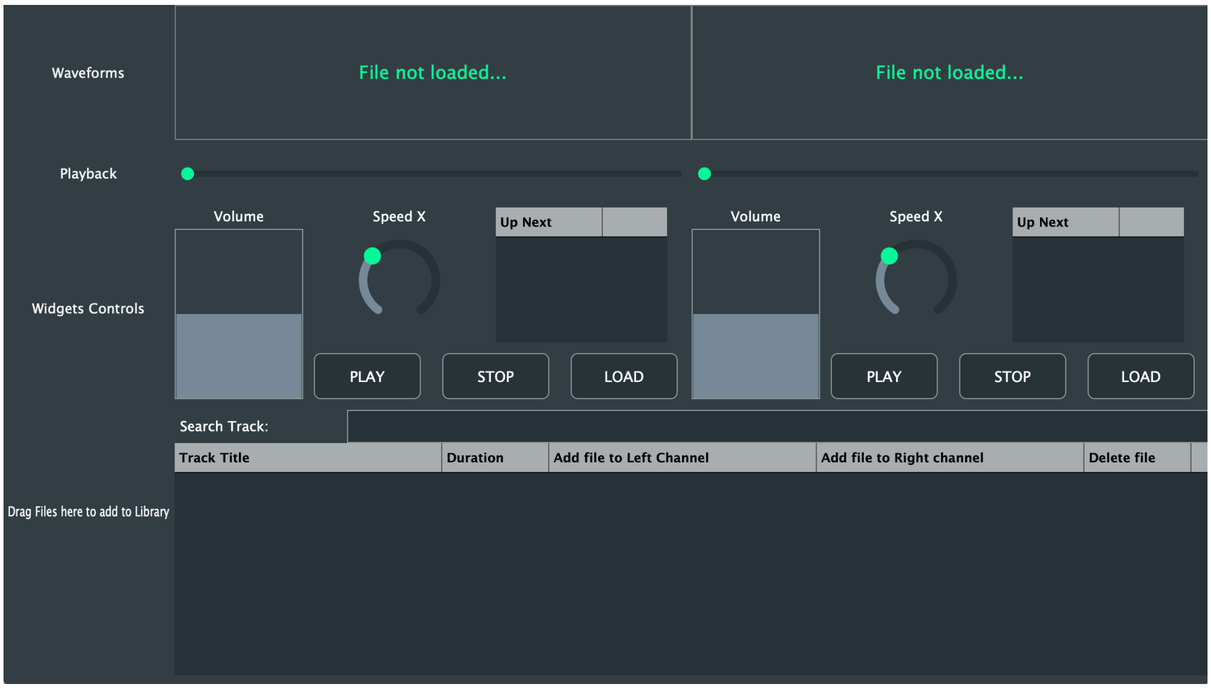
Introduction of Otodecks

- An introduction describing the purpose of Otodecks:

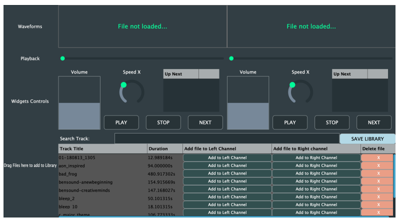
  As indicated in the coursework's introduction, Otodecks is a basic DJ application that enables the user to have means of manipulating and remixing music tracks that are loaded into the application. The application contains a playlist for music tracks to be loaded onto the deck of the user interface to be edited. 2 tracks are able to be loaded to be played simultaneously.

  The compiler that I used was Xcode (MacOS) and the picture below shows the interface of the Otodecks application when it's first compiled and built:

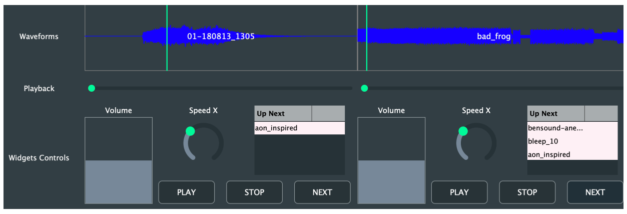
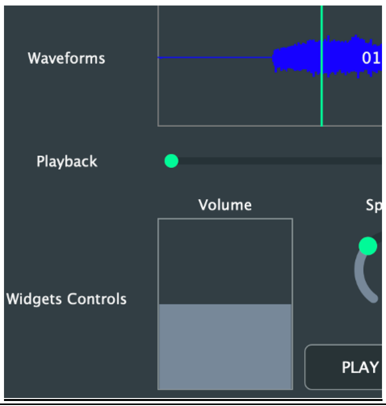
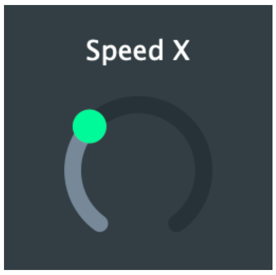
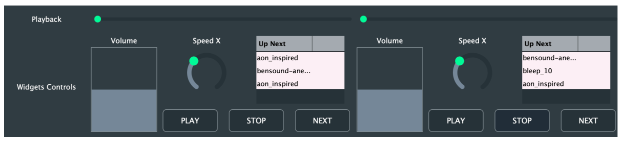
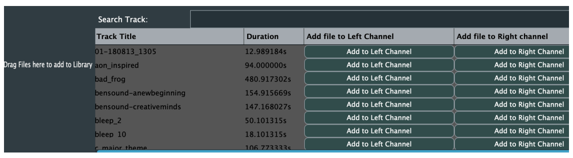


The table (Table 1) below tabulates the various requirements of the coursework with regards to the Otodecks application.

Table 1:

Requirement	Sub-Requirement	Functionality	Display Image
R1	R1A	User is able to load audio into audio players	 Waveforms File not loaded... File not loaded... Playback Volume Speed X Up Next Volume Speed X Up Next Widgets Controls PLAY STOP NEXT PLAY STOP NEXT Search Track: SAVE LIBRARY Drag Files here to add to Library

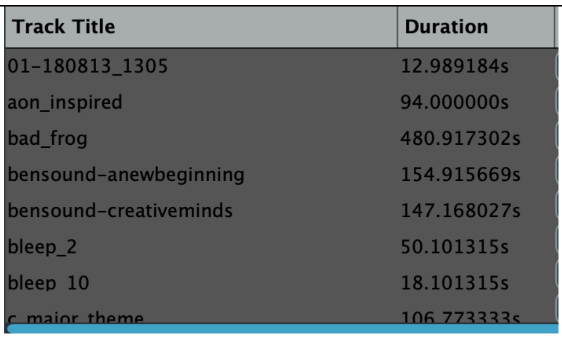
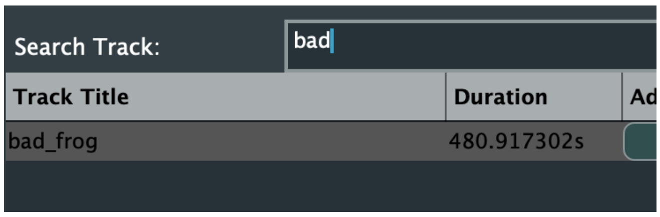
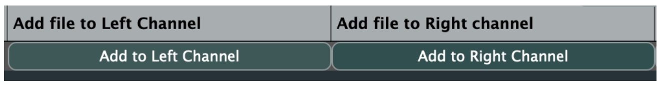
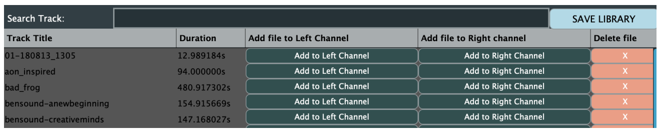
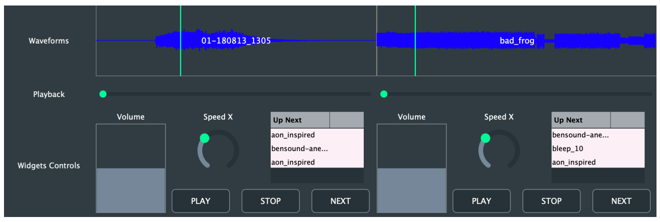
Track Title	Duration	Add file to Left Channel	Add file to Right channel	Delete file
01-180813_1305	12.989184s	Add to Left Channel	Add to Right Channel	X
aon_inspired	94.000000s	Add to Left Channel	Add to Right Channel	X
bad_frog	480.917302s	Add to Left Channel	Add to Right Channel	X
bensound-anewbeginning	154.915669s	Add to Left Channel	Add to Right Channel	X
bensound-creativeminds	147.168027s	Add to Left Channel	Add to Right Channel	X
bleep_2	50.101315s	Add to Left Channel	Add to Right Channel	X
bleep_10	18.101315s	Add to Left Channel	Add to Right Channel	X
c_major_theme	106.773333s	Add to Left Channel	Add to Right Channel	X

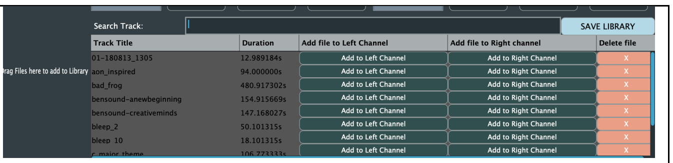
	R1B	User can play 2 or more tracks	 Waveforms 01-180813_1305 bad_frog Playback Widgets Controls Volume Speed X Up Next aon_inspired PLAY STOP NEXT Volume Speed X Up Next bensound-ane... bleep_10 aon_inspired PLAY STOP NEXT
	R1C	User can mix audio tracks by varying their volumes	 Waveforms 01 Playback Volume Sp Widgets Controls PLAY
	R1D	User can speed up and slow down the audio tracks	
R2	R2A	Components with custom graphics in Deck	 Playback Widgets Controls Volume Speed X Up Next aon_inspired bensound-ane... aon_inspired PLAY STOP NEXT Volume Speed X Up Next bensound-ane... bleep_10 aon_inspired PLAY STOP NEXT
	R2B	Component that enable user to playback audio in Deck	
R3	R3A	Component that allows user to add audio track file to library (Playlist Component)	

Search Track:

Drag Files here to add to Library

Track Title	Duration	Add file to Left Channel	Add file to Right channel
01-180813_1305	12.989184s	Add to Left Channel	Add to Right Channel
aon_inspired	94.000000s	Add to Left Channel	Add to Right Channel
bad_frog	480.917302s	Add to Left Channel	Add to Right Channel
bensound-anewbeginning	154.915669s	Add to Left Channel	Add to Right Channel
bensound-creativeminds	147.168027s	Add to Left Channel	Add to Right Channel
bleep_2	50.101315s	Add to Left Channel	Add to Right Channel
bleep_10	18.101315s	Add to Left Channel	Add to Right Channel
c_major_theme	106.773333s	Add to Left Channel	Add to Right Channel

	R3B	Component that parses audio track file’s title and duration	 Track Title | Duration 01-180813_1305 | 12.989184s aon_inspired | 94.000000s bad_frog | 480.917302s bensound-anewbeginning | 154.915669s bensound-creativeminds | 147.168027s bleep_2 | 50.101315s bleep_10 | 18.101315s c_major_theme | 106.773333s
	R3C	Component that allows user to search for audio track files in library(playlist)	 Search Track: bad Track Title | Duration | Ad bad_frog | 480.917302s
	R3D	Component that allows user to load track files from library(playlist) to Deck	 Add file to Left Channel | Add file to Right channel Add to Left Channel | Add to Right Channel
	R3E	Music track library restores when application is exited and restarted	 Search Track: | SAVE LIBRARY Track Title | Duration | Add file to Left Channel | Add file to Right channel | Delete file 01-180813_1305 | 12.989184s | Add to Left Channel | Add to Right Channel | X aon_inspired | 94.000000s | Add to Left Channel | Add to Right Channel | X bad_frog | 480.917302s | Add to Left Channel | Add to Right Channel | X bensound-anewbeginning | 154.915669s | Add to Left Channel | Add to Right Channel | X bensound-creativeminds | 147.168027s | Add to Left Channel | Add to Right Channel | X
R4	R4A	GUI layout is different from class with extra controls	 Up Next: aon_inspired, bensound-ane..., aon_inspired Up Next: bensound-ane..., bleep_10, aon_inspired Waveforms 01-180813_1305 bad_frog Playback Widgets Controls Volume Speed X PLAY STOP NEXT Volume Speed X PLAY STOP NEXT
	R4B	GUI layout for R2 is different from class	-

