Name: Sau Ting Lam
Student Number: 200594390

Object Oriented Programming

Coursework for Endterm

Otodecks Report

Content:

Name: Sau Ting Lam
Student Number: 200594390

Introduction

This is a new design DJ application based on Otodecks. This new application was added a custom deck control Component and a music library Component. The music files in the music library Component can be added to Deck GUI.

The new deck control Component have a new custom graphics with new layout design and colour theme. The music library component allow user to manage a library of music within the application. User can search the music name by typing the file name in the music library search bar. After selecting the file, user can load music into the decks.

The application can save the loaded file in the music library automatically, it will show the same loaded files when user reopen the application.

Name: Sau Ting Lam
Student Number: 200594390

R1: The application should contain all the basic functionality shown in class:

(**Image 1**)

R1A: can load audio files into audio players

The application allow user to load file in DeckGUI1 and DeckGUI2 separately. User can either click the button “LOAD A MUSIC FILE” or simply “Drag and drop a file” to load a single music file. Then the file will be loaded into the audio player and presented in the Waveform display box.

R1B: can play two or more tracks

The application allow user to play two tracks at the same time. In (image 1), it shows that music can be played in two different players.

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

The application allow user to change the volumes in each DeckGUI. The volume slider is a rotary style associate with a label “Volume” at the left top corner, so that the user can see clearly. And the volume is between 0 – 1, when the slider changes, the number will be changed as well. Two players can play at the same time, so the user can mix the tracks by varying the volumes in each DeckGUI.

R1D: can speed up and slow down the tracks

The speed slider is a rotary style, user can slide left for slow down and slide right to speed up. The range for speed is 0 to 10. The label “Speed” at the left top corner is associated with the slider. And the speed number will be changed when the slider change.

Name: Sau Ting Lam
Student Number: 200594390

R2: Implementation of a custom deck control Component with custom graphics which allows the user to control deck playback in some way that is more advanced than stop/ start.

(**Image 2**)

R2A: Component has custom graphics implemented in a paint function

In (image 2), it shows that the application has a new layout and style. The waveform display was moved to the top of the Deck. Some new label such as “Position”, “Volume” and “Speed” were added to remind user how to use the sliders. The colour of the outline around the component and text colour were changed.

```
/** volSlider: set the volume style */
volSlider.setRange(0.0, 1.0);
volSlider.setTextBoxStyle(Slider::TextBoxLeft, false, 40, volSlider.getTextBoxHeight());
volSlider.setNumDecimalPlacesToDisplay(2);
volSlider.setValue(0.5);
volSlider.setSkewFactorFromMidPoint(0.5);
/** volSlider: rotary slider */
volSlider.setSliderStyle(Slider::Rotary);
volSlider.setTextBoxStyle(Slider::TextEntryBoxPosition::TextBoxBelow, false, 200, 20);
/** volLabel: create a label next to Volume slider */
volLabel.setText("Volume", dontSendNotification);
volLabel.attachToComponent(&volSlider, false);
```

DeckGUI.cpp (Code 1)

Take an example of volume Slider. In (code 1), it shows that the slider style changed to rotary, the volume number set to 2 decimals instead of the original long number. The set up of the value is 0.5 and the point of the slider set to the middle, when the user open the application, all the initial setting will be just point in the centre middle. The label is created and just attached to the volume slider component. It is the same design the speed and position slider.

