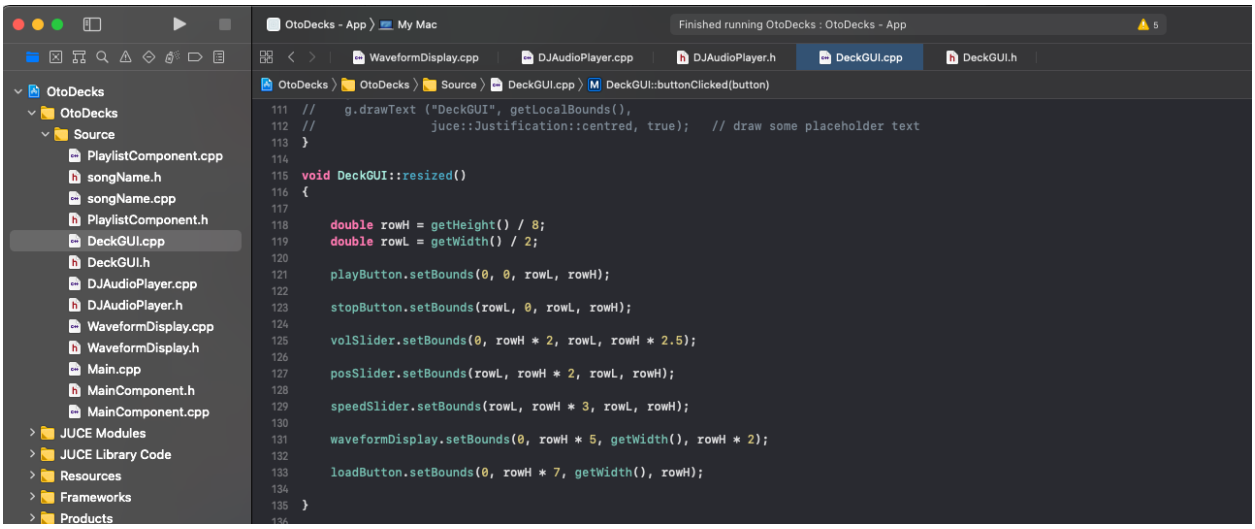
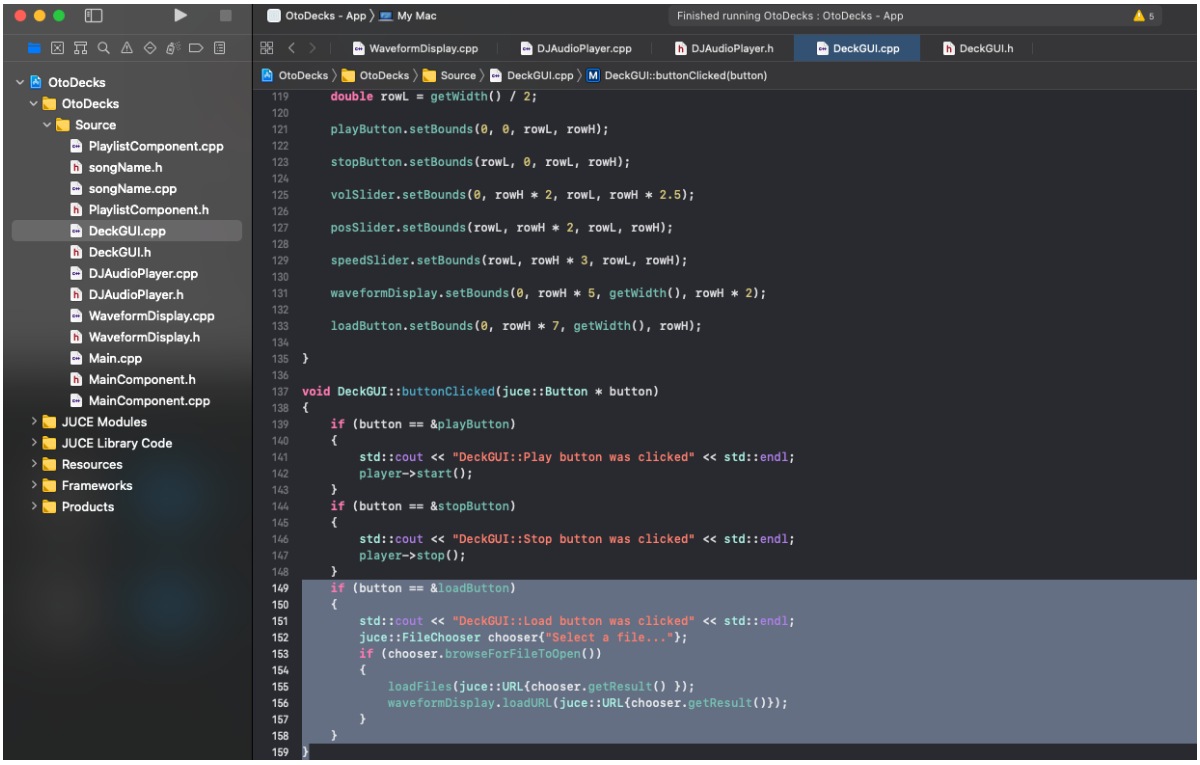
Requirements:

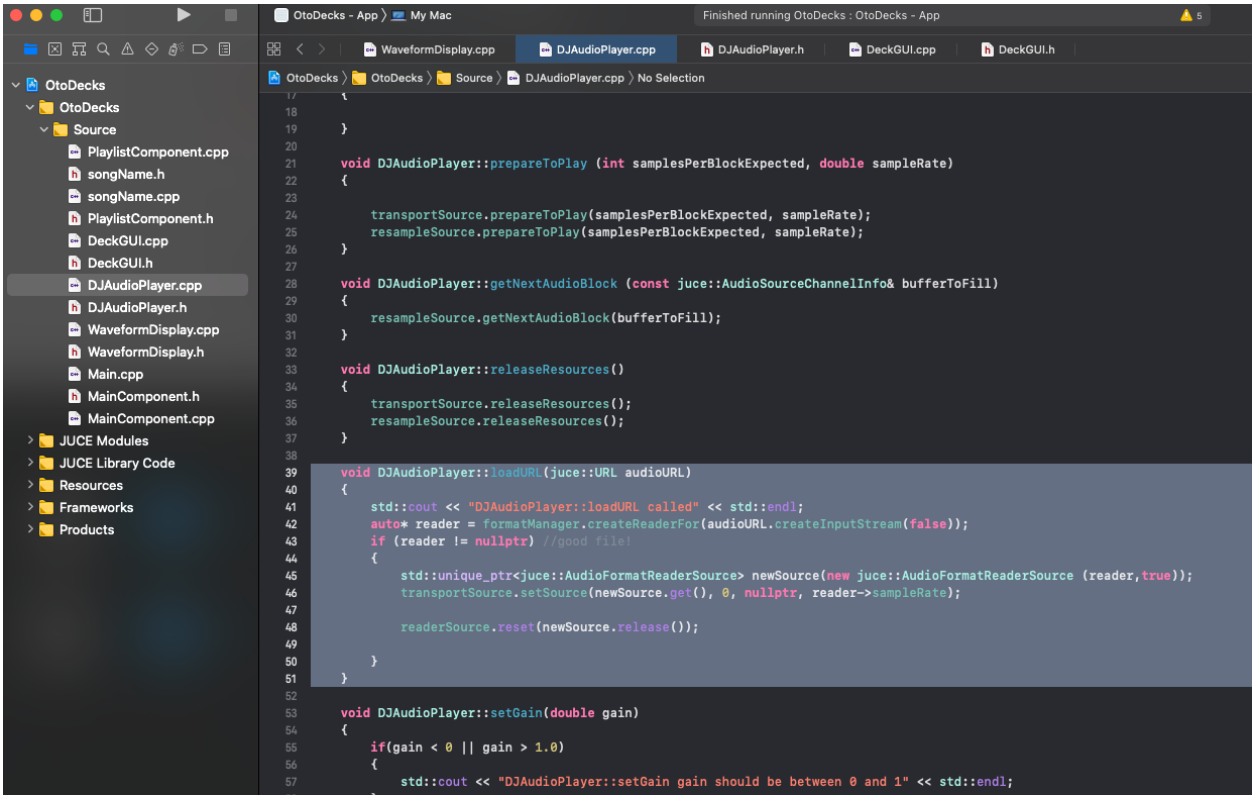
R1A: can load audio files into audio players



This shows where all the buttons will be placed. Load Button will be placed at the most bottom part of the player. Meanwhile, from this image itself, we can analyse where the buttons and sliders will be placed in the player.

