JUCE Music player

NOTE:

In juce add **“JUCE_MODAL_LOOPS_PERMITTED=1”** into your Preprocessor Definitions settings or some functions will be treated as an error

R1A:

We create a load button.

```
TextButton loadButton{"LOAD"};
```
- From DeckGUI.h

```
addAndMakeVisible(loadButton); // Add load button
```
- from DeckGUI.cpp

Whenever the load button is pressed, below function is called. Code from DeckGUI.cpp

```
if (button == &loadButton) // If load button clicked
{
    FileChooser chooser {"Select a file..."}; // default placeholder
    if (chooser.browseForFileToOpen()) // If a file is selected
    {
        player->loadURL(URL{chooser.getResult()});          // Call loadURL to read the file
        double length = player->getlength();                // Call getlength to get the duration of the file
        waveformDisplay.loadURL(URL{chooser.getResult()}); // Call loadURL into the waveformdisplay

        File Musicfile(chooser.getResult());               // get file name
        String stringfile = Musicfile.getFileName();       // turn file name into string
        DBG("The name of the file is " << stringfile);
        DBG("The length of the file is " << length);
    }
}
```

It opens a file explorer to select a file and calls the loadURL onto the selected file to read it and set the selected file as the current audio source. Below code is from DJAudioPlayer.cpp

```
void DJAudioPlayer::loadURL(URL audioURL) // Global file reader
{
    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());
    }
    else
    {
        DBG("Bad file");
    }
}
```

It also load the waveformdisplay using loadURL to read the file and create a waveform form it. Below code is from WaveFormDisplay.cpp

```
void WaveformDisplay::loadURL(URL audioURL) // check file to be an audio file, if not return a message
{
  audioThumb.clear();
  fileLoaded  = audioThumb.setSource(new URLInputSource(audioURL)); // checking file is an audio file
  if (fileLoaded)
  {
      DBG("wfd: loaded! ");
      FileName = audioURL.getFileName(); // get file name
      repaint();
  }
  else {
      DBG("wfd: not loaded! ");
  }

}
```

R1B:

By creating multiple DeckGUI and different player streams to host the file on

```
DJAudioPlayer player1{ formatManager };
DeckGUI deckGUI1{ &player1, formatManager, thumbCache };

DJAudioPlayer player2{ formatManager };
DeckGUI deckGUI2{ &player2, formatManager, thumbCache };
```

- MainComponent.h

They both are created by deckGUI.cpp, but under different players, hence letting them play different songs independently

Then set their bounds so they are not colliding with each other or other components

```
deckGUI1.setBounds(0, 0, getWidth() / 4, getHeight());
deckGUI2.setBounds((getWidth() / 4) * 3, 0, getWidth() / 4, getHeight());
```

- MainComponent.cpp

R1C:

First is to create the volume slider, named volSlider

```
Slider volSlider;
```

- from DeckGUI.h

```
addAndMakeVisible(volSlider);    // Add volume slider
```

- from DeckGUI.cpp

Then add a listener to the volSlider

```
void sliderValueChanged (Slider *slider) override;
```

- from DeckGUI.h

```
volSlider.addListener(this);     // volSlider listener
```

- from DeckGUI.cpp

When the VolSlider value is changed, it procs below code, below code is from DeckGUI.cpp

```
void DeckGUI::sliderValueChanged (Slider *slider)
{
    if (slider == &volSlider) // If volSlider change value
    {
        player->setGain(slider->getValue()); // Change gain to the new value
    }
```

The code above call the code below to set the volume to the newest value in the slider

```
void DJAudioPlayer::setGain(double gain)
{
    if (gain < 0 || gain > 10.0)
    {
        DBG("DJAudioPlayer::setGain gain should be between 0 and 1");
    }
    else
    {
        transportSource.setGain(gain); // Call setGain on new value
    }

}
```

- DJAudioPlayer.cpp

R1D:

Same as in R1C, the method is similar.