Name: Sau Ting Lam
Student Number: 200594390

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

g.setColour (Colours::cadetblue);
g.setFont (14.0f);
g.drawText ("DeckGUI   ", getLocalBounds(),
            juce::Justification::centredRight, true);   // draw some placeholder text
```

DeckGUI.cpp (Code 2)

In (code 2) shows that the outline around the DeckGUI component changed to hotpink colour for highlighting the different between Deck and playlist. The placeholder text changed to cadetblue and placed at the right, as the colour is slightly lighter than normal blue so it will not be too eye catching. It can remind the user where the area they are using.

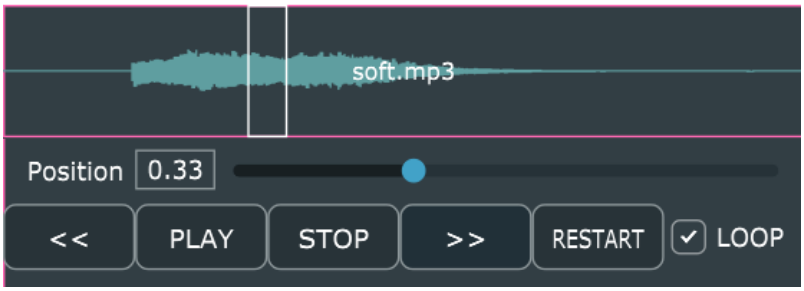
```
g.setColour (juce::Colours::hotpink);
g.drawRect (getLocalBounds(), 1);   // draw an outline around the component

g.setColour (juce::Colours::cadetblue);
if (fileLoaded)
{
    audioThumb.drawChannel(g,
        getLocalBounds(),
        0,
        audioThumb.getTotalLength(),
        0,
        1.0f
    );
    g.setColour(Colours::white);
    g.drawRect(position * getWidth(), 0, getWidth() / 20, getHeight());
    g.drawText(trackName, getLocalBounds(), Justification::centred, true);
}
else
{
    g.setFont(20.0f);
    g.drawText("Drag and drop a file", getLocalBounds(),
                juce::Justification::centred, true);   // draw some placeholder text
}
```

WaveformDisplay.cpp (Code 3)

From (Code 3) and (Image 1), in the waveform display box, it shows the text “Drag and drop a file” with cadetblue colour if the file hasn’t been loaded. If the file is loaded, it displays the file name in the centre with a white colour to remind user what they are playing. And the outline of the component uses the same hotpink colour for highlight.

Name: Sau Ting Lam
Student Number: 200594390

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



(Image 3)

```
/** Buttons  */
TextButton playButton{ "PLAY" };
TextButton stopButton{ "STOP" };
TextButton restartButton{ "RESTART" };
TextButton loadButton{ "LOAD A MUSIC FILE" };
TextButton rewindButton{ "<<" };
TextButton forwardButton{ ">>" };
ToggleButton loopButton{ "LOOP" };
```

DeckGUI.h (Code 4)

```
void DJAudioPlayer::rewind()
{
    //Check if the new position value is more than 0
    if (transportSource.getCurrentPosition() - 1 > 0)
    {
        //set the new position by reducing 1.8 position from the current position
        transportSource.setPosition(transportSource.getCurrentPosition() - 1.8);
    }
}
 //Moving the playback forward
void DJAudioPlayer::forward()
{
    //the last position of transportSource
    double lastPos{ transportSource.getLengthInSeconds() };

