

            }
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
```

```
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Application.Exit();
            }

        }


// Code to update the current user
        private void btnUpdateCurrentUser_Click(object sender, EventArgs e)
        {
            try
            {

                // 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 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");
                }



                // 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");
        }

    }


}
catch (Exception ex)
{
    // Write error to log file
    StreamWriter sw = File.AppendText(@"./log.txt");
    sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
    sw.Close();

    // Display error message to user
    MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

    // Close the application
    Application.Exit();
}


}
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 = "";
        }
    }
}
```

Reports.cs

```
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;
using System.Text.RegularExpressions;
using System.Windows.Forms.DataVisualization.Charting;
using System.Globalization;
using System.IO;

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

        public Reports()
        {
            InitializeComponent();
        }

        private void Reports_Load(object sender, EventArgs e)
        {
            try
            {

                // Connect to the database
                mySqlConn.Open();

            }
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Application.Exit();
            }


            //obtainShowData(11);

            setupShowHistoryGraph();

            // Setup the max and min date ranges on the Song History report
            monthCalendarStartDate.MaxDate = DateTime.Now;
            monthCalendarEndDate.MaxDate = DateTime.Now;

            // Set the default values for the combo boxes for start and end hours
            cbStartHour.SelectedIndex = 0;
            cbEndHour.SelectedIndex = 23;

            // Update the monthly report month ranges
```

```
            updatePeakMonthRanges();

            // Setup month peak graph
            setupMonthlyPeakGraph();

        }

        // Show History Report

        // List shows from DB
        private void listShows(string query)
        {
            try
            {

                // 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();

            }
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Application.Exit();
            }

        }

        // Obtain show data
        private void obtainShowData(int showID)
        {
            try
            {

                // Create a command to obtain show data
                MySqlCommand command = mySqlConn.CreateCommand();
                command.CommandText = $"SELECT Show_ID, Show_Name, Show_Presenter,
Show_Day, Show_Starttime, Show_Endtime, DATE(Show_StartDate) AS 'Show_StartDate',
```

```
DATE(Show_EndDate) AS 'Show_EndDate', Show_Active 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();
                    // Split the date to just display the date and drop the time off the
end
                    lblStartDate.Text = reader["Show_StartDate"].ToString().Split('
')[0];
                    lblEndDate.Text = reader["Show_EndDate"].ToString().Split(' ')[0];

                    // 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";
                    }


                }

                // 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();

                // Select the last item in the combo box
                comboBoxDate.SelectedIndex = comboBoxDate.Items.Count - 1;

                // check to see if there are any values in the combo box
                if (comboBoxDate.Items.Count == 0)
                {
                    //Clear graph so data can be plotted
                    chartShowHistory.Series[0].Points.Clear();

                    // Set title to say no data could be loaded
                    chartShowHistory.Titles[0].Visible = true;

                    // hide the cursor
                    chartShowHistory.ChartAreas[0].CursorX.IsUserSelectionEnabled =
false;
                    chartShowHistory.ChartAreas[0].CursorY.LineWidth = 0;

                    // disable the show navigation buttons
                    btnNextShow.Enabled = false;
                    btnPreviousShow.Enabled = false;
                }
                else
                {
                    // Hide the title on the graph as it is not needed
                    chartShowHistory.Titles[0].Visible = false;

                    // Enable the cursors again
                    chartShowHistory.ChartAreas[0].CursorX.IsUserSelectionEnabled = true;
                    chartShowHistory.ChartAreas[0].CursorY.LineWidth = 2;
                }

            }
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Application.Exit();
            }


        }

private void btnSearch_Click(object sender, EventArgs e)
        {
            // List shows based off the query entered in the search box
            listShows(txtQuery.Text);
        }

