**Report on enhancing the Otodecks DJ application**

**Introduction**

This report outlines the improvements made to the Otodecks DJ application. The design has been reworked, and a custom deck control component has been added. The play/stop buttons were removed, and the playlist functionality has been expanded. Now, users can load tracks even faster from the root folder.

## Core Functionality (R1)

### Description of the Implemented Basic Functionalities

#### R1A: Loading Audio Files into Players

Tracks are loaded using the loadTrack method in the DeckGUI class. This method calls player->loadURL(audioURL) to load the track into DJAudioPlayer and waveformDisplay.loadURL(audioURL) to display the waveform.

#### R1B: Playing Multiple Tracks Simultaneously

Tracks are played using the playButton. When the button is pressed, the buttonClicked method checks if a track is playing and either calls player->start() to start playback or player->stop() to stop it. The button state is updated using the playButtonSetColor method.

#### R1C: Mixing Tracks by Varying Volume Levels

Volume control is handled using the volSlider. When the slider value changes, the sliderValueChanged method calls player->setGain(volSlider.getValue()) to adjust the volume.

#### R1D: Adjusting Playback Speed

Playback speed is controlled using the speedSlider. When the slider value changes, sliderValueChanged calls player->setSpeed(speedSlider.getValue()) to modify the playback speed.

## User Interface Customization (R2)

The interface was significantly redesigned:

* All sliders were changed from horizontal to vertical.
* A file tree was added on the left side of the playlist.
* The playlist now allows batch track loading from a folder with a single click.
* Users can now choose which player to load a track into.
* The track position can now be adjusted on the waveform display or rotated within the deck layout.
* The "Load" button was removed.
* A new Play/Stop button was added.
* Each slider now has a label.
* Minor color scheme adjustments were made for the waveform display and buttons.

### Before and After Screenshots

(Insert before and after images here)

## New Feature Implementation (R3)

### Research

The new features were inspired by the VirtualDJ software, as shown in this video: <https://www.youtube.com/watch?v=GIPHwnokWiw&t=1149s>

I liked the simplicity and functionality of VirtualDJ, so I decided to incorporate a deck and a file tree for loading tracks. This approach is also seen in other DJ applications: <https://www.youtube.com/watch?v=gaVJ-8qjulI&t=1s>

To better understand JUCE's capabilities, I reviewed all the examples provided in Projucer.

### Implementation

#### FileBrowserComponent and PlaylistComponent

##### FileBrowserComponent

**Classes Used:**

* juce::FileTreeComponent
* juce::DirectoryContentsList
* juce::TimeSliceThread

**Methods:**

* FileBrowserComponent(PlaylistComponent& playlist)
* void resized()
* void selectionChanged()
* void fileClicked(const juce::File& file, const juce::MouseEvent& event)
* void fileDoubleClicked(const juce::File& file)
* void browserRootChanged(const juce::File& newRoot)
* void scanAndAddFiles(const juce::File& directory)

##### PlaylistComponent

**Track Storage:** The tracks are stored using the following structure:

struct TrackInfo {

juce::String path;

juce::String title;

double duration = 0.0;

};

**Classes Used:**

* juce::Component
* juce::TableListBoxModel
* juce::Button::Listener
* DeckGUI
* TrackInfo

**Methods:**

* PlaylistComponent(DeckGUI\* deckGUI1, DeckGUI\* deckGUI2)
* ~PlaylistComponent()
* void paint(juce::Graphics& g)
* void resized()
* int getNumRows()
* void paintRowBackground(juce::Graphics& g, int rowNumber, int width, int height, bool rowIsSelected)
* void paintCell(juce::Graphics& g, int rowNumber, int columnId, int width, int height, bool rowIsSelected)
* juce::Component\* refreshComponentForCell(int rowNumber, int columnId, bool isRowSelected, juce::Component\* existingComponentToUpdate)
* void buttonClicked(juce::Button\* button)
* void addTrack(const juce::String& trackPath)

### Object-Oriented Programming Principles

* **Encapsulation:** Each class encapsulates its functionality, such as file browsing and playlist management.
* **Composition:** PlaylistComponent and DeckGUI work together to manage tracks.
* **Inheritance:** PlaylistComponent extends JUCE components, allowing customization.