    //check if the new position is not equal or greater than the last position
    if (transportSource.getCurrentPosition() + 0.5 != lastPos || transportSource.getCurrentPosition() + 0.5 > lastPos)
    {
        //update the new position by adding 1.5 position
        transportSource.setPosition(transportSource.getCurrentPosition() + 1.5);
    }
}
```

DJAudioPlayer.cpp (Code 5)

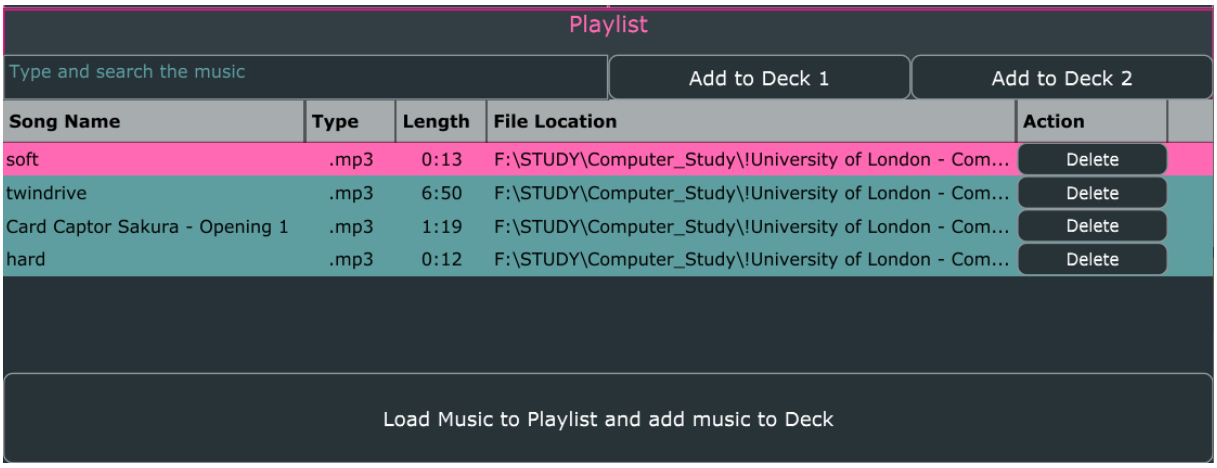
The application keeps the basic play and stop button shown in (image 3). There are some new buttons “restart”, “loop”, “rewind” and “forward” shown in (code 4). The restart button allow user to start over again the music and set back to the 0 position while the music is playing. When the loop button is toggled, it shows a tick in the small box. It means the track will start again the 0 position when the position goes to the end.

User can click “<<” to go backward and “>>” to go forward. In (Code 5) rewind(), I decided to set the position back 1.8 position instead of 1.5 like in forward(). It is because the song keeps moving forward, it will be more obvious when its user can see a big jump back.

To make it nice, it can be adding a hover text so the user can see the hidden message if they don’t understand what the buttons mean.

Name: Sau Ting Lam
Student Number: 200594390

R3: Implementation of a music library component which allows the user to manage their music library



(Image 4)

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

```
//initialize file chooser
auto dlgFlags =
    FileBrowserComponent::openMode |
    FileBrowserComponent::canSelectFiles;

this->chooser.launchAsync(dlgFlags,
    [this](const FileChooser& chooser)
    {
        for (const File& file : chooser.getResults())
        {
            String fileName{ file.getFileNameWithoutExtension() };
            if (!isInPlaylist(fileName)) // if there is not repeated file
            {
                //load file and get the file meta data to the playlist
                TrackData importedFile{ file };
                URL audioURL{ file };
                importedFile.length = getLength(audioURL);
                tracks.push_back(importedFile);
            }
            playlist.updateContent();
        }
    });
```

PlaylistComponent.cpp (Code 6)

In (image 4) shows that the application allow user to load files by clicking the button “Load Music to Playlist and add music to Deck”. As I am using the latest juce projucer and Visual Studio Code 2022, some code has to be changed. For example, the file chooser is changed a bit in (Code 6). The user load and add file one by one to the playlist. Then the playlist will update the content after the file is loaded. And it will read and display the meta data in the playlist.

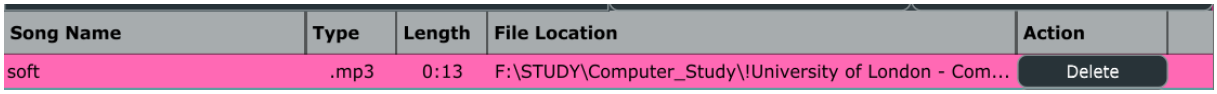
Name: Sau Ting Lam
Student Number: 200594390

```
bool isInterestedInFileDrag(const StringArray& files) override;
void filesDropped(const StringArray& files, int x, int y) override;
```

PlaylistComponent.h (Code 7)

I managed to add drop and drag function in (Code 7), but the playlist couldn't read the meta data. When I hold a file and ready to put in the playlist, the mouse hover changed to "+", but it has no response in the playlist after the mouse released. It is one of the miner things I can not fix before the submission. As the load function is working fine, the user can still load the music file in the playlist.

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



(Image 5)

```
/** Playlist Header */
playlist.getHeader().addColumn("Song Name", 1, 200);
playlist.getHeader().addColumn("Type", 2, 60);
playlist.getHeader().addColumn("Length", 3, 60);
playlist.getHeader().addColumn("File Location", 4, 350);
playlist.getHeader().addColumn("Action", 5, 100);
playlist.setModel(this);
```

PlaylistComponent.cpp (Code 7)

