ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ  
Федеральное государственное образовательное бюджетное учреждение   
высшего профессионального образования  
«Санкт-Петербургский государственный университет телекоммуникаций  
им. проф. М. А. Бонч-Бруевича»

Факультет инфокоммуникационных сетей и систем

Кафедра программной инженерии и вычислительной техники

ОТЧЁТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Предприятие: ПАО «Интелтех»

Выполнил:

студент 3 курса группы ИКПИ-15

Коновалов Илья Александрович

Руководитель практики от предприятия:

Муравьева Ирина Дмитриевна

Руководитель практики от кафедры:

Муравьева Ирина Дмитриевна

Санкт-Петербург

2024

Оглавление

[1. О предприятии 3](#_Toc171860629)

[2. Цель 4](#_Toc171860630)

[3. Задачи 5](#_Toc171860631)

[4. Ход выполнения работы 6](#_Toc171860632)

[4.1. Макет и его описание 6](#_Toc171860633)

[4.2. Программный код 8](#_Toc171860634)

[4.2.1 Знакомство с JUCE 8](#_Toc171860635)

[4.2.2. Понятие о нативности и кроссплатформенности 8](#_Toc171860636)

[4.2.3. Обработка событий и основные компоненты 9](#_Toc171860637)

[4.2.4. Структура проекта 10](#_Toc171860638)

[5. Выводы 12](#_Toc171860639)

[6. Список источников 13](#_Toc171860640)

[7. Ссылки на инструменты 14](#_Toc171860641)

[8. Приложение 15](#_Toc171860642)

[8.1. Скриншоты программы 15](#_Toc171860643)

[8.2. Листинг программы 17](#_Toc171860644)

# 1. О предприятии

ПАО «Интелтех» является одним из ведущих российских предприятий в области научных исследований и производства автоматизированных систем управления и связи, включая аппаратно-программные комплексы, специализированное телекоммуникационное оборудование и программное обеспечение.

Начиная с середины ХХ века предприятие наращивало опыт разработки, внедрения и эксплуатации программно-технических комплексов, цифровых сетей конфиденциальной связи, систем интеллектуального управления и телекоммуникационного оборудования для государственных ведомств, гражданских и коммерческих организаций России.

Комплексное использование мультисервисного оборудования ПАО «Интелтех» обеспечивает заказчикам возможность реализации гибкой стратегии развития ведомственных систем управления и связи на основе принципов конвергенции и поэтапного внедрения новых сетевых технологий и расширения спектра информационных услуг.

В соответствии с мировыми тенденциями развития информационных и связных технологий предприятие ориентируется на создание универсальной телекоммуникационной техники нового поколения, решающей задачи системной и сетевой интеграции и обеспечения безопасности информационного обмена.

Выпускаемое предприятием оборудование характеризуется использованием новейших аппаратных и программных платформ, высокой надежностью и безопасностью, компактностью и малым энергопотреблением, что соответствует современному уровню требований. Конструкторские решения позволяют использовать оборудование в необходимом заказчикам диапазоне внешних условий.

# 2. Цель

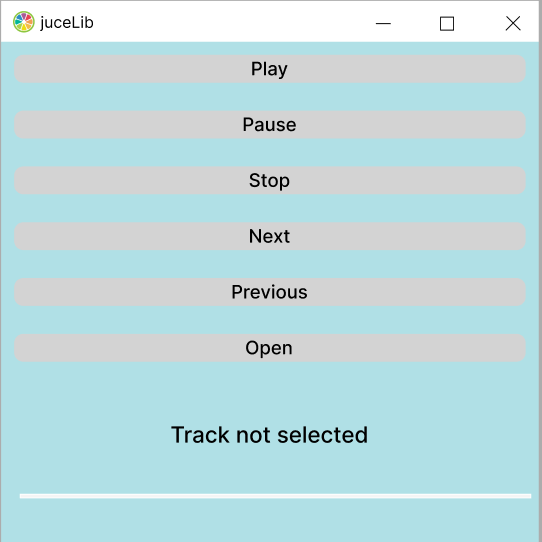
Цель: разработать GUI приложение «Аудио плеер» на языке программирования C++ с помощью кроссплатформенного инструментария JUCE

# 3. Задачи

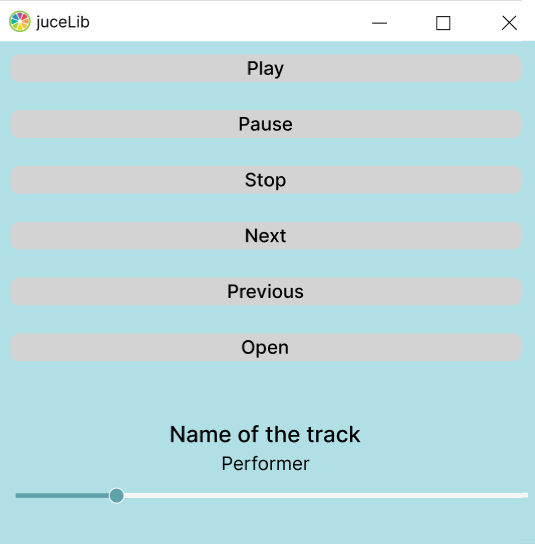
* создание макета графического интерфейса разрабатываемого приложения
* написание программного кода
* написание комментариев к коду
* отладка приложения