### Additional Feature: Animated Deck Wheel

Inspired by [InfiniteRotarySlider](https://github.com/deorst/InfiniteRotarySlider), I implemented an interactive, animated deck wheel that allows track scrubbing.

### InfiniteRotarySliderComponent Logic

**Key Methods:**

* setRange(double newMinimum, double newMaximum)
* setValue(double newValue)
* setAngle(float newAngle)
* paint(juce::Graphics& g)
* void resized()
* m\_processSliderValue(float value)

**OOP Principles:**

* **Encapsulation:** The class encapsulates the rotary slider behavior.
* **Composition:** Uses juce::Slider for user interaction.
* **Polymorphism:** Overrides paint() and resized() for custom rendering.

## Music Library Component

The music library component enables:

* Searching and managing a track library.
* Loading tracks into decks.
* Persisting between application loads using a data file.

## Code Structure and Best Practices

* **C1:** Code is split into .h and .cpp files.
* **C2:** Class interfaces include comments for clarity.
* **C3:** Consistent indentation and layout.
* **C4:** Functions have clear inputs, outputs, and purposes.
* **C5:** Stateless design where possible, using data passing.
* **C6:** Meaningful naming conventions.
* **C7:** Functions only modify class/global variables when necessary.

1. **Introduction**

Краткое описание улучшений приложения. Переработан дизайн приложения и добавлено deck управления сэмплом. Удалены кнопки управления play/stop. Расширен функционал play list. Теперь можно загружать треки еще быстрее из корня папки.

1. **Core Functionality (R1)**

Description of the implemented basic functionalities:

* **R1A**: Loading audio files into players.

Треки загружаются с помощью метода loadTrack в классе DeckGUI. Этот метод вызывает player->loadURL(audioURL) для загрузки трека в DJAudioPlayer и waveformDisplay.loadURL(audioURL) для отображения волновой формы трека.

* **R1B**: Playing multiple tracks simultaneously.

Воспроизведение треков осуществляется с помощью кнопки playButton. Когда кнопка нажата, метод buttonClicked проверяет, воспроизводится ли трек в данный момент, и вызывает **player->start()** для начала воспроизведения или **player->stop()** для остановки воспроизведения. Состояние кнопки обновляется с помощью метода playButtonSetColor.

* **R1C**: Mixing tracks by varying volume levels.

Регулировка громкости осуществляется с помощью слайдера volSlider. Когда значение слайдера изменяется, метод sliderValueChanged вызывает **player->setGain(volSlider.getValue())**, чтобы установить новую громкость.

* **R1D**: Adjusting playback speed.