	R4C	GUI layout for library (playlist) is different from class	

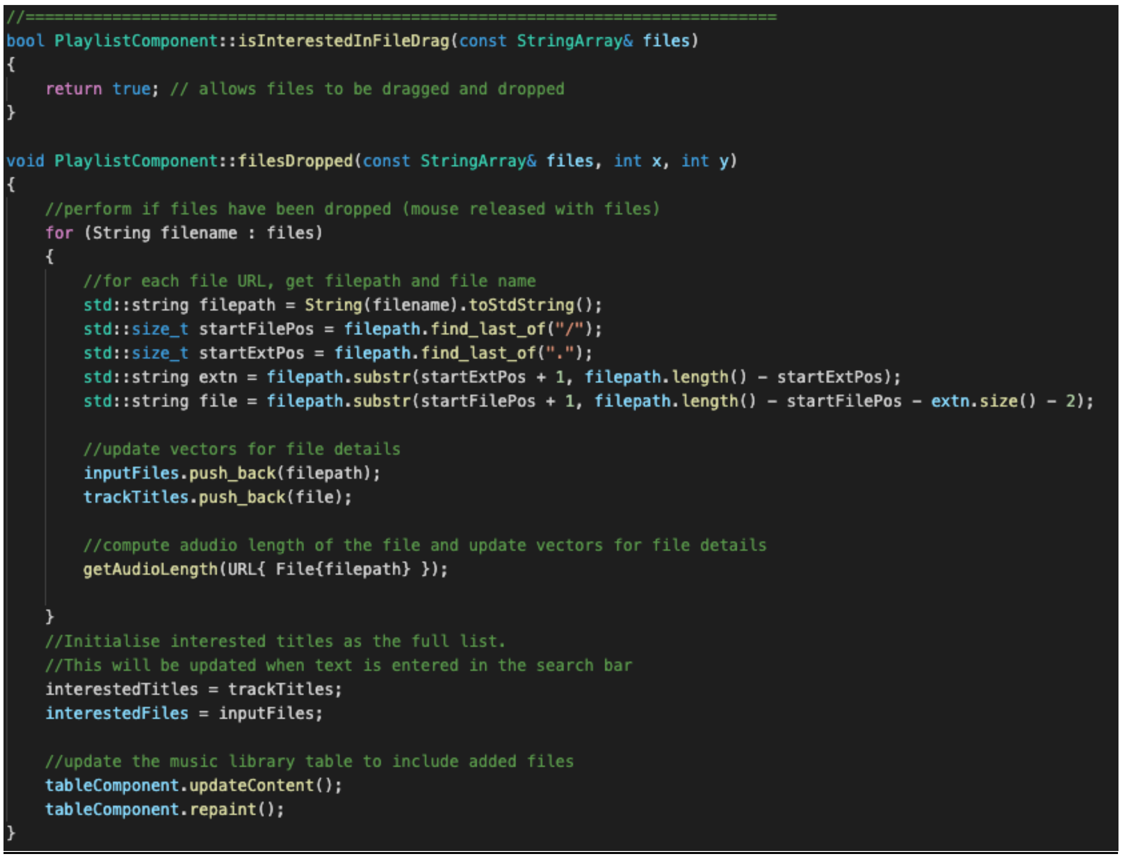
Search Track: SAVE LIBRARY

Drag Files here to add to Library

Track Title	Duration	Add file to Left Channel	Add file to Right channel	Delete file
01-180813_1305	12.989184s	Add to Left Channel	Add to Right Channel	X
aon_inspired	94.000000s	Add to Left Channel	Add to Right Channel	X
bad_frog	480.917302s	Add to Left Channel	Add to Right Channel	X
bensound-anewbeginning	154.915669s	Add to Left Channel	Add to Right Channel	X
bensound-creativeminds	147.168027s	Add to Left Channel	Add to Right Channel	X
bleep_2	50.101315s	Add to Left Channel	Add to Right Channel	X
bleep_10	18.101315s	Add to Left Channel	Add to Right Channel	X
c_major_theme	106.773333s	Add to Left Channel	Add to Right Channel	X

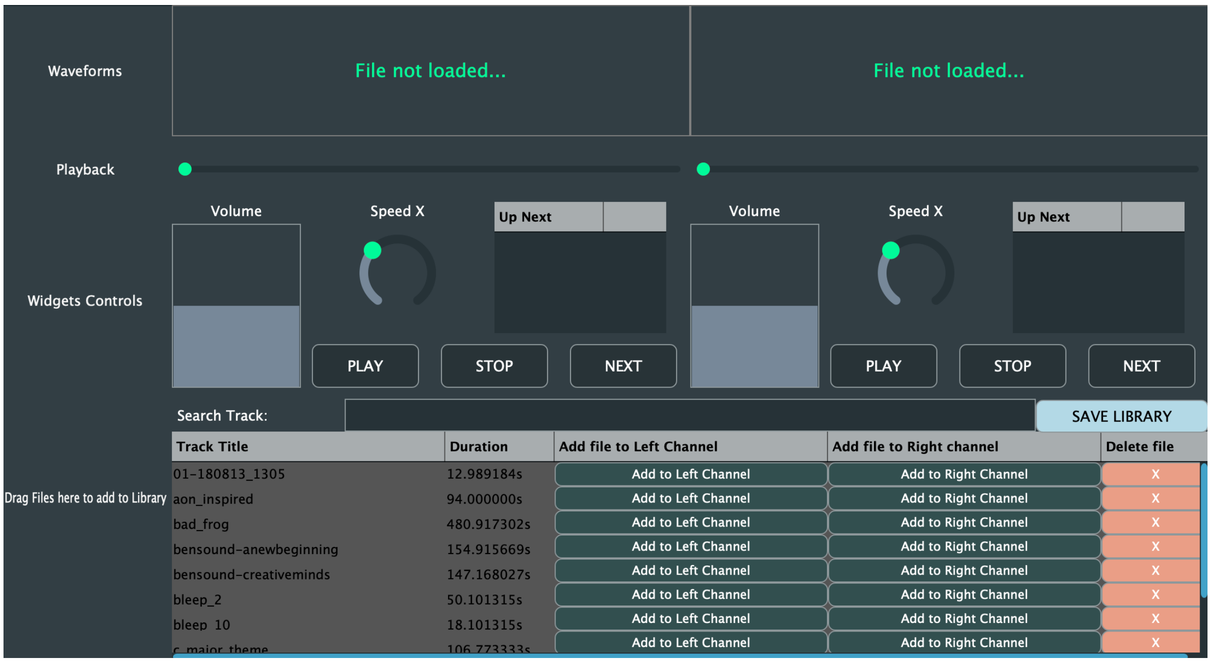
R1A: Loading Audio into Player

In the code extract below, which is found in my PlaylistComponent.cpp, I enable a feature where the user is able to drag and drop the audio files from their music folders into Otodecks. Once the files are dropped into the Otodecks, it will exist in the playlist, and the user is able to click on buttons to add to either the left audio player or right audio player, upon the user's intention to load the tracks into the audio player.

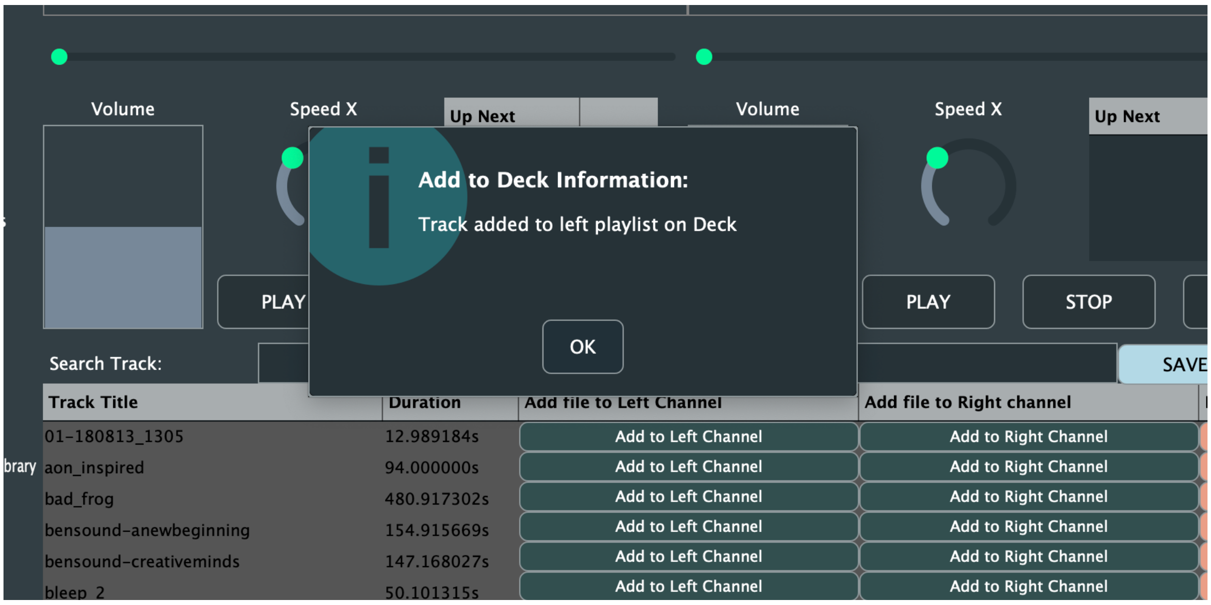


```cpp
//==============================================================================
bool PlaylistComponent::isInterestedInFileDrag(const StringArray& files)
{
    return true; // allows files to be dragged and dropped
}

void PlaylistComponent::filesDropped(const StringArray& files, int x, int y)
{
    //perform if files have been dropped (mouse released with files)
    for (String filename : files)
    {
        //for each file URL, get filepath and file name
        std::string filepath = String(filename).toStdString();
        std::size_t startFilePos = filepath.find_last_of("/");
        std::size_t startExtPos = filepath.find_last_of(".");
        std::string extn = filepath.substr(startExtPos + 1, filepath.length() - startExtPos);
        std::string file = filepath.substr(startFilePos + 1, filepath.length() - startFilePos - extn.size() - 2);

        //update vectors for file details
        inputFiles.push_back(filepath);
        trackTitles.push_back(file);

        //compute adudio length of the file and update vectors for file details
        getAudioLength(URL{ File{filepath} });

    }
    //Initialise interested titles as the full list.
    //This will be updated when text is entered in the search bar
    interestedTitles = trackTitles;
    interestedFiles = inputFiles;

    //update the music library table to include added files
    tableComponent.updateContent();
    tableComponent.repaint();
}
```

The image below shows the Otodecks interface when the user drag and drops the files into the playlist. Upon clicking the “Add to Left Channel” or “Add to Right Channel” buttons, the tracks will then be loaded into the audio player above. A pop out message will show the affirmation of the track being added.



