Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

**Кафедра «Компьютерные интеллектуальные технологии»**

**КУРСОВАЯ РАБОТА**

**Каталогизация медиа-файлов**

по дисциплине «Технологии объектно-ориентированного программирования»

Выполнил

студент гр. гр.23536/2 Е.Ю. Шолохова

Руководитель

Ст. преподаватель К. А. Туральчук

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 201\_\_ г.

Санкт-Петербург

2019

СОДЕРЖАНИЕ

ВВЕДЕНИЕ. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1. ОРГАНИЗАЦИЯ ХРАНЕНИЯ ДАННЫХ. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.1. Постановка задачи. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

1.2. Анализ предметной области. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

1.3. Описание программы . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

1.4. Выбор средств и методов реализации. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

2. РАЗРАБОТКА ПРОГРАММЫ ДЛЯ КАТАЛОГИЗАЦИИ

2.1. Декомпозиция задачи . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

2.2. Разработка графического интерфейса пользователя.. . . . . . . . . . . . . . . . . . . . . . . 10

2.3. Описание бизнес-логики. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

2.3.1. Выборка и отображение медиа-файлов . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

2.3.2. Сортировка файлов по директориям. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

2.3.3. Генерация коллекции. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

ЗАКЛЮЧЕНИЕ. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

СПИСОК ЛИТЕРАТУРЫ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

ПРИЛОЖЕНИЕ 1. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

ПРИЛОЖЕНИЕ 2. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

ПРИЛОЖЕНИЕ 3. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

ВВЕДЕНИЕ

Данная работапосвящена изучению этапов разработки программного обеспечения. Основная цель - показать умение применять объектно-ориентированный подход и использовать различный инструментарий в процессе разработки на примере программы-каталогизатора медиа-файлов. Исходя из поставленной цели, можно сформулировать следующие задачи:

* проанализировать предметную область, выбрать средства разработки;
* декомпозировать задачу;
* реализовать основной функционал программы и протестировать его.

Отчет содержит в себе как теоретические сведения, отобранные и систематизированные в процессе работы, так и подробное описание реализации программы, начиная с декомпозиции и заканчивая замечаниями по итогам работы.

ОРГАНИЗАЦИЯ ХРАНЕНИЯ ДАННЫХ. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

## Постановка задачи

Задача, которая ставится перед нами в данной работе, - создание приложения для каталогизации медиа-файлов. Под медиа-файлами в данном случае будем понимать аудио-, видео- файлы – именно с этими типами файлов должно работать наше приложение. Поиск файлов должен осуществляться в выбранной пользователем директории и ее поддиректориях. В нем также должна быть предусмотрена возможность копировании файлов в указанную директорию в каталогизированном в виде, т.е. рассортированными по тем каталогам, которые для них определил пользователь.

## Анализ предметной области

## Программа должна производить поиск и отбор файлов по некоторому критерию. Обратимся к теории хранения данных, и определим для себя понятия [файл и файловая система](http://knowledge.allbest.ru/programming/3c0a65625a3bd79b4d43a88521206c27_0.html).

## Вся информация в компьютере хранится в долговременной памяти в виде файлов. Таким образом, файл - это именованная область памяти на каком-либо носителе информации. Имя файла состоит из двух частей, разделенных точкой: собственно имя файла и расширение, определяющее его тип (программа, данные и т.д.).

Порядок хранения файлов на диске определяется установленной файловой системой. Файловая система - это система хранения файлов и организации каталогов. Каталог (папка, директория) - это, с одной стороны, набор файлов, сгруппированных пользователем по некоторому принципу, с другой же, он сам по себе является файлом и содержит системную информацию о группе файлов, его составляющих. Каталоги могут образовывать иерархическую структуру за счет того, что каталог более низкого уровня может входить в каталог более высокого уровня. Если на диске хранятся сотни и тысячи файлов, то для удобства поиска файлы организуются в многоуровневую иерархическую файловую систему, которая имеет «древовидную» структуру («дерево каталогов»).

Для того чтобы найти файл в иерархической файловой структуре необходимо указать путь к файлу. Путь к файлу - это последовательность из имен вложенных друг в друга каталогов, в последнем из которых находится нужный файл, разделённых символом «\». Корневой каталог - начальный каталог в структуре каталогов. Путь к файлу от корневого каталога называют абсолютным путём, а от текущего — относительным путём. Путь к каталогу файла и имя файла, разделённые «\», перед которыми указано имя диска, представляет собой полное имя файла.

## Описание работы программы

Программа представляет из себя оконное приложение с удобным и функциональным пользовательским интерфейсом.

Главное окно программы поделено на 2 части. Справа - таблица, предназначенная для отображения информации о найденных файлах. Слева – список уж созданных пользователем каталогов. Над этими элементами расположено поле для ввода с надписью Path to the directory. В начале работы пользователь должен ввести в это поле путь до папки, в которой будет производиться поиск файлов его аудио-/видео- коллекции. Рядом расположены кнопки Browse и Search. Первая позволяет задать путь через файловый обозреватель, а вторая - провести поиск в указанной папке. Поиск можно ограничить, выбрав файлы только с определенными расширениями. Для этого достаточно напечатать эти расширения через разделитель в строке Extensions. Слева от нее кнопка Filters, открывающая доступ к функциям фильтрации найденных файлов. В нижней части главного окна расположены 2 кнопки – AddToDir и Generate Catalog.

Для добавления файлов в каталог, их нужно сначала пометить галочками, а потом нажать AddToDir. Во всплывающем окне можно будет указать название каталога. После того как пользователь закончил сортировку своих файлов, он может либо сгенерировать свой библиотеку медиа-файлов, либо просто сохранить информацию о сортировке, чтобы продолжить позднее. Кнопка Generate Catalog отвечает за создание уже каталогизированной библиотеки медиа-файлов. При нажатии на нее программа сначала запрашивает у пользователя путь для генерации каталога и имя каталога. Далее происходит проверка наличия свободного места на указанном диске и копирование файлов.