# 4. Ход выполнения работы

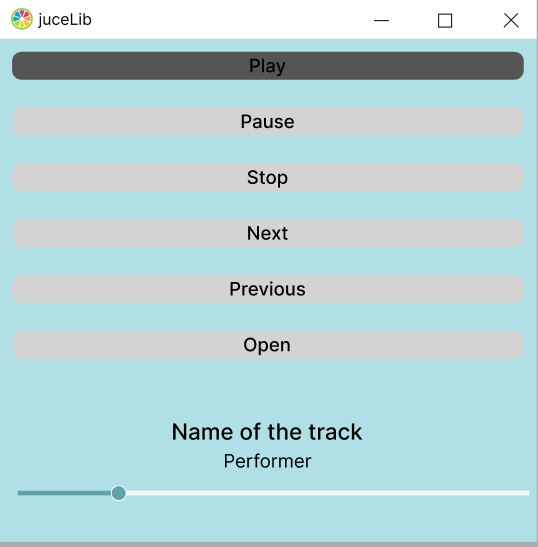
# 4.1. Макет и его описание



*Рис 1. Макет приложения без выбранного трека*



*Рис 2. Макет приложения после добавления файлов через кнопку “open”*



*Рис 3. Макет приложения в момент нажатия на одну из кнопок*

Приложение должно иметь несколько ключевых элементов:

1. Кнопка “Open” (рис 1). Кнопка, открывающая окно выбора файлов формата mp3, aiff, mav. Любая кнопка при нажатии должна менять цвет (рис 3)
2. Label. Если файлы не загружены в приложение – должно выводиться «Track not selected” (рис 1), иначе – название трека и с новой строки более мелким текстом название исполнителя (рис 2 и рис 3)
3. Slider. Слайдер показывает состояние трека и позволяет перемещаться по нему
4. Кнопка “Play”. Позволяет продолжить или начать прослушивание трека
5. Кнопка “Pause”. Позволяет поставить трек на паузу
6. Кнопка “Stop”. Позволяет сбросить состояние прослушиваемого трека
7. Кнопки “Next” и “Previous”. Позволяют перемещаться между выбранными через кнопку “Open” треками

# 4.2. Программный код

# 4.2.1 Знакомство с JUCE

JUCE — наиболее широко используемый фреймворк для разработки аудиоприложений и плагинов. Это открытая кодовая база C++, которую можно использовать для создания автономного программного обеспечения на Windows, macOS, Linux, iOS и Android, а также плагинов VST, VST3, AU, AUv3, AAX и LV2.

JUCE позволяет разработчикам сосредоточиться на самых ценных частях своего программного обеспечения, учитывая различия между операционными системами (как настольными, так и мобильными) и форматами плагинов. С помощью библиотеки строительных блоков цифровой обработки звука (DSP) JUCE вы можете быстро создавать прототипы и выпускать собственные приложения и плагины с единообразным пользовательским интерфейсом на всех поддерживаемых платформах. Использование JUCE также обеспечивает вашим продуктам защиту от будущих обновлений операционной системы и хоста плагинов.

# 4.2.2. Понятие о нативности и кроссплатформенности

Нативность означает, что приложение или библиотека написаны с использованием платформо-специфических API и ресурсов, что позволяет им работать напрямую с операционной системой без промежуточных слоев. Приложения, разработанные с использованием JUCE, взаимодействуют с нативными компонентами и API, что обеспечивает им высокую производительность и доступ к функциональности ОС. Основные аспекты нативности в JUCE:

Использование системных ресурсов: JUCE предоставляет доступ к нативным аудио, графическим и системным ресурсам, таким как таймеры, файловая система и сетевые функции.

Производительность: Прямое взаимодействие с нативными API позволяет добиться высокой производительности.

Интеграция с ОС: JUCE приложения могут интегрироваться с функциями операционной системы, такими как системные меню, уведомления и т. д.

Кроссплатформенность означает, что программа может работать на нескольких операционных системах без необходимости изменения кода. JUCE разработан как кроссплатформенный фреймворк, который позволяет разработчикам писать код один раз и запускать его на различных платформах. Основные аспекты кроссплатформенности в JUCE:

Поддержка нескольких ОС: JUCE поддерживает Windows, macOS, Linux, iOS и Android. Это позволяет создавать приложения, которые могут быть запущены на всех этих платформах.

Абстракция платформо-специфических деталей: JUCE абстрагирует многие платформо-специфические детали, предоставляя унифицированные API для работы с аудио, графикой, вводом-выводом и другими функциями.

Единый код: Разработчики могут писать код один раз и использовать его на всех поддерживаемых платформах, что сокращает время разработки и упрощает обслуживание кода.и кроссплатформенности в JUCE

# 4.2.3. Обработка событий и основные компоненты

Обработка событий в JUCE — это ключевой механизм, который позволяет приложениям реагировать на взаимодействия пользователя, такие как нажатия кнопок, перемещения мыши, изменение значений слайдеров и другие действия. Основные концепции и подходы к обработке событий в JUCE включают:

Основные понятия

* События: Любое взаимодействие пользователя с интерфейсом, такое как нажатие кнопки, перемещение мыши, ввод текста и т.д.
* Обработчики событий (Listeners): Объекты или методы, которые реагируют на определенные события, выполняя заданные действия.
* Основные компоненты и их события
* juce::Component: Базовый класс для всех визуальных элементов. Может обрабатывать события мыши и клавиатуры.
* juce::Button: Кнопки, которые могут обрабатывать события нажатия.
* juce::Slider: Слайдеры, которые могут обрабатывать события изменения значения.
* juce::Timer: Таймеры, которые могут генерировать события через регулярные интервалы времени.