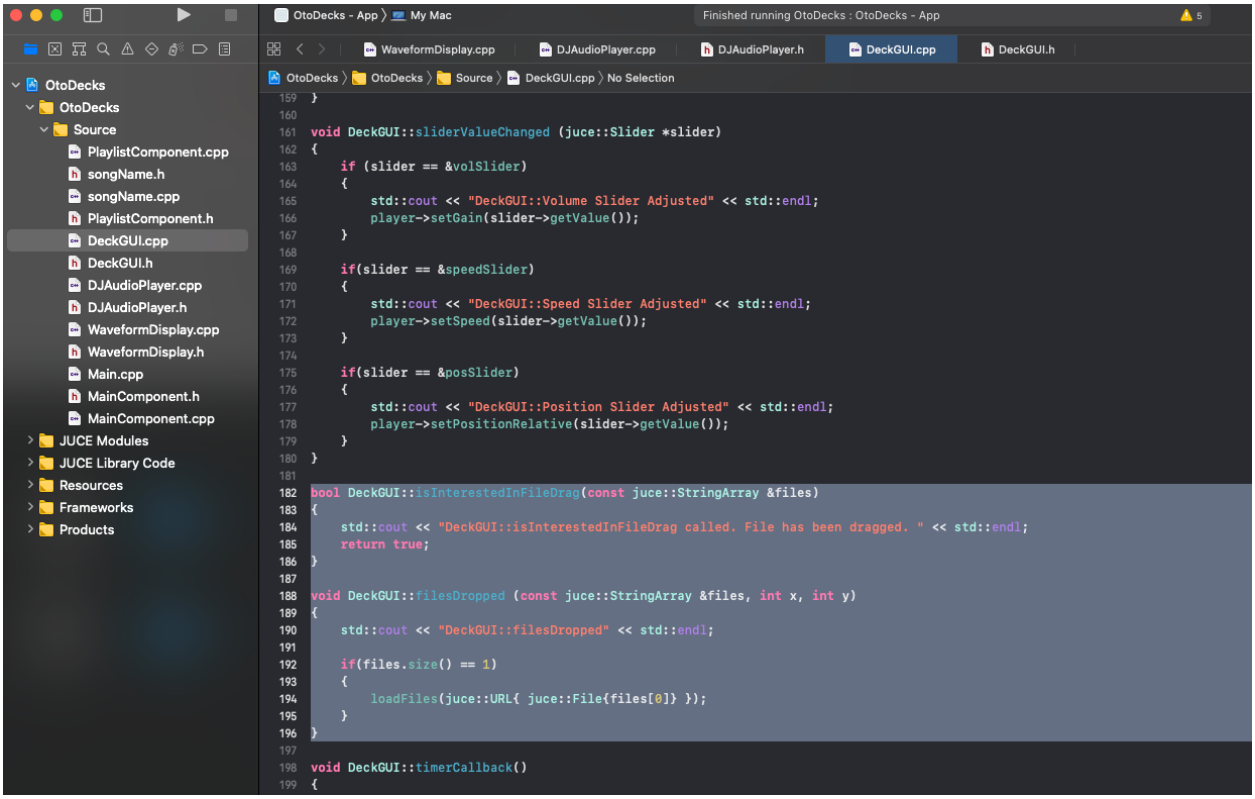


The highlighted part above will implement the ButtonClicked function. Therefore, when the loadButton is being clicked, it will prompt the user to select a song from their computer, in which it will be loaded into the player.




```
206 }
207
208 void DeckGUI::loadFiles(juce::URL audioFileURL)
209 {
210     std::cout << "DeckGUI::loadFiles called." << std::endl;
211     player -> loadURL(audioFileURL);
212     waveformDisplay.loadURL(audioFileURL);
213 }
214
```

The highlighted part above will help to load the song into the player, and read it as well.



