D3: Requirements: for each sub requirement (R1A → R4C) state how this was achieved or if it was not achieved. Explain where it can be found in the code. Use focused, short code extracts if they make your explanation clearer. Use annotated screenshots to describe your GUI components.

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

- R1A: can load audio files into audio players

The class DJAudioPlayer : public AudioSource is mainly responsible for the audio stream working. The functionality of loading audio files into the audio player is ensured by the public function void loadURL(URL audioURL). In the class DeckGUI there is also the TextButton loadButton which displays “LOAD” for the user to launch the audio.

- R1B: can play two or more tracks

This is achieved through the public function void play(). In the class DeckGUI there is also the TextButton playButton which displays “PLAY” for the user to launch the audio. Also, the function void stop() is ensuring that the track will stop playing. In the class DeckGUI there is also the TextButton stopButton, which displays “STOP” for the user to stop the play. The possibility of mixing two tracks is given by setting in the MainComponent class a deckGUI1 and a deckGUI1 for DeckGUI, together with a player1 and player2 for DJAudioPlayer.

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

The volume of the tracks is controlled by the setGain(double gain) function, present in the DJAudioPlayer class.

- R1D: can speed up and slow down the tracks

The speed of the tracks is set by the setSpeed(double ratio) function, present in the DJAudioPlayer class.

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.

- R2A: Component has custom graphics implemented in a paint function

LookAndFeel_V4, the latest JUCE look-and-feel style, is used in the CustomComponent class in order to design a basic graphics for the sliders for volume, speed and positioning. The drawRotarySlider() function is drawing the ellipses and rectangles in order to give sliders the aspect of vinyl records. The AffineTransform JUCE class is used to create the 2D rotation of the rectangle around the ellipses, mimicking the movement of records. This way the user experience is that of a DJ who spins records.

Another aspect to mention is the class WaveformDisplay, which is using the JUCE user-interface component in order to create a 2D waveform which detects when the song is being loaded (through the boolean fileLoaded) and also the positioning of the song (through double position). The waveform graphical layout is implemented through the paint (Graphics&) function.

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

The deck is controlled by the play and start buttons, but also in the player we can notice that there are some extra controls of the songs played:

- the volume, which adjusts the volume of the song in DJAudioPlayer class, in the setGain() function, from greater than 0 to smaller then 5.0. The default value is 2.5 and is set in DeckGUI.cpp through volDefaultValue.
- the speed, which adjusts the speed of the song in DJAudioPlayer class, in the setSpeed() function, from a ratio of greater than 0.10 to smaller than 3.0. The default value is set in DeckGUI.cpp through speedDefaultValue at 1.0, which is the not-modified / original speed version of the song.
- the position of the song, which gives control of the position of the song in DJAudioPlayer class, in the getPositionRelative() function, from a position pos greater than 0 to smaller than 1.0 (from the beginning of the song, to the end). The default value is set in DeckGUI.cpp through speedDefaultValue.

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

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

In the PlaylistComponent class, the function loadToMusicLibrary() is mainly responsible for allowing the user to add files to their music library by setting a file chooser. If the song chosen to be added is already present in the library, an Info Box (through the JUCE class AlertWindow) will pop up informing the user that the song is already present. Also the loadFileInLibraryButton button is helping the user load the track they wish from their device.

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

The metadata is displayed through the functions getLength() and the JUCE string component secondsToMinutes, which finds seconds / minutes and converts them into a string, while also formatting them. The function getLength() is investigating the metadata and then secondsToMinutes will return it as a string in an adequate format, making it possible to be shown to the user in the Length column from the music library.

- R3C: Component allows the user to search for files

The search functionality is insured mainly in the PlaylistComponent class by the functions searchMusicLibrary() and whereInSongs(), with the boolean foundInSongs. The function searchMusicLibrary() is searching for songs whose name contains searchText, by calling whereInSongs, and selects it. The function whereInSongs() is searching for a song whose name contains searchText. The boolean foundInSongs is testing if one of the songs has a certain file name. On the user interface, the search functionality is ensured by the “SEARCH (Track + Enter)” field, which is configured through searchTrackInLibrary that uses searchMusicLibrary.

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

The function loadInPlayer() present in the PlaylistComponent class is ensuring that the user can load the selected file in the player (through selectedRow that uses the juce::ListBox::getSelectedRow, which returns the row number of a selected row). The file is loaded into the player and can be tested also with the Waveform, which will change the graphics displayed on the screen, as an indicator that the file is loaded from the music library. On the user interface, the playInDeck1Button and playInDeck2Button will guide the user into adding the selected song into the desired deck.

If there is no song selected and the user presses one of the two buttons, an Info Box (which makes use of the JUCE AlertWindow class) will be displayed with the information to choose a track to be added to one of the decks.

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

The music library persists when the user exists and then restarts the application through the saveMusicLibrary() function. This makes use of offstrem that will create a new csv file and a for loop that makes all songs in the list to be outputted in the csv file.

The function loadMusicLibrary() ensures that the music library will be loaded, by opening the csv file, creating two strings filePath and length, making use of the getline() standard library function to read the filePath and length and then add the information in the song list.

The music library also has a deleteSong() function which makes it possible for the user to delete songs from the list by identifying the song (int id) and pressing the “X” TextButton.

The class Song is included in the PlaylistComponent class. It is meant to offer general access to the the files in order to be used and also keeps all the data about the songs, bu storing the file itself (File file), so it can be read, the name and length of the song (strings name and length) and a JUCE URL out of the file name. It also uses the bool operator, which is a function for equality check, where objects are being compared by their name. If a comparison of the songs will ever happen, it will be true if the name is the same.

R4: Implementation of a complete custom GUI

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

The GUI layout has a different aspect than the basic DeckGUI shown in class, as can be seen in the screenshot from Figure 1.

Figure 1

The elements displayed (buttons, players, waveform, layout etc) are placed in different positions. The custom component offers a different graphical representation of the sliders for Volume, Speed and Position, giving them the aspect of vinyl records.

The music library from the bottom has significant additions, extra buttons, a search field and displays information such as the length of the songs.

- R4B: GUI layout includes the custom Component from R2

The custom component from R2 is included in the GUI layout. The DeckGUI.h file includes CustomComponent.h file (#include "CustomComponent.h").

- R4C: GUI layout includes the music library component from R3

The custom component from R3 is included in the GUI layout. The PlaylistComponent.h file includes the DeckGUI.h file (#include "DeckGUI.h").