Основные методы и интерфейсы

* Listener интерфейсы: В JUCE для каждого типа события существует соответствующий интерфейс-слушатель, который должен быть реализован классом для обработки этих событий. Например:
* juce::Button::Listener для обработки нажатий кнопок.
* juce::Slider::Listener для обработки изменений значений слайдеров.
* juce::FileDragAndDropTarget для обработки событий перетаскивания файлов.

Обобщение

* Наследование и реализация интерфейсов: JUCE использует наследование и реализацию интерфейсов для обработки событий. Классы реализуют необходимые интерфейсы и переопределяют методы для обработки событий.
* Регистрация слушателей: Компоненты должны регистрировать слушателей (например, addListener для кнопок и слайдеров) для получения событий.
* Периодическое выполнение: Для выполнения действий через регулярные интервалы времени используются таймеры (juce::Timer).

# 4.2.4. Структура проекта

Проект состоит из нескольких файлов, каждый из которых отвечает за определенные аспекты работы приложения.

Заголовочные файлы

Application.h

Описание: Определяет основной класс приложения, который управляет жизненным циклом приложения (запуск, закрытие и т.д.).

Основные функции: Создание главного окна, инициализация ресурсов.

CustomButton.h

Описание: Определяет класс кастомной кнопки, наследующейся от стандартной кнопки JUCE.

Основные функции: Расширяет функциональность стандартной кнопки, добавляя дополнительные стили и обработчики событий.

MainComponent.h

Описание: Основной компонент приложения, отвечающий за отображение интерфейса и управление аудиотреками.

Основные функции: Обработчики событий кнопок, слайдеров, перетаскивания файлов и таймеров. Включает в себя кнопки управления воспроизведением, метку для отображения текущего файла и слайдер для управления позицией воспроизведения.

MainFrame.h

Описание: Определяет главный фрейм (окно) приложения.

Основные функции: Управление основным окном приложения, включая создание и отображение основного компонента.

Исходные файлы

Main.cpp

Описание: Точка входа в приложение.

Основные функции: Создание экземпляра класса приложения и запуск приложения.

MainComponent.cpp

Описание: Реализация методов, объявленных в MainComponent.h.

Основные функции: Определение поведения кнопок, слайдеров и других элементов интерфейса. Реализация методов для воспроизведения аудио, обработки перетаскивания файлов и обновления меток.

MainFrame.cpp

Описание: Реализация методов, объявленных в MainFrame.h.

Основные функции: Определение поведения главного окна приложения, управление его отображением и взаимодействием с пользователем.

# 5. Выводы

В ходе проделанной работы была выполнена серия задач по разработке и документированию приложения «Аудио плеер» на базе JUCE

Основные функции программы включают воспроизведение, паузу, остановку, переход к следующему или предыдущему треку, а также открытие и выбор аудиофайлов. Программа поддерживает перетаскивание файлов, отображает название текущего трека (без расширения) и позволяет управлять позицией воспроизведения с помощью слайдера.

Для создания интуитивно понятного интерфейса были разработаны пользовательские элементы управления, такие как кнопки воспроизведения, паузы, остановки и другие. Также была реализована функциональность для отображения текущего состояния воспроизведения и управления им через графический интерфейс.

В процессе работы были приобретены следующие знания и навыки:

* Работа с библиотекой JUCE для создания кроссплатформенных приложений.
* Понимание принципов обработки событий в JUCE и использование таймеров для обновления интерфейса.
* Навыки работы с аудиофайлами, включая их открытие, чтение и воспроизведение.
* Создание пользовательских элементов управления и использование стандартных компонентов JUCE, таких как кнопки, метки и слайдеры.
* Работа с Doxygen для документирования кода и генерации документации.
* Понимание принципов кроссплатформенной разработки и использования AudioDeviceManager для работы с аудиоустройствами.

Для разработки программы были использованы следующие инструменты:

* JUCE: Основная библиотека для создания графического интерфейса и обработки аудиофайлов.
* Doxygen: Инструмент для генерации документации из комментариев в коде.
* C++: Язык программирования, на котором написано приложение.
* IDE Visual Studio: Интегрированная среда разработки, такая как Visual Studio или CLion, для написания и отладки кода.
* Git: Система контроля версий для управления изменениями в коде и совместной работы.

Для дальнейшего развития программы можно рассмотреть следующие доработки и улучшения:

* Поддержка плейлистов: Добавить возможность создания и управления плейлистами, чтобы пользователи могли составлять свои наборы треков.
* Эквалайзер и эффекты: Реализовать эквалайзер и различные аудиоэффекты, такие как реверберация, эхо и другие, для улучшения качества воспроизведения.
* Интерфейс пользователя: Улучшить пользовательский интерфейс, добавив анимации, более продвинутую графику, используя отрисованные кнопки вместо текстовых шаблонных кнопок
* Расширенные настройки аудио: Предоставить пользователям доступ к настройкам аудиоустройств и параметрам воспроизведения, таким как баланс, громкость и качество звука.
* Поддержка дополнительных форматов файлов: Расширить поддержку форматов аудиофайлов для обеспечения совместимости с большим числом аудиоисточников.
* Сохранение состояния: Реализовать функциональность для сохранения текущего состояния программы (позиция трека, плейлист и т.д.) при закрытии и его восстановления при следующем запуске.
* Тестирование и отладка: Провести тщательное тестирование и устранение ошибок, а также оптимизировать производительность приложения.

