Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

 К ЗАЩИТЕ ДОПУСТИТЬ

\_\_\_\_\_\_\_\_\_\_\_ М. Г. Базанов

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

«Кроссплатформенный медиаплеер»

по дисциплине

«Системное программное обеспечение вычислительных машин»

БГУИР КР 1-40 02 01 505 ПЗ

Выполнил:                                                                   Руководитель:

студент гр. 650505                                                     Базанов М. Г.

Болошенко С. В.

МИНСК 2018

Учреждение образования

«Белорусский государственный университет информатики

и радиоэлектроники»

Факультет: Компьтерных систем и сетей

Кафедра: Электронных вычислительных машин

Специальность: 40 02 01 «Вычислительные машины, системы и сети»

Предмет: Системное и программное обеспечение электронных вычислительных машин

УТВЕРЖДАЮ Заведующий кафедрой ЭВМ \_\_\_\_\_\_\_\_\_ Д. И. Самаль

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 20\_\_ г.

ЗАДАНИЕ

по курсовому проектированию студента

Болошенко Степана Витальевича

1. Тема проекта: «Кроссплатформенный медиаплеер»

2. Дата выдачи задания: 20.02.2018

3. Предоставление студентом готового проекта: 22.05.2018

4. Решаемые задачи и функционал разрабатываемого ПО:

4.1 Работа с аудиофайлами и видеофайлами.

4.1.1 Возможность воспроизведения аудиофайлов и видеофайлов.

4.1.2 Управление воспроизведением аудиофайла и видеофайлов.

4.2 Создание собственного плейлиста.

4.2.1 Добавления аудиофайлов и видеофайлов в список проигрывания.

4.3 Наложение звуковой дорожки на видеоряд.

4.3.1 Выбор звуковой дорожки.

4.4 Отображение гистограммы частот звука.

4.5 Настройка видео.

4.5.1 Контрастность, яркость, оттенок, насыщенность.  
 4.6 Настройка скорости воспроизведения.

5. ОС и средства разработки:

5.1. *Qt Creator.*

5.2. *Windows* 10.

5.3. *Ubuntu* 16.04.3.

6. Другое:

7. Содержание пояснительной записки:

Введение. 1. Постановка задачи. 2. Структурное проектирование. 3. Функциональное проектирование. 4. Разработка программных модулей. 5. Программа и методика испытаний. 6. Руководство пользователя. Заключение. Литература. Приложения.

8. Перечень графического материала:

8.1. Кроссплатформенный медиаплеер. Схема структурная. Формат А4.

8.2. Кроссплатформенный медиаплеер. Диаграмма классов. Формат А3.

8.3 Определение положения курсора. Схема программы. Формат А4.

8.3. Добавление аудиофайла в плейлист. Схема программы. Формат А4.

8.4 Листинг программы. Формат А4

9. Контрольные точки готовности частей проекта:

9.1. 01.03 – введение, глава 1 (постановка задачи), лист задания.

9.2. 01.04 – предварительная структурная схема и диаграмма классов.

9.3. 01.05 – предварительная демонстрация ПО.

9.4. 10.05 – структурная схема, диаграмма классов, главы 1-7.

РУКОВОДИТЕЛЬ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Базанов М. Г.

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Болошенко С. В.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ…...…………………………………………………………….……..4

1.ПОСТАНОВКА ЗАДАЧИ ……………………………………....………….…5

1.1 Основные термины…………………………………………………………...5

1.2 Определение основных функций…….……………………………………...5

1.3 Выбор технологии для создания проекта…………………………………...5

1.4 Сравнительный анализ программных конкурентов………………………..6

2 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ…………………………………….....7

2.1 Общее структурное описание состава…..…………………………………...7

2.2 Описание деление проекта…......…….………….…………………………...7

2.3 Краткое описание сторонних программных компонент......…………..…...8

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ…….……………………….....9

* 1. Описание «\*.mp3» и «\*.wav» файлов………………………………….……9

3.2 Описание «\*.avi» файла……………….……………………………………14

3.3 Описание функционирования программы…………………...……………14

4 РАЗРАБОТКА ПРОГРАМНЫХ МОДУЛЕЙ………………………..………15

4.1 Описание теоретических основ для разработки алгоритмов……………..15

* 1. Описание алгоритмов, разработанных самостоятельно…………………...17

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ…..………………..………..19

5.1 Проверка работоспособности……………………………….………………19

* 1. Качественные и количественные характеристики……………………....…19

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ…………………………………………20

* 1. Системные требования……………...……………………………………….20
  2. Запуск программы…………………...…………………………………….…20

6.3 Использование программы…………….……………..……………………..20

ЗАКЛЮЧЕНИЕ………………………………………………………………….24

СПИСОК ИСТОЧНИКОВ………………………………………………………25

ПРИЛОЖЕНИЕ А……………………………………………………………….26

ПРИЛОЖЕНИЕ Б……………………………………………………………….27

ПРИЛОЖЕНИЕ В……………………………………………………………….28

ПРИЛОЖЕНИЕ Г……………………………………………………………….29 ПРИЛОЖЕНИЕ Д……………………………………………………………….30 ПРИЛОЖЕНИЕ Е……………………………………………………………….42

**ВВЕДЕНИЕ**

Целью данной работы является разработка программного обеспечения для воспроизведения и проигрывания аудиофайлов (\*.mp3 и \*.wav файлов) и видеофайлов (\*.avi и \*.mp4), а также настройка изображения видеофайла и наложение звуковой дорожки на видеофайл.   
 Идея данного проекта заключается в создании общедоступного и комфортного в использовании медиаплеера.

Разработка данного проекта производиться для упрощения работы с пользователя с аудиофайлами и видеофайлами.

Проект будет включать следующие функции:

- воспроизведение аудиофайлов;

- составление плейлиста;

- отображение гистограммы частот звука;

- настройка изображения видеофайла.

- настройка скорости воспроизведения.

Личная заинтересованность заключается:

* в развитии навыков работы с музыкальными файлами и видеофайлами;
* в развитии и поддержке данного проекта в будущем;
* в использовании конечного продукта.

При дальнейшем развитии проекта планируется создать собственный сервер, на котором пользователи смогут делиться композициями и оценивать их, а также перенос медеаплеера на мобильное устройство.

1. **ПОСТАНОВКА ЗАДАЧИ**

**1.1 Основные термины**

Использованные термины:

* **Пользовательский интерфейс** – совокупность средств и методов, с помощью которых пользователь может взаимодействовать с различными процессами и аппаратурой;
* **Графический пользовательский интерфейс (от англ. GUI – “Graphical User Interface”)** – разновидность пользовательского интерфейса, в котором средства управления представлены графически;
* **Фреймворк** (от англ. Framework) – программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

**1.2 Определение основных функций**

Основные функции, выполняемые разрабатываемым проектом:

* воспроизведение (воспроизведение установленного музыкального файла или видеофайла, при нажатии соответствующей клавиши);
* остановка проигрывания (остановка установленного музыкального файла или видеофайла, при нажатии соответствующей клавиши);
* пролистывание (переключение установленного музыкального файла или видеофайл, при нажатии соответствующей клавиши, на следующий аудиофайл или видеофайл в списке);
* настройка громкости (уменьшение или увеличение громкости);
* выбор порядка воспроизведения (в хаотичном порядке или последовательно);
* совместное воспроизведение видеофайла и звуковой дорожки.
* изменение аудиодорожки в видеоряде путем наложение ее

на этот видеоряд;

* настройка видео путем редактирования контрастности, яркости, насыщенности и т.д.

**1.3 Выбор технологии для создания проекта**

При разработке проекта было определено использовать уже готовые разработки в плане:

1. Среда разработки для языка программирования C++ **- IDE Microsoft Visual Studio 2015**.
2. Для разработки пользовательского графического интерфейса – **Фреймворк Qt**.

При разработке проекта были использованы следующие источники:

* Для языка программирования С++:

*Р.Лафоре – «Объектно-ориентированное программирование в C++».*

* Для работы с фреймворком Qt:
* документация Qt <http://doc.qt.io/>.
* Учебное пособие по программированию в среде Qt – *Е. Р. Алексеев, Г. Г. Злобин, Д. А. Костюк, О.В. Чеснокова, А. С.*

*Чмыхало – «Программирование на языке C++ в среде Qt Creator»*

**1.4 Сравнительный анализ аппаратных конкурентов**

Главным конкурентом данного проекта является приложение *jetAudio* от компании *Cowon*[1]. Оно позволяет не только воспроизводить, но и преобразовывать типы файлов.