Track Title	Duration	Add file to Left Channel	Add file to Right channel	Delete file
01-180813_1305	12.989184s	Add to Left Channel	Add to Right Channel	X
aon_inspired	94.000000s	Add to Left Channel	Add to Right Channel	X
bad_frog	480.917302s	Add to Left Channel	Add to Right Channel	X
bensound-anewbeginning	154.915669s	Add to Left Channel	Add to Right Channel	X
bensound-creativeminds	147.168027s	Add to Left Channel	Add to Right Channel	X
bleep_2	50.101315s	Add to Left Channel	Add to Right Channel	X
bleep_10	18.101315s	Add to Left Channel	Add to Right Channel	X
c_major_theme	106.773333s	Add to Left Channel	Add to Right Channel	X

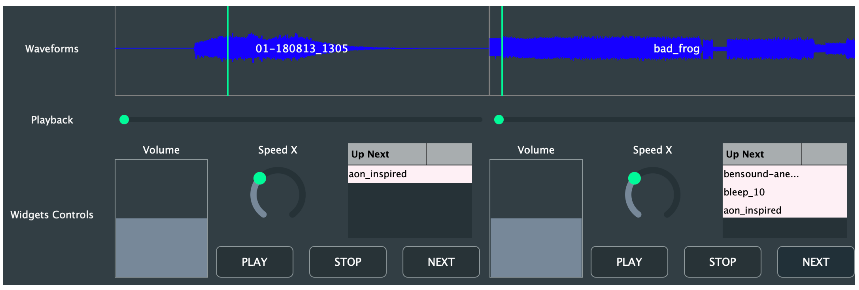


Add to Deck Information:

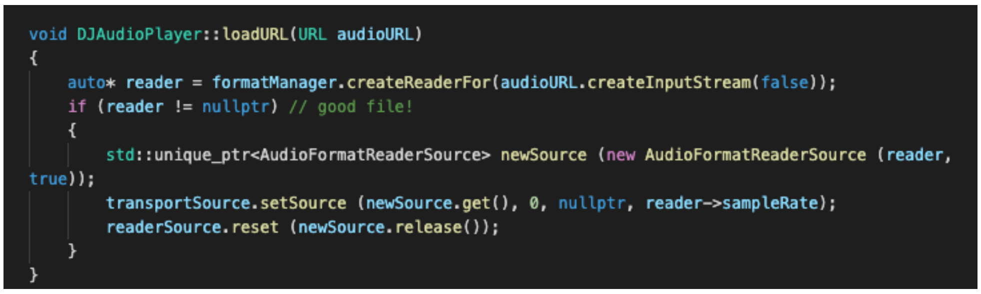
Track added to left playlist on Deck

Track Title	Duration	Add file to Left Channel	Add file to Right channel
01-180813_1305	12.989184s	Add to Left Channel	Add to Right Channel
aon_inspired	94.000000s	Add to Left Channel	Add to Right Channel
bad_frog	480.917302s	Add to Left Channel	Add to Right Channel
bensound-anewbeginning	154.915669s	Add to Left Channel	Add to Right Channel
bensound-creativeminds	147.168027s	Add to Left Channel	Add to Right Channel
bleep_2	50.101315s	Add to Left Channel	Add to Right Channel

When the user clicks on the “Next” button, it will queue the tracks in “Up Next”.



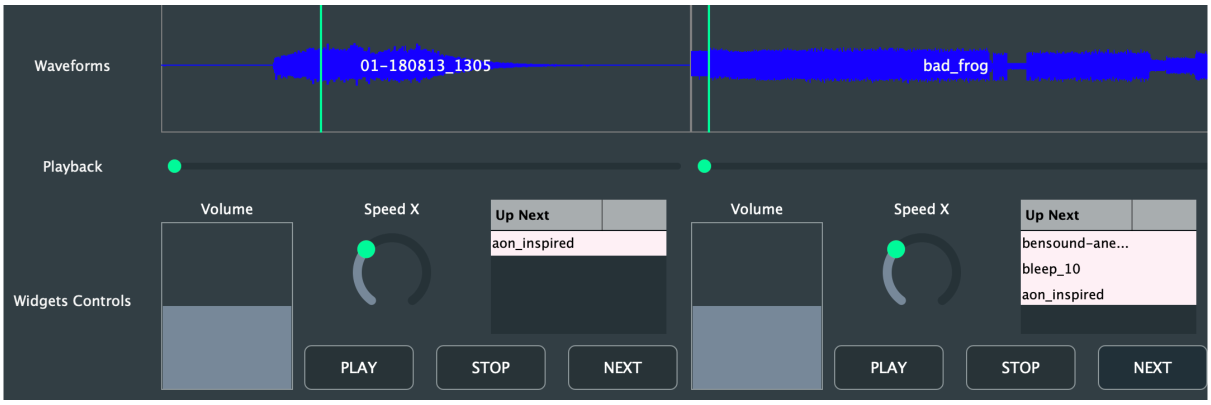
In my DJAudioPlayer.cpp, there is a function that will enable the loading of the audio track file's URL to be read and played, as shown below.



```
void DJAudioPlayer::loadURL(URL audioURL)
{
    auto* reader = formatManager.createReaderFor(audioURL.createInputStream(false));
    if (reader != nullptr) // good file!
    {
        std::unique_ptr<AudioFormatReaderSource> newSource (new AudioFormatReaderSource (reader,
true));
        transportSource.setSource (newSource.get(), 0, nullptr, reader->sampleRate);
        readerSource.reset (newSource.release());
    }
}
```

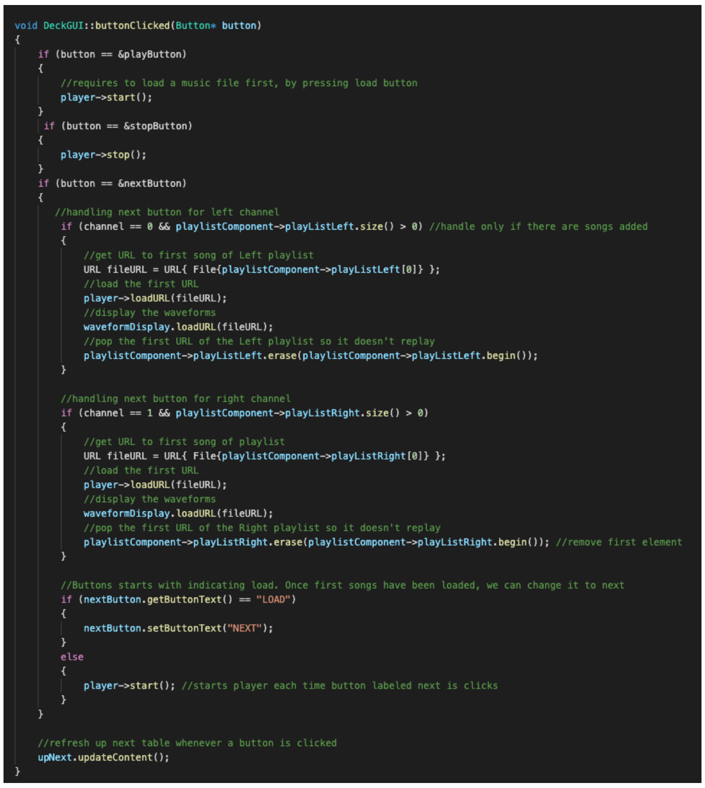
<u>R1B: Playing 2 Tracks</u>

The user is able to play 2 audio tracks at the same time, by loading into the audio track files into the left and right channels of the audio deck player.



Upon loading the audio track files into either the left or right channel, they are queued and the user is able to click on "Play" to start playing the audio tracks.

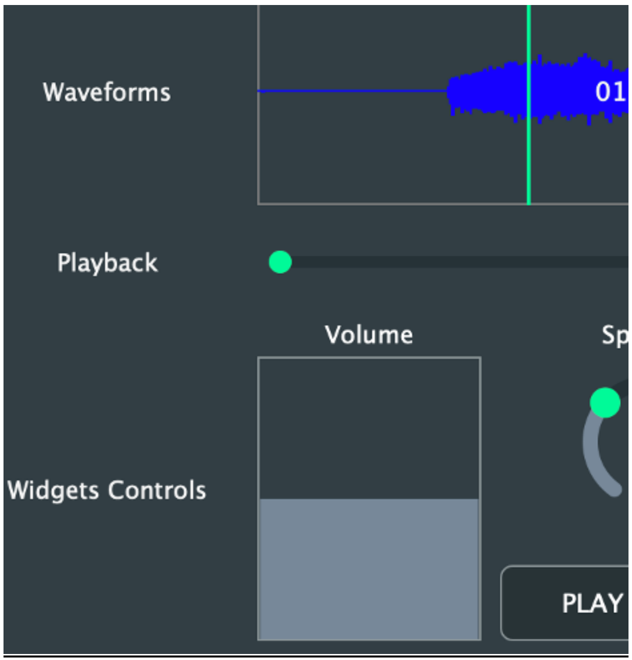
The code extract below, which is from my DeckGUI.cpp, functions to enable the audio track files to be loaded upon clicking on the buttons, such as “Play”, “Stop” and “Next”. The “Next button has to be instantiated first for the audio track files to be loaded from the playlist to the audio player.



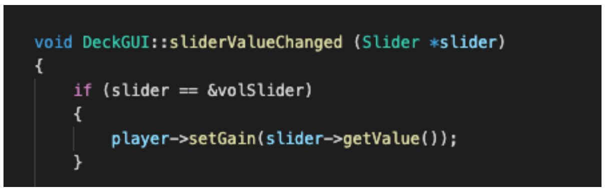
```cpp
void DeckGUI::buttonClicked(Button* button)
{
    if (button == &playButton)
    {
        //requires to load a music file first, by pressing load button
        player->start();
    }
     if (button == &stopButton)
    {
        player->stop();
    }
    if (button == &nextButton)
    {
       //handling next button for left channel
        if (channel == 0 && playlistComponent->playListLeft.size() > 0) //handle only if there are songs added
        {
            //get URL to first song of Left playlist
            URL fileURL = URL{ File{playlistComponent->playListLeft[0]} };
            //load the first URL
            player->loadURL(fileURL);
            //display the waveforms
            waveformDisplay.loadURL(fileURL);
            //pop the first URL of the Left playlist so it doesn't replay
            playlistComponent->playListLeft.erase(playlistComponent->playListLeft.begin());
        }

        //handling next button for right channel
        if (channel == 1 && playlistComponent->playListRight.size() > 0)
        {
            //get URL to first song of playlist
            URL fileURL = URL{ File{playlistComponent->playListRight[0]} };
            //load the first URL
            player->loadURL(fileURL);
            //display the waveforms
            waveformDisplay.loadURL(fileURL);
            //pop the first URL of the Right playlist so it doesn't replay
            playlistComponent->playListRight.erase(playlistComponent->playListRight.begin()); //remove first element
        }

        //Buttons starts with indicating load. Once first songs have been loaded, we can change it to next
        if (nextButton.getButtonText() == "LOAD")
        {
            nextButton.setButtonText("NEXT");
        }
        else
        {
            player->start(); //starts player each time button labeled next is clicks
        }
    }

    //refresh up next table whenever a button is clicked
    upNext.updateContent();
}
```

R1C: Varying volumes of Audio Tracks

As shown in the picture below, the user has a slider to enable the manipulation of audio volumes when the audio track is being played.

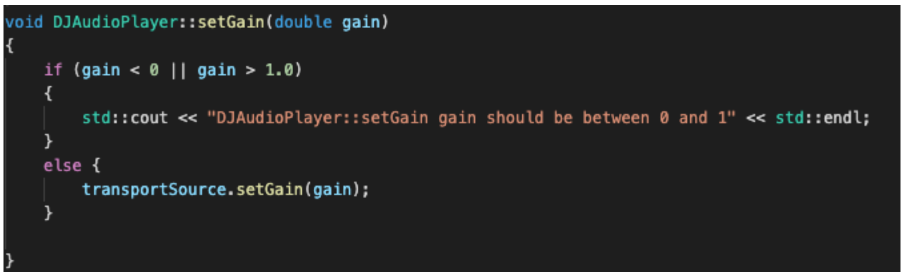


In my DeckGUI.cpp, there is a slider included for the volume slider to be visualized, with the address of the volume slider in an "if" condition.