# 6. Список источников

Официальная документация JUCE: <https://docs.juce.com/master/index.html>

Репозиторий JUCE на GitHub: <https://github.com/juce-framework/JUCE>

Официальная документация Doxygen: <https://www.doxygen.nl/manual/index.html>

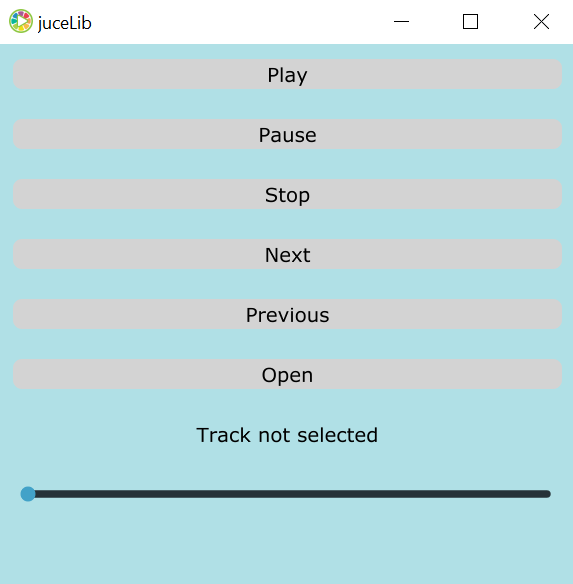
Лекции по работе с JUCE от «ИНТУИТ»: <https://intuit.ru/studies/courses/2338/638/lecture/13837>

# 7. Ссылки на инструменты

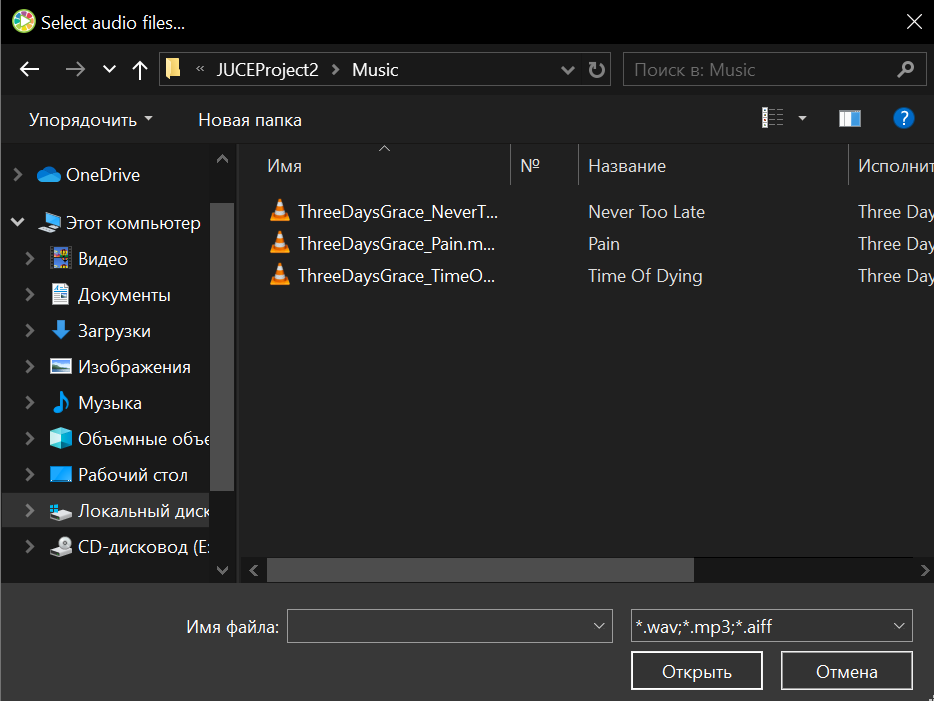
Репозиторий GiHub: <https://github.com/IlyaDev1/IntelKek>