Однако это приложение является платным и сложным в использовании, а также с закрытым исходным кодом. Поэтому главным отличием разрабатываемого проекта от приведённого аналога заключается в доступности, а также плюсом является совместное воспроизведение аудио и видеоряда.

[1]- http://www.jetaudio.com/

1. **СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ**
   1. **Общее структурное описание состава**

Разрабатываемый проект содержит в себе 3 основных модуля:

1) Менеджер медиафайлов.

2) Хранилище медиафайлов.

3) Проигрыватель медиафайлов:

- контроллер проигрывателя;

- плеер.

Структурная схема разрабатываемого проекта представлена на рисунке 2.1.



Рисунок 2.1 – структурная схема медиаплеера

* 1. **Описание деления проекта**

1. ***GUI*** – блок, отвечающий за средства связи приложения с пользователем. Cвязан с блоками *Audio Player, Video Player* и *Files Manager*. Для *Audio Player* отображает пользователю клавиши, позволяющие переключать аудиофайлы, настраивать громкость, перематывать аудиофайл, настраивать режимы воспроизведения. Для *, Video Player* отображает клавиши управления видеофайлом, редактирования видеофайла. Для *Files Manager* отображает файлы, уже хранящиеся в *Files Storage*, и позволяет пользователю добавлять или удалять из него аудио файлы.
2. ***Files Storage*** – блок, отвечающий за управление связями с аудиофайлами и видеофайлами. Связан с *GUI, Files Managere*. Для *GUI* передаёт список уже хранящихся аудиофайлов и видеофайлов для отображения. Для *Files Manager* позволяет добавлять и удалять аудио файлы.
3. ***Files Manager***– блок, отвечающий за хранение путей к аудиофайлам. Связан с *Audio Player, Video Player* и *GUI.* Для *Audio Player* передает аудиофайлы для воспроизведения, для *Video Player* передает видеофайл для воспроизведения, для *GUI* позволяет добавлять аудиофайлы в плейлист.
4. ***Audio Player*** – блок, отвечающий за воспроизведение аудиофайлов. Связан с *GUI*, *Files Manager.* Для *GUI* позволяет переключать аудиофайлы, настраивать громкость. Для *Files Storage* получает файлы для воспроизведения.
5. ***Video Player*** – блок, отвечающий за воспроизведение видеофайлов. Связан с *GUI*, *Files Manager.* Для *GUI* позволяет управлять видеофайлом(переключать), менять звуковые дорожки, редактировать видео. Для *Files Manager* получает файлы для воспроизведения.
   1. **Краткое описание сторонних программных компонент**

Для того, чтобы упростить разработку использовался *фреймворк Qt*. Этот *фреймворк* удобен для дизайна графического интерфейса, а также работы с аудио и видеофайлами.

1. **ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**
   1. **Описание «\*.mp3» и «\*.wav» файлов**

Музыкальные файлы имеют расширение «\*.*mp3*», «\*.*wav*», рассмотрим их поподробнее. Любой звук представляет из себя колебания с частотой, которая характеризует тон звучания, и амплитудой, которая характеризует громкость звучания. Компьютер, с помощью различных устройств вывода звуковой информации может генерировать колебания заданной частотой и амплитудой.

В звуковом файле содержатся базовые элементы – фреймы. Фрейм – основная единица в звуковом файле. В ней содержаться данные, с помощью которых компьютер и может генерировать конкретный звук, а именно – частота и амплитуда. Так, как любая музыкальная композиция обязательно должна содержать изменения тонов, то и при воспроизведении всего файла, необходимо воспроизводить конкретный фрейм в конкретном диапазоне времени. Именно из-за этого существует понятие скорости воспроизведения музыкального файла - битрейт (битов в секунду). Для «*wa»v* эта скорость постоянна, а для «*mp3*» – хранится в специально отведённом для этого месте в файле – в заголовке фрейма.

Рассмотрим чуть глубже «*mp3»* файлы. В музыкальной композиции часто используется несколько инструментов. Для хранения музыкальной композиции в файле существует разбиение всей композиции на несколько каналов. При воспроизведении все каналы должны воспроизводиться параллельно, однако хранение в файле не может осуществляться параллельно. Поэтому для хранения каналов последовательно применяется теорема Котельникова – “Любую функцию *F(t)*, состоящую из частот от 0 до *f*1 можно непрерывно передавать с любой точностью при помощи чисел, следующих друг за другом через *1/(*2*f*1*)* секунд”. Применительно к музыкальным файлам это означает, что n каналов будут кодироваться как n фреймов, расположенных друг за другом. Информация о количестве каналов, вместе с остальной информацией о «*mp3»* файле храниться в заголовке.

На данный момент существуют 2 версии заголовка «*mp3»* файла: «*ID3V1*» и «*ID3V2*». Заголовок файла храниться в конце или начале файла и хранит информацию о названии трека, альбома, имени исполнителя и т. д., которая используются в различных медиа-проигрывателях.

Структура «*ID3V1»* представлена в таблице 3.1.

Хранится в конце файла и имеет постоянную длину. Данные заголовка используются медиа-проигрывателями, чтобы сортировать список треков по исполнителю, году записи, жанру, а также отображать всю эту информацию при воспроизведении.

Таблица 3.1 – Структура «*ID3V1*»

|  |  |  |
| --- | --- | --- |
| Поле | Длина, байт | Описание |
| *Header* | 3 | Маркер, всегда содержит “TAG” |
| *Title* | 30 | Название трека |
| *Artist* | 30 | Исполнитель |
| *Album* | 30 | Название альбома |
| *Year* | 4 | Год |
| *Comment* | 28(ID3V1.1) или 30(ID3V1.0) | Комментарий |
| *Zero-byte* | 1(ID3V1.1) | 1 – если указан номер трека |
| *Track* | 1(ID3V1.1) | Номер трека |
| *Genre* | 1 | Индекс жанра. Если не указан - 255 |

Структура «*ID3V2*» представлена в таблице 3.2:

Таблица 3.2 – Структура «*ID3V2*»

|  |  |  |
| --- | --- | --- |
| Поле | Длина, байт | Описание |
| *Header* | 3 | Маркер. Всегда содержит “ID3” |
| *Version* | 1 | Индекс версии заголовка |
| *Subversion* | 1 | Индекс суб-версии заголовка |
| *Tags* | 1 | Используется 3 бита: *abc*00000  a (“*Unsynchronisation*”) – используется только с MPEG-2 и MPEG-2.5 форматами  b (“*Extended header*”) – указывает наличие расширенного заголовка  c (“*Experimental indicator*”) – экспериментальный индикатор |
| *Length* | 4 | Длина тегов. 7 бит в каждом байте не используется и всегда установлен в 0. |

Длина указывается для всех следующих после него тегов. седьмой бит в каждом байте в длине не учитывается, например:

Таблица 3.3 – Пример битов

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 2 | 1 | 0 |
| 00000000 | 00000000 | 00000111 | 01110110 |

Полученное значение – 11101110110b байт

Итоговое значение – 1111110110b байт

В десятичной форме – 1014d байт

У каждого фрейма в «*mp3*» файле есть свой заголовок, размером в 4 байта. Его структура представлена в таблице 3.4:

Таблица 3.4 – Структура заголовка фрейма

|  |  |  |
| --- | --- | --- |
| Поле | Занимаемые биты | Описание |
| *Frame sync* | 0-10 | Маркер. Полностью заполнен 1 |
| *Audio version ID* | 11-12 | Индекс версии MPEG:  00 – MPEG-2.5  01 – не используется  10 – MPEG-2  11 – MPEG-1 |
| *Layer index* | 13-14 | Индекс версии Layer:  00 – не используется  01 – Layer III  10 – Layer II  11 – Layer I |
| *Protection bit* | 15 | Бит защиты:  1 – нет защиты  0 – заголовок защищён 16 бит, следует за заголовком |

|  |  |  |
| --- | --- | --- |
| *Bitrate index* | 16-19 | Индекс битрейта, представлен в таблице 3.4 |
| *Sampling rate index* | 20-21 | Индекс частоты дискретизации, представлен в таблице 3.5 |
| *Padding bit* | 22 | Бит смещения. Если он установлен, то данные смещаются на 1 байт |
| *Private* | 23 | Бит Private только для информации |
| *Channel mode* | 24-25 | Индекс режима канала:  00 – Stereo  01 – Joint stereo  10 – Dual channel  11 – Mono |
| *Mode extension* | 26-27 | Расширение режима канала, используется с Joint stereo |
| *Copyright bit* | 28 | Бит только для информации |
| *Original bit* | 29 | Бит только для информации |
| *Emphasis* | 30-31 | Акцент, в настоящее время практически не используется |

*Продолжение таблицы 3.4*

С помощью данного заголовка можно определить основную информацию о самом фрейме. При проектировании программы были применены библиотеки, позволяющие декодировать полученные фреймы и передать на аудио-выводящее устройство компьютера.