        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();
            }
        }
```

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

            // Wipe current data source data
            dataGridView1.DataSource = null;

            int showid;

            // Check to make sure an index is selected
            if (int.TryParse(listBoxShows.Text.Split('-')[0].Trim(' '), out showid))
            {
                // Obtain show data for the current selected show
                obtainShowData(showid);
            }



            // Reset the zoom level on the chart, needs a while loop
            // as there is no way of knowing how many times the chart was zoomed in
             while (chartShowHistory.ChartAreas[0].AxisX.ScaleView.IsZoomed)
             {
                 chartShowHistory.ChartAreas[0].AxisX.ScaleView.ZoomReset();
             }
        }


        private void obtainShowStats(int showID, int startHour, int endHour)
        {
            try
            {

                // Create new query to obtain show history data
                MySqlCommand command = mySqlConn.CreateCommand();

                // Convert date to SQL form
                String date = DateTime.Parse(comboBoxDate.Text).ToString("yyyy-MM-dd");

                // Create command to query database
                command.CommandText = $"SELECT TIME(Log_DateTime) AS 'Time',
ROUND(AVG(Log_ListenerCount), 0) AS 'Count', Log_CurrentSong AS 'Current Song' FROM log
WHERE Log_ShowID = {showID} AND Log_DateTime between \"{date} {startHour}:00:00\" and
\"{date} {endHour - 1}:59:59\" GROUP BY MINUTE(log.Log_DateTime) ORDER BY Log_DateTime
ASC;";

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

                // Create a data table to hold the data so it can be displayed in the
data grid view
                DataTable dt = new DataTable();
                dt.Load(reader);

                // Assign data source to table
                dataGridView1.DataSource = dt;

                // Set 3rd column to expand
                dataGridView1.Columns[2].AutoSizeMode =
DataGridViewAutoSizeColumnMode.Fill;

                //Clear graph so data can be plotted
                chartShowHistory.Series[0].Points.Clear();

                // Create dictionary average to hold data to plot onto graph
                Dictionary<string, int> dictionaryAverage = new Dictionary<string,
int>();

                // Data reader from data table
                reader = dt.CreateDataReader();
```

```
                // Loop through all results returned and add them to the dictionary
                while (reader.Read())
                {
                    dictionaryAverage.Add(reader["Time"].ToString(),
int.Parse(reader["Count"].ToString()));
                }

                // Create a MAX integer to hold the peak count for the show
                int max = 0;

                // Cycle through each item in the dictionary and add it to the graph
                foreach (KeyValuePair<string, int> point in dictionaryAverage)
                {
                    chartShowHistory.Series[0].Points.AddXY(point.Key, point.Value);

                    // update the max count so it holds the peak value so far
                    max = Math.Max(max, point.Value);
                }

                // Set this value as the YCursor position on the graph
                chartShowHistory.ChartAreas[0].CursorY.Position = max;

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

            }
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Application.Exit();
            }

        }

        private void comboBoxDate_SelectedIndexChanged(object sender, EventArgs e)
        {
            // Check to see if there are any shows prior or after to enable the
respective buttons
            if(comboBoxDate.Items.Count < 2)
            {
                // If there are 0 or 1 show, then there is nowhere to navigate to, so
disable both buttons
                btnNextShow.Enabled = false;
                btnPreviousShow.Enabled = false;
            }
            else if (comboBoxDate.SelectedIndex == comboBoxDate.Items.Count - 1)
            {
                // If the last item in the listbox is selected, disable the next show
button and enable the previous button
                btnNextShow.Enabled = false;
                btnPreviousShow.Enabled = true;
            }
            else if(comboBoxDate.SelectedIndex == 0)
            {
                // If the first index is selected enable the next show button, but
disable the previous show button
```

```
                btnNextShow.Enabled = true;
                btnPreviousShow.Enabled = false;
            }
            else
            {
                // Either direction is fine as there is a show above and below the
current point so enable both buttons
                btnNextShow.Enabled = true;
                btnPreviousShow.Enabled = true;
            }


            // Obtain show stats for current show
            obtainShowStats(int.Parse(listBoxShows.Text.Split('-')[0].Trim(' ')),
int.Parse(lblStartHour.Text), int.Parse(lblEndHour.Text));
        }

        private void setupShowHistoryGraph()
        {
            // Setup X Axis
            chartShowHistory.ChartAreas[0].Axes[0].Title = "Time";
            chartShowHistory.ChartAreas[0].Axes[0].IsLabelAutoFit = true;
            chartShowHistory.ChartAreas[0].Axes[0].Interval = 1;
            chartShowHistory.ChartAreas[0].Axes[0].IsMarginVisible = false;

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

            // Setup Y Axis
            chartShowHistory.ChartAreas[0].Axes[1].Title = "No. Listeners";
            chartShowHistory.ChartAreas[0].Axes[1].IsLabelAutoFit = true;
            chartShowHistory.ChartAreas[0].Axes[1].LabelAutoFitStyle =
LabelAutoFitStyles.LabelsAngleStep30;
            chartShowHistory.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
            chartShowHistory.Series[0].ChartType = SeriesChartType.StepLine;
            chartShowHistory.Series[0].LegendText = "Listener Count";
            chartShowHistory.Series[0].BorderWidth = 4;
            chartShowHistory.Series[0].Color = Settings.colours.Blue;
            chartShowHistory.Series[0].LabelForeColor = Settings.colours.Blue;

            //Add selection and peak line setup
            chartShowHistory.ChartAreas[0].CursorX.IsUserSelectionEnabled = true;
            chartShowHistory.ChartAreas[0].CursorY.LineWidth = 2;
            chartShowHistory.ChartAreas[0].CursorY.LineDashStyle = ChartDashStyle.Dash;

            // Add title for no data
            chartShowHistory.Titles.Add("No data for selected show...");
            chartShowHistory.Titles[0].Visible = false;

        }

        private void btnPreviousShow_Click(object sender, EventArgs e)
        {
            // Move backwards one show (change the index of the combo box)
            comboBoxDate.SelectedIndex = comboBoxDate.SelectedIndex - 1;

        }

        private void btnNextShow_Click(object sender, EventArgs e)
        {
            // Move forwards one show (change the index of the combo box)
            comboBoxDate.SelectedIndex = comboBoxDate.SelectedIndex + 1;
        }

        private void tabPage2_Click(object sender, EventArgs e)
        {
```