# 8. Приложение

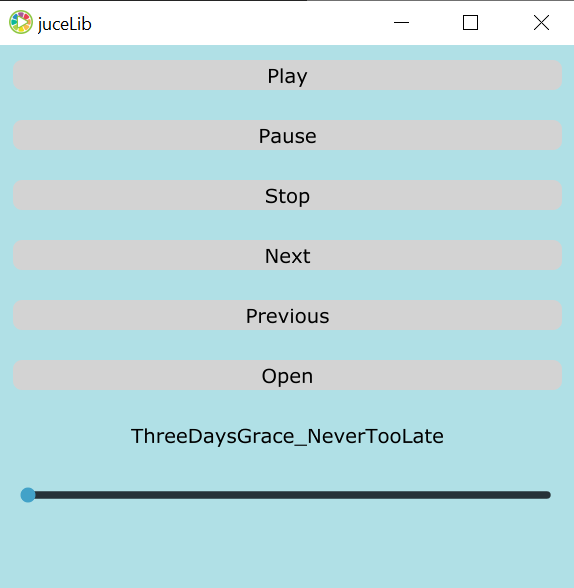
# 8.1. Скриншоты программы



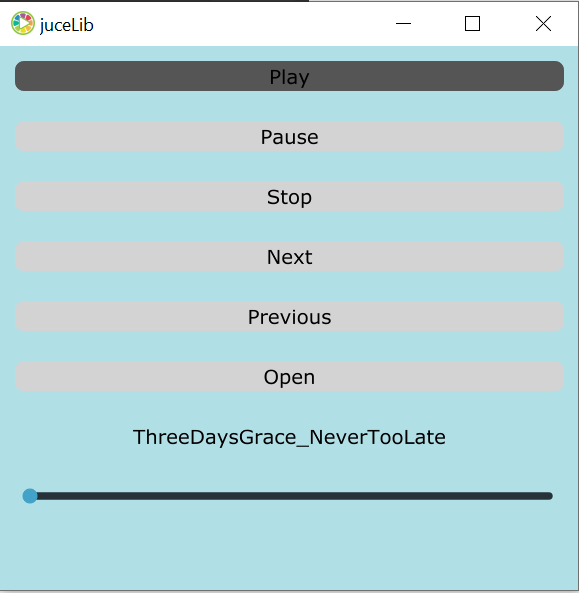
*Рис 4. Программа после запуска*



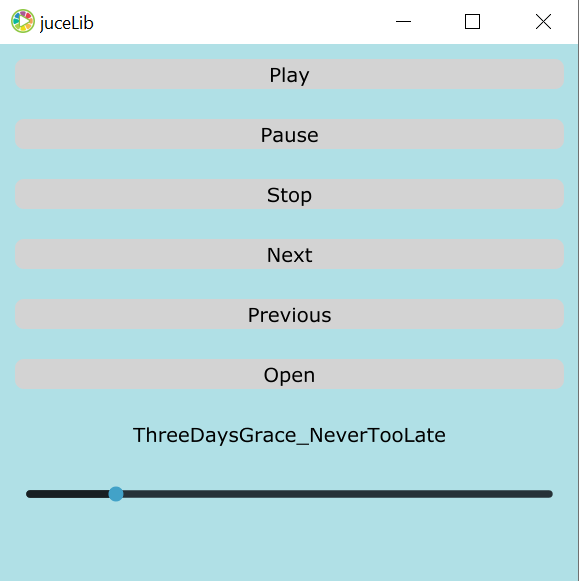
*Рис 5. Открытие диалогового окна после нажатия на кнопку «Open»*



*Рис 6. Программа после выбора файлов*



*Рис 7. Изменение цвета заливки кнопки по ее нажатии*



*Рис 8. Движение ползунка слайдера по мере проигрывания трека*

# 8.2. Листинг программы

|  |
| --- |
| Application.h |
| #pragma once  //  #include "../JuceLibraryCode/JuceHeader.h"  #include "../Include/MainFrame.h"  //  class PracticeApplication : public JUCEApplication  {  public:  PracticeApplication(void) {}  ~PracticeApplication(void) override {}  //  const String getApplicationName(void) override { return ProjectInfo::projectName; }  const String getApplicationVersion(void) override { return ProjectInfo::versionString; }  bool moreThanOneInstanceAllowed(void) override { return true; }  //  void initialise(const String&) override  {  mainFrame.reset(new MainFrame(getApplicationName()));  }  //  void shutdown(void) override { mainFrame = nullptr; }  //  void systemRequestedQuit(void) override { quit(); }  void anotherInstanceStarted(const String&) override {}  //  private:  std::unique\_ptr<MainFrame> mainFrame;  };  //  START\_JUCE\_APPLICATION(PracticeApplication) |

*Таблица 1. Файл Application.h*

|  |
| --- |
| CustomButton.h |
| #pragma once  #include "../JuceLibraryCode/JuceHeader.h"  class CustomButton : public juce::Button  {  public:  CustomButton(const juce::String& buttonName) : juce::Button(buttonName) {}  void paintButton(juce::Graphics& g, bool isMouseOverButton, bool isButtonDown) override  {  auto bounds = getLocalBounds().toFloat();  g.setColour(isButtonDown ? juce::Colours::darkgrey : juce::Colours::lightgrey);  g.fillRoundedRectangle(bounds, 6.0f);  g.setColour(juce::Colours::black);  g.setFont(juce::Font(16.0f));  g.drawText(getButtonText(), bounds, juce::Justification::centred, true);  }  }; |