Таблица 3.5 – Значения индекса битрейта

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Значение | MPEG-1 Layer I | MPEG-1 Layer II | MPEG-1 Layer III | MPEG-2 Layer I MPEG-2.5 Layer I | MPEG-2 Layer II, Layer III и MPEG-2.5 Layer II, Layer III |
| 0000 | не используется | | | | |
| 0001 | 32кб/с | 32 кб/с | 32 кб/с | 32 кб/с | 8 кб/с |
| 0010 | 64кб/с | 48 кб/с | 40 кб/с | 48 кб/с | 16 кб/с |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0011 | 96кб/с | 56 кб/с | 48 кб/с | 56 кб/с | 24 кб/с |
| 0100 | 128кб/с | 64 кб/с | 56 кб/с | 64 кб/с | 32 кб/с |
| 0101 | 160кб/с | 80 кб/с | 64 кб/с | 80 кб/с | 40 кб/с |
| 0110 | 192кб/с | 96 кб/с | 80 кб/с | 96 кб/с | 48 кб/с |
| 0111 | 224кб/с | 112 кб/с | 96 кб/с | 112 кб/с | 56 кб/с |
| 1000 | 256кб/с | 128 кб/с | 112 кб/с | 128 кб/с | 64 кб/с |
| 1001 | 288кб/с | 160 кб/с | 128 кб/с | 144 кб/с | 80 кб/с |
| 1010 | 320кб/с | 192 кб/с | 160 кб/с | 160 кб/с | 96 кб/с |
| 1011 | 352кб/с | 224 кб/с | 192 кб/с | 176 кб/с | 112 кб/с |
| 1100 | 384кб/с | 256 кб/с | 224 кб/с | 192 кб/с | 128 кб/с |
| 1101 | 416кб/с | 320 кб/с | 256 кб/с | 224 кб/с | 144 кб/с |
| 1110 | 448кб/с | 384 кб/с | 320 кб/с | 256 кб/с | 160 кб/с |
| 1111 | не используется | | | | |

*Продолжение таблицы 3.5*

Таблица 3.6 – Значения индекса частоты дискретизации

|  |  |  |  |
| --- | --- | --- | --- |
| Значение | MPEG-1 | MPEG-2 | MPEG-2.5 |
| 00 | 44100 Гц | 22050 Гц | 11025 Гц |
| 01 | 48000 Гц | 24000 Гц | 12000 Гц |
| 10 | 32000 Гц | 16000 Гц | 8000 Гц |
| 11 | не используется | | |

* 1. **Описание «\*.avi» файла**

Рассмотрим подробнее форматы *«\*.avi»*. Формат данных с расширением *«\*.avi»* может содержать видео и аудиоданные, сжатые с использованием разных комбинаций кодеков, что позволяет синхронно воспроизводить видео со звуком. Минусы такого формата это невозможность поддержки аудио-потоков с переменным битрейтом (*VBR*).

Данные в *«avi»*. хранятся в виде последовательностей записей, каждая из которых состоит из одного кадра и соответствующего звукового сопровождения.

* 1. **Описание функционирования программы**

Данное программное обеспечение, исходя из его функционала, должно воспроизводить аудио и видеофайлы. Данный функционал реализован с помощью передачи аудиофайла или видеофайла на выходное аудиоустройство при нажатии соответствующей клавиши. При нажатии клавиши “Stop” программа должна проверить, воспроизводится ли файл и остановить его, также при нажатии клавиш перемотки.

Длительность воспроизведения файла контролируется с помощью таймера. Громкость регулируется изменением значения переменной, от которой и зависит громкость. Настройка яркости, контрастности и т.д. осуществляется с помощью перемещения ползунка. Хранилище аудио или видеофайлов представлено в виде списка строк, в которых хранится название и путь к самим аудиофайлам.

1. **РАЗРАБОТКА ПРОГРАМНЫХ МОДУЛЕЙ**
   1. **Описание теоретических основ для разработки алгоритмов**

При разработке программы был использован принцип сигналов и слотов, реализованный в *фреймворке Qt*. Такой принцип основан на том, что, когда происходит какое-либо событие, оно отправляет сигнал, и этот сигнал вызывает выполнение связанного с этим сигналом слота. При помощи такого принципа и осуществляется воспроизведение видео или аудиофайла.

Все функции программы реализованы в проекте с помощью следующих классов, указанных в диаграмме классов (Приложение А).

***Widget*** – главный класс, который осуществляет связь действий пользователя с самой программой.

Основные методы:

**-** *void**on\_btn\_add\_clicked()* – добавляет список видимых аудиофайлов в плейлист.

- *void on\_btn\_del\_clicked()* – удаляет аудиофайл из списка воспроизведения, а также останавливает поток его воспроизведения.

- *void on\_btn\_random\_clicked()* – переключение типа воспроизведения в произвольном или в алфавитном порядках.

- *void setPosition(int)* – устанавливает время воспроизведении при перемотке аудиофайлов.

- *void createTaskbar*() – отображает в панели задач прогресс воспроизведения аудиофайла.

- *void createThumbnailToolBar()* – добавляет кнопки остановки/проигрывания аудиофайла, а также перемотки в панель задач.

- *void mouseMoveEvent(QMouseEvent\*)* – устанавливает в какой области находится курсор мыши и в зависимости от этого осуществляется управление клавишами плеера, а также размерами окна.

- *void mousePressEvent(QMouseEvent\*)* - устанавливает, была ли нажата левая клавиша мыши.

- *void mouseReleaseEvent(QMouseEvent\*)* – устанавливает, была ли отжата левая клавиша мыши.

***Player*** – класс, который осуществляет воспроизведение видеофайлов, а также накладывание звуковой дорожки.

Основные методы:

**-** *void open() –* копирование путей видеофайлов, которые нужно добавить в плейлист. **-** *void open\_music() –* копирование путей аудиофайлов, которые нужно добавить в плейлист.

**-** *void del() –* удаление путей и самих видеофайлов из плейлиста.

**-** *void del\_music() –* удаление путей и самих аудиофайлов из плейлиста.

**-** *void addToPlaylist(const QList<QUrl> &) –* добавление видеозаписей по путям в плейлист.

**-** *void addToPlaylis\_music(const QList<QUrl> &) –* добавление аудиозаписей по путям в плейлист.

***PlayerControls*** – класс, отвечающий за кнопки управления видео или аудиофайлом: остановка, пауза, запуск, перемотка.

***PlaylistModel*** – класс, отвечающий за модель плейлиста, в который будут добавляться видео или аудиофайлы.

***FrameProcessor*** – класс, за отображение гистограммы для аудио и видеофайла.

Основные методы:

*- void paintEvent(QPaintEvent \*) override –* метод, отвечающий за отображение гистограммы.

*- void processFrame(QVideoFrame, int) –* метод, отвечающий за получение значений частот.

*- QVector<qreal> getBufferLevels(const QAudioBuffer&) ) –* метод, который возвращает уровень для каждого канала.

***VolumeButton* –** класс клавиши регулирования громкости.

Основные методы:

- *int volume() const* – возвращает текущее значение громкости, установленное в классе *Widget*.

- *void increaseVolume() –* перемещаеткаретку на ползунке при увеличении громкости вправо.

- *void descreaseVolume()* – перемещает каретку на ползунке при уменьшении громкости влево.

- *void setVolume(int)* – устанавливает громкость воспроизведения.

***Style*** – класс, отвечающий за внешний вид плеера.

Основные методы:

- *static QString getWindowStyleSheet() –* внешний вид главного окна.

*- static QString getCloseStyleSheet() –* кнопка закрытия приложения.

*- static QString getPlayStyleSheet() –* кнопка воспроизведения.

*- static QString getTableViewStyleSheet() –* внешний вид плейлиста.

*- static QString getSliderStyleSheet() –* внешний вид ползунка перемотки аудиофайла.

*- static QString getSettingsStyleSheet() –* внешний вид окна настройки изображения видеофайла.