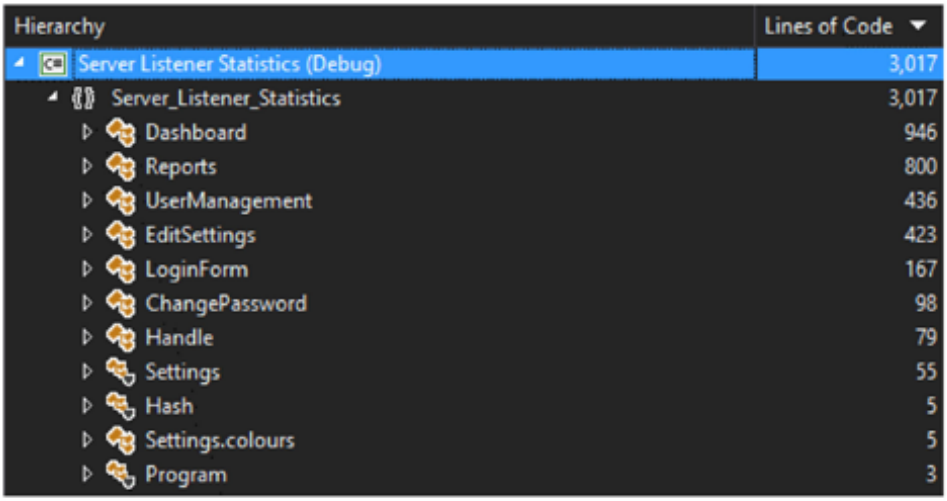
```
        }


        // Song history section
        private void btnViewData_Click(object sender, EventArgs e)
        {
            try
            {

                // Create new query to obtain song history data
                MySqlCommand command = mySqlConn.CreateCommand();

                // Create command to query database
                command.CommandText = $"SELECT Log_DateTime AS 'Date/Time',
Log_CurrentSong AS 'Song Name',Log_ListenerCount AS 'Listeners', (SELECT
CONCAT(Show_Name, ' with ',Show_Presenter) FROM shows WHERE Show_ID=Log_ShowID) AS
'Current Show', Log_ShowID AS 'Show ID' FROM log WHERE Log_DateTime between
\"{monthCalendarStartDate.SelectionStart.ToString("yyyy-MM-dd")}
{cbStartHour.Text.ToString()}:00:00\" and
\"{monthCalendarEndDate.SelectionStart.ToString("yyyy-MM-dd")}
{int.Parse(cbEndHour.Text.ToString()) - 1}:59:59\" AND Log_CurrentSong LIKE @query ORDER
BY Log_DateTime ASC;";
                command.Parameters.AddWithValue("@query",
$"%{txtSongQuery.Text.ToString()}%");

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


                // Create a data table to hold the data so it can be displayed in the
data grid view
                DataTable dt = new DataTable();
                dt.Load(reader);

                // Assign data source to table
                dataGridViewSongHistory.DataSource = dt;

                // Adjust data grid view columns
                dataGridViewSongHistory.Columns[0].AutoSizeMode =
DataGridViewAutoSizeColumnMode.AllCells;
                dataGridViewSongHistory.Columns[1].AutoSizeMode =
DataGridViewAutoSizeColumnMode.Fill;
                dataGridViewSongHistory.Columns[2].AutoSizeMode =
DataGridViewAutoSizeColumnMode.AllCells;
                dataGridViewSongHistory.Columns[3].AutoSizeMode =
DataGridViewAutoSizeColumnMode.AllCells;
                dataGridViewSongHistory.Columns[4].Visible = false;

                // Check to see if the current show needs to be displayed
                if (!checkBoxDisplayShows.Checked)
                {
                    dataGridViewSongHistory.Columns[3].Visible = false;
                }
                else
                {
                    dataGridViewSongHistory.Columns[3].Visible = true;
                }

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

            }
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
```

```
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Application.Exit();
            }


        }

        private void monthCalendarStartDate_DateChanged(object sender, DateRangeEventArgs
e)
        {
            // Set the minimum date on the end date to the current selected date
            monthCalendarEndDate.MinDate = monthCalendarStartDate.SelectionStart;
        }

        // Show leader board
        private void monthCalendarShowLeaderboardStart_DateChanged(object sender,
DateRangeEventArgs e)
        {
            // Set the minimum date on the end date to the current selected date
            monthCalendarShowLeaderboardEnd.MinDate =
monthCalendarShowLeaderboardStart.SelectionStart;
        }

        private void btnShowLeaderBoard_Click(object sender, EventArgs e)
        {
            try
            {

                // Create new query to obtain song history data
                MySqlCommand command = mySqlConn.CreateCommand();