Create the Slider, named speedSlider

```
Slider speedSlider;
```
- From DeckGUI.h

```
addAndMakeVisible(speedSlider); // Add speed slider
```
- From DeckGUI.cpp

Then add a listener to the speedSlider

```
Slider speedSlider;
```
- from DeckGUI.cpp

When the speedSlider value is changed, it procs below code, below code from DeckGUI.cpp

```
if (slider == &speedSlider) // If speedSlider change value
{
    player->setSpeed(slider->getValue()); // Change speed to the new value
}
```

The code above call the code below to set the speed to the newest value in the slider

```
void DJAudioPlayer::setSpeed(double ratio)
{
  if (ratio < 0 || ratio > 10.0)
    {
      DBG("DJAudioPlayer::setSpeed ratio should be between 0 and 100");
    }
    else
    {
        resampleSource.setResamplingRatio(ratio); // Call setResamplingRatio on new value
    }
}
```
- DJAudioPlayer.cpp

R2A:

Within the paint function, the code below create the waveform when it is loaded and a green line is also created to show which part of the song you are currently at, default it starts at the beginning.

Default have a placeholder text to show nothing is loaded.

```
if (fileLoaded)
{
  audioThumb.drawChannel(g, getLocalBounds(), 0, audioThumb.getTotalLength(), 0, 1.0f); // Drawing the lines
  g.setColour(Colours::lightgreen);
  g.drawRect(position * getWidth(), 0, 1, getHeight()); // Drawing the position box
}
else
{
  g.setFont (20.0f);
  g.drawText ("No Music?", getLocalBounds(), Justification::centred, true); // Placeholder text for empty box
}
```

^ From WaveFormDisplay.cpp

R2B:

Same as in R1C and R1D, the method is similar

Create the Slider, named posSlider

```
Slider posSlider;
```
- From DeckGUI.h

```
addAndMakeVisible(posSlider);    // Add position slider
```
- From DeckGUI.cpp

Then add a listener to the posSlider

```
posSlider.addListener(this);     // posSlider listener
```
- From DeckGUI.cpp

When the posSlider value is changed, it procs below code, below code is from DeckGUI.cpp

```
if (slider == &posSlider) // If posSlider change value
{
    player->setPositionRelative(slider->getValue()); // Change position to the new value
}
```

The code above call the code below to set the position to the newest value in the slider

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

R3A, R3B:

To achieve this, I used a “load” button to add files into a library, so create the button…

```
TextButton AddButton{ "ADD TO PLAYLIST" };
```
- PlaylistComponent.h

```
addAndMakeVisible(AddButton);
```
- PlaylistComponent.cpp

Add a listener for the “load” button

```
void buttonClicked(Button* button) override;
```
- PlaylistComponent.h

```
AddButton.addListener(this);  // Add AddButton Listener
```
- PlaylistComponent.cpp

So, when the button is clicked, the code below procs, calling AddToLibrary

```
void PlaylistComponent::buttonClicked(Button* button)
{
    if (button == &AddButton)
    {
        DBG("Add button clicked");
        AddToLibrary();     // Call AddToLibrary
    }
```
- PlaylistComponent.cpp

The “AddToLibrary” code first open your file explorer to select a file to be added into the library, when the file is selected, it finds the file name, duration and fit them to a class, “Song” in Songs.h. Then send it into “Songs” vector containing both name and duration. Unless it’s already inside the library, then it will print out a console message.

```
std::vector<Song> Songs;
```
- PlaylistComponent.h

```
void PlaylistComponent::AddToLibrary()
{
    FileChooser chooser{ "Select files" };
    if (chooser.browseForFileToOpen())
    {
        for (File& file : chooser.getResults())
        {
            String FileNameNoExt = file.getFileNameWithoutExtension(); // return file name

            if (!InsideSong(FileNameNoExt)) // if not already loaded
            {
                Song AddSong{ file };
                URL audioURL{ file };
                AddSong.Length = Getlength(audioURL);
                Songs.push_back(AddSong);
                DBG( "loaded file: " << AddSong.Title );
            }
            else // display info message
            {
                DBG( "AddToLibrary:: " << FileNameNoExt << " already loaded" );
            }
        }
    }
}
```
^ PlaylistComponent.cpp

```
String PlaylistComponent::Getlength(URL audioURL) // Get the duration of the file
{
    player->loadURL(audioURL);
    double sec = player->getlength();
    int roundedsec = (std::round(sec)); //round up/down the seconds

    String stringmin = std::to_string(roundedsec / 60); // convert secs into mins
    String stringsec = std::to_string(roundedsec % 60); // find the remanding secs

    if (stringsec.length() < 2) // if seconds are 1 digit or less
    {
        stringsec = stringsec.paddedLeft('0', 2); //add '0' to seconds until seconds = 2
    }

    String Duration{ stringmin + ":" + stringsec }; // Set duration string

    return Duration;
}
```