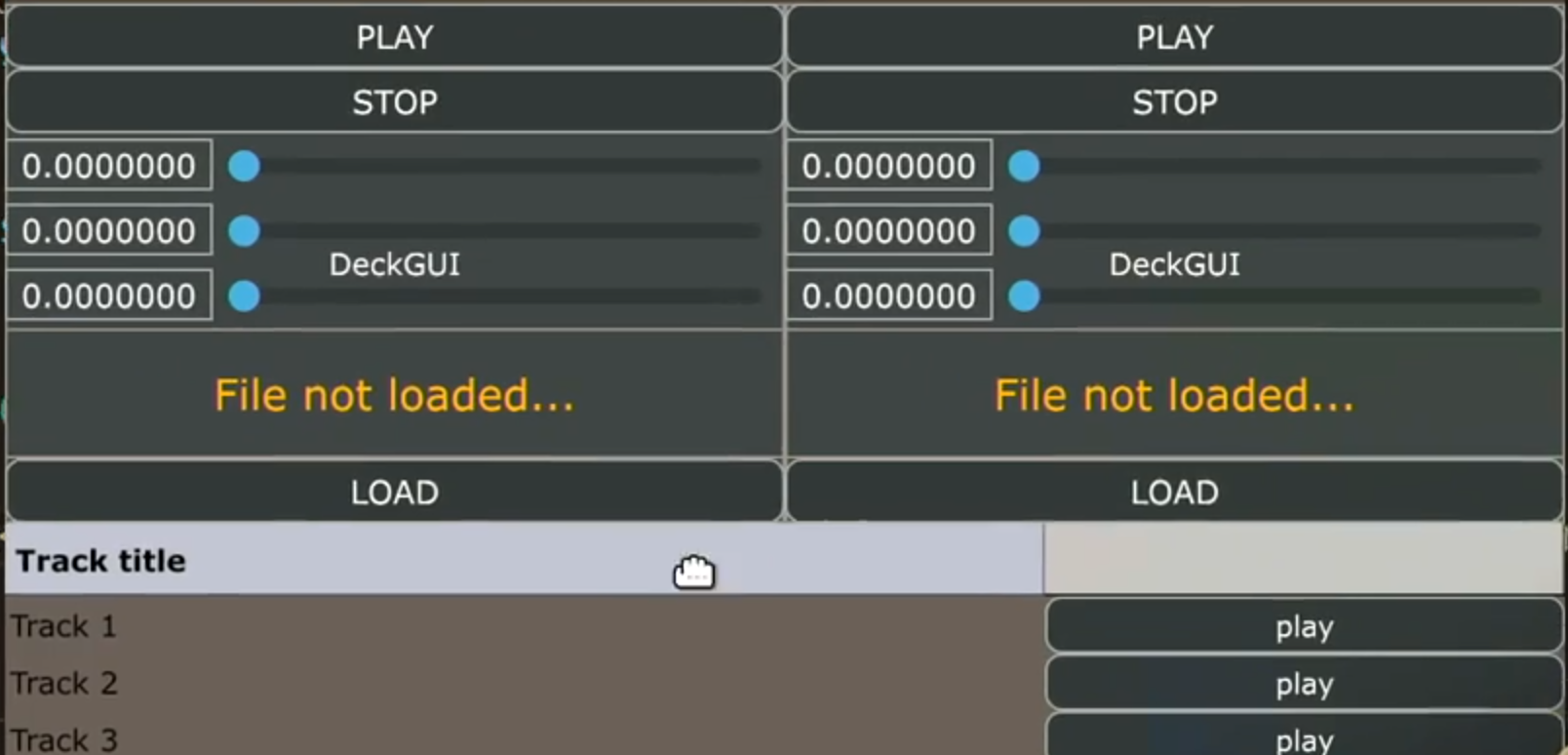
Изменение скорости воспроизведения осуществляется с помощью слайдера speedSlider. Когда значение слайдера изменяется, метод sliderValueChanged вызывает **player->setSpeed(speedSlider.getValue())**, чтобы установить новую скорость воспроизведения.