The highlighted part above will allow the user to drag a song from their computer into the player, and once dragged, it will be loaded into the player.

R1B: can play two or more tracks

```
136
137 void DeckGUI::buttonClicked(juce::Button * button)
138 {
139     if (button == &playButton)
140     {
141         std::cout << "DeckGUI::Play button was clicked" << std::endl;
142         player->start();
143     }
144     if (button == &stopButton)
145     {
146         std::cout << "DeckGUI::Stop button was clicked" << std::endl;
147         player->stop();
148     }
149     if (button == &loadButton)
150     {
151         std::cout << "DeckGUI::Load button was clicked" << std::endl;
152         juce::FileChooser chooser{"Select a file..."};
153         if (chooser.browseForFileToOpen())
154         {
155             loadFiles(juce::URL{chooser.getResult() });
156             waveformDisplay.loadURL(juce::URL{chooser.getResult()});
157         }
158     }
159 }
160
```

This image above shows what happens when the play button is being clicked. Once play button is clicked, the song which has been loaded will start playing. Since I have two deckGUI, I am able to load two respective songs on respective DeckGUI, and play two songs by clicking the "PLAY" button.

R1C: can mix the tracks by varying each of their volumes

```
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);
    }
}
```

The image above shows that Volume has been set from a range of 0 to 1.0. 0 means that there will be no sound at all, while 1.0 being the loudest. The volume can be adjusted via the slider.

```
void DeckGUI::sliderValueChanged (juce::Slider *slider)
{
    if (slider == &volSlider)
    {
        std::cout << "DeckGUI::Volume Slider Adjusted" << std::endl;
        player->setGain(slider->getValue());
    }
```

The image above shows that Volume can be adjusted via a slider. It will also show the value of the Volume on the player.

R1D: can speed up and slow down the tracks

```
    if(slider == &speedSlider)
    {
        std::cout << "DeckGUI::Speed Slider Adjusted" << std::endl;
        player->setSpeed(slider->getValue());
    }
```

The image above shows that Speed can be adjusted via a slider. It will also show the value of the Speed on the player.

```
void DJAudioPlayer::setSpeed(double ratio)
{
    if(ratio < 0.10 || ratio > 5.0 )
    {
        std::cout << "DJAudioPlayer::setSpeed ratio should be between 0.10 and 5.0" << std::endl;
    }
    else {
        resampleSource.setResamplingRatio(ratio);
    }
}
```

The image above shows that Speed has been set from a range of 0.10 to 5.0. The lower the value, the lower the speed and 0.1 being the minimum. While 5.0, the maximum value, being the fastest. The Speed can be adjusted via the slider.

R2A: Component has custom graphics implemented in a paint function

```
class OtherLookAndFeel : public juce::LookAndFeel_V4
{
public:
    OtherLookAndFeel()
    {
        setColour(juce::Slider::rotarySliderOutlineColourId, juce::Colours::blanchedalmond);
        setColour(juce::Slider::rotarySliderFillColourId, juce::Colours::sandybrown);
        setColour(juce::Slider::thumbColourId, juce::Colours::red);

    }
};
```

From this image above, we are trying to change the appearance of the slider, especially the thumbcolour and the rotary slider fill colours.

R2B: Component enables the user to control the playback of a deck somehow

```
void DeckGUI::buttonClicked(juce::Button * button)
{
    if (button == &playButton)
    {
        std::cout << "DeckGUI::Play button was clicked" << std::endl;
        player->start();
    }
    if (button == &stopButton)
    {
        std::cout << "DeckGUI::Stop button was clicked" << std::endl;
        player->stop();
    }
    if (button == &loadButton)
    {
        std::cout << "DeckGUI::Load button was clicked" << std::endl;
        juce::FileChooser chooser{"Select a file..."};
        if (chooser.browseForFileToOpen())
        {
            loadFiles(juce::URL{chooser.getResult() });
            waveformDisplay.loadURL(juce::URL{chooser.getResult()});
        }
    }
}

void DeckGUI::sliderValueChanged (juce::Slider *slider)
{
    if (slider == &volSlider)
    {
        std::cout << "DeckGUI::Volume Slider Adjusted" << std::endl;
        player->setGain(slider->getValue());
    }

    if(slider == &speedSlider)
    {
        std::cout << "DeckGUI::Speed Slider Adjusted" << std::endl;
        player->setSpeed(slider->getValue());
    }

    if(slider == &posSlider)
    {
        std::cout << "DeckGUI::Position Slider Adjusted" << std::endl;
        player->setPositionRelative(slider->getValue());
    }
}
```