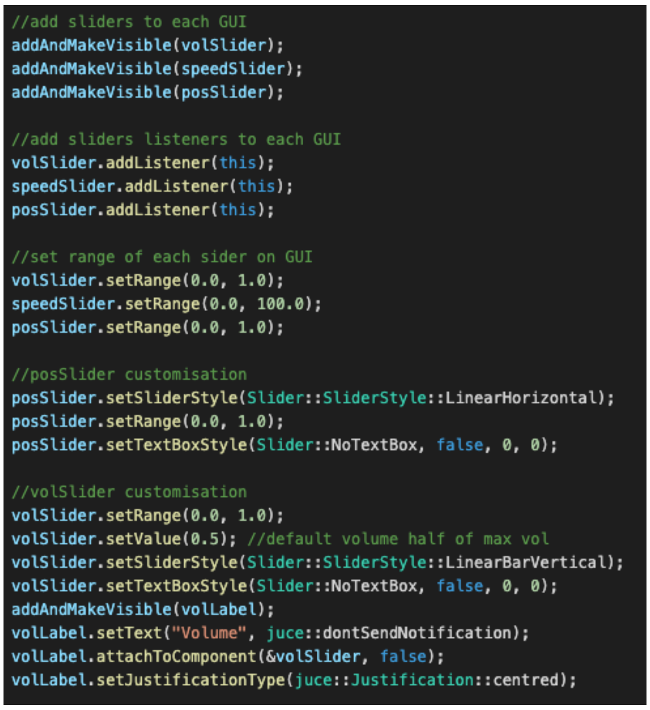
```
void DeckGUI::sliderValueChanged (Slider *slider)
{
    if (slider == &volSlider)
    {
        player->setGain(slider->getValue());
    }
```

Setting the gain of my volume slider with "if" conditions.



```
void DJAudioPlayer::setGain(double gain)
{
    if (gain < 0 || gain > 1.0)
    {
        std::cout << "DJAudioPlayer::setGain gain should be between 0 and 1" << std::endl;
    }
    else {
        transportSource.setGain(gain);
    }

}
```

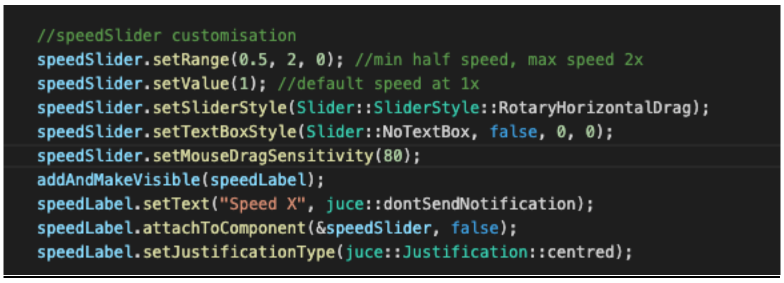
Also in DeckGUI.cpp, I've added slider listeners and customisations for my sliders, like the volume slider for instance.



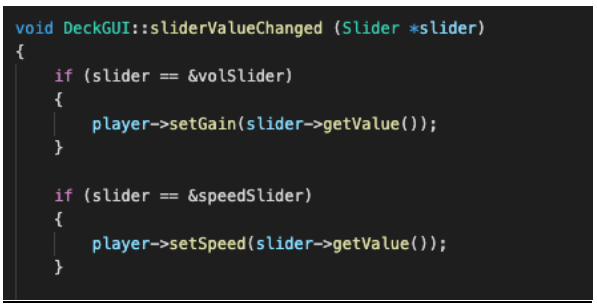
```
//add sliders to each GUI
addAndMakeVisible(volSlider);
addAndMakeVisible(speedSlider);
addAndMakeVisible(posSlider);

//add sliders listeners to each GUI
volSlider.addListener(this);
speedSlider.addListener(this);
posSlider.addListener(this);

//set range of each sider on GUI
volSlider.setRange(0.0, 1.0);
speedSlider.setRange(0.0, 100.0);
posSlider.setRange(0.0, 1.0);

//posSlider customisation
posSlider.setSliderStyle(Slider::SliderStyle::LinearHorizontal);
posSlider.setRange(0.0, 1.0);
posSlider.setTextBoxStyle(Slider::NoTextBox, false, 0, 0);

//volSlider customisation
volSlider.setRange(0.0, 1.0);
volSlider.setValue(0.5); //default volume half of max vol
volSlider.setSliderStyle(Slider::SliderStyle::LinearBarVertical);
volSlider.setTextBoxStyle(Slider::NoTextBox, false, 0, 0);
addAndMakeVisible(volLabel);
volLabel.setText("Volume", juce::dontSendNotification);
volLabel.attachToComponent(&volSlider, false);
volLabel.setJustificationType(juce::Justification::centred);
```

R1D: Varying Speed of Audio Tracks

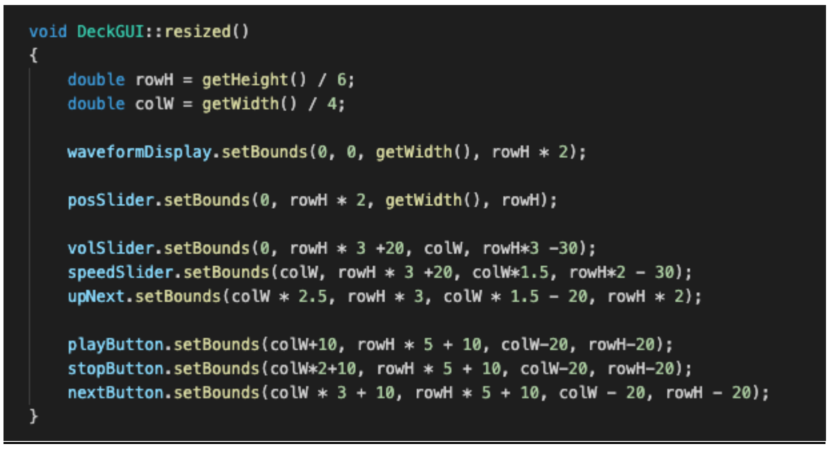
In the code extract below, that is found in my DeckGUI.cpp, I've added a customised rotary slider for the manipulation of the speed of the audio track, adjusting the positioning and sensitivity of the slider as well.



```
    //speedSlider customisation
    speedSlider.setRange(0.5, 2, 0); //min half speed, max speed 2x
    speedSlider.setValue(1); //default speed at 1x
    speedSlider.setSliderStyle(Slider::SliderStyle::RotaryHorizontalDrag);
    speedSlider.setTextBoxStyle(Slider::NoTextBox, false, 0, 0);
    speedSlider.setMouseDragSensitivity(80);
    addAndMakeVisible(speedLabel);
    speedLabel.setText("Speed X", juce::dontSendNotification);
    speedLabel.attachToComponent(&speedSlider, false);
    speedLabel.setJustificationType(juce::Justification::centred);
```



```
void DeckGUI::sliderValueChanged (Slider *slider)
{
    if (slider == &volSlider)
    {
        player->setGain(slider->getValue());
    }

    if (slider == &speedSlider)
    {
        player->setSpeed(slider->getValue());
    }
```

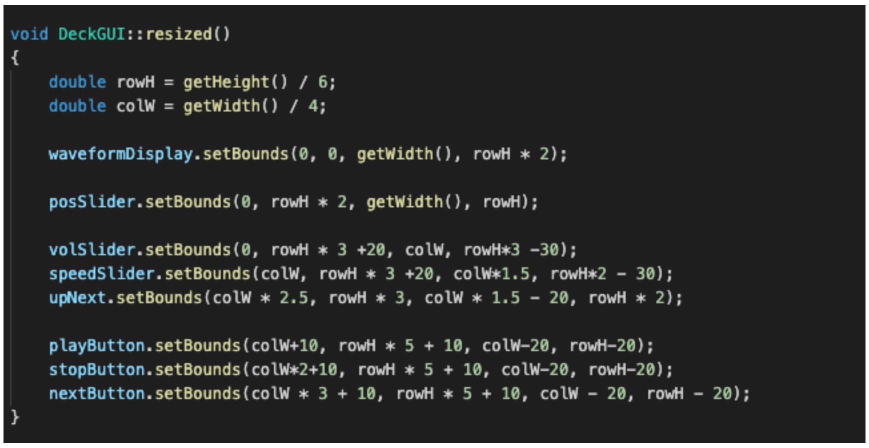


```
void DeckGUI::resized()
{
    double rowH = getHeight() / 6;
    double colW = getWidth() / 4;

    waveformDisplay.setBounds(0, 0, getWidth(), rowH * 2);

    posSlider.setBounds(0, rowH * 2, getWidth(), rowH);

    volSlider.setBounds(0, rowH * 3 +20, colW, rowH*3 -30);
    speedSlider.setBounds(colW, rowH * 3 +20, colW*1.5, rowH*2 - 30);
    upNext.setBounds(colW * 2.5, rowH * 3, colW * 1.5 - 20, rowH * 2);

    playButton.setBounds(colW+10, rowH * 5 + 10, colW-20, rowH-20);
    stopButton.setBounds(colW*2+10, rowH * 5 + 10, colW-20, rowH-20);
    nextButton.setBounds(colW * 3 + 10, rowH * 5 + 10, colW - 20, rowH - 20);
}
```

The same adjustments hold true for my other sliders too.

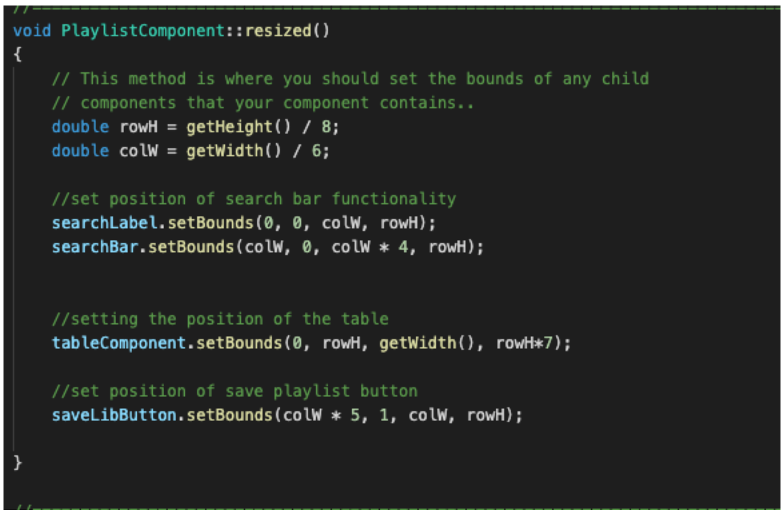
R2A: Customise Graphics of Interface

I've reposition my user controls for my application interface, as well as the playlist components by adjusting the rows and columns positions. The code extract below is found in my DeckGUI.cpp that positions the customised controls for the user.



```cpp
void DeckGUI::resized()
{
    double rowH = getHeight() / 6;
    double colW = getWidth() / 4;

    waveformDisplay.setBounds(0, 0, getWidth(), rowH * 2);

    posSlider.setBounds(0, rowH * 2, getWidth(), rowH);

    volSlider.setBounds(0, rowH * 3 +20, colW, rowH*3 -30);
    speedSlider.setBounds(colW, rowH * 3 +20, colW*1.5, rowH*2 - 30);
    upNext.setBounds(colW * 2.5, rowH * 3, colW * 1.5 - 20, rowH * 2);

    playButton.setBounds(colW+10, rowH * 5 + 10, colW-20, rowH-20);
    stopButton.setBounds(colW*2+10, rowH * 5 + 10, colW-20, rowH-20);
    nextButton.setBounds(colW * 3 + 10, rowH * 5 + 10, colW - 20, rowH - 20);
}
```