```
TrackData::TrackData(File _file) : file(_file),
fileName(_file.getFileNameWithoutExtension()),
fileType(_file.getFileExtension()),
fileLocation(_file.getFullPathName()),
```

TrackData.cpp (Code 8)

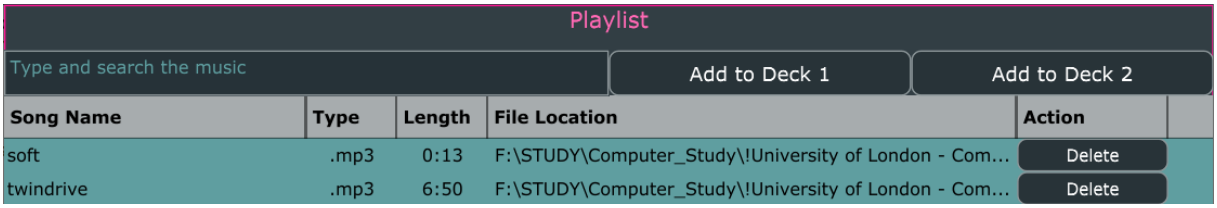
```
String PlaylistComponent::getLength(URL audioURL)
{
    djAudioPlayer->loadURL(audioURL);
    double seconds{ djAudioPlayer->getLengthInSeconds() };
    String minutes{ secondToMinute(seconds) };
    return minutes;
}

String PlaylistComponent::secondToMinute(int seconds)
{
    //Minutes in string
    String minStr{ std::to_string(seconds / 60) };
    //Seconds in string
    String secStr{ std::to_string(seconds % 60) };
    //return minutes and seconds in a single string
    return String{ minStr + ":" + secStr };
}
```

PlaylistComponent.cpp (Code 9)

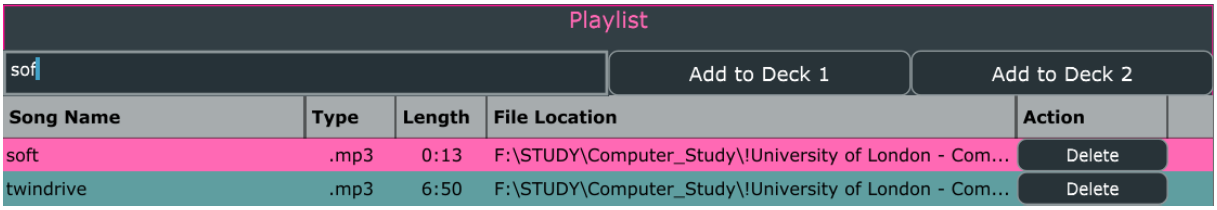
There are 5 columns in the paintCell() will be displayed in the playlist(Image 5). In (Code 7, 8) The "Song Name" is linked to the getFileNameWithoutExtension() that get the simple file name. The "Type" linked to getFileExtension() that get the file format such as .mp3 or wav. The "File Location" linked to getFullPathName() that get the music file directory. The "Length" is the most complicated part (Code 9), the Length is from the audioURL file and push back to the track at the end. Before adding back to the playlist, it has to be converted from second to minute so the user can easily see how the length of the music will be.

Name: Sau Ting Lam
Student Number: 200594390

R3C: Component allows the user to search for files



(Image 6)



(Image 7)

The user can search the music by typing in the search bar (Image 7). Before user typing in the search bar (Image 6), it shows some words “Type and search the music” with cadetblue colour for remining user they can use this search function.