*- static QString getComboBoxStyleSheet() –* внешний вид клавиши выбора скорости воспроизведения.

* 1. **Описание алгоритмов, разработанных самостоятельно**

Основные алгоритмы программы представлены в приложениях: определение положения курсора мыши (Приложение В) и добавление аудиофайла в плейлист (Приложение Г).

Рассмотрим подробно отслеживание курсора мыши:

void Widget:: mouseMoveEvent(QMouseEvent \*eventMouse)

{

Далее проверяем статус нажатия кнопки мыши:

switch (m\_leftMouseButtonPressed) {

case Move: {

Проверка открыто ли окно на максимум:

if (isMaximized()) {

Необходимо вернуть окно в стандартное состояние:

lt\_1()->setMargin(9);

auto prt = eventMouse ->screenPos().x() / width();

this->showNormal();

auto offsetX = width() \* prt;

Пересчитываем положение окна относительно стандартных размеров:

setGeometry(eventMouse->screenPos().x() - offsetX, 0,

width(), height());

setPreviousPosition(QPoint(offsetX, eventMouse->y()));

} else {

Иначе просто перемещаем окно:

auto newX = eventMouse->x() - m\_previousPosition.x();

auto newY = eventMouse->y() - m\_previousPosition.y();

setGeometry(x() + newX, y() + newY, width(), height());

}

break;

}

case Top: {

Для изменения размера окна пересчитаем его координаты:

if (!isMaximized()) {

auto newY = eventMouse->y() - m\_previousPosition.y();

setGeometry(x(), y() + newY, width(), height() - newY);

}

break;

}

case Bottom: {

if (!isMaximized()) {

auto newY = eventMouse ->y() - m\_previousPosition.y();

setGeometry(x(), y(), width(), height() + newY);

setPreviousPosition(eventMouse ->pos());

}

break;

}

case Left: {

if (!isMaximized()) {

auto newX = eventMouse ->x() - m\_previousPosition.x();

setGeometry(x() + newX, y(), width() - newX, height());

}

break;

}

case Right: {

if (!isMaximized()) {

auto newX = eventMouse ->x() - m\_previousPosition.x();

setGeometry(x(), y(), width() + newX, height());

setPreviousPosition(eventMouse->pos());

}

break;

}

default:

Курсор перемещается по окну без зажатой кнопки:

checkResizableField(eventMouse);

break;

}

return QWidget::mouseMoveEvent(eventMouse);

}

**5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ**

**5.1 Проверка работоспособности**

Данный медиаплеер тестировался на следующей конфигурации устройства:

-Операционная система: Windows 10 Pro;

-Процессор: Intel® Core™ i3-5005U CPU @ 2.00 GHz 2.00 GHz;

-Оперативная память(ОЗУ): 8 Гб.

При запуске программы через *mediaplayer.exe* появляется окно (см. рис. 6.2). После добавления аудиофайлов в плейлист и нажатия клавиши «*Play*»

медиаплеер начнет воспроизведение.

Допустимые форматы воспроизведения: «\*.*mp3*» и «\*.*wav*», «\*.*avi*», другие форматы не будет отображаться в окне выбора медиафайлов.

**6.2 Качественные и количественные характеристики**

При формировании качественных характеристик были проведены тесты, включающие в себя попытку воспроизведения других типов данных, открытия большого количества медиафайлов. Ко всем тестам программа была устойчива. Также была проведено тестирование быстродействия программы, с которым она также справилась.

**6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

**6.1 Системные требования**

Операционная система: Windows 7 64 bit и выше.

Оперативная память: 128 MB.

Жесткий диск: 60 MB свободного места.

**6.2 Запуск программы**

Для запуска программы необходимо:

1) скачать архив *mediaplayer.zip*.

2) разархивировать в любую папку.

3) запустить исполняемый файл *mediaplayer.exe*. Перед этим убедившись, что установлены все требуемые библиотеки.

**6.3 Использование программы**

При запуске программы, откроется главное окно программы (см. рисунок 6.1). Первым делом необходимо выбрать: аудиоплеер или видеоплеер. Для этого необходимо кликнуть по кнопке «*MUSIC*» или «*VIDEO*» на в окне программы (1).



Рисунок 6.1 – медиаплеер до выбора способа работы

По нажатию клавиши «*MUSIC*» появится кнопки управления аудиоплеером. А при нажатии клавиши «*VIDEO*» откроется новое окно управления видеоплеером.



Рисунок 6.2 – аудиоплеер после выбора аудиофайлов

Для добавления аудиофайла в плейлист надо нажать на клавишу (3) (см. рисунок 6.2). Далее откроется диалоговое окно выбора аудиофайлов.

После выбора желаемого звукового файла и его воспроизведения, название текущего аудиофайла будет отображаться в поле (8). Управление плейлистом производится клавишами (5), удаление выбранного аудиофайла из плейлиста производится нажатием клавиши «*Remove track*», расположенной рядом с добавлением (4). Также перемотку аудиофайла назад или вперед можно осуществлять перемещением ползунка влево или вправо соответственно (6). Выбор типа воспроизведения аудиофайлов осуществляется нажатием клавиши «*Type of payback*». Гистограмма частот аудио видна на панели номер 7.

Далее представлены типы воспроизведения:

1.  - в произвольном порядке;
2.  - последовательно.

Регулировка уровня громкости осуществляется клавишей «*Volume*», после нажатия появляется меню с ползунком, который отвечает за уровень громкости (см. рисунок 6.3).



Рисунок 6.3 – регулировка уровня громкости

Поподробнее разберем поле с управляющими клавишами (5). При нажатии на клавишу «*Play*» плеер начнет воспроизведение аудиофайла, выбранного в плейлисте. Для того, чтобы поставить аудиофайл на паузу надо нажать на клавишу «*Pause*», для остановки воспроизведения аудиофайла надо нажать на клавишу «*Stop*», переключение на следующий или предыдущий аудиофайл в плейлисте осуществляется нажатием на клавиши «*Next track*» и «*Previous track*» соответственно.



Рис 6.4 – отображение медиаплеера на панели задач (только в *Windows*)

При нажатии на клавишу (1) (см. рисунок 6.4) осуществляется остановка или воспроизведение аудиофайла, при нажатии на клавиши (2) и (3) происходит перемотка аудиофайла назад или вперед соответственно. В поле (4) отображается текущее время воспроизведения аудиофайла.



Рисунок 6.5 – видеоплеер после выбора видеофайлов

Добавление видеофайлов в плейлист производится точно также, как и в аудиоплеере. В поле (1) (см. рисунок 6.5) отображается текущий видеофайл, в поле (2) –отображается список видеофайлов, доступных для воспроизведения, в поле (3) – список аудиофайлов для наложения на видеоряд. Нажатием клавиш в поле (4) происходит регулировка скорости воспроизведения. Нажатием клавиши (5) осуществляется переход в полноэкранный режим, при нажатии клавиши (6) осуществляется настройка видео. В полях (7) и (8) отображается гистограмма видео и аудиофайла соответственно.



Рисунок 6.6 – настройка видео

Настройка видео осуществляется регулированием соответствующего ползунка (см. рисуное 6.6).

**ЗАКЛЮЧЕНИЕ**

Результатом данной курсовой работы является ПО, предназначенное для проигрывания формата «\*.*mp3*», «\*.*wav*» и «\*.*avi*», а также наложение звуковой дорожки на видеофайл, настройки изображения видео, настройки скорости воспроизведения, отображения гистограммы частот звука, настройки громкости и выбора режима воспроизведения. Приложение не имеет жестоких системных требований.

В будущем планируется продолжать поддержку проекта, расширить существующий функционал, и сделать проект более доступным. Планируется создать ресурс, где пользователи данного приложения смогут делится своими плейлистами друг с другом. Также в будущем планируется перенос данного приложения на мобильные устройства для удобства использования удаленно от компьютера.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Документация QtCreator [Электронный ресурс]. – Режим доступа : <http://doc.qt.io/>

[2] Документация по QSS [Электронный ресурс]. – <http://doc.crossplatform.ru/qt/4.6.x/>

[3] *Е.Р. Алексеев, Г. Г. Злобин, Д. А.Костюк,О.В.Чеснокова, А.С.Чмыхало.* Программирование на языке С++ в среде Qt Creator. - М. : ALT Linux, 2015. -448 с. : ил. - (Библиотека ALT Linux).

[4] *Керниган Б., Ритчи Д.* Язык программирования Си. - М.: Вильямс, 2013. –304 с.