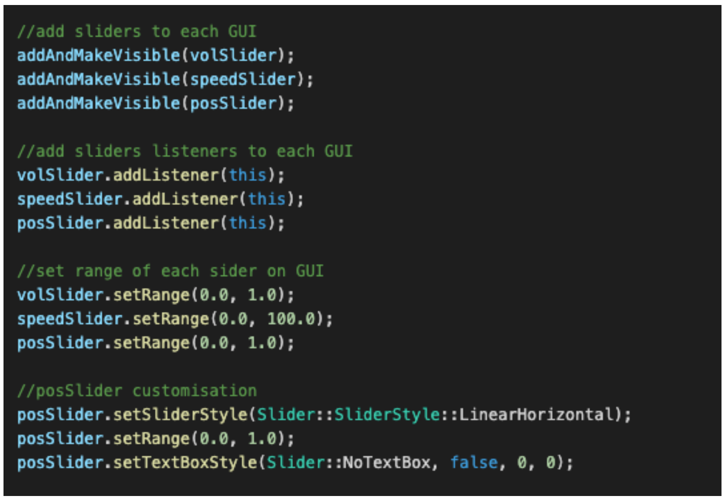
As for my PlaylistComponent.cpp, I've also customised the positions of the search bar and the playlist that contains the audio tracks that are dropped into the interface, waiting to be loaded into the DJ audio player above.



```cpp
void PlaylistComponent::resized()
{
    // This method is where you should set the bounds of any child
    // components that your component contains..
    double rowH = getHeight() / 8;
    double colW = getWidth() / 6;

    //set position of search bar functionality
    searchLabel.setBounds(0, 0, colW, rowH);
    searchBar.setBounds(colW, 0, colW * 4, rowH);


    //setting the position of the table
    tableComponent.setBounds(0, rowH, getWidth(), rowH*7);

    //set position of save playlist button
    saveLibButton.setBounds(colW * 5, 1, colW, rowH);

}
```

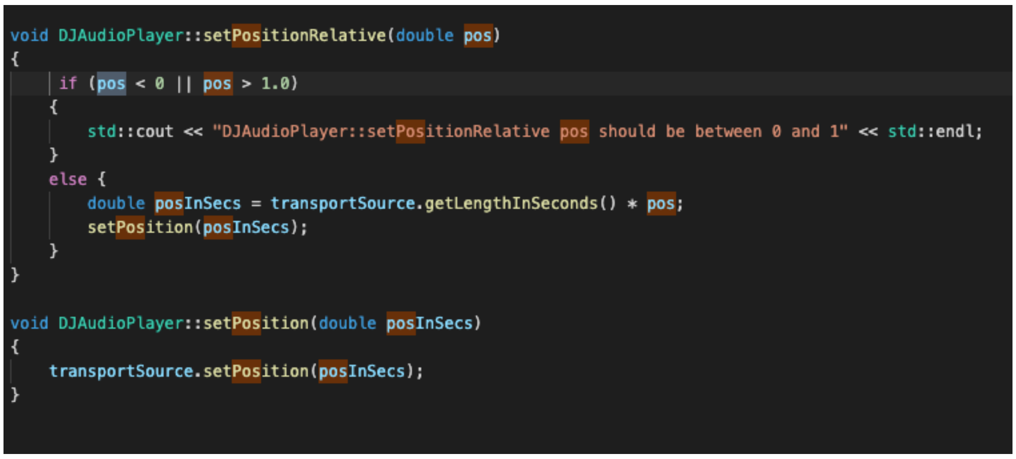
R2B: Enable Playback of Audio Tracks

To enable the playback of the audio tracks, I've also included a position slider, in my DeckGUI.cpp



```
//add sliders to each GUI
addAndMakeVisible(volSlider);
addAndMakeVisible(speedSlider);
addAndMakeVisible(posSlider);

//add sliders listeners to each GUI
volSlider.addListener(this);
speedSlider.addListener(this);
posSlider.addListener(this);

//set range of each sider on GUI
volSlider.setRange(0.0, 1.0);
speedSlider.setRange(0.0, 100.0);
posSlider.setRange(0.0, 1.0);

//posSlider customisation
posSlider.setSliderStyle(Slider::SliderStyle::LinearHorizontal);
posSlider.setRange(0.0, 1.0);
posSlider.setTextBoxStyle(Slider::NoTextBox, false, 0, 0);
```

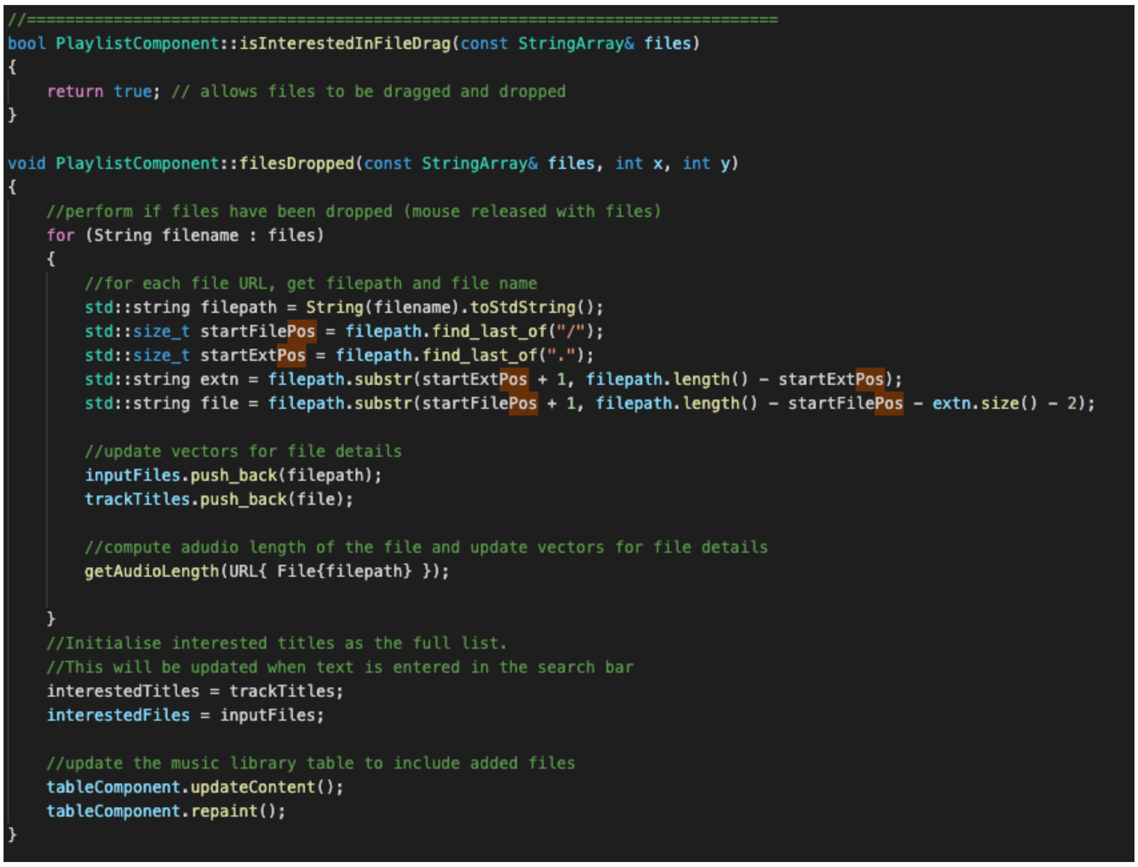
In my DJAudioPlayer.cpp, I've included the conditions of which the position slider for audio playback should be asserted.



```
void DJAudioPlayer::setPositionRelative(double pos)
{
    if (pos < 0 || pos > 1.0)
    {
        std::cout << "DJAudioPlayer::setPositionRelative pos should be between 0 and 1" << std::endl;
    }
    else {
        double posInSecs = transportSource.getLengthInSeconds() * pos;
        setPosition(posInSecs);
    }
}

void DJAudioPlayer::setPosition(double posInSecs)
{
    transportSource.setPosition(posInSecs);
}
```

R3A: Adding Audio Track Files to Library (Playlist Component)

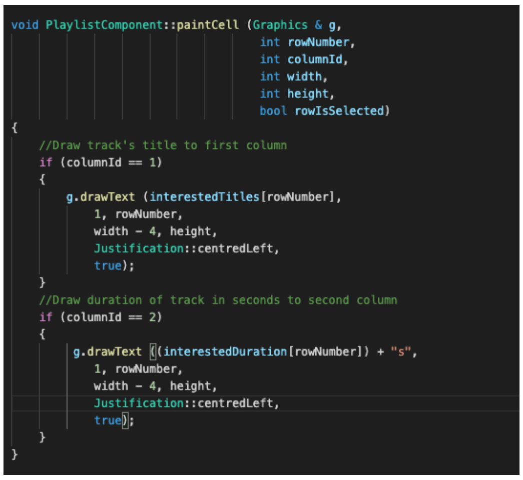
In the code extract, from my PlaylistComponent.cpp, it allows the user to drag and drop audio track files into the playlist library of Otodecks, be reading the file's URL and pathway.



```cpp
//==============================================================================
bool PlaylistComponent::isInterestedInFileDrag(const StringArray& files)
{
    return true; // allows files to be dragged and dropped
}

void PlaylistComponent::filesDropped(const StringArray& files, int x, int y)
{
    //perform if files have been dropped (mouse released with files)
    for (String filename : files)
    {
        //for each file URL, get filepath and file name
        std::string filepath = String(filename).toStdString();
        std::size_t startFilePos = filepath.find_last_of("/");
        std::size_t startExtPos = filepath.find_last_of(".");
        std::string extn = filepath.substr(startExtPos + 1, filepath.length() - startExtPos);
        std::string file = filepath.substr(startFilePos + 1, filepath.length() - startFilePos - extn.size() - 2);

        //update vectors for file details
        inputFiles.push_back(filepath);
        trackTitles.push_back(file);

        //compute adudio length of the file and update vectors for file details
        getAudioLength(URL{ File{filepath} });

    }
    //Initialise interested titles as the full list.
    //This will be updated when text is entered in the search bar
    interestedTitles = trackTitles;
    interestedFiles = inputFiles;

    //update the music library table to include added files
    tableComponent.updateContent();
    tableComponent.repaint();
}
```

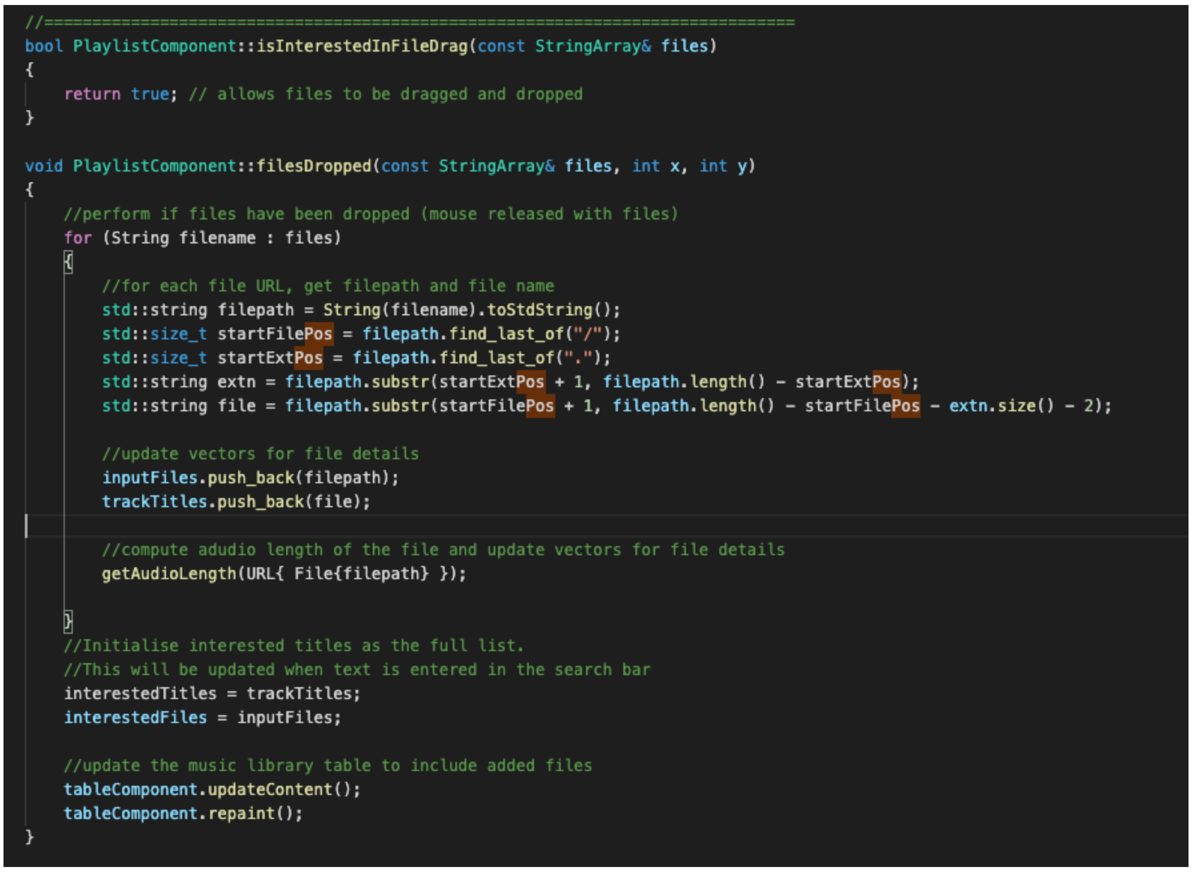
R3B: Parses Filename and Track Duration of Audio Tracks