^ PlaylistComponent.cpp

With “Getlength” to find the duration and convert it into minutes and seconds as a string, and if the seconds are 1 digit, it will print a ‘0’ to pad the seconds.

R3C:

To search, must first create a search bar

```
TextEditor Searchbar;
```
- PlaylistComponent.h

```
addAndMakeVisible(Searchbar);
Searchbar.setTextToShowWhenEmpty("Search Songs (enter to submit) (CAPS SENSITIVE)", Colours::orange);
```
^ PlaylistComponent.cpp

For the searchbar to work, there need to be a function activated somehow, so I decided to use enter to be the activator.

```
Searchbar.onReturnKey = [this] { SearchLibrary(Searchbar.getText()); };
```
- PlaylistComponent.cpp

This code above will call “SearchLibrary” on the text inside the searchbar on pressing ‘enter’

```
void SearchLibrary(String Searchbar);
```
- PlaylistComponent.h

```
int SongSearch(String Searchbar);
```
- PlaylistComponent.h

```
void PlaylistComponent::SearchLibrary(String Searchbar)
{
    DBG("Searching library for: " << Searchbar);
    if (Searchbar != "") // If searchbar has something
    {
        int rowNumber = SongSearch(Searchbar); // Find rowNumber based on SongSearch result
        tableComponent.selectRow(rowNumber);   // Select row on rowNumber
    }
    else
    {
        tableComponent.deselectAllRows();     // If no result, deselect all rows
    }
}
```

^ PlaylistComponent.cpp

The “SearchLibrary”, if searchbar is not empty, will call “SongSearch” on the searchbar text and return the row of the found object, highlighting it

```
int PlaylistComponent::SongSearch(String Searchbar)
{
    // Search Song on searchbar beginning letter and ending letter, returning an int
    auto find = find_if(Songs.begin(), Songs.end(), [&Searchbar](const Song& obj) {return obj.Title.contains(Searchbar); });
    int i = -1;                                 // It must be -1 one so if any search no match, it wont highlight anything

    if (find != Songs.end())
    {
        i = std::distance(Songs.begin(), find); // Find the rowNumber
    }

    return i;
}
```

^ PlaylistComponent.cpp

The “SongSearch” will compare the searchbar text and the library songs and any closest match will return an int for number of rows crossed from the first, giving us the proper row number.

If no match were found, since i is originally -1, it won’t select anything.

R3D:

I've added a function to load the file of the selected row onto deck1 or deck2 based on the user choice. By first making buttons on each row

Below code creates 2 new TextButton in each row under column 3 and 4, then creates a unique ID from the row number combined with 'A' or 'B' to determined which button is clicked on the same row. Also adds a listener to each button.

```
Component* PlaylistComponent::refreshComponentForCell(int rowNumber, int columnId, bool isRowSelected, Component* existingComponentToUpdate)
{
    if (columnId == 3) //player1 button
    {
        if (existingComponentToUpdate == nullptr)
        {
            TextButton* btn = new TextButton { "Load into player1" };   // Add textbutton "Load into player1"
            String uniqueid { "A" + std::to_string(rowNumber)};         // String A for delete
            btn->setComponentID(uniqueid);                              // Set ID component string A

            btn->addListener(this);
            existingComponentToUpdate = btn;
        }
    }

    if (columnId == 4) //player2 button
    {
        if (existingComponentToUpdate == nullptr)
        {
            TextButton* btn = new TextButton { "Load into player2" };   // Add textbutton "Load into player2"
            String uniqueid { "B" + std::to_string(rowNumber) };        // String B for delete
            btn->setComponentID(uniqueid);                              // Set ID component string B

            btn->addListener(this);
            existingComponentToUpdate = btn;
        }
    }
```

^ PlaylistComponent.cpp

Then under "buttonClicked", it finds which button was clicked by componentID containing 'A' for Deck1 or 'B' for Deck2, then selects the appropriate row by deciphering the componentID into the rownumber by removing the 'A' or 'B' from the componentID string then convert it into the rownumber.