This image shows that the users will be able to start and stop a song. Besides that, they are able to adjust the volume of the song too, and make necessary changes on the speed as well. They are able to play the song from a desired position.

R3A: Component allows the user to add files to their library

```
198
199 void PlaylistComponent::loadInPlayer(DeckGUI* deckGUI)
200 {
201     int selectedRow{songsLibrary.getSelectedRow() };
202     if (selectedRow != -1)
203     {
204         std::cout<< songs[selectedRow].title << " - PlaylistComponent:: Added to Library " << std::endl;
205         deckGUI->loadFiles(songs[selectedRow].URL);
206     }
207     else
208     {
209         juce::AlertWindow::showMessageBox(juce::AlertWindow::AlertIconType::QuestionIcon,
210             "MORE INFORMATION:",
211             "Please pick a song above to add to deck!",
212             "OK",
213             nullptr
214         );
215     }
216 }
217
218 void PlaylistComponent::importToSongsLibrary()
219 {
220     std::cout<< "PlaylistComponent::importToLibrary called" << std::endl;
221
222     //start to initiate song chooser
223     juce::FileChooser chooser{ "Select songs" };
224     if (chooser.browseForMultipleFilesToOpen())
225     {
226         for (const juce::File& file : chooser.getResults())
227         {
228             juce::String songNameWithoutExtension{ file.getFileNameWithoutExtension() };
229             if (!isInSongs(songNameWithoutExtension)) // if song has not been loaded
230             {
231                 songName newSong{ file };
232                 juce::URL audioFileURL{ file };
233                 songs.push_back(newSong);
234                 std::cout<< "Loaded file: " << newSong.title << std::endl;
235             }
236             else
237                 // error message will pop out
238             {
239                 juce::AlertWindow::showMessageBox(juce::AlertWindow::AlertIconType::WarningIcon,
240                     "ERROR:",
241                     songNameWithoutExtension + " has already been loaded!",
242                     "CLOSE",
243                     nullptr
244                 );
245             }
246         }
247     }
248 }
249
```

From the image above, this will allow users to add new songs into their library. User need to click “Import New Songs” on the top left hand corner, choose the song(s) which they want to load, then it will be loaded in the library. However, if the song has already been loaded, a message windows box will appear, stating that the song chosen has already been loaded in the library.

R3B: Component parses and displays meta data such as filename and song length

```
// setting up the songs Library with 3 columns
songsLibrary.getHeader().addColumn("Songs", 1, 1);
songsLibrary.getHeader().addColumn("Discard?", 2, 1);
songsLibrary.getHeader().addColumn("Length", 3, 1);
songsLibrary.setModel(this);
loadSongsFromLibrary();
```

From the image above, this will set up my Songs Library with 3 columns. It will display the Song Name, an option whether you want to remove the song, and the length as the header.

```
void PlaylistComponent::importToSongsLibrary()
{
    std::cout<< "PlaylistComponent::importToLibrary called" << std::endl;

    //start to initiate song chooser
    juce::FileChooser chooser{ "Select songs" };
    if (chooser.browseForMultipleFilesToOpen())
    {
        for (const juce::File& file : chooser.getResults())
        {
            juce::String songNameWithoutExtension{ file.getFileNameWithoutExtension() };
            if (!isInSongs(songNameWithoutExtension)) // if song has not been loaded
            {
                songName newSong{ file };
                juce::URL audioFileURL{ file };
                songs.push_back(newSong);
                std::cout<< "Loaded file: " << newSong.title << std::endl;
            }
```

This will make the name of songs be visible on the Songs Library.