The code extract below is the from PlaylistComponent.cpp, specifically the paintcell() function, and in this function I've drawn the track titles and track duration in the playlist, in a table format. I've specially indicated "s'" behind the tracks' total duration to denote that it is in seconds.



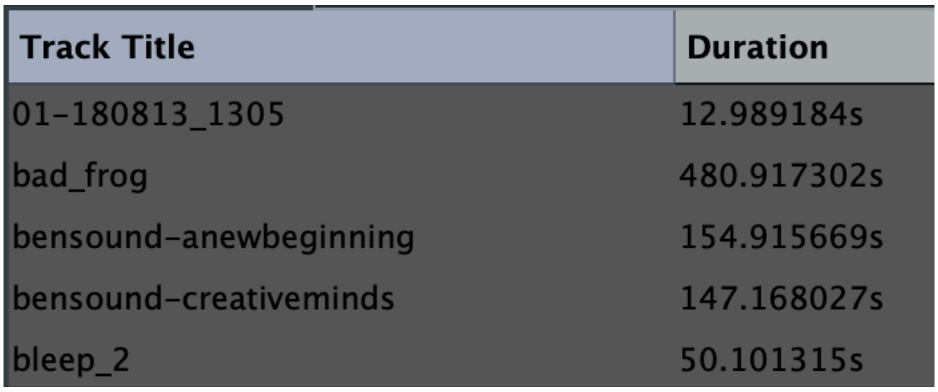
```cpp
void PlaylistComponent::paintCell (Graphics & g,
                                   int rowNumber,
                                   int columnId,
                                   int width,
                                   int height,
                                   bool rowIsSelected)
{
    //Draw track's title to first column
    if (columnId == 1)
    {
        g.drawText (interestedTitles[rowNumber],
            1, rowNumber,
            width - 4, height,
            Justification::centredLeft,
            true);
    }
    //Draw duration of track in seconds to second column
    if (columnId == 2)
    {
         g.drawText ((interestedDuration[rowNumber]) + "s",
            1, rowNumber,
            width - 4, height,
            Justification::centredLeft,
            true);
    }
}
```

I've created vectors for each filename and filepath, to instantiate them as strings and to reflect only the filename of the audio tracks.



```
//==============================================================================
bool PlaylistComponent::isInterestedInFileDrag(const StringArray& files)
{
    return true; // allows files to be dragged and dropped
}

void PlaylistComponent::filesDropped(const StringArray& files, int x, int y)
{
    //perform if files have been dropped (mouse released with files)
    for (String filename : files)
    {
        //for each file URL, get filepath and file name
        std::string filepath = String(filename).toStdString();
        std::size_t startFilePos = filepath.find_last_of("/");
        std::size_t startExtPos = filepath.find_last_of(".");
        std::string extn = filepath.substr(startExtPos + 1, filepath.length() - startExtPos);
        std::string file = filepath.substr(startFilePos + 1, filepath.length() - startFilePos - extn.size() - 2);

        //update vectors for file details
        inputFiles.push_back(filepath);
        trackTitles.push_back(file);

        //compute adudio length of the file and update vectors for file details
        getAudioLength(URL{ File{filepath} });

    }
    //Initialise interested titles as the full list.
    //This will be updated when text is entered in the search bar
    interestedTitles = trackTitles;
    interestedFiles = inputFiles;

    //update the music library table to include added files
    tableComponent.updateContent();
    tableComponent.repaint();
}
```

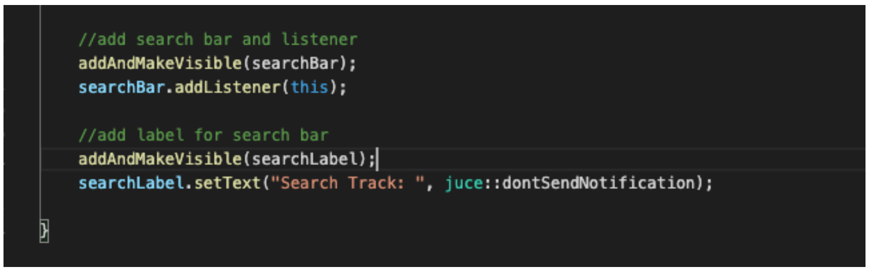
Hence, the outcome would be showing only track titles and the total duration of tracks denoted in seconds, to enable the user to estimate the time of track for remixing.



Track Title	Duration
01-180813_1305	12.989184s
bad_frog	480.917302s
bensound-anewbeginning	154.915669s
bensound-creativeminds	147.168027s
bleep_2	50.101315s

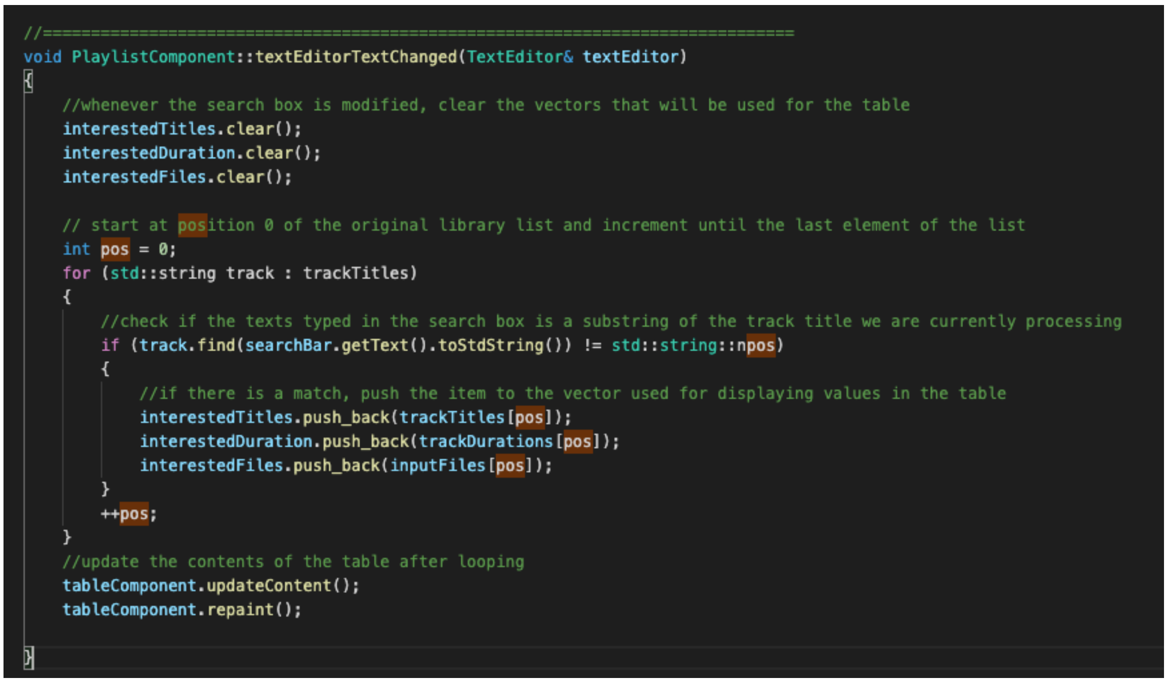
R3C: Component to Search for Audio Tracks in Library (Playlist Component)

In my PlaylistComponent.cpp, I've made a listener for the search bar and label it, so that users are able to search for particular audio tracks in the audio tracks dropped in the playlist.



```
    //add search bar and listener
    addAndMakeVisible(searchBar);
    searchBar.addListener(this);

    //add label for search bar
    addAndMakeVisible(searchLabel);
    searchLabel.setText("Search Track: ", juce::dontSendNotification);

}
```

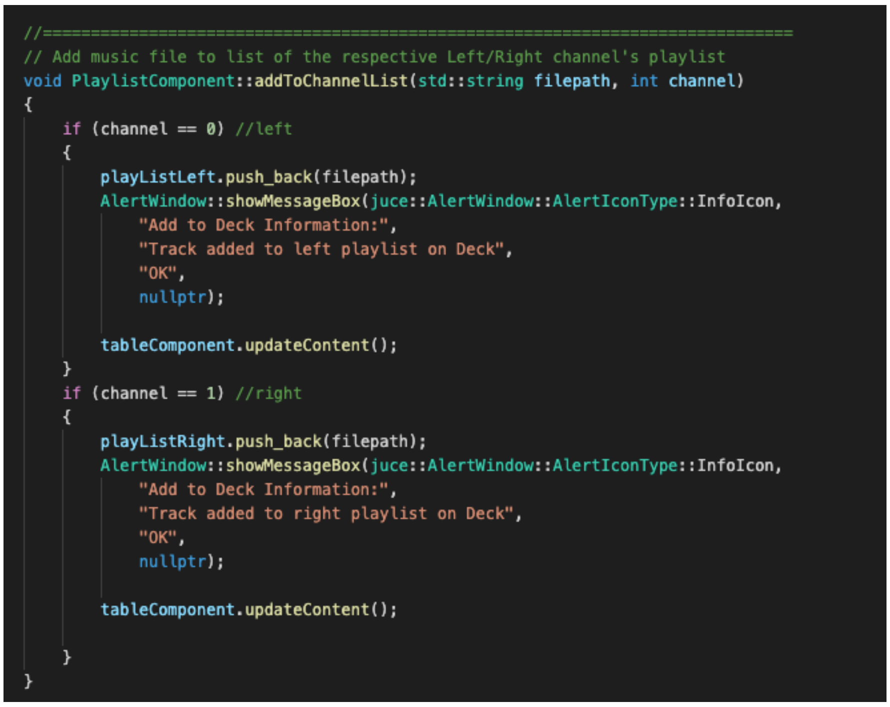
Also in my PlaylistComponent.cpp, I have a textEditor function that is for the search bar to be cleared and retrieve the titles of the specific track the user wants to find, by matching the string elements, as the function searches through from the first position to the last position of the track in the playlist library, as a string.



```
//==============================================================================
void PlaylistComponent::textEditorTextChanged(TextEditor& textEditor)
{
    //whenever the search box is modified, clear the vectors that will be used for the table
    interestedTitles.clear();
    interestedDuration.clear();
    interestedFiles.clear();

    // start at position 0 of the original library list and increment until the last element of the list
    int pos = 0;
    for (std::string track : trackTitles)
    {
        //check if the texts typed in the search box is a substring of the track title we are currently processing
        if (track.find(searchBar.getText().toStdString()) != std::string::npos)
        {
            //if there is a match, push the item to the vector used for displaying values in the table
            interestedTitles.push_back(trackTitles[pos]);
            interestedDuration.push_back(trackDurations[pos]);
            interestedFiles.push_back(inputFiles[pos]);
        }
        ++pos;
    }
    //update the contents of the table after looping
    tableComponent.updateContent();
    tableComponent.repaint();

}
```

<u>R3D: Enable User to Load Audio Track Files from Playlist to Deck</u>

To load the audio track files from the playlist to the deck audio player, I've created a "addToChannelList" function to be called upon, when the user clicks on the button to load tracks, to either the left audio player or the right audio player.