                // Create command to query database
                // {monthCalendarEndDate.SelectionStart.ToString("yyyy-MM-dd")}
                //{monthCalendarStartDate.SelectionStart.ToString("yyyy-MM-dd")}
                command.CommandText = $"SELECT MAX(Log_ListenerCount) AS 'Peak', (SELECT
CONCAT(Show_Name, ' with ',Show_Presenter,' (',Show_Day,')') FROM shows WHERE
Show_ID=Log_ShowID) AS 'Show' FROM log WHERE DATE(Log_DateTime) BETWEEN
\"{monthCalendarShowLeaderboardStart.SelectionStart.ToString("yyyy-MM-dd")}\" and
\"{monthCalendarShowLeaderboardEnd.SelectionStart.ToString("yyyy-MM-dd")}\" AND
Log_ShowID IS NOT NULL GROUP BY Log_ShowID ORDER BY Peak DESC";

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


                // Create a data table to hold the data so it can be displayed in the
data grid view
                DataTable dt = new DataTable();
                dt.Load(reader);

                // Assign data source to table
                dataGridViewShowLeaderboard.DataSource = dt;

                // Adjust data grid view columns so the show name fills all of the space,
after the listener count has been set as small as possible
                dataGridViewShowLeaderboard.Columns[0].AutoSizeMode =
DataGridViewAutoSizeColumnMode.AllCells;
                dataGridViewShowLeaderboard.Columns[1].AutoSizeMode =
DataGridViewAutoSizeColumnMode.Fill;

                // Close the reader so it can be used again
```

```
                reader.Close();

            }
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Application.Exit();
            }

        }

        private void button1_Click(object sender, EventArgs e)
        {

        }

        // Month peak listeners
        private void updatePeakMonthRanges()
        {
            try
            {

                // Update the peak ranges
                // Clear the values currently in the combo boxes
                comboBoxStartMonth.Items.Clear();
                comboBoxEndMonth.Items.Clear();

                // Create new query to obtain the ranges of dates available
                MySqlCommand command = mySqlConn.CreateCommand();
                command.CommandText = $"SELECT CONCAT(MONTHNAME(Log_DateTime), ' ',
YEAR(Log_DateTime)) AS 'Months' FROM log GROUP BY MONTH(Log_DateTime) ORDER BY
Log_DateTime ASC";

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

                // Loop through each result and add them to the combo boxes
                while (reader.Read())
                {
                    comboBoxStartMonth.Items.Add(reader[0].ToString());
                    comboBoxEndMonth.Items.Add(reader[0].ToString());
                }

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

                // Select the first item in the list for the start month, and the last
one for the end month
                comboBoxStartMonth.SelectedIndex = 0;
                comboBoxEndMonth.SelectedIndex = comboBoxEndMonth.Items.Count - 1;

            }
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
```

```
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Application.Exit();
            }


        }

 private void btnMonthlyPeakListenersShow_Click(object sender, EventArgs e)
        {
            try
            {

                // Create variables to hold the start and end dates
                string startdate = "";
                string enddate = "";