Then calls "LoadPlayer" onto the specific deck.

```
else
{
    std::string uniqueID = button->getComponentID().toStdString();
    //DBG("This button id is " << uniqueID);                        // For debugging

    if (uniqueID.find("A") != std::string::npos) // Player1 button clicked
    {
        int RowNumber = std::stoi(uniqueID.erase(0, 1)); // Erase the A from the string, turn into int

        tableComponent.selectRow(RowNumber);    // Select Appropriate row
        loadPlayer(Deck1);                      // load into Deck1

        //DBG("PlaylistComponent::buttonClicked Player1");          // For debugging
    }

    if (uniqueID.find("B") != std::string::npos) // Player2 button clicked
    {
        int RowNumber = std::stoi(uniqueID.erase(0, 1)); // Erase the B from the string, turn into int

        tableComponent.selectRow(RowNumber);    // Select Appropriate row
        loadPlayer(Deck2);                      // load into Deck2

        //DBG("PlaylistComponent::buttonClicked Player2");          // For debugging
    }
```

^ PlaylistComponent.cpp

The “LoadPlayer” finds the selected row, then call “playlistplayer” in deckGUI, like the load button, it calls both DJAudioPlayer and WaveformDisplay “LoadURL” in the selected Deck.

```
void PlaylistComponent::loadPlayer(DeckGUI* deckGUI)
{
    int selectedrow{ tableComponent.getSelectedRow() }; // Selected row is used to determined which row is being loaded

    DBG("PlaylistComponent::loadPlayer selectedrow is " << selectedrow);
    if (selectedrow >= 0) // If there is selected row
    {
        DBG("Adding: " << Songs[selectedrow].Title << " to Player");
        deckGUI->playlistplayer(Songs[selectedrow].URL); // Call playlistplayer on the song URL in the selected row
    }
    else
    {
        DBG("loadPlayer:: please select a valid file");
    }
}
```

^ PlaylistComponent.cpp

```
void DeckGUI::playlistplayer(URL URL) // function for playlistComponent
{
    DBG("DeckGUI::playlistplayer used");
    player->loadURL(URL);           // Call LoadURL to a targeted file/URL to read
    waveformDisplay.loadURL(URL);   // Call LoadURL to a targeted file/URL to form waveform
}
```

- DeckGUI.cpp

R3E:

Create a function to save the playlist that is loaded onto a file and another to load the file into the playlist. So we create 2 functions, “saveLibrary” and “loadLibrary”.

```
void saveLibrary();
void loadLibrary();
```

- PlaylistComponent.h

The “saveLibrary” first check if the save file exist, if not create the save file: “Library.csv”, then create a output stream to “Library.csv” and for every class in the vector, it inserts the class filepath and duration into the save file.

This function procs every time the program closes.

```
void PlaylistComponent::saveLibrary()
{
    File TrackFile = File::getCurrentWorkingDirectory().getChildFile("Library.csv");    // File in the same working directory
    if (!TrackFile.existsAsFile())                                                      // If file dont exist, create it
    {
        DBG("File doesn't exist, creating now");
        TrackFile.create();
    }

    std::ofstream myLibrary("Library.csv"); // input stream

    for (Song& Song : Songs)
    {
        myLibrary << Song.MFile.getFullPathName() << "," << Song.Length << "\n"; // Insert into CSV, fileFillpath and duration
    }
}
```

^ PlaylistComponent.cpp

```
PlaylistComponent::~PlaylistComponent()
{
    saveLibrary();
}
```

^ PlaylistComponent.cpp

The “loadLibrary” starts whenever the program boots up.

It creates an input stream to “Library.csv” and reads every string within the file, then puts it into a Song class to push into the Songs vector.