1. **User Interface Customization (R2)**

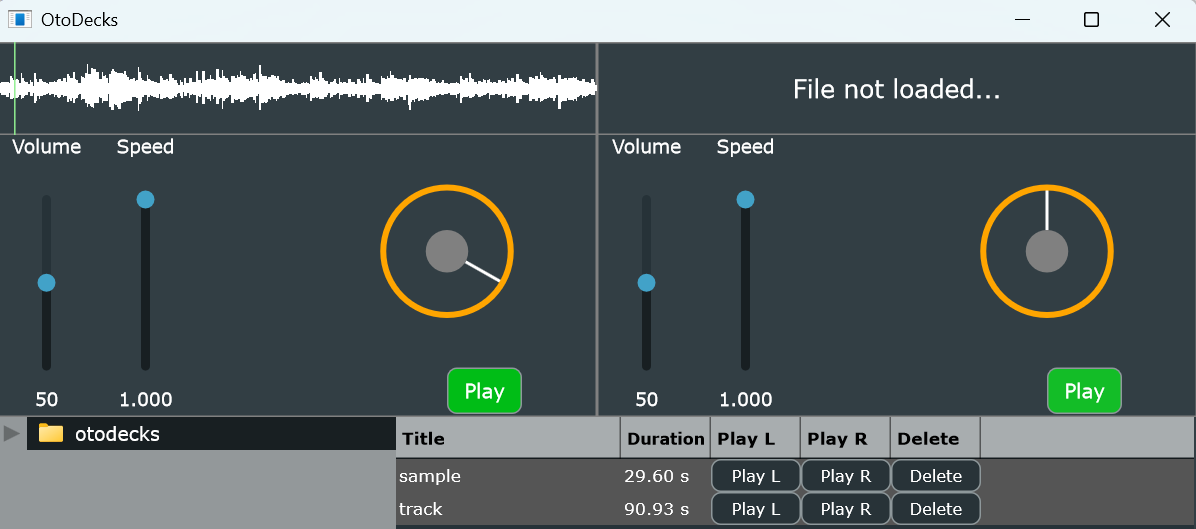
Был координально изменен интерфейс. Все sliders были перевернуты из горизонтального положения в вертикальное. Добавлен дерево файлов слева от play list. В play list можно загрызить все треки из папки по одному клику. Добавлен выбор в какой player загрузить трек. Теперь позицую трека можно поменять на waveform или повернуть укоядку deck, которая отображает воспроизведение трека. Удалена кнопка load. Добавлена кнопка Play/Stop. Теперб каждый слайдер подписан

Небольшое изменние в цевотовой схеме waveform и кнопки.

До



после



1. **New Feature Implementation (R3)**

Исследование:

Все свои идеи мне пришли после просмотра ролика программы virtualDJ

<https://www.youtube.com/watch?v=GIPHwnokWiw&t=1149s>

Мне понравилось простота программы и функционал. Было решенно взять deck и дерево загрузки треков.

Этот же набор и встречался в других программах DJ, которые представлены здесь   
<https://www.youtube.com/watch?v=gaVJ-8qjulI&t=1s>

Для полного понимания что можно сделать используя juce были пересмотрены все примеры из Projucer.  
Скриншоты с youtube https://www.youtube.com/watch?v=gaVJ-8qjulI&t=1s  

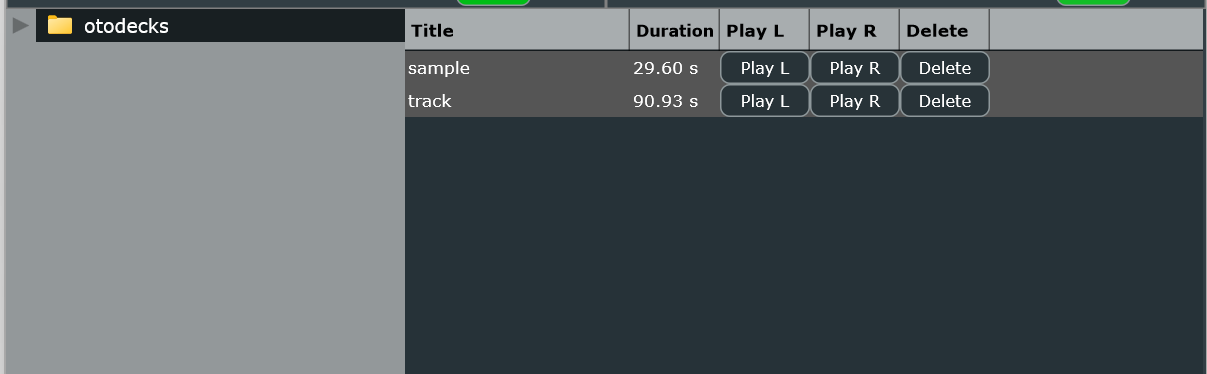



Этого было достаточно для представления как это реализовать.