```
/** Text in the box to remind people to search a song */
searchBar.setTextToShowWhenEmpty("Type and search the music", juce::Colours::cadetblue);
/** change the color when the search input is much with the keyword */
searchBar.onTextChange = [this] { searchInPlaylist(searchBar.getText()); };
```

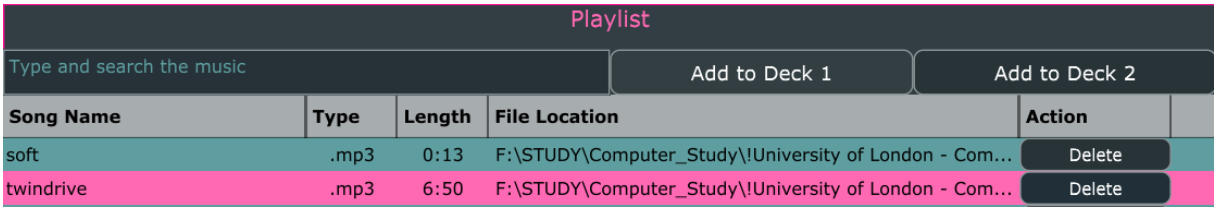
```
void PlaylistComponent::searchInPlaylist(String keyword)
{
    //Search the track in the playlist
    if (keyword != "")
    {
        int rowIdNumber = findTrackName(keyword);
        //change the track color when there are matching keyword
        playlist.selectRow(rowIdNumber);
    }
    else
    {
        //not highlight anythig
        playlist.deselectAllRows();
    }
}
int PlaylistComponent::findTrackName(String keyword)
{
    // finds the index id where the input search contains the keyword
    auto music = find_if(tracks.begin(), tracks.end(),
        [&keyword](const TrackData& obj) {return obj.fileName.contains(keyword); });
    int i = -1;

    if (music != tracks.end())
    {
        //track is found if it is not the end of the search
        i = std::distance(tracks.begin(), music);
    }
    return i;
}
```

PlaylistComponent.cpp (Code 10)

When user type a word (Code 10), the searchInPlaylist() will catch the keyword and pass it in findTrackName(). If the keyword matches the music file name, the selected row in the playlist will highlight with pink colour. If the keyword does not match any file name in the playlist, it highlights nothing.

Name: Sau Ting Lam
Student Number: 200594390

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



(**Image 8**)

```
void PlaylistComponent::buttonClicked(Button* button)
{
  else if (button == &addToDeckGUI1)
  {
      //Load selected music file to deckGUI 1 when button was clicked
      loadToDeckGUI(deckGUI1);
  }
  else if (button == &addToDeckGUI2)
  {
      //Load selected music file to deckGUI 2 when button was clicked
      loadToDeckGUI(deckGUI2);
  }
```

```
//Function to load tracks to deck
void PlaylistComponent::loadToDeckGUI(DeckGUI* deckGUI)
{
    int selectedRow{ playlist.getSelectedRow() };
    if (selectedRow != -1)
    {
        //Add the selected music to the DeckGUI
        deckGUI->loadFile(tracks[selectedRow].URL);
    }
}
```

PlaylistComponent.cpp (Code 11)

When user want to load a music file from playlist library to deck. User can simply select a row in the playlist (Image 8), the row will change colour and wait the user to click the “Add to Deck 1” or “Add to Deck 2” button.

For example, when “Add to Deck 1” button is clicked (Code 11), the loadToDeckGUI(deckGUI1) will be loaded. And the deckGUI will point to the loadFile to load the URL file to the audio player and display in the waveform display.

Name: Sau Ting Lam
Student Number: 200594390

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

```
void PlaylistComponent::saveInPlaylist()
{
    // create a text file and save in the playlist
    std::ofstream playlist("playlist.txt");

    for (TrackData& t : tracks)
    {
        playlist << t.file.getFullPathName() << "," << t.length << "\n";
    }
}
```

```
void PlaylistComponent::loadPlaylist()
{
    //Read from the saved playlist.txt
    std::ifstream playlist("playlist.txt");
    std::string path;
    std::string length;