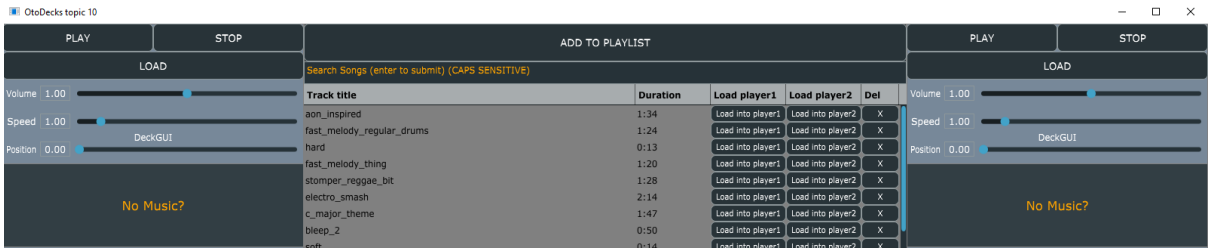
Once it read finish, it closes the input stream.

```
void PlaylistComponent::loadLibrary() // Load everytime it boots up
{
    std::ifstream MyLibrary("Library.csv");
    std::string filePath;
    std::string length;

    if (MyLibrary.is_open())
    {
        while (getline(MyLibrary, filePath, ',')) // While reading the file strings
        {
            File file = filePath; // Convert the std::string into juce::File
            Song newTrack = file; // Have the file be put into Song class

            getline(MyLibrary, length); // Read file string, put into length
            newTrack.Length = length;   // Add found length into class Length
            Songs.push_back(newTrack);  // Add the class into the class vector Songs
        }
    }
    MyLibrary.close();
}
```

^ PlaylistComponent.cpp

R4A:

Current Layout of the program:



Made by tweaking the setsize and resize functions.

```
setSize (1600, 300);
```
- MainComponent.cpp

```
void MainComponent::resized()
{
    deckGUI1.setBounds(0, 0, getWidth() / 4, getHeight());
    deckGUI2.setBounds((getWidth() / 4) * 3, 0, getWidth() / 4, getHeight());

    playlistComponent.setBounds(getWidth() / 4, 0, getWidth() / 2, getHeight());
}
```
- MainComponent.cpp

An extra control comes in the playlist being able to delete songs from the playlist.

This code creates the button an attach a unique ComponentID to each button, same way in R3D, it finds the appropriate row by deciphering the componentID into the rownumber by removing the ‘C’ from the componentID string then convert it into the rownumber.

Then calls “DeleteFromLibrary” on the requested row that erase the appropriate song from the vector.

```
if (columnId == 5) //Delete button
{
    if (existingComponentToUpdate == nullptr)
    {
        TextButton* btn = new TextButton{ " X " };                  // Add textbutton " X "
        String uniqueid { "C" + std::to_string(rowNumber) };        // String C for delete
        btn->setComponentID(uniqueid);                              // Set ID component string C

        btn->addListener(this);
        existingComponentToUpdate = btn;
    }
}
```

^ PlaylistComponent.cpp

```
if (uniqueID.find("C") != std::string::npos) // Delete button clicked
{
    int RowNumber = std::stoi(uniqueID.erase(0, 1)); // Erase the C from the string, turn into int

    DeleteFromLibrary(RowNumber);           // Call delete on Appropriate row

    //DBG(Songs[RowNumber].Title + " removed from Library");    // For debugging
    //DBG("PlaylistComponent::buttonClicked Delete");           // For debugging
}
```

^ PlaylistComponent.cpp

```
void PlaylistComponent::DeleteFromLibrary(int id)
{
    //DBG("Deleting: " << id); // For debugging
    Songs.erase(Songs.begin() + id); // Delete songs beginning with the selected row ID
}
```
- PlaylistComponent.cpp

R4B, R4C:

The GUI contains the green line, from R2A to show the progress of the loaded track.

The playlist from R3 is in the middle and Deck1 is on the left while Deck2 is on the right, both songs are loaded from the playlist via the buttons within the playlist.

Deck1 played for a bit while Deck2 has it progress set at 50%.