                // Get start and end dates
                startdate = comboBoxStartMonth.SelectedItem.ToString().Split(' ')[1] + "-
" +
(DateTimeFormatInfo.CurrentInfo.MonthNames.ToList().IndexOf(comboBoxStartMonth.SelectedIt
em.ToString().Split(' ')[0]) + 1).ToString() + "-1";
                enddate = comboBoxEndMonth.SelectedItem.ToString().Split(' ')[1] + "-" +
(DateTimeFormatInfo.CurrentInfo.MonthNames.ToList().IndexOf(comboBoxEndMonth.SelectedItem
.ToString().Split(' ')[0]) + 2).ToString() + "-1";

                // Create query
                MySqlCommand command = mySqlConn.CreateCommand();
                command.CommandText = $"SELECT CONCAT(MONTHNAME(Log_DateTime), ' ',
YEAR(Log_DateTime)) AS 'Months', MAX(Log_ListenerCount) AS 'Peak' FROM log WHERE
Log_DateTime BETWEEN \"{startdate}\" and \"{enddate}\" GROUP BY MONTH(Log_DateTime) ORDER
BY Log_DateTime ASC";

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

                // Create dictionary to hold data
                Dictionary<string, int> dictionaryMonthPeak = new Dictionary<string,
int>();

                // Loop through each of the results returned
                while (reader.Read())
                {
                    dictionaryMonthPeak.Add(reader["Months"].ToString(),
int.Parse(reader["Peak"].ToString()));
                }

                // Clear data currently on graph
                chartMonthlyPeak.Series[0].Points.Clear();

                // Cycle through each item in the dictionary and add it to the graph
                foreach (KeyValuePair<string, int> point in dictionaryMonthPeak)
                {
                    chartMonthlyPeak.Series[0].Points.AddXY(point.Key, point.Value);
                }

                // Close reader
                reader.Close();

            }
```

```
            catch (Exception ex)
            {
                // Write error to log file
                StreamWriter sw = File.AppendText(@"./log.txt");
                sw.WriteLine($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\n");
                sw.Close();

                // Display error message to user
                MessageBox.Show($"{DateTime.Now.ToString()} - [ERROR] {ex.Message}
{ex.InnerException}\nThe application will now exit. Please manually restart the
application once the error has been resolved", "An Error has occurred",
MessageBoxButtons.OK, MessageBoxIcon.Error);

                // Close the application
                Application.Exit();
            }


        }

        private void setupMonthlyPeakGraph()
        {
            // Setup X Axis
            chartMonthlyPeak.ChartAreas[0].Axes[0].Title = "Month";
            chartMonthlyPeak.ChartAreas[0].Axes[0].IsLabelAutoFit = true;
            chartMonthlyPeak.ChartAreas[0].Axes[0].Interval = 1;
            chartMonthlyPeak.ChartAreas[0].Axes[0].IsMarginVisible = false;

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

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

            // Setup the series to hold the monthly listener peak count
            chartMonthlyPeak.Series[0].ChartType = SeriesChartType.Line;
            chartMonthlyPeak.Series[0].LegendText = "Listener Count";
            chartMonthlyPeak.Series[0].BorderWidth = 4;
            chartMonthlyPeak.Series[0].Color = Settings.colours.Blue;
            chartMonthlyPeak.Series[0].LabelForeColor = Settings.colours.Blue;

        }
    }

}
```