    // Read data from playlist.txt line by line
    if (playlist.is_open())
    {
        while (getline(playlist, path, ',')) {
            File file{ path };
            TrackData loadPlaylist{ file };

            getline(playlist, length);
            loadPlaylist.length = length;
            tracks.push_back(loadPlaylist);
        }
    }
    playlist.close();
}
```

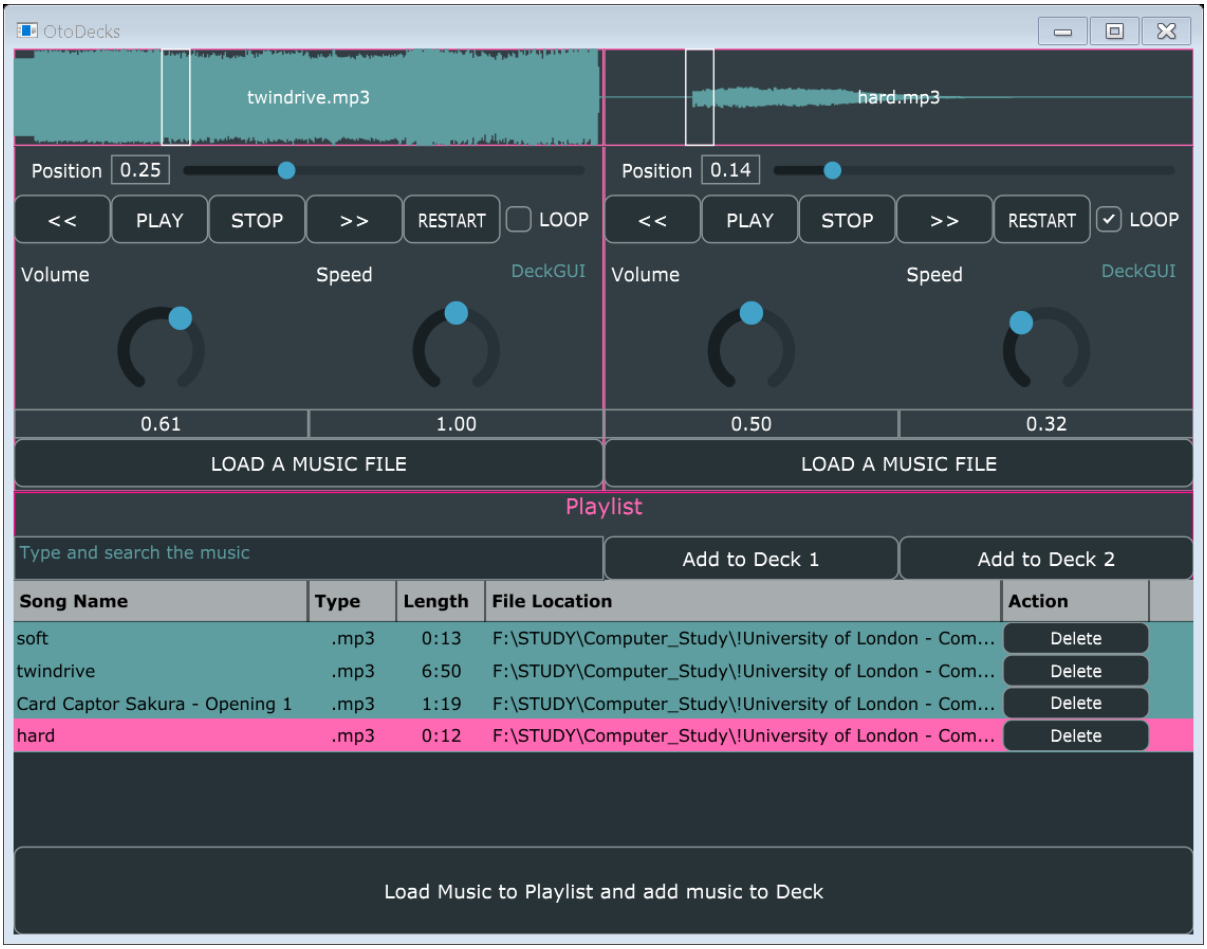
PlaylistComponent.cpp（Code 12)

The application will store the files loaded in the playlist without manually save and export a file. When user reopen again the application, it will load and read the playlist.txt file and reload in the playlist automatically.

In (Code 12) shows that when the application close, saveInPlaylist() will generate a playlist.txt file. The meta data will be saved is the from the new class TrackData. When the application start again, loadPlaylist() will read the playlist.txt line by line and republish them into the music playlist. Then the user will see the same playlist file stored from last modified.

Name: Sau Ting Lam
Student Number: 200594390

R4: Implementation of a complete custom GUI



(**Image 9**)

R4A: GUI layout is significantly different from the basic DeckGUI shown in class, with extra controls

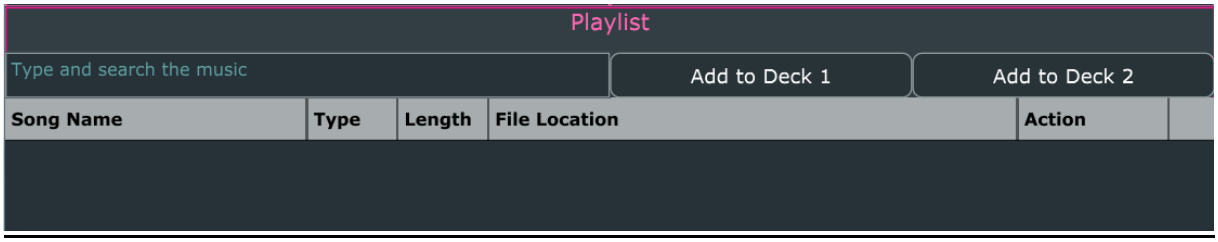
It is the new layout of the application after a new development (Image 9). The waveform display moved to the top of the deck. New button is added, new slider design with some text labels. The colour theme has changed.

R4B: GUI layout includes the custom Component from R2

In (Image 9), there are some new buttons such as “Restart”, “Loop”, “<<” which means rewind and “>>” which means forward. In order to make the application more user friendly, text label was added such as “Volume”, “Speed” and “Position”. To make the value number more aesthetic, I change the number value for the sliders to only 2 decimal number.

Name: Sau Ting Lam
Student Number: 200594390

R4C: GUI layout includes the music library component from R3



(**Image 10**)