*Таблица 2. Файл CustomButton.h*

|  |
| --- |
| MainComponent.h |
| /\*\*  \* @file MainComponent.h  \*  \* @brief Основной компонент приложения для управления аудиотреками.  \*/  #pragma once  #include "../JuceLibraryCode/JuceHeader.h"  #include "CustomButton.h"  /\*\*  \* @class MainComponent  \*  \* @brief Основной компонент приложения, отвечающий за отображение интерфейса и управление аудиотреками.  \*/  class MainComponent : public juce::Component,  public juce::Button::Listener,  public juce::FileDragAndDropTarget,  public juce::Slider::Listener,  public juce::Timer  {  public:  /\*\*  \* @brief Конструктор MainComponent.  \*/  MainComponent();  /\*\*  \* @brief Деструктор MainComponent.  \*/  ~MainComponent() override;  /\*\*  \* @brief Переопределенная функция отрисовки компонента.  \*  \* @param g Объект Graphics для рисования.  \*/  void paint(juce::Graphics& g) override;  /\*\*  \* @brief Переопределенная функция изменения размеров компонента.  \*/  void resized() override;  /\*\*  \* @brief Обработчик события нажатия кнопок.  \*  \* @param button Указатель на нажатую кнопку.  \*/  void buttonClicked(juce::Button\* button) override;  /\*\*  \* @brief Проверка, заинтересован ли компонент в перетаскивании файлов.  \*  \* @param files Массив путей к перетаскиваемым файлам.  \* @return true, если компонент заинтересован в перетаскивании файлов, иначе false.  \*/  bool isInterestedInFileDrag(const juce::StringArray& files) override;  /\*\*  \* @brief Обработчик события перетаскивания файлов в компонент.  \*  \* @param files Массив путей к перетаскиваемым файлам.  \* @param x Координата X точки, где произошло событие.  \* @param y Координата Y точки, где произошло событие.  \*/  void filesDropped(const juce::StringArray& files, int x, int y) override;  /\*\*  \* @brief Обработчик изменения значения слайдера.  \*  \* @param slider Указатель на слайдер, значение которого изменилось.  \*/  void sliderValueChanged(juce::Slider\* slider) override;  /\*\*  \* @brief Callback-функция таймера, вызывается периодически.  \*/  void timerCallback() override;  private:  CustomButton playButton{ "Play" }; /\*\*< Кнопка воспроизведения. \*/  CustomButton pauseButton{ "Pause" }; /\*\*< Кнопка паузы. \*/  CustomButton stopButton{ "Stop" }; /\*\*< Кнопка остановки. \*/  CustomButton nextButton{ "Next" }; /\*\*< Кнопка следующего трека. \*/  CustomButton prevButton{ "Previous" }; /\*\*< Кнопка предыдущего трека. \*/  CustomButton openButton{ "Open" }; /\*\*< Кнопка открытия файла. \*/  juce::Label currentFileLabel; /\*\*< Метка для отображения текущего файла. \*/  juce::Slider positionSlider; /\*\*< Слайдер для управления позицией воспроизведения. \*/  AudioFormatManager formatManager; /\*\*< Менеджер форматов аудиофайлов. \*/  std::unique\_ptr<AudioFormatReaderSource> readerSource; /\*\*< Источник данных для воспроизведения аудио. \*/  AudioTransportSource transportSource; /\*\*< Объект для управления воспроизведением аудио. \*/  AudioSourcePlayer sourcePlayer; /\*\*< Плеер аудиоданных. \*/  AudioDeviceManager deviceManager; /\*\*< Менеджер аудиоустройств. \*/  /\*\*  \* @brief Обработчик нажатия кнопки открытия файла.  \*/  void openButtonClicked();  /\*\*  \* @brief Воспроизведение следующего трека.  \*/  void playNextTrack();  /\*\*  \* @brief Воспроизведение предыдущего трека.  \*/  void playPreviousTrack();  /\*\*  \* @brief Обновление метки текущего файла.  \*/  void updateCurrentFileLabel();  Array<File> audioFiles; /\*\*< Массив выбранных аудиофайлов. \*/  int currentIndex = -1; /\*\*< Индекс текущего выбранного трека. \*/  JUCE\_DECLARE\_NON\_COPYABLE\_WITH\_LEAK\_DETECTOR(MainComponent)  }; |

*Таблица 3. Файл MainComponent.h*

|  |
| --- |
| MainFrame.h |
| #pragma once  //  #include "../JuceLibraryCode/JuceHeader.h"  #include "../Include/MainComponent.h"  //  class MainFrame : public DocumentWindow  {  public:  //  MainFrame(const String& name);  //  void closeButtonPressed(void) override { JUCEApplication::getInstance()->systemRequestedQuit(); }  //  MainComponent& getMainComponent(void) { return \*dynamic\_cast<MainComponent\*> (getContentComponent()); }  //  private:  JUCE\_DECLARE\_NON\_COPYABLE\_WITH\_LEAK\_DETECTOR(MainFrame)  }; |