R3C: Component allows the user to search for files

```
251 void PlaylistComponent::searchSongsLibrary(juce::String searchForText)
252 {
253     std::cout<< "PlaylistComponent:: Searching Songs Library for " << searchForText << std::endl;
254
255     if (searchForText != "")
256     {
257         int rowNumber = whereInSongs(searchForText);
258         songsLibrary.selectRow(rowNumber);
259     }
260     else
261     {
262         songsLibrary.deselectAllRows();
263     }
264 }
265
266 int PlaylistComponent::whereInSongs(juce::String searchForText)
267 {
268     // searches the index where song has the searchBox
269     auto it = find_if(songs.begin(), songs.end(),
270         [&searchForText](const songName& obj) {return obj.title.contains(searchForText); });
271     int i = -1;
272
273     if (it != songs.end())
274     {
275         i = std::distance(songs.begin(), it);      ⚠ Implicit conversion loses integer precision: 'typename iterator_traits<__
276     }
277
278     return i;
279 }
```

From the image above, user can search the songs from the library. All they need to do is to type inside the “Search for desired song - press enter” If the file is found in the library, a light blue row will be highlighted. However, once typed, and if the song is not available in the library, no rows will be highlighted;

R3D: Component allows the user to load files from the library into a deck

```
    }
    else if (button == &addToDeck1Button)
    {
        std::cout<<"PlaylistComponent::Add To Deck1 Button Clicked " << std::endl;
        loadInPlayer(deckGUI1);
    }
    else if (button == &addToDeck2Button)
    {
        std::cout<<"PlaylistComponent::Add To Deck2 Button Clicked " << std::endl;
        loadInPlayer(deckGUI2);
    }
    else
    {
    int id = std::stoi(button->getComponentID().toStdString());
    std::cout<< songs[id].title << " - PlaylistComponent:: Removed from Library " << std::endl;
    removeSongsFromLibrary(id);
    songsLibrary.updateContent();

//    std::cout<<"PlaylistComponent::buttonClicked " << songs[id] << std::endl;

    }
}
```

From the image above, user can choose a song from in the library, then once it is highlighted in light blue, they can have the option to add the song into Deck: 1 or Deck: 2 by clicking the respective buttons on the bottom right corner, depending on which Deck they want to load it to.

R3E: The music library persists so that it is restored when the user exits and restarts the app

```
282 void PlaylistComponent::saveSongsToLibrary()
283 {
284     // generate a .csv file to store Songs library
285     std::ofstream myLibrary("mysongslibrary.csv");
286 
287     // save Songs library to .csv file
288     for (songName& s : songs)
289     {
290         myLibrary << s.file.getFullPathName() << "," << s.length << "\n";
291     }
292 }
293 
294 
295 void PlaylistComponent::loadSongsFromLibrary()
296 {
297     // create input stream from saved Songs library
298     std::ifstream myLibrary("mysongslibrary.csv");
299     std::string filePath;
300     std::string length;
301 
302     // to analyse the data
303     if (myLibrary.is_open())
304     {
305         while (getline(myLibrary, filePath, ',')) {
306             juce::File file{ filePath };
307             songName newSong{ file };
308 
309             getline(myLibrary, length);
310             newSong.length = length;
311             songs.push_back(newSong);
312         }
313     }
314     myLibrary.close();
315 }
316 
```

All songs that has been loaded into the library, will be stored automatically in the mysongslibrary.csv file. Therefore, when user exits the application, and then relaunches the application, the loaded songs will still appear in the library.

R4A: GUI layout is significantly different from the basic DeckGUI shown in class

```
class OtherLookAndFeel : public juce::LookAndFeel_V4
{
public:
    OtherLookAndFeel()
    {
        setColour(juce::Slider::rotarySliderOutlineColourId, juce::Colours::blanchedalmond);
        setColour(juce::Slider::rotarySliderFillColourId, juce::Colours::sandybrown);
        setColour(juce::Slider::thumbColourId, juce::Colours::red);

    }
};
```

R4B: GUI layout includes the custom Component from R2

```
class OtherLookAndFeel : public juce::LookAndFeel_V4
{
public:
    OtherLookAndFeel()
    {
        setColour(juce::Slider::rotarySliderOutlineColourId, juce::Colours::blanchedalmond);
        setColour(juce::Slider::rotarySliderFillColourId, juce::Colours::sandybrown);
        setColour(juce::Slider::thumbColourId, juce::Colours::red);

    }
};
```