Так как playlist уже был реализован было решено добавить дерево как это показоно на этом скрине 

Классы FileBrowserComponent и PlaylistComponent

Как это выглядит в моем приложении



FileBrowserComponent

**Используемые классы:**

* juce::FileTreeComponent
* **juce::DirectoryContentsList**
* **juce::TimeSliceThread**
* PlaylistComponent

**Методы:**

* **FileBrowserComponent(PlaylistComponent& playlist)**
* **void resized()**
* **void selectionChanged()**
* **void fileClicked(const juce::File& file, const juce::MouseEvent& event)**
* **void fileDoubleClicked(const juce::File& file)**
* **void browserRootChanged(const juce::File& newRoot)**
* **void scanAndAddFiles(const juce::File& directory)**

**OOP-принципы:**

* **Инкапсуляция:** Класс инкапсулирует функциональность для отображения файловой системы и обработки событий, связанных с выбором файлов.
* **Композиция:** Класс использует объекты PlaylistComponent, juce::FileTreeComponent, **juce::DirectoryContentsList** и **juce::TimeSliceThread** для выполнения своих задач.

PlaylistComponent

**Используемые классы:**

* juce::Component
* **juce::TableListBoxModel**
* juce::Button::Listener
* DeckGUI
* TrackInfo

**Методы:**

* **PlaylistComponent(DeckGUI\* deckGUI1, DeckGUI\* deckGUI2)**
* **~PlaylistComponent()**
* **void paint(juce::Graphics& g)**
* **void resized()**
* **int getNumRows()**
* **void paintRowBackground(juce::Graphics& g, int rowNumber, int width, int height, bool rowIsSelected)**
* **void paintCell(juce::Graphics& g, int rowNumber, int columnId, int width, int height, bool rowIsSelected)**
* **juce::Component\* refreshComponentForCell(int rowNumber, int columnId, bool isRowSelected, juce::Component\* existingComponentToUpdate)**
* **void buttonClicked(juce::Button\* button)**
* **void addTrack(const juce::String& trackPath)**

**OOP-принципы:**

* **Наследование:** Класс наследуется от juce::Component, **juce::TableListBoxModel** и juce::Button::Listener, что позволяет ему переопределять методы и использовать их функциональность.
* **Инкапсуляция:** Класс инкапсулирует функциональность для управления плейлистом, включая добавление треков и обработку событий кнопок.
* **Композиция:** Класс использует объекты DeckGUI и TrackInfo для выполнения своих задач.

**Схема работы**

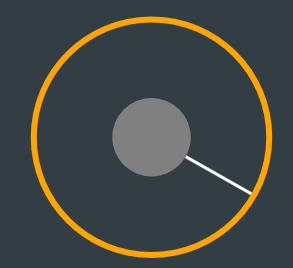
1. **FileBrowserComponent:**
   * Отображает файловую систему с помощью juce::FileTreeComponent.
   * При выборе или двойном щелчке по файлу вызывает соответствующие методы для обработки событий.
   * При выборе директории сканирует и добавляет файлы в плейлист с помощью метода scanAndAddFiles.
2. **PlaylistComponent:**
   * Отображает список треков в виде таблицы с помощью juce::TableListBox.
   * Обрабатывает события кнопок для воспроизведения треков на разных деках или удаления треков из плейлиста.
   * Добавляет треки в плейлист с помощью метода addTrack.

**Взаимодействие классов**

* FileBrowserComponent взаимодействует с PlaylistComponent для добавления треков в плейлист.
* PlaylistComponent взаимодействует с DeckGUI для воспроизведения треков на разных деках.

Эти классы вместе обеспечивают функциональность для загрузки, отображения и управления треками в приложении.

В файле MainComponent.h FileBrowserComponent получает указатель на PlaylistComponent , для того чтобы добавлять треки.

Следущее нововедение было реализует анимацию воспроизведения трека. Так же это deck колесо кликабельное и его можно крутить тем самым делать перемотку трека вперед и назад. Реализация данного класса была позаимствованна с <https://github.com/deorst/InfiniteRotarySlider>.   
Вот как это выглядит в моем приложении. 