*Таблица 4. Файл MainFrame.h*

|  |
| --- |
| Main.cpp |
| /\*  ==============================================================================  This file is part of the JUCE library.  Copyright (c) 2022 - Raw Material Software Limited  JUCE is an open source library subject to commercial or open-source  licensing.  By using JUCE, you agree to the terms of both the JUCE 7 End-User License  Agreement and JUCE Privacy Policy.  End User License Agreement: www.juce.com/juce-7-licence  Privacy Policy: www.juce.com/juce-privacy-policy  Or: You may also use this code under the terms of the GPL v3 (see  www.gnu.org/licenses).  JUCE IS PROVIDED "AS IS" WITHOUT ANY WARRANTY, AND ALL WARRANTIES, WHETHER  EXPRESSED OR IMPLIED, INCLUDING MERCHANTABILITY AND FITNESS FOR PURPOSE, ARE  DISCLAIMED.  ==============================================================================  \*/  #include <JuceHeader.h>  #include "../../Assets/DemoUtilities.h"  #include "UI/MainComponent.h"  //==============================================================================  #if JUCE\_MAC || JUCE\_WINDOWS || JUCE\_LINUX || JUCE\_BSD  // Just add a simple icon to the Window system tray area or Mac menu bar..  struct DemoTaskbarComponent : public SystemTrayIconComponent,  private Timer  {  DemoTaskbarComponent()  {  setIconImage (getImageFromAssets ("juce\_icon.png"),  getImageFromAssets ("juce\_icon\_template.png"));  setIconTooltip ("JUCE demo runner!");  }  void mouseDown (const MouseEvent&) override  {  // On OSX, there can be problems launching a menu when we're not the foreground  // process, so just in case, we'll first make our process active, and then use a  // timer to wait a moment before opening our menu, which gives the OS some time to  // get its act together and bring our windows to the front.  Process::makeForegroundProcess();  startTimer (50);  }  // This is invoked when the menu is clicked or dismissed  static void menuInvocationCallback (int chosenItemID, DemoTaskbarComponent\*)  {  if (chosenItemID == 1)  JUCEApplication::getInstance()->systemRequestedQuit();  }  void timerCallback() override  {  stopTimer();  PopupMenu m;  m.addItem (1, "Quit");  // It's always better to open menus asynchronously when possible.  m.showMenuAsync (PopupMenu::Options(),  ModalCallbackFunction::forComponent (menuInvocationCallback, this));  }  };  #endif  std::unique\_ptr<AudioDeviceManager> sharedAudioDeviceManager;  //==============================================================================  class DemoRunnerApplication : public JUCEApplication  {  public:  //==============================================================================  DemoRunnerApplication() {}  ~DemoRunnerApplication() override  {  sharedAudioDeviceManager.reset();  }  const String getApplicationName() override { return ProjectInfo::projectName; }  const String getApplicationVersion() override { return ProjectInfo::versionString; }  bool moreThanOneInstanceAllowed() override { return true; }  //==============================================================================  void initialise (const String& commandLine) override  {  registerAllDemos();  #if JUCE\_MAC || JUCE\_WINDOWS || JUCE\_LINUX || JUCE\_BSD  // (This function call is for one of the demos, which involves launching a child process)  if (invokeChildProcessDemo (commandLine))  return;  #else  ignoreUnused (commandLine);  #endif  mainWindow.reset (new MainAppWindow (getApplicationName()));  }  bool backButtonPressed() override { mainWindow->getMainComponent().getSidePanel().showOrHide (false); return true; }  void shutdown() override { mainWindow = nullptr; }  //==============================================================================  void systemRequestedQuit() override { quit(); }  void anotherInstanceStarted (const String&) override {}  ApplicationCommandManager& getGlobalCommandManager() { return commandManager; }  private:  class MainAppWindow : public DocumentWindow  {  public:  MainAppWindow (const String& name)  : DocumentWindow (name, Desktop::getInstance().getDefaultLookAndFeel()  .findColour (ResizableWindow::backgroundColourId),  DocumentWindow::allButtons)  {  setUsingNativeTitleBar (true);  setResizable (true, false);  setResizeLimits (400, 400, 10000, 10000);  #if JUCE\_IOS || JUCE\_ANDROID  setFullScreen (true);  auto& desktop = Desktop::getInstance();  desktop.setOrientationsEnabled (Desktop::allOrientations);  desktop.setKioskModeComponent (this);  #else  setBounds ((int) (0.1f \* (float) getParentWidth()),  (int) (0.1f \* (float) getParentHeight()),  jmax (850, (int) (0.5f \* (float) getParentWidth())),  jmax (600, (int) (0.7f \* (float) getParentHeight())));  #endif  setContentOwned (new MainComponent(), false);  setVisible (true);  #if JUCE\_MAC || JUCE\_WINDOWS || JUCE\_LINUX || JUCE\_BSD  taskbarIcon.reset (new DemoTaskbarComponent());  #endif  }  void closeButtonPressed() override { JUCEApplication::getInstance()->systemRequestedQuit(); }  #if JUCE\_IOS || JUCE\_ANDROID  void parentSizeChanged() override  {  getMainComponent().resized();  }  #endif  //==============================================================================  MainComponent& getMainComponent() { return \*dynamic\_cast<MainComponent\*> (getContentComponent()); }  private:  std::unique\_ptr<Component> taskbarIcon;  JUCE\_DECLARE\_NON\_COPYABLE\_WITH\_LEAK\_DETECTOR (MainAppWindow)  };  std::unique\_ptr<MainAppWindow> mainWindow;  ApplicationCommandManager commandManager;  };  ApplicationCommandManager& getGlobalCommandManager()  {  return dynamic\_cast<DemoRunnerApplication\*> (JUCEApplication::getInstance())->getGlobalCommandManager();  }  //==============================================================================  // This macro generates the main() routine that launches the app.  START\_JUCE\_APPLICATION (DemoRunnerApplication) |