From this image above, we are trying to change the appearance of the slider, especially the thumbcolour and the rotary slider fill colours. Therefore, the appearance of the Sliders are different.

```
void WaveformDisplay::paint (juce::Graphics& g)
{
    /* This demo code just fills the component's background and
       draws some placeholder text to get you started.

       You should replace everything in this method with your own
       drawing code..
    */

    g.fillAll (getLookAndFeel().findColour (juce::ResizableWindow::backgroundColourId));   // clear the background

    g.setFont(20.0f);
    g.setColour (juce::Colours::darkorange); //deck label colour
    g.drawText(" Deck: " + std::to_string(id), getLocalBounds(),
              juce::Justification::centredTop, true);

    g.setColour (juce::Colours::lightgrey);
    g.drawRect (getLocalBounds(), 1);   // draw an outline around the component

    g.setColour (juce::Colours::seagreen); // wavelength colour

    if(fileLoaded)
    {
        audioThumb.drawChannel(g,
                               getLocalBounds(),
                               0,
                               audioThumb.getTotalLength(),
                               0,
                               1.0f
                               );
        g.setColour(juce::Colours::white); //file title colour
        g.setFont (20.0f);
        g.drawText(fileTitle, getLocalBounds(),
                juce::Justification::bottomRight, true);

        g.setColour(juce::Colours::darkviolet); // playhead wavelength box colour
        g.drawRect(position * getWidth(), 0, getWidth() / 20, getHeight());
    }
    else
    {
        g.setFont (25.0f);
        g.setColour (juce::Colours::steelblue); // filenotloaded text colour
        g.drawText ("File not loaded...", getLocalBounds(),
                    juce::Justification::centred, true);   // draw some placeholder text
    }

}

```

From the image above, I am trying to change the appearance of the wavelength, and font colours and size.

R4C: GUI layout includes the music library component from R3

```
// setting up the search Box
searchBox.setTextToShowWhenEmpty("Search for Desired Song - Press Enter",
                                 juce::Colours::yellow);
searchBox.onReturnKey = [this] { searchSongsLibrary (searchBox.getText()); };

// setting up the songs Library with 3 columns
songsLibrary.getHeader().addColumn("Songs", 1, 1);
songsLibrary.getHeader().addColumn("Discard?", 2, 1);
songsLibrary.getHeader().addColumn("Length", 3, 1);
songsLibrary.setModel(this);
loadSongsFromLibrary();
```

I have set up a Search Box function where user can search the songs which are already added into the Songs Library. I have also added a table, for my Songs library, with 3 columns, “Songs, Discard and Length”.

```
importButton.setBounds(0, 0, getWidth(), 2 * getHeight() / 18);
searchBox.setBounds(0, 2 * getHeight() / 18, getWidth(), getHeight()/18);
songsLibrary.setBounds(0, 3 * getHeight() / 18, getWidth(), 15 * getHeight() / 18);
addToDeck1Button.setBounds(0, 17 * getHeight() / 18, getWidth() / 2, getHeight() / 18);
addToDeck2Button.setBounds(getWidth() / 2, 17 * getHeight() / 18, getWidth() / 2, getHeight() / 18);

//construct columns
songsLibrary.getHeader().setColumnWidth(1, 10 * getWidth() / 20);
songsLibrary.getHeader().setColumnWidth(2, 4 * getWidth() / 18);
songsLibrary.getHeader().setColumnWidth(3, 5 * getWidth() / 18);
```

This show how I set up my Songs library.

```
106
107 void PlaylistComponent::paintRowBackground(juce::Graphics & g,
108                         int rowNumber,
109                         int width,
110                         int height,
111                         bool rowIsSelected)
112 {
113     if (rowIsSelected)
114     {
115         g.fillAll(juce::Colours::cyan);
116     }
117     else
118     {
119         g.fillAll(juce::Colours::slategrey);
120     }
121 }
122
```

This shows that when row is selected, it will turn cyan in colour. However, when the row is not selected, it remains slate grey in colour.