**ПРИЛОЖЕНИЕ А**

(*обязательное*)

Схема структурная

**ПРИЛОЖЕНИЕ Б**

(*обязательное*)

Диаграмма классов

**ПРИЛОЖЕНИЕ В**

(*обязательное*)

Схема определения положения курсора

**ПРИЛОЖЕНИЕ Г**

(*обязательное*)

Схема добавления аудиофайла в плейлист

**ПРИЛОЖЕНИЕ Д**

(*обязательное*)

Листинг программы

#ifndef WIDGET\_H

#define WIDGET\_H

#include <QWidget>

#include <QStandardItemModel>

#include <QMediaPlayer>

#include <QMediaPlaylist>

#include <QMouseEvent>

#include <QFileDialog>

#include <QDir>

#include <QGraphicsDropShadowEffect>

#include <QMediaMetaData>

#ifdef WIN32

#include <QtWinExtras>

#endif

#include "volumebutton.h"

#include "histogramwidget.h"

#include "player.h"

#include "style.h"

QT\_FORWARD\_DECLARE\_CLASS(QSlider)

QT\_FORWARD\_DECLARE\_CLASS(QWinTaskbarButton)

QT\_FORWARD\_DECLARE\_CLASS(QWinTaskbarProgress)

QT\_FORWARD\_DECLARE\_CLASS(QAudioProbe)

*class* HistogramWidget;

*class* VolumeButton;

*namespace* Ui {

*class* Widget;

}

*class* Widget : *public* QWidget

{

Q\_OBJECT

*//* *Свойство* *с* *точкой* *предыдущей* *позиции* *мыши*

*//* *Относительно* *данной* *точки* *идёт* *пересчёт* *позиции* *окна*

*//* *Или* *размеров* *окна.* *При* *этом* *свойство* *устанавливается* *при* *нажатии* *мыши*

*//* *по* *окну* *и* *в* *ряде* *иных* *случаев*

Q\_PROPERTY(QPoint previousPosition READ previousPosition WRITE setPreviousPosition NOTIFY previousPositionChanged)

*//* *тип* *клика* *мыши,* *при* *перемещении* *курсора* *по* *этому* *типу* *будем* *определять*

*//* *что* *именно* *нужно* *сделать,* *перенести* *окно,* *или* *изменить* *его* *размер* *с* *одной* *из* *сторон*

*enum* MouseType {

*None* = 0,

*Top*,

*Bottom*,

*Left*,

*Right*,

*Move*

};

*private*:

Ui::Widget \*ui = *nullptr*;

Player \*player = *nullptr*;

HistogramWidget \*m\_audioHistogram = *nullptr*;

QAudioProbe \*m\_audioProbe = *nullptr*;

QStandardItemModel \*m\_playListModel = *nullptr*;

QMediaPlayer \*m\_player = *nullptr*;

QMediaPlaylist \*m\_playlist = *nullptr*;

VolumeButton \*m\_volumeButton = *nullptr*;

#ifdef WIN32

QWinTaskbarProgress \*m\_taskbarProgress = *nullptr*;

QWinTaskbarButton \*m\_taskbarButton = *nullptr*;

QWinThumbnailToolBar \*thumbnailToolBar = *nullptr*;

QWinThumbnailToolButton \*playToolButton = *nullptr*;

QWinThumbnailToolButton \*forwardToolButton = *nullptr*;

QWinThumbnailToolButton \*backwardToolButton = *nullptr*;

#endif

*//* *Переменная,* *от* *которой* *будем* *отталкиваться* *при* *работе* *с* *перемещением* *и* //*изменением* *размера* *окна*

MouseType m\_leftMouseButtonPressed;

QPoint m\_previousPosition;

MouseType checkResizableField(QMouseEvent\*);

#ifdef WIN32

void createTaskbar();

void createThumbnailToolBar();

#endif

void clearHistogram();

*private* *slots*:

void on\_btn\_add\_clicked();

void on\_btn\_del\_clicked();

void on\_btn\_random\_clicked();

void updatePosition(qint64);

void updateDuration(qint64);

void setPosition(int);

#ifdef WIN32

void updateTaskbar();

void updateThumbnailToolBar();

#endif

void on\_btn\_music\_clicked();

void on\_btn\_video\_clicked();

void on\_btn\_close\_clicked();

*protected*:

void *mousePressEvent*(QMouseEvent\*);

void *mouseReleaseEvent*(QMouseEvent\*);

void *mouseMoveEvent*(QMouseEvent\*);

*public*:

*explicit* Widget(QWidget \*parent = *nullptr*);

~*Widget*();

QPoint previousPosition() *const*;

void addToPlaylist(*const* QList<QUrl> &urls);

*public* *slots*:

void setPreviousPosition(QPoint);

void togglePlayback();

void seekForward();

void seekBackward();

*signals*:

void previousPositionChanged(QPoint);

};

#endif *//* *WIDGET\_H*

#include <QAudioProbe>

#include "widget.h"

#include "ui\_widget.h"

#include "player.h"

#include "histogramwidget.h"

Widget::Widget(QWidget \*parent) :

QWidget(parent),

ui(*new* Ui::Widget),

m\_leftMouseButtonPressed(*None*)

{

ui->setupUi(*this*);

*///* *Настройка* *UI*

*this*->setWindowFlags(Qt::*FramelessWindowHint*); *//* *Отключаем* *оформление* *окна*

*this*->setAttribute(Qt::*WA\_TranslucentBackground*); *//* *Делаем* *фон* *главного* *виджета* *прозрачным*

*this*->setStyleSheet(Style::getWindowStyleSheet()); *//* *Устанавливаем* *стиль* *виджета*

*this*->setMouseTracking(*true*); *//* *Включаем* *отслеживание* *курсора* *без* *нажатых* *кнопок*

*///* *Создаём* *эффект* *тени*

QGraphicsDropShadowEffect \*shadowEffect = *new* QGraphicsDropShadowEffect(*this*);

shadowEffect->setBlurRadius(9); *//* *Устанавливаем* *радиус* *размытия*

shadowEffect->setOffset(0); *//* *Устанавливаем* *смещение* *тени*

ui->widgetInterface->setGraphicsEffect(shadowEffect); *//* *Устанавливаем* *эффект* *тени* *на* *окно*

ui->widgetInterface->layout()->setMargin(0); *//* *Устанавливаем* *размер* *полей*

ui->widgetInterface->layout()->setSpacing(0);

ui->label->setAlignment(Qt::*AlignHCenter* | Qt::*AlignVCenter*);

*///* *Установка* *стилей* *для* *всех* *элементов*

ui->currentTrack->setStyleSheet("color:#c1c1c1;");

ui->btn\_close->setStyleSheet(Style::getCloseStyleSheet());

ui->btn\_maximize->setStyleSheet(Style::getMaximizeStyleSheet());

ui->btn\_minimize->setStyleSheet(Style::getMinimizeStyleSheet());

ui->btn\_next->setStyleSheet(Style::getNextStyleSheet());

ui->btn\_previous->setStyleSheet(Style::getPreviousStyleSheet());

ui->btn\_stop->setStyleSheet(Style::getStopStyleSheet());

ui->btn\_play->setStyleSheet(Style::getPlayStyleSheet());

ui->btn\_pause->setStyleSheet(Style::getPauseStyleSheet());

ui->btn\_add->setStyleSheet(Style::getMenuStyleSheet());

ui->btn\_del->setStyleSheet(Style::getRemoveStyleSheet());

ui->btn\_add->setStyleSheet(Style::getAddStyleSheet());

ui->btn\_random->setStyleSheet(Style::getRandomStyleSheet());

ui->btn\_music->setStyleSheet(Style::getBtnToolStyleSheet());

ui->btn\_video->setStyleSheet(Style::getBtnToolStyleSheet());

ui->playlistView->setStyleSheet(Style::getTableViewStyleSheet());

ui->positionSlider->setStyleSheet(Style::getSliderStyleSheet());

ui->playlistView->verticalScrollBar()->setStyleSheet(Style::getVerticalScrollBarStyleSheet());

ui->playlistView->horizontalScrollBar()->setStyleSheet(Style::getHorizontalScrollBarStyleSheet());

ui->btn\_add->setToolTip("Add track");

ui->btn\_del->setToolTip("Remove track");

ui->btn\_next->setToolTip("Next track");

ui->btn\_pause->setToolTip("Pause");

ui->btn\_play->setToolTip("Play");