Hierarchy	Lines of Code
Server Listener Statistics (Debug)	3,017
Server_Listener_Statistics	3,017
Dashboard	946
Reports	800
UserManagement	436
EditSettings	423
LoginForm	167
ChangePassword	98
Handle	79
Settings	55
Hash	5
Settings.colours	5
Program	3

Examiner commentary

Question/Part: AO 2.2 Analysis

Marks: 10/10

The Analysis contains no content from section 2.2.2 of the H446 specification on computational methods, aside from the use of the term 'computational approach' in the problem introduction.

The student has identified a real life problem which means the stakeholder information and following explanation about how they make use of the potential solution is detailed and well explained.

The following investigation is sensible and extracts the essential features to take forward into the students own solution. Each of these are specific, measurable and justified clearly.

The limitations section does not really comment on anything specific, simply saying the solution won't be expanded further with no detail as to what 'anything else' could be (e.g. the report issues identified in Design on pages 11). This section could be improved.

Although the Analysis is a good section, 10 marks is slightly generous.

Question/Part: AO 3.1 Design

Marks: 15/15

Several small structure diagrams show decomposition of the problem. This is further clarified through each algorithm design. It is clear the student fully understands the processes needed to build their solution.

Each interface screen is explained and several usability features and validation techniques are identified throughout this section although these are not made explicit and not all are clearly explained.

A series of flowcharts with accompanying pseudocode are used to demonstrate clearly how the system will be created. Each is explained in detail and the set of algorithms from a comprehensive overall picture.

There is no data dictionary, although it is clear from the algorithms that there are several key variables to be stored alongside the data in the database itself. The database table structures are planned and explained well.

The test plan section explains how the system will be tested in general and at what stages. Although a full test table is not expected at his stage, some examples of tests and data to be used could have been identified and justified. A small test table describes a number of post-development tests that will be carried out.

We need to consider the best-fit for this piece of work. Whilst it is agreed that it should be considered worthy of the top mark band, there are some areas that fall slightly short of full justification and lack depth.

Question/Part: AO 3.2 Developing the coded solution

Marks: 15/15

The development section shows a thorough walkthrough of exactly how the student created the solution. Whilst this is excellent, it is unnecessary to show each step of development, particularly where the steps are non-code related such as customising settings and arranging items on forms.

As the section is so detailed, subheadings would assist the marking and moderation process to make it clearer where the range of evidence is shown.

The evidence clearly shows both code and interface working together, along with tests to prove the elements are functional. This testing includes a range of valid and invalid data and demonstrates working validation in several areas of the data entry forms.

The explanation and review at each stage allows errors and changes made to be explained and justified.

Code is well structured and variables are suitably named. There are comments throughout and this makes the code very easy to maintain.

Question/Part: AO 3.2 Testing to inform development

Marks: 10/10

A wide range of tests are shown throughout the iterative development that makes it clear how the system was tested as it was built. There are a number of remedial actions taken and explained, with before and after screenshots as evidence of where the work was improved.

Agree with the teacher's mark and comments.

Question/Part: AO 3.3 Testing to inform evaluation

Marks: 5/5

Post-development testing evidence begins on page 172. This follows the plan from the Design section and any tweaks needed are evidenced well. All functionality is tested to ensure the robustness of the system.

There is no specific focus on usability but it is clear from the depth of testing that the system is usable, however particularly as there was a specific stakeholder for the project, it is unusual that end-user testing was not carried out to verify the usability features.

The teacher's mark can be agreed.

Question/Part: AO 3.3 Evaluation of solution

Marks: 15/15

The user requirements are well written and nicely cross-referenced to the evidence in the project to demonstrate its success. As with the testing section, the focus is largely on the

functionality of the system rather than usability, particularly from the end user's perspective. This has more of an impact on the section as there are two descriptors that focus on this aspect of the solution.

There is a short section discussing accessibility which partly addresses one usability feature. There are also some overall comments at the end of the section which mention the end user and their opinion of the solution. This feedback could have been included in earlier parts of the evaluation to reinforce the students own opinion of their work.

One non-essential criterion was not met; this is explained and how this could be developed in the future is explained and demonstrated nicely. Other limitations are further discussed in the final section of the evaluation, addressing how those features could be built in.

Maintenance issues are appropriate and explained well, giving possible solutions to the issues.

The teacher's mark is slightly generous considering the lack of evaluative comment on usability, however the overall quality of the evaluation means the mark can be agreed.

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