*Таблица 5. Файл Main.cpp*

|  |
| --- |
| MainComponent.cpp |
| #include "../Include/MainComponent.h"  MainComponent::MainComponent(void)  {  setOpaque(true);  setSize(800, 800);  // Инициализация кнопок  addAndMakeVisible(playButton);  playButton.addListener(this);  addAndMakeVisible(pauseButton);  pauseButton.addListener(this);  addAndMakeVisible(stopButton);  stopButton.addListener(this);  addAndMakeVisible(nextButton);  nextButton.addListener(this);  addAndMakeVisible(prevButton);  prevButton.addListener(this);  addAndMakeVisible(openButton);  openButton.addListener(this);  // Инициализация метки  addAndMakeVisible(currentFileLabel);  currentFileLabel.setJustificationType(juce::Justification::centred);  currentFileLabel.setFont(juce::Font(16.0f));  currentFileLabel.setColour(Label::textColourId, Colours::black); // Сделать текст черным  updateCurrentFileLabel(); // Обновление начального текста  // Инициализация слайдера  addAndMakeVisible(positionSlider);  positionSlider.setRange(0.0, 1.0);  positionSlider.setSliderStyle(juce::Slider::LinearHorizontal);  positionSlider.setTextBoxStyle(juce::Slider::NoTextBox, true, 0, 0);  positionSlider.addListener(this);  // Настройка аудио  formatManager.registerBasicFormats();  deviceManager.initialise(0, 2, nullptr, true);  deviceManager.addAudioCallback(&sourcePlayer);  sourcePlayer.setSource(&transportSource);  startTimer(100); // Запуск таймера с интервалом 100 мс  }  MainComponent::~MainComponent(void)  {  stopTimer(); // Остановка таймера  sourcePlayer.setSource(nullptr);  deviceManager.removeAudioCallback(&sourcePlayer);  }  void MainComponent::paint(Graphics& g)  {  g.fillAll(Colours::powderblue);  }  void MainComponent::resized(void)  {  auto area = getLocalBounds();  playButton.setBounds(area.removeFromTop(40).reduced(10));  pauseButton.setBounds(area.removeFromTop(40).reduced(10));  stopButton.setBounds(area.removeFromTop(40).reduced(10));  nextButton.setBounds(area.removeFromTop(40).reduced(10));  prevButton.setBounds(area.removeFromTop(40).reduced(10));  openButton.setBounds(area.removeFromTop(40).reduced(10));  currentFileLabel.setBounds(area.removeFromTop(40).reduced(10));  positionSlider.setBounds(area.removeFromTop(40).reduced(10));  }  void MainComponent::buttonClicked(Button\* button)  {  if (button == &playButton)  {  transportSource.start();  }  else if (button == &pauseButton)  {  transportSource.stop();  }  else if (button == &stopButton)  {  transportSource.stop();  transportSource.setPosition(0.0);  }  else if (button == &nextButton)  {  playNextTrack();  }  else if (button == &prevButton)  {  playPreviousTrack();  }  else if (button == &openButton)  {  openButtonClicked();  }  }  bool MainComponent::isInterestedInFileDrag(const StringArray& files)  {  for (auto& file : files)  if (formatManager.findFormatForFileExtension(File(file).getFileExtension()) != nullptr)  return true;  return false;  }  void MainComponent::filesDropped(const StringArray& files, int x, int y)  {  for (auto& file : files)  {  auto\* reader = formatManager.createReaderFor(File(file));  if (reader != nullptr)  {  std::unique\_ptr<AudioFormatReaderSource> newSource(new AudioFormatReaderSource(reader, true));  transportSource.setSource(newSource.get(), 0, nullptr, reader->sampleRate);  readerSource.reset(newSource.release());  break;  }  }  updateCurrentFileLabel();  }  void MainComponent::openButtonClicked()  {  FileChooser chooser("Select audio files...",  File::getSpecialLocation(File::userMusicDirectory),  "\*.wav;\*.mp3;\*.aiff");  if (chooser.browseForMultipleFilesToOpen())  {  audioFiles.clear();  Array<File> chosenFiles = chooser.getResults();  for (const auto& file : chosenFiles)  {  audioFiles.add(file);  }  if (!audioFiles.isEmpty())  {  currentIndex = 0;  auto\* reader = formatManager.createReaderFor(audioFiles[currentIndex]);  if (reader != nullptr)  {  std::unique\_ptr<AudioFormatReaderSource> newSource(new AudioFormatReaderSource(reader, true));  transportSource.setSource(newSource.get(), 0, nullptr, reader->sampleRate);  readerSource.reset(newSource.release());  }  }  }  updateCurrentFileLabel();  }  void MainComponent::playNextTrack()  {  if (!audioFiles.isEmpty() && currentIndex < audioFiles.size() - 1)  {  currentIndex++;  auto\* reader = formatManager.createReaderFor(audioFiles[currentIndex]);  if (reader != nullptr)  {  std::unique\_ptr<AudioFormatReaderSource> newSource(new AudioFormatReaderSource(reader, true));  transportSource.setSource(newSource.get(), 0, nullptr, reader->sampleRate);  readerSource.reset(newSource.release());  transportSource.start();  }  }  updateCurrentFileLabel();  }  void MainComponent::playPreviousTrack()  {  if (!audioFiles.isEmpty() && currentIndex > 0)  {  currentIndex--;  auto\* reader = formatManager.createReaderFor(audioFiles[currentIndex]);  if (reader != nullptr)  {  std::unique\_ptr<AudioFormatReaderSource> newSource(new AudioFormatReaderSource(reader, true));  transportSource.setSource(newSource.get(), 0, nullptr, reader->sampleRate);  readerSource.reset(newSource.release());  transportSource.start();  }  }  updateCurrentFileLabel();  }  void MainComponent::updateCurrentFileLabel()  {  if (currentIndex >= 0 && currentIndex < audioFiles.size())  {  currentFileLabel.setText(audioFiles[currentIndex].getFileNameWithoutExtension(), juce::dontSendNotification);  }  else  {  currentFileLabel.setText("Track not selected", juce::dontSendNotification);  }  }  void MainComponent::sliderValueChanged(juce::Slider\* slider)  {  if (slider == &positionSlider)  {  if (transportSource.isPlaying())  {  transportSource.setPosition(transportSource.getLengthInSeconds() \* positionSlider.getValue());  }  }  }  void MainComponent::timerCallback()  {  if (transportSource.isPlaying())  {  double currentPosition = transportSource.getCurrentPosition();  double trackLength = transportSource.getLengthInSeconds();  if (trackLength > 0.0)  {  positionSlider.setValue(currentPosition / trackLength, dontSendNotification);  }  }  } |

*Таблица 6. Файл MainComponent.cpp*

|  |
| --- |
| MainFrame.cpp |
| #include "../JuceLibraryCode/JuceHeader.h"  #include "../Include/MainFrame.h"  #include "../Include/Application.h"  MainFrame::MainFrame(const String& name)  : DocumentWindow(name, Desktop::getInstance().getDefaultLookAndFeel()  .findColour(ResizableWindow::backgroundColourId),  DocumentWindow::allButtons)  {  setUsingNativeTitleBar(true);  setResizable(true, false);  setResizeLimits(400, 400, 10000, 10000);  setContentOwned(new MainComponent(), false);  setVisible(true);  } |

*Таблица 7. Файл MainFrame.cpp*