ui->btn\_previous->setToolTip("Previous track");

ui->btn\_stop->setToolTip("Stop");

ui->btn\_random->setToolTip("Type of payback");

ui->btn\_next->setCursor(Qt::*PointingHandCursor*);

ui->btn\_previous->setCursor(Qt::*PointingHandCursor*);

ui->btn\_stop->setCursor(Qt::*PointingHandCursor*);

ui->btn\_play->setCursor(Qt::*PointingHandCursor*);

ui->btn\_pause->setCursor(Qt::*PointingHandCursor*);

m\_playListModel = *new* QStandardItemModel(*this*);

ui->playlistView->*setModel*(m\_playListModel);

ui->horizontalLayout->setSpacing(0);

ui->horizontalLayout\_2->setSpacing(0);

ui->playlistView->verticalHeader()->*setVisible*(*false*);

ui->playlistView->setSelectionBehavior(QAbstractItemView::*SelectRows*);

ui->playlistView->setSelectionMode(QAbstractItemView::*SingleSelection*);

ui->playlistView->setEditTriggers(QAbstractItemView::*NoEditTriggers*);

ui->playlistView->horizontalHeader()->setStretchLastSection(*true*);

m\_player = *new* QMediaPlayer(*this*);

m\_playlist = *new* QMediaPlaylist(m\_player);

m\_player->setPlaylist(m\_playlist);

m\_playlist->setPlaybackMode(QMediaPlaylist::*Loop*);

connect(ui->btn\_previous, &QToolButton::clicked, m\_playlist, &QMediaPlaylist::previous);

connect(ui->btn\_next, &QToolButton::clicked, m\_playlist, &QMediaPlaylist::next);

connect(ui->btn\_play, &QToolButton::clicked, m\_player, &QMediaPlayer::play);

connect(ui->btn\_pause, &QToolButton::clicked, m\_player, &QMediaPlayer::pause);

connect(ui->btn\_stop, &QToolButton::clicked, m\_player, &QMediaPlayer::stop);

connect(ui->btn\_del, &QToolButton::clicked, m\_player, &QMediaPlayer::stop);

*///* *Устанавливаем* *громкость* *воспроизведения* *треков*

m\_volumeButton = *new* VolumeButton(*this*);

m\_volumeButton->setToolTip(tr("Volume"));

m\_volumeButton->setVolume(m\_player->volume());

m\_volumeButton->setStyleSheet("QToolButton::menu-indicator{image:none;}");

connect(m\_volumeButton, &VolumeButton::volumeChanged, m\_player, &QMediaPlayer::setVolume);

ui->gridLayout\_3->addWidget(m\_volumeButton,0,0);

*///Устанавливаем* *перемотку* *треков* *и* *время* *вопроизыведения*

connect(m\_player, &QMediaPlayer::positionChanged, *this*, &Widget::updatePosition);

connect(m\_player, &QMediaPlayer::durationChanged, *this*, &Widget::updateDuration);

connect(ui->positionSlider, &QAbstractSlider::valueChanged, *this*, &Widget::setPosition);

*///* *коннекты* *для* *кнопок* *сворачивания/максимизации/минимизации/закрытия*

connect(ui->btn\_minimize, &QToolButton::clicked, *this*, &QWidget::showMinimized);

connect(ui->btn\_maximize, &QToolButton::clicked, [*this*](){

*if* (*this*->isMaximized()) {

ui->btn\_maximize->setStyleSheet(Style::getMaximizeStyleSheet());

*this*->layout()->setMargin(9);

*this*->showNormal();

} *else* {

ui->btn\_maximize->setStyleSheet(Style::getRestoreStyleSheet());

*this*->layout()->setMargin(0);

*this*->showMaximized();

}

});

connect(ui->btn\_close, &QToolButton::clicked, *this*, &QWidget::close);

connect(ui->playlistView, &QTableView::doubleClicked, [*this*](*const* QModelIndex &index){

m\_playlist->setCurrentIndex(index.row());});

connect(m\_playlist, &QMediaPlaylist::currentIndexChanged, [*this*](int index){

ui->currentTrack->setText(m\_playListModel->*data*(m\_playListModel->*index*(index, 0)).toString());});

connect(m\_playlist, &QMediaPlaylist::currentIndexChanged, [*this*](int index){

ui->playlistView->selectRow(index);});

m\_audioProbe = *new* QAudioProbe(*this*);

m\_audioHistogram = *new* HistogramWidget(*this*);

QHBoxLayout \*histogramLayout = *new* QHBoxLayout;

histogramLayout->addWidget(m\_audioHistogram);

histogramLayout->setSpacing(0);

ui->horizontalLayout\_6->addLayout(histogramLayout);

ui->horizontalLayout\_6->setSpacing(0);

connect(m\_audioProbe, &QAudioProbe::audioBufferProbed, m\_audioHistogram, &HistogramWidget::processBuffer);

m\_audioProbe->setSource(m\_player);

ui->positionSlider->*setVisible*(*false*);

ui->positionLabel->*setVisible*(*false*);

ui->btn\_stop->*setVisible*(*false*);

ui->btn\_pause->*setVisible*(*false*);

ui->btn\_next->*setVisible*(*false*);

ui->btn\_play->*setVisible*(*false*);

ui->btn\_previous->*setVisible*(*false*);

ui->btn\_random->*setVisible*(*false*);

ui->currentTrack->*setVisible*(*false*);

m\_volumeButton->*setVisible*(*false*);

ui->btn\_add->*setVisible*(*false*);

ui->btn\_del->*setVisible*(*false*);

m\_audioHistogram->*setVisible*(*false*);

ui->currentTrack->setText("");

setWindowTitle("Audio Player");

}

#ifdef WIN32

void Widget::**on\_btn\_add\_clicked**()

{

QStringList files = QFileDialog::getOpenFileNames(*this*,

tr("Open files"),

QString(),

tr("Audio Files(\*.wav \*.mp3)"));

*foreach* (QString filePath, files) {

QList<QStandardItem \*> items;

items.append(*new* QStandardItem(QDir(filePath).dirName()));

items.append(*new* QStandardItem(filePath));

m\_playListModel->appendRow(items);

m\_playlist->addMedia(QUrl(filePath));

};

m\_playListModel->setHorizontalHeaderLabels(QStringList() << tr("AUDIO TRACK")

<< tr("FILE PATH"));

}

#endif

#ifdef \_\_linux\_\_

void Widget::on\_btn\_add\_clicked()

{

QFileDialog fileDialog(*this*);

fileDialog.setAcceptMode(QFileDialog::AcceptOpen);

QStringList supportedMimeTypes;

supportedMimeTypes << "audio/mp3"

<< "audio/wav";

*if* (!supportedMimeTypes.isEmpty()) {

fileDialog.setMimeTypeFilters(supportedMimeTypes);

}

fileDialog.setDirectory(QStandardPaths::standardLocations(QStandardPaths::MusicLocation).value(0, QDir::homePath()));

*if* (fileDialog.exec() == QDialog::Accepted) {

addToPlaylist(fileDialog.selectedUrls());

QList<QStandardItem \*> items;

*for*(*auto* &url: fileDialog.selectedUrls()){

items.append(*new* QStandardItem(QDir(url.toString()).dirName()));

items.append(*new* QStandardItem(url.toString()));

m\_playListModel->appendRow(items);

}

}

m\_playListModel->setHorizontalHeaderLabels(QStringList() << tr("AUDIO TRACK")

<< tr("FILE PATH"));

}

#endif

#ifndef PLAYER\_H

#define PLAYER\_H

#include <QWidget>

#include <QMediaPlayer>

#include <QMediaPlaylist>

#include <QToolButton>

#ifdef WIN32

#include <QtWinExtras>

#endif

#include "style.h"

QT\_FORWARD\_DECLARE\_CLASS(QAbstractItemView)

QT\_FORWARD\_DECLARE\_CLASS(QLabel)

QT\_FORWARD\_DECLARE\_CLASS(QMediaPlayer)

QT\_FORWARD\_DECLARE\_CLASS(QModelIndex)

QT\_FORWARD\_DECLARE\_CLASS(QPushButton)

QT\_FORWARD\_DECLARE\_CLASS(QToolButton)

QT\_FORWARD\_DECLARE\_CLASS(QSlider)

QT\_FORWARD\_DECLARE\_CLASS(QVideoProbe)

QT\_FORWARD\_DECLARE\_CLASS(QVideoWidget)