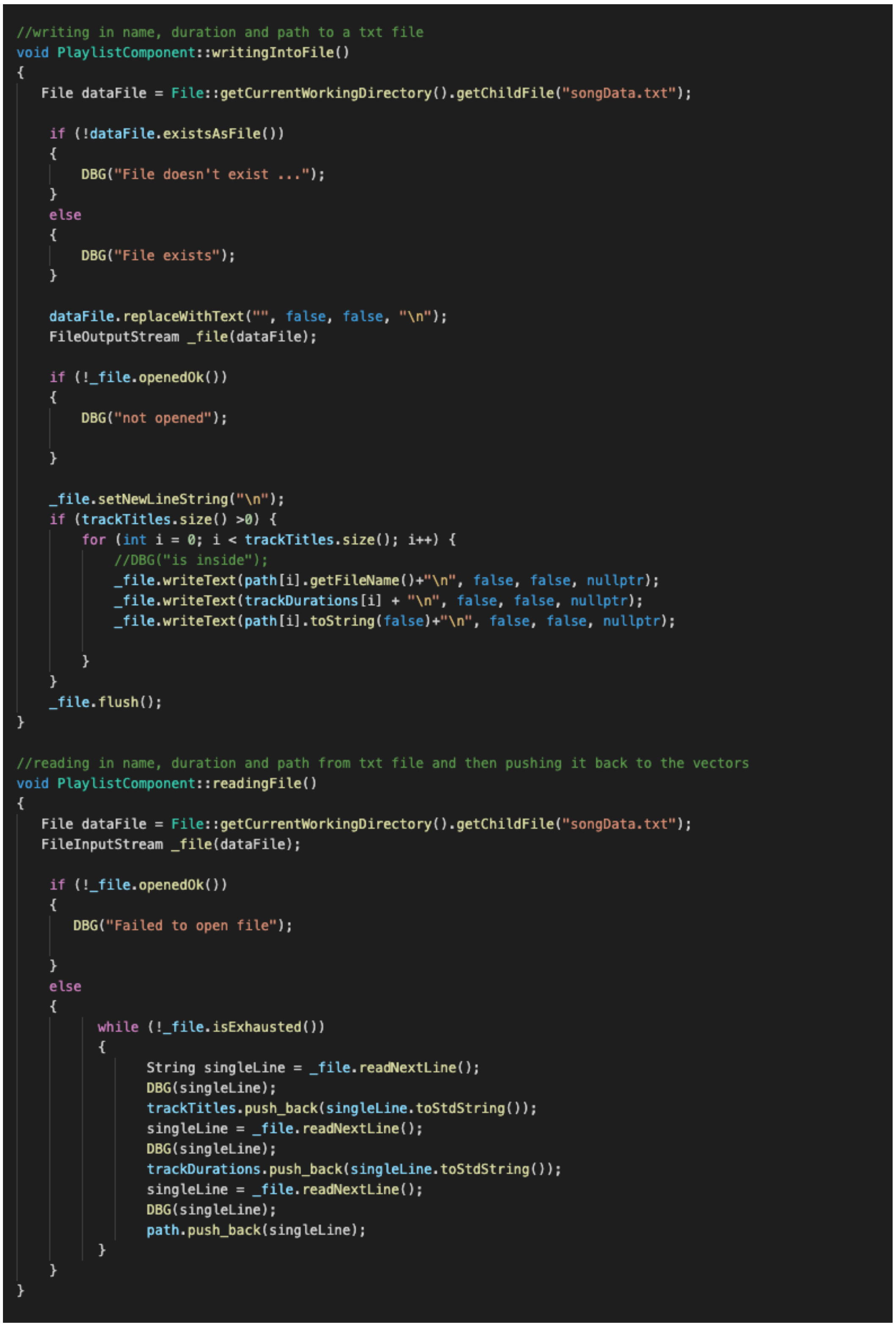


```cpp
//==============================================================================
// Add music file to list of the respective Left/Right channel's playlist
void PlaylistComponent::addToChannelList(std::string filepath, int channel)
{
    if (channel == 0) //left
    {
        playListLeft.push_back(filepath);
        AlertWindow::showMessageBox(juce::AlertWindow::AlertIconType::InfoIcon,
            "Add to Deck Information:",
            "Track added to left playlist on Deck",
            "OK",
            nullptr);

        tableComponent.updateContent();
    }
    if (channel == 1) //right
    {
        playListRight.push_back(filepath);
        AlertWindow::showMessageBox(juce::AlertWindow::AlertIconType::InfoIcon,
            "Add to Deck Information:",
            "Track added to right playlist on Deck",
            "OK",
            nullptr);

        tableComponent.updateContent();

    }
}
```

In the function, the user will also receive a pop up message to indicate that the audio track has been added to the correct deck (AlertWindow).

R3E: Storing and Restoring of Playlist

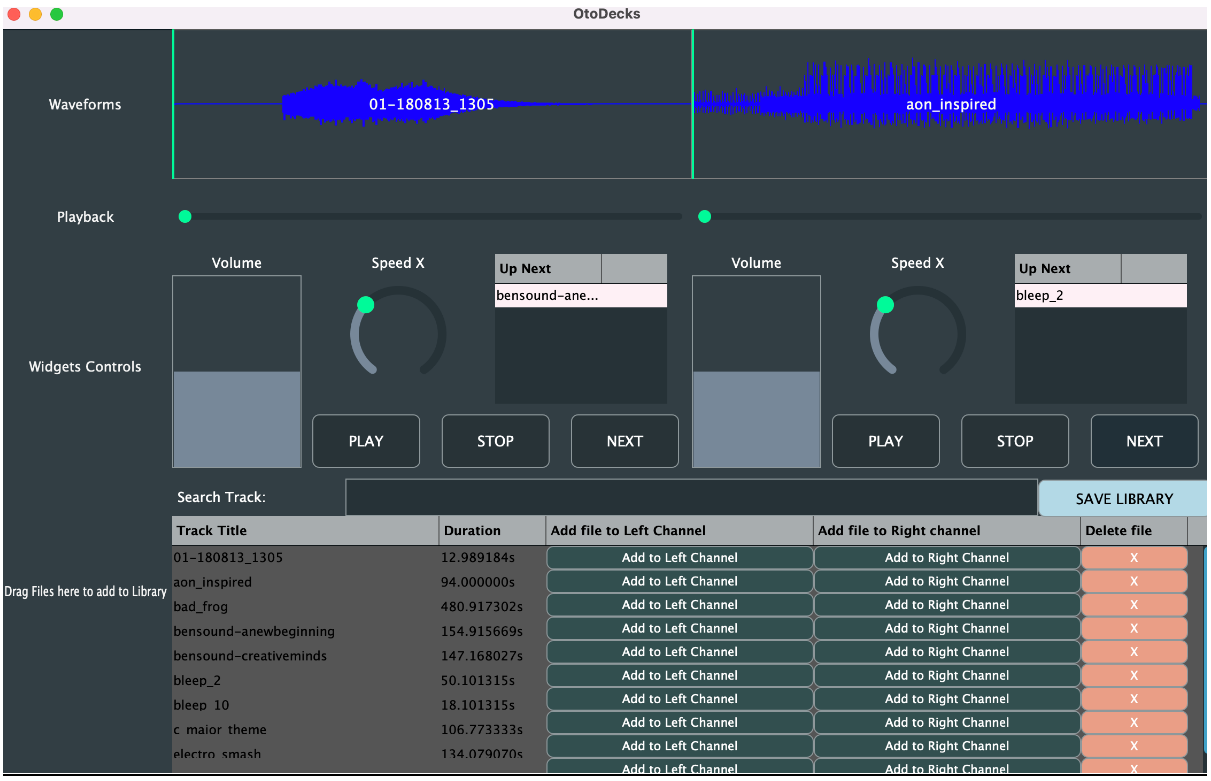
For the storing of library of audio tracks in the playlist, I've implemented a conversion of data from the library of audio tracks in the playlist to be extracted into a text file. The respective fields to be saved are the track titles, duration, and the file's pathway. The function will create a text file for the data to be saved in, and when the Otodecks application is opened again, it will automatically read the text.file and input the data in the playlist again. The data are saved as strings and spaces to differentiate the different fields.



```cpp
//writing in name, duration and path to a txt file
void PlaylistComponent::writingIntoFile()
{
   File dataFile = File::getCurrentWorkingDirectory().getChildFile("songData.txt");

    if (!dataFile.existsAsFile())
    {
        DBG("File doesn't exist ...");
    }
    else
    {
        DBG("File exists");
    }

    dataFile.replaceWithText("", false, false, "\n");
    FileOutputStream _file(dataFile);

    if (!_file.openedOk())
    {
        DBG("not opened");

    }

    _file.setNewLineString("\n");
    if (trackTitles.size() >0) {
        for (int i = 0; i < trackTitles.size(); i++) {
            //DBG("is inside");
            _file.writeText(path[i].getFileName()+"\n", false, false, nullptr);
            _file.writeText(trackDurations[i] + "\n", false, false, nullptr);
            _file.writeText(path[i].toString(false)+"\n", false, false, nullptr);

        }
    }
    _file.flush();
}

//reading in name, duration and path from txt file and then pushing it back to the vectors
void PlaylistComponent::readingFile()
{
   File dataFile = File::getCurrentWorkingDirectory().getChildFile("songData.txt");
   FileInputStream _file(dataFile);

    if (!_file.openedOk())
    {
       DBG("Failed to open file");

    }
    else
    {
          while (!_file.isExhausted())
          {
                String singleLine = _file.readNextLine();
                DBG(singleLine);
                trackTitles.push_back(singleLine.toStdString());
                singleLine = _file.readNextLine();
                DBG(singleLine);
                trackDurations.push_back(singleLine.toStdString());
                singleLine = _file.readNextLine();
                DBG(singleLine);
                path.push_back(singleLine);
          }
    }
}
```

R4: Customising GUI layout

I've completely customised my GUI layout for my Otodecks application, differing from the program taught in class, with keeping it basic and user-friendly at the same time



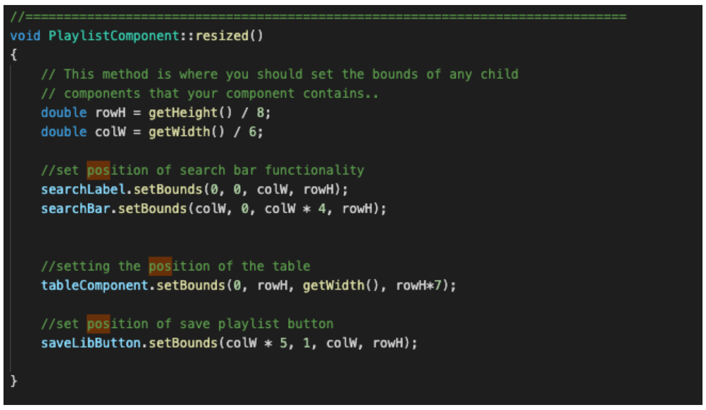
Track Title	Duration	Add file to Left Channel	Add file to Right channel	Delete file
01-180813_1305	12.989184s	Add to Left Channel	Add to Right Channel	X
aon_inspired	94.000000s	Add to Left Channel	Add to Right Channel	X
bad_frog	480.917302s	Add to Left Channel	Add to Right Channel	X
bensound-anewbeginning	154.915669s	Add to Left Channel	Add to Right Channel	X
bensound-creativeminds	147.168027s	Add to Left Channel	Add to Right Channel	X
bleep_2	50.101315s	Add to Left Channel	Add to Right Channel	X
bleep_10	18.101315s	Add to Left Channel	Add to Right Channel	X
c_major_theme	106.773333s	Add to Left Channel	Add to Right Channel	X
electro_smash	134.079070s	Add to Left Channel	Add to Right Channel	X
		Add to Left Channel	Add to Right Channel	X

The coding for the customisations are made in each component, namely DeckGUI, PlaylistComponent, and DJAudioPlayer.