In (Image 9), it is a new music playlist layout with the Playlist title in hotpink colour in the middle top of the music library. The new buttons are “Add to Deck1”, “Add to Deck2” and “Load Music to Playlist and add music to Deck”. Those button name have a clear message so user know the object when using the application. The search bar is added just on top of the Song name, before typing anything, there is a message “Type and search the music” to make sure the user knows what it is.

Before loading any files in the playlist (Image 10), the trackData component shows 5 headers. They are “Song Name” “Type”, “Length”, “File Location” and “Action”. When user load a file in the playlist, the meta data from the music file will be added in the playlist.

The “delete” bottom is added as well in order to remove file when user not longer need to files. User can simply click the delete button, the whole row will be erased.

Name: Sau Ting Lam
Student Number: 200594390

Reference

Ivor Horton and Peter Van Weert (2018). Beginning C++17 : from novice to professional. New York, Ny: Apress.

SliderStyle – Juce Documentation Available at: https://docs.juce.com/develop/classSlider.html#af1caee82552143dd9ff0fc9f0cdc0888 (Accessed: 2022/02/28)

FileChooser() – Juce Documentation Available at: https://docs.juce.com/develop/classFileChooser.html#afd55c3c97c97e23a0b18818156dfb71c (Accessed: 2022/03/01)

TextEditor Class Reference – Juce Documentation Available at: https://docs.juce.com/develop/classTextEditor.html (Accessed: 2022/03/02)

Label::Listener Class Reference – Juce Documentation Available at: https://docs.juce.com/develop/classLabel_1_1Listener.html#ad2a3e79ba106bfa64cab523e197b1514 (Accessed: 2022/03/02)

FileDragAndDropTarget Class Reference – Juce Documentation Available at: https://docs.juce.com/develop/classFileDragAndDropTarget.html (Accessed: 2022/03/07)

ifstream - C++ Reference. Available at: https://www.cplusplus.com/reference/fstream/ifstream/. (Accessed: 2022/03/08)

std::find_if - C++ Reference. Available at: https://www.cplusplus.com/reference/algorithm/find_if/. (Accessed: 2022/03/08)

std::distance - C++ Reference. Available at: https://www.cplusplus.com/reference/iterator/distance/. (Accessed: 2022/03/08)

std::vector::erase - C++ Reference. Available at: https://www.cplusplus.com/reference/vector/vector/erase/ (Accessed: 2022/03/08)