QT\_FORWARD\_DECLARE\_CLASS(QAudioProbe)

*class* PlaylistModel;

*class* HistogramWidget;

*class* Player : *public* QWidget

{

Q\_OBJECT

*private*:

void clearHistogram();

void setTrackInfo(*const* QString &info);

void setStatusInfo(*const* QString &info);

void handleCursor(QMediaPlayer::MediaStatus status);

void updateDurationInfo(qint64 currentInfo);

void updateDurationInfo\_music(qint64 currentInfo);

QMediaPlaylist \*m\_playlist = *nullptr*;

QMediaPlaylist \*m\_playlist\_music = *nullptr*;

QVideoWidget \*m\_videoWidget = *nullptr*;

QLabel \*m\_coverLabel = *nullptr*;

QSlider \*m\_slider = *nullptr*;

QSlider \*m\_slider\_music = *nullptr*;

QLabel \*m\_labelDuration = *nullptr*;

QLabel \*m\_labelDuration\_music = *nullptr*;

QToolButton \*m\_fullScreenButton = *nullptr*;

QToolButton \*m\_colorButton = *nullptr*;

QDialog \*m\_colorDialog = *nullptr*;

HistogramWidget \*m\_videoHistogram = *nullptr*;

HistogramWidget \*m\_audioHistogram = *nullptr*;

QVideoProbe \*m\_videoProbe = *nullptr*;

QAudioProbe \*m\_audioProbe = *nullptr*;

PlaylistModel \*m\_playlistModel = *nullptr*;

QAbstractItemView \*m\_playlistView = *nullptr*;

PlaylistModel \*m\_playlistModel\_music = *nullptr*;

QAbstractItemView \*m\_playlistView\_music = *nullptr*;

QString m\_trackInfo;

QString m\_statusInfo;

qint64 m\_duration;

qint64 m\_duration\_music;

#ifdef WIN32

void createTaskbar();

void createThumbnailToolBar();

QWinTaskbarProgress \*m\_taskbarProgress = *nullptr*;

QWinTaskbarButton \*m\_taskbarButton = *nullptr*;

QWinThumbnailToolBar \*thumbnailToolBar = *nullptr*;

QWinThumbnailToolButton \*playToolButton = *nullptr*;

QWinThumbnailToolButton \*forwardToolButton = *nullptr*;

QWinThumbnailToolButton \*backwardToolButton = *nullptr*;

#endif

*private* *slots*:

void open();

void open\_music();

void del();

void del\_music();

void durationChanged(qint64 duration);

void positionChanged(qint64 progress);

void durationChanged\_music(qint64 duration);

void positionChanged\_music(qint64 progress);

void metaDataChanged();

void metaDataChanged\_music();

void previousClicked();

void seek(int seconds);

void seek\_music(int seconds);

void jump(*const* QModelIndex &index);

void jump\_music(*const* QModelIndex &index);

void playlistPositionChanged(int);

void playlistPositionChanged\_music(int);

void statusChanged(QMediaPlayer::MediaStatus status);

void stateChanged(QMediaPlayer::State state);

void bufferingProgress(int progress);

void videoAvailableChanged(bool available);

void displayErrorMessage();

void showColorDialog();

#ifdef WIN32

void updateTaskbar();

void updateThumbnailToolBar();

#endif

*public*:

*explicit* Player(QWidget \*parent = *nullptr*);

~*Player*();

QMediaPlayer \*m\_player = *nullptr*;

QMediaPlayer \*m\_player\_music = *nullptr*;

bool isPlayerAvailable() *const*;

void addToPlaylist(*const* QList<QUrl> &urls);

void addToPlaylist\_music(*const* QList<QUrl> &urls);

*public* *slots*:

void togglePlayback();

void seekForward();

void seekBackward();

*signals*:

void fullScreenChanged(bool fullScreen);

};

#endif *//* *PLAYER\_H*

#include <QMediaService>

#include <QMediaPlaylist>

#include <QVideoProbe>

#include <QAudioProbe>

#include <QMediaMetaData>

#include <QtWidgets>

#include "player.h"

#include "playercontrols.h"

#include "playlistmodel.h"

#include "histogramwidget.h"

#include "videowidget.h"

Player::Player(QWidget \*parent)

: QWidget(parent)

{

*this*->setStyleSheet(Style::getWindowStyleSheet());

*this*->setWindowTitle("Video player");

*this*->setFixedSize(700, 250);

setWindowFlags(Qt::*Window*

| Qt::*WindowTitleHint*

| Qt::*WindowSystemMenuHint*

| Qt::*WindowMinimizeButtonHint*

| Qt::*WindowFullscreenButtonHint*

| Qt::*CustomizeWindowHint*);

m\_player = *new* QMediaPlayer(*this*);

m\_player\_music = *new* QMediaPlayer(*this*);

m\_player->setAudioRole(QAudio::*VideoRole*);

m\_player\_music->setAudioRole(QAudio::*MusicRole*);

m\_playlist = *new* QMediaPlaylist();

m\_playlist\_music = *new* QMediaPlaylist();

m\_player->setPlaylist(m\_playlist);

m\_player\_music->setPlaylist(m\_playlist\_music);

m\_videoWidget = *new* VideoWidget(*this*);

m\_player->setVideoOutput(m\_videoWidget);

m\_playlistModel = *new* PlaylistModel(*this*);

m\_playlistModel->setPlaylist(m\_playlist);

m\_playlistModel\_music = *new* PlaylistModel(*this*);

m\_playlistModel\_music->setPlaylist(m\_playlist\_music);

m\_playlistView = *new* QListView(*this*);

m\_playlistView\_music = *new* QListView(*this*);

m\_playlistView->*setModel*(m\_playlistModel);

m\_playlistView\_music->*setModel*(m\_playlistModel\_music);

m\_playlistView->setCurrentIndex(m\_playlistModel->*index*(m\_playlist->currentIndex(), 0));

m\_playlistView\_music->setCurrentIndex(m\_playlistModel\_music->*index*(m\_playlist\_music->currentIndex(), 0));

m\_playlistView->verticalScrollBar()->setStyleSheet(Style::getVerticalScrollBarStyleSheet());

m\_playlistView->horizontalScrollBar()->setStyleSheet(Style::getHorizontalScrollBarStyleSheet());

m\_playlistView\_music->verticalScrollBar()->setStyleSheet(Style::getVerticalScrollBarStyleSheet());

m\_playlistView\_music->horizontalScrollBar()->setStyleSheet(Style::getHorizontalScrollBarStyleSheet());

m\_slider = *new* QSlider(Qt::*Horizontal*, *this*);

m\_slider->setStyleSheet(Style::getSliderStyleSheet());

m\_slider->setRange(0, m\_player->duration() / 1000);

m\_slider\_music = *new* QSlider(Qt::*Horizontal*, *this*);

m\_slider\_music->setStyleSheet(Style::getSliderStyleSheet());

m\_slider\_music->setRange(0, m\_player\_music->duration() / 1000);

m\_labelDuration = *new* QLabel(*this*);

m\_labelDuration\_music = *new* QLabel(*this*);

m\_videoProbe = *new* QVideoProbe(*this*);

m\_audioProbe = *new* QAudioProbe(*this*);

m\_videoHistogram = *new* HistogramWidget(*this*);

m\_audioHistogram = *new* HistogramWidget(*this*);

QHBoxLayout \*histogramLayout = *new* QHBoxLayout;

histogramLayout->addWidget(m\_videoHistogram, 1);

histogramLayout->addWidget(m\_audioHistogram, 2);

QToolButton \*openButton = *new* QToolButton(*this*);

QToolButton \*openButton\_music = *new* QToolButton(*this*);

openButton->setStyleSheet(Style::getAddStyleSheet());

openButton\_music->setStyleSheet(Style::getAddStyleSheet());

QToolButton \*delButton = *new* QToolButton(*this*);

QToolButton \*delButton\_music = *new* QToolButton(*this*);

delButton->setStyleSheet(Style::getRemoveStyleSheet());

delButton\_music->setStyleSheet(Style::getRemoveStyleSheet());

PlayerControls \*controls = *new* PlayerControls(*this*);