The customisations are mostly done in the "paint" and "resized" of the different components. The buttons and colours are also changed for it to be more prominent for the user.

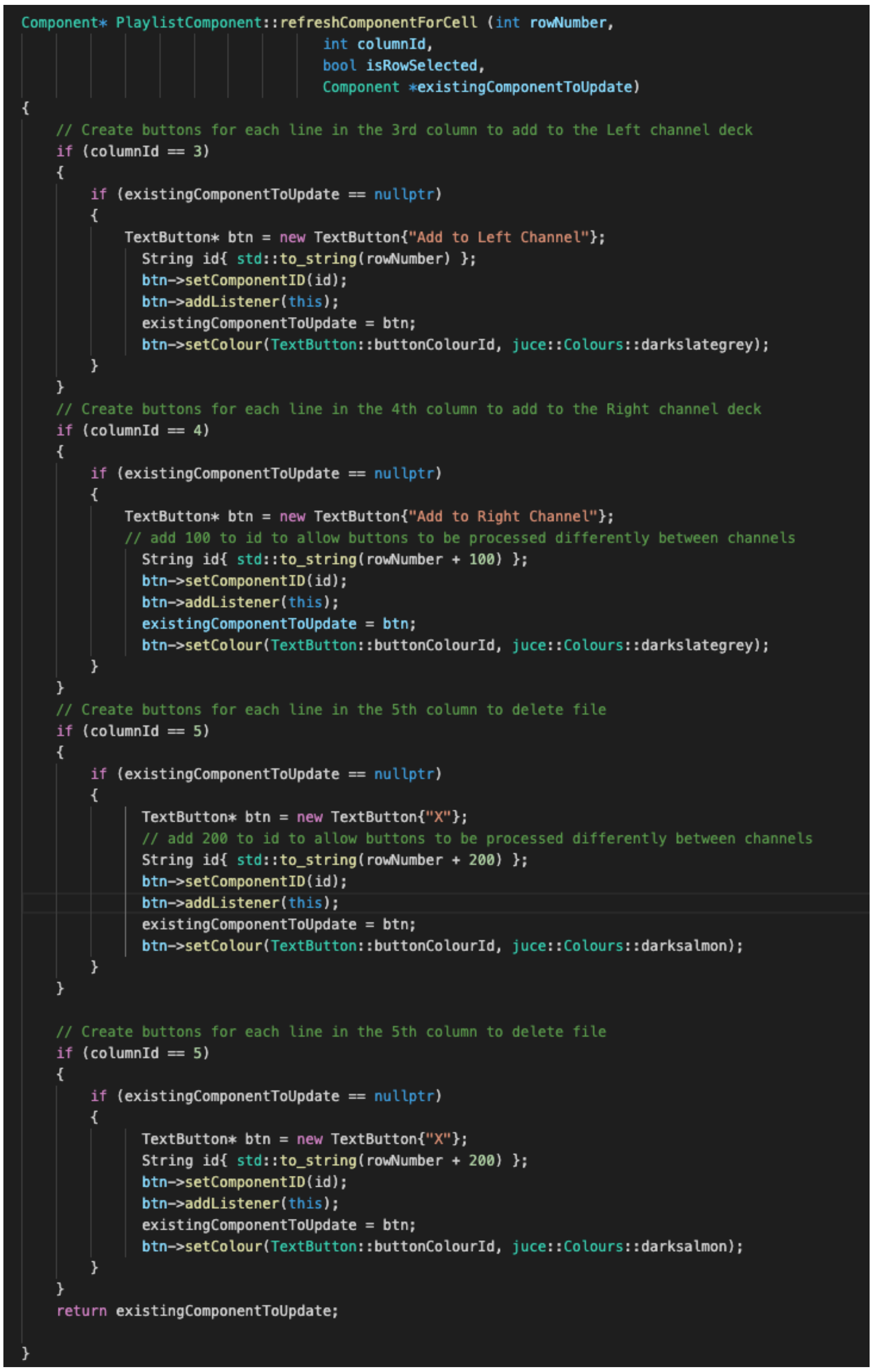
The code extracts below are examples of the customisations that are done:

PlaylistComponents.cpp; for the formatting of the playlist (library)



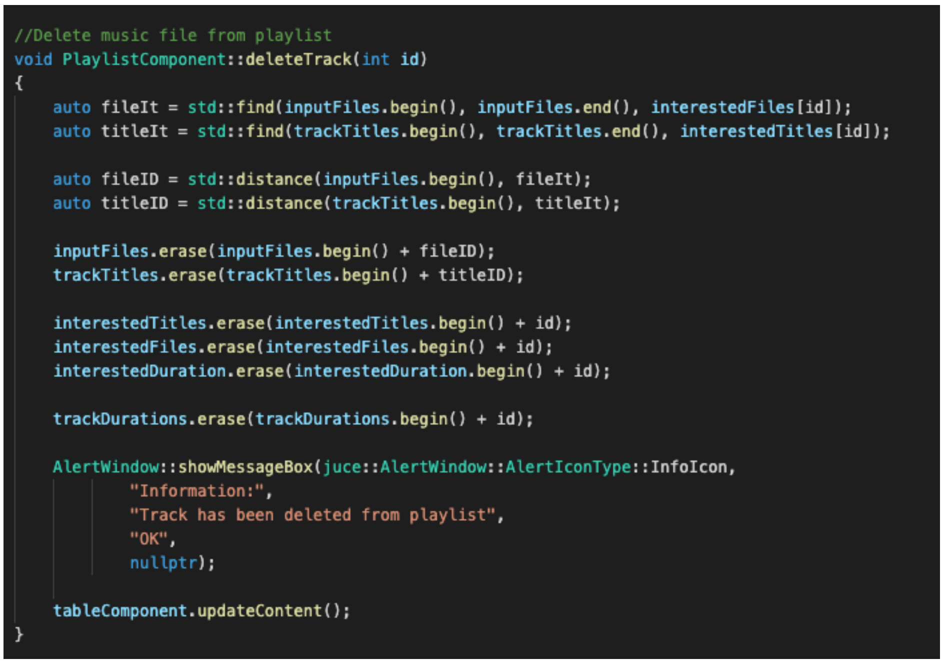
```
//==============================================================================
void PlaylistComponent::resized()
{
    // This method is where you should set the bounds of any child
    // components that your component contains..
    double rowH = getHeight() / 8;
    double colW = getWidth() / 6;

    //set position of search bar functionality
    searchLabel.setBounds(0, 0, colW, rowH);
    searchBar.setBounds(colW, 0, colW * 4, rowH);


    //setting the position of the table
    tableComponent.setBounds(0, rowH, getWidth(), rowH*7);

    //set position of save playlist button
    saveLibButton.setBounds(colW * 5, 1, colW, rowH);

}
```

PlaylistComponents.cpp; for buttons in playlist (library)



```cpp
Component* PlaylistComponent::refreshComponentForCell (int rowNumber,
                                                        int columnId,
                                                        bool isRowSelected,
                                                        Component *existingComponentToUpdate)
{
    // Create buttons for each line in the 3rd column to add to the Left channel deck
    if (columnId == 3)
    {
        if (existingComponentToUpdate == nullptr)
        {
            TextButton* btn = new TextButton{"Add to Left Channel"};
              String id{ std::to_string(rowNumber) };
              btn->setComponentID(id);
              btn->addListener(this);
              existingComponentToUpdate = btn;
              btn->setColour(TextButton::buttonColourId, juce::Colours::darkslategrey);
        }
    }
    // Create buttons for each line in the 4th column to add to the Right channel deck
    if (columnId == 4)
    {
        if (existingComponentToUpdate == nullptr)
        {
            TextButton* btn = new TextButton{"Add to Right Channel"};
            // add 100 to id to allow buttons to be processed differently between channels
              String id{ std::to_string(rowNumber + 100) };
              btn->setComponentID(id);
              btn->addListener(this);
              existingComponentToUpdate = btn;
              btn->setColour(TextButton::buttonColourId, juce::Colours::darkslategrey);
        }
    }
    // Create buttons for each line in the 5th column to delete file
    if (columnId == 5)
    {
        if (existingComponentToUpdate == nullptr)
        {
              TextButton* btn = new TextButton{"X"};
              // add 200 to id to allow buttons to be processed differently between channels
              String id{ std::to_string(rowNumber + 200) };
              btn->setComponentID(id);
              btn->addListener(this);
              existingComponentToUpdate = btn;
              btn->setColour(TextButton::buttonColourId, juce::Colours::darksalmon);
        }
    }

    // Create buttons for each line in the 5th column to delete file
    if (columnId == 5)
    {
        if (existingComponentToUpdate == nullptr)
        {
              TextButton* btn = new TextButton{"X"};
              String id{ std::to_string(rowNumber + 200) };
              btn->setComponentID(id);
              btn->addListener(this);
              existingComponentToUpdate = btn;
              btn->setColour(TextButton::buttonColourId, juce::Colours::darksalmon);
        }
    }
    return existingComponentToUpdate;

}
```

The code extract below is the function to be called upon when the "Delete" button is clicked on, to remove the audio tracks from the playlist. It is performed by erasing the pathways and id(s) of the files from all vectors.



```
//Delete music file from playlist
void PlaylistComponent::deleteTrack(int id)
{
    auto fileIt = std::find(inputFiles.begin(), inputFiles.end(), interestedFiles[id]);
    auto titleIt = std::find(trackTitles.begin(), trackTitles.end(), interestedTitles[id]);

    auto fileID = std::distance(inputFiles.begin(), fileIt);
    auto titleID = std::distance(trackTitles.begin(), titleIt);

    inputFiles.erase(inputFiles.begin() + fileID);
    trackTitles.erase(trackTitles.begin() + titleID);

    interestedTitles.erase(interestedTitles.begin() + id);
    interestedFiles.erase(interestedFiles.begin() + id);
    interestedDuration.erase(interestedDuration.begin() + id);

    trackDurations.erase(trackDurations.begin() + id);

    AlertWindow::showMessageBox(juce::AlertWindow::AlertIconType::InfoIcon,
            "Information:",
            "Track has been deleted from playlist",
            "OK",
            nullptr);

    tableComponent.updateContent();
}
```

<u>Conclusion:</u>

In conclusion, the Otodecks application is an extremely simplified DJ audio player that allows new budding DJs to experiment remixing of audio tracks with a user-friendly interface. Simple elements like volume, speed and playback of audio tracks, enable a user that has the intention to remix audio tracks to innovate and audialize how their remix sound.

A further development that can be made is to allow the user to remix more audio tracks, with more than 2 players, and to allow the user to save the remixed track in their archive for future reference. Also, further elements to remix the tracks like disk scratching and looping of audio tracks can be looked into to enhance the remix experience.

The process of developing and programming this application is not easy as it's a pretty new concept to me and a lot of experimentation with codes are done to achieve the desired outcome. Some functions like the delete function and the save and load playlist function may have an error when it is being called upon and instantiated, but I've tried my utmost best in enabling the functions to work.