**Логика работы InfiniteRotarySliderComponent**

**Класс InfiniteRotarySliderComponent** представляет собой компонент, который позволяет пользователю управлять значением с помощью бесконечного вращающегося слайдера. Этот компонент использует juce::Slider для обработки ввода пользователя и предоставляет дополнительные методы для настройки и управления поведением слайдера.

Основные методы и их функции:

* **Конструктор и деструктор:**
  + **InfiniteRotarySliderComponent()**: Инициализирует слайдер, устанавливает его стиль и параметры, а также добавляет обработчики событий для изменения значения и начала/окончания перетаскивания.
  + **~InfiniteRotarySliderComponent()**: Деструктор, освобождающий ресурсы.
* **Методы для настройки диапазона и значений:**
  + **void setRange(double newMinimum, double newMaximum)**: Устанавливает минимальное и максимальное значения для слайдера.
  + **void setValue(double newValue)**: Устанавливает новое значение для слайдера.
  + **void setAngle(float newAngle)**: Устанавливает новый угол для слайдера и обновляет конечный угол.
  + **void setOneRevolutionSliderMin(float newMin)**: Устанавливает минимальное значение для одного оборота слайдера.
  + **void setOneRevolutionSliderMax(float newMax)**: Устанавливает максимальное значение для одного оборота слайдера.
  + **void setEndAngle(float newEndAngle)**: Устанавливает конечный угол.
  + **void setStartValue(float newStartValue)**: Устанавливает начальное значение.
  + **void setStartValueNeedUpdate(bool newStartValueNeedUpdate)**: Устанавливает флаг необходимости обновления начального значения.
  + **void setStep(float newStep)**: Устанавливает шаг изменения значения.
  + **void setAdjustedValue(float newAdjustedValue)**: Устанавливает скорректированное значение.
* **Методы для получения значений:**
  + **float getValue()**: Возвращает текущее значение слайдера.
  + **float getAngle()**: Возвращает текущий угол слайдера.
  + **float getAdjustedValue()**: Возвращает скорректированное значение слайдера.
* **Методы для обработки событий:**
  + **void paint(juce::Graphics& g)**: Отрисовывает компонент, включая метку и ручку слайдера.
  + **void resized()**: Устанавливает размеры слайдера в соответствии с размерами компонента.
  + **void m\_processSliderValue(float value)**: Обрабатывает изменение значения слайдера, обновляет угол и скорректированное значение.
  + **void m\_setValueStartNeedUpdate()**: Устанавливает флаг необходимости обновления начального значения.
  + **void m\_setAdjustedAngleEnd()**: Устанавливает конечный угол.
  + **float m\_getSliderRange()**: Возвращает диапазон значений слайдера.

Принципы ООП:

* **Инкапсуляция:** Класс инкапсулирует всю логику работы бесконечного вращающегося слайдера, предоставляя методы для настройки и получения значений, а также для обработки событий.
* **Композиция:** Класс использует объект juce::Slider для обработки ввода пользователя и управления значениями слайдера.
* **Полиморфизм:** Класс переопределяет методы paint и resized из базового класса juce::Component для кастомной отрисовки и изменения размеров компонента.

**Схема работы**

* **Инициализация:**
  + В конструкторе инициализируется слайдер, устанавливаются его параметры и добавляются обработчики событий.
* **Настройка:**
  + Методы setRange, setValue, setAngle и другие используются для настройки диапазона, значений и углов слайдера.
* **Обработка событий:**
  + При изменении значения слайдера вызывается метод m\_processSliderValue, который обновляет угол и скорректированное значение.
  + При начале и окончании перетаскивания вызываются методы m\_setValueStartNeedUpdate и m\_setAdjustedAngleEnd соответственно.
* **Отрисовка:**
  + Метод paint отвечает за отрисовку компонента, включая метку и ручку слайдера.

Этот компонент предоставляет гибкий и настраиваемый интерфейс для управления значениями с помощью бесконечного вращающегося слайдера, что может быть полезно в различных аудиоприложениях.

1. **Music Library Component**
2. **Code Structure and Best Practices**
3. **Conclusion**
4. **References**