PlayerControls \*controls\_music = *new* PlayerControls(*this*);

controls->setState(m\_player->state());

controls->setVolume(m\_player->volume());

controls->setMuted(controls->isMuted());

controls\_music->setState(m\_player\_music->state());

controls\_music->setVolume(m\_player\_music->volume());

controls\_music->setMuted(controls\_music->isMuted());

m\_fullScreenButton = *new* QToolButton(*this*);

m\_fullScreenButton->setStyleSheet(Style::getFullStyleSheet());

m\_fullScreenButton->setCheckable(*true*);

m\_colorButton = *new* QToolButton(*this*);

m\_colorButton->setEnabled(*false*);

m\_colorButton->setStyleSheet(Style::getSettingsStyleSheet());

connect(m\_colorButton, &QPushButton::clicked, *this*, &Player::showColorDialog);

connect(m\_slider, &QSlider::sliderMoved, *this*, &Player::seek);

connect(m\_slider\_music, &QSlider::sliderMoved, *this*, &Player::seek\_music);

connect(openButton, &QPushButton::clicked, *this*, &Player::open);

connect(openButton\_music, &QPushButton::clicked, *this*, &Player::open\_music);

connect(delButton, &QPushButton::clicked, *this*, &Player::del);

connect(delButton\_music, &QPushButton::clicked, *this*, &Player::del\_music);

connect(delButton, &QToolButton::clicked, m\_player, &QMediaPlayer::stop);

connect(delButton\_music, &QToolButton::clicked, m\_player\_music, &QMediaPlayer::stop);

connect(m\_videoProbe, &QVideoProbe::videoFrameProbed, m\_videoHistogram, &HistogramWidget::processFrame);

m\_videoProbe->setSource(m\_player);

connect(m\_audioProbe, &QAudioProbe::audioBufferProbed, m\_audioHistogram, &HistogramWidget::processBuffer);

m\_audioProbe->setSource(m\_player\_music);

connect(m\_playlistView, &QAbstractItemView::activated, *this*, &Player::jump);

connect(m\_playlistView\_music, &QAbstractItemView::activated, *this*, &Player::jump\_music);

connect(controls, &PlayerControls::play, m\_player, &QMediaPlayer::play);

connect(controls, &PlayerControls::play, m\_player\_music, &QMediaPlayer::play);

connect(controls, &PlayerControls::pause, m\_player, &QMediaPlayer::pause);

connect(controls, &PlayerControls::pause, m\_player\_music, &QMediaPlayer::pause);

connect(controls, &PlayerControls::stop, m\_player, &QMediaPlayer::stop);

connect(controls, &PlayerControls::stop, m\_player\_music, &QMediaPlayer::stop);

connect(controls, &PlayerControls::next, m\_playlist, &QMediaPlaylist::next);

connect(controls, &PlayerControls::previous, *this*, &Player::previousClicked);

connect(controls, &PlayerControls::changeVolume, m\_player, &QMediaPlayer::setVolume);

connect(controls, &PlayerControls::changeMuting, m\_player, &QMediaPlayer::setMuted);

connect(controls, &PlayerControls::changeRate, m\_player, &QMediaPlayer::setPlaybackRate);

connect(controls, &PlayerControls::stop, m\_videoWidget, QOverload<>::of(&QVideoWidget::update));

connect(controls\_music, &PlayerControls::play, m\_player\_music, &QMediaPlayer::play);

connect(controls\_music, &PlayerControls::pause, m\_player\_music, &QMediaPlayer::pause);

connect(controls\_music, &PlayerControls::stop, m\_player\_music, &QMediaPlayer::stop);

connect(controls\_music, &PlayerControls::next, m\_playlist\_music, &QMediaPlaylist::next);

connect(controls\_music, &PlayerControls::previous, m\_playlist\_music, &QMediaPlaylist::previous);

connect(controls\_music, &PlayerControls::changeVolume, m\_player\_music, &QMediaPlayer::setVolume);

connect(controls\_music, &PlayerControls::changeMuting, m\_player\_music, &QMediaPlayer::setMuted);

connect(controls\_music, &PlayerControls::changeRate, m\_player\_music, &QMediaPlayer::setPlaybackRate);

connect(m\_player, &QMediaPlayer::stateChanged, controls, &PlayerControls::setState);

connect(m\_player, &QMediaPlayer::volumeChanged, controls, &PlayerControls::setVolume);

connect(m\_player, &QMediaPlayer::mutedChanged, controls, &PlayerControls::setMuted);

connect(m\_player\_music, &QMediaPlayer::stateChanged, controls\_music, &PlayerControls::setState);

connect(m\_player\_music, &QMediaPlayer::volumeChanged, controls\_music, &PlayerControls::setVolume);

connect(m\_player\_music, &QMediaPlayer::mutedChanged, controls\_music, &PlayerControls::setMuted);

connect(m\_player, &QMediaPlayer::durationChanged, *this*, &Player::durationChanged);

connect(m\_player, &QMediaPlayer::positionChanged, *this*, &Player::positionChanged);

connect(m\_player, QOverload<>::of(&QMediaPlayer::metaDataChanged), *this*, &Player::metaDataChanged);

connect(m\_playlist, &QMediaPlaylist::currentIndexChanged, *this*, &Player::playlistPositionChanged);

connect(m\_player, &QMediaPlayer::mediaStatusChanged, *this*, &Player::statusChanged);

connect(m\_player, &QMediaPlayer::bufferStatusChanged, *this*, &Player::bufferingProgress);

connect(m\_player, &QMediaPlayer::videoAvailableChanged, *this*, &Player::videoAvailableChanged);

connect(m\_player, QOverload<QMediaPlayer::Error>::of(&QMediaPlayer::error), *this*, &Player::displayErrorMessage);

connect(m\_player, &QMediaPlayer::stateChanged, *this*, &Player::stateChanged);

connect(m\_player\_music, &QMediaPlayer::durationChanged, *this*, &Player::durationChanged\_music);

connect(m\_player\_music, &QMediaPlayer::positionChanged, *this*, &Player::positionChanged\_music);

connect(m\_player\_music, QOverload<>::of(&QMediaPlayer::metaDataChanged), *this*, &Player::metaDataChanged\_music);

connect(m\_playlist\_music, &QMediaPlaylist::currentIndexChanged, *this*, &Player::playlistPositionChanged\_music);

connect(m\_player\_music, &QMediaPlayer::mediaStatusChanged, *this*, &Player::statusChanged);

connect(m\_player\_music, &QMediaPlayer::bufferStatusChanged, *this*, &Player::bufferingProgress);

connect(m\_player\_music, QOverload<QMediaPlayer::Error>::of(&QMediaPlayer::error), *this*, &Player::displayErrorMessage);

connect(m\_player\_music, &QMediaPlayer::stateChanged, *this*, &Player::stateChanged);

QBoxLayout \*displayLayout = *new* QHBoxLayout;

displayLayout->addWidget(m\_videoWidget, 2);

displayLayout->addWidget(m\_playlistView);

displayLayout->addWidget(m\_playlistView\_music);

QBoxLayout \*controlLayout = *new* QHBoxLayout;

controlLayout->setMargin(0);

controlLayout->addWidget(openButton);

controlLayout->addWidget(delButton);

controlLayout->addWidget(controls);

controlLayout->addWidget(m\_fullScreenButton);

controlLayout->addWidget(m\_colorButton);

controlLayout->addStretch(1);

QBoxLayout \*controlLayout\_music = *new* QHBoxLayout;

controlLayout\_music->setMargin(0);

controlLayout\_music->addWidget(openButton\_music);

controlLayout\_music->addWidget(delButton\_music);

controlLayout\_music->addWidget(controls\_music);

controlLayout\_music->addStretch(1);

QBoxLayout \*layout = *new* QVBoxLayout;

layout->addLayout(displayLayout);

QHBoxLayout \*hLayout = *new* QHBoxLayout;

hLayout->addWidget(m\_slider);

hLayout->addWidget(m\_labelDuration);

QHBoxLayout \*hLayout\_music = *new* QHBoxLayout;

hLayout\_music->addWidget(m\_slider\_music);

hLayout\_music->addWidget(m\_labelDuration\_music);

layout->addLayout(hLayout);

layout->addLayout(hLayout\_music);

layout->addLayout(controlLayout);

layout->addLayout(controlLayout\_music);

layout->addLayout(histogramLayout);

m\_slider->*setVisible*(*false*);

m\_labelDuration->*setVisible*(*false*);

m\_slider\_music->*setVisible*(*false*);

m\_labelDuration\_music->*setVisible*(*false*);

setLayout(layout);

#ifdef WIN32

createTaskbar();

createThumbnailToolBar();

#endif

metaDataChanged();

}

**ПРИЛОЖЕНИЕ Е**

(*обязательное*)

Ведомость документов