## Выбор средств и методов реализации (ООП+СPP+MVC+QT)

### Применение объектно-ориентированного подхода

В разработке программы мы прежде всего будем полагаться на объектно-ориентированный подход. Именно его используют большинство современных программ. Суть данного подхода заключается в представлении программного кода, как системы взаимодействующих друг с другом объектов. Каждый объект является экземпляром некоторого класса, где класс - самостоятельная сущность, выделяемая программистом при декомпозиции задачи. Классы могут как состоять в иерархии, так и быть независимы друг от друга или дружественными друг другу. Иерархи подразумевает, что есть более общие по смыслу классы (они стоят вверху иерархии), а есть более конкретизированные (они наследуют свойства от более общих и добавляют свои свойства, как бы уточняя характеристики описываемой сущности). Связь между экземплярами классов осуществляется с помощью методов (действий, доступных каждому классу). На деле, метода мало чем отличаются от функций, а свойства от переменных. Но такая организация программного кода позволяет сосредоточиться на общей логике работы программы, не отвлекаясь на внутреннее устройство каждого отдельно взятого объекта.

### С++

Теперь для того, чтобы перейти к написанию кода, нам необходимо определиться с языком программирования. Остановим свой выбор на языке С++. На заре появления ООП язык С++ считался языком программирования высокого уровня, но сейчас его причисляют скорее к языкам среднего уровня. Благодаря своей гибкости, высокой производительности, поддержке объектно-ориентированного программирования, наличию библиотеки STL, данный язык находит самое широкое применение. Но главные сферы его использования это, безусловно, системное программирование, высоконагруженные сервера, встраиваемые системы, а также кроссплатформенные приложения (при использовании фреймворков; например, Qt, wxWidgets, GTK) и разработка игр (скрипты для Unreal Engine). C++, как язык, не стоит на месте, его активно развивают. За последние 10 лет вышло сразу несколько новых стандартов языка - C++11, C++14 и C++17. Со всеми нововведениями С++ теперь не уступает самым продвинутым языкам программирования.

### MVC

Помимо объектно-ориентированного подхода, в современной теории разработки

существуют и другие стандартные методы, которые помогают проектировать ваше приложение – например, различные шаблоны или паттерны.

Один из них – MVC - мы и попытаемся внедрить в нашу программу. MVC — это архитектурный шаблон, который описывает способ построения структуры приложения (расшифровывается как Model-View-Controller или Модель-Представление-Контроллер). Основная суть этого способа заключается в отделении программной логики от пользовательского интерфейса и от способа представления информации пользователю. Все эти 3 компонента в архитектуре MVC независимы, их код не пересекается и чаще всего можно четко выделить, какая часть программы (какие классы) за что отвечает.

MVC один из базовых архитектурных шаблонов и наиболее часто применяется именно в случае оконных приложений. Построение программы на основе MVC-архитектуры непосредственно связано с декомпозицией задачи. Поэтому более подробно мы рассмотрим данный вопрос в соответствующем параграфе.

### Фреймворк QT

Для создания приложения с графическим интерфейсом удобно использовать фреймворк или библиотеку, заточенную под такие цели. В качестве такого средства мы будем использовать фреймворк Qt. Qt - это кроссплатформенный инструмент для создания прикладных программ. Данный фреймворк предоставляет множество возможностей и часто используется именно в связке с С++. Его единственный недостаток – не самая лучшая оптимизация потребляемых программой ресурсов- с лихвой перекрывается множеством достоинств. Библиотека фреймворка содержит огромной количество готовых и классов, а также абстрактных классов, от которых можно наследовать свои, если потребности вашей задачи выходят за рамки стандартных инструментов библиотеки. Qt, что не маловажно, имеет прекрасную документацию с большим количеством примеров. Разработчики поддерживают и активно развивают его. А, благодаря обширному коммьюнити, в сети интернет можно найти решение практически любо проблемы, возникшей в процессе разработки.

У Qt есть также Qt Designer – case-средство для проектирования графического интерфейса. Но для лучшего понимания устройства фреймворка, в частности, того как в нем реализован объектно-ориентированный подход, мы не будем использовать Qt Designer.

РАЗРАБОТКА ПРОГРАММЫ ДЛЯ КАТАЛОГИЗАЦИИ

## Декомпозиция задачи

## В процессе декомпозиции были выделены подзадачи, которые необходимо выполнить для достижения результата, и определена их последовательность (см.рисунок).

Рисунок 2.1 - Диаграмма IDEF0

Каждая подзадача отражает определенную последовательность действий

Подзадачи были выделены на основе логической последовательности действий, которую пользователь должен выполнить, чтобы организовать свою коллекцию файлов.

Была проведена также объектная декомпозиции проектируемой программы, в ходе которой были определены классы, объекты которых нам потребуются при написании программной логики.

Исходные данные - коллекция медиа-файлов, которую необходимо систематизировать. Каждый медиа-файл представляет собой некую запись в этой коллекции. На основе этих утверждений были выделены следующие сущности:

* медиа-запись;
* каталог (коллекция).

«Медиа-запись», по сути, является «оберткой» для медиа-файла, позволяющей производить с ним все действия, требуемые по условию задачи.

«Каталог» - класс-контейнер, содержащий имеющиеся у пользователя «медиа-записи»; способен систематизировать «медиа-записи» и сохранять структуру организации коллекции (дерево каталогов, созданных пользователем).

Остальные спроектированные классы относятся к графическому интерфейсу и будут рассмотрены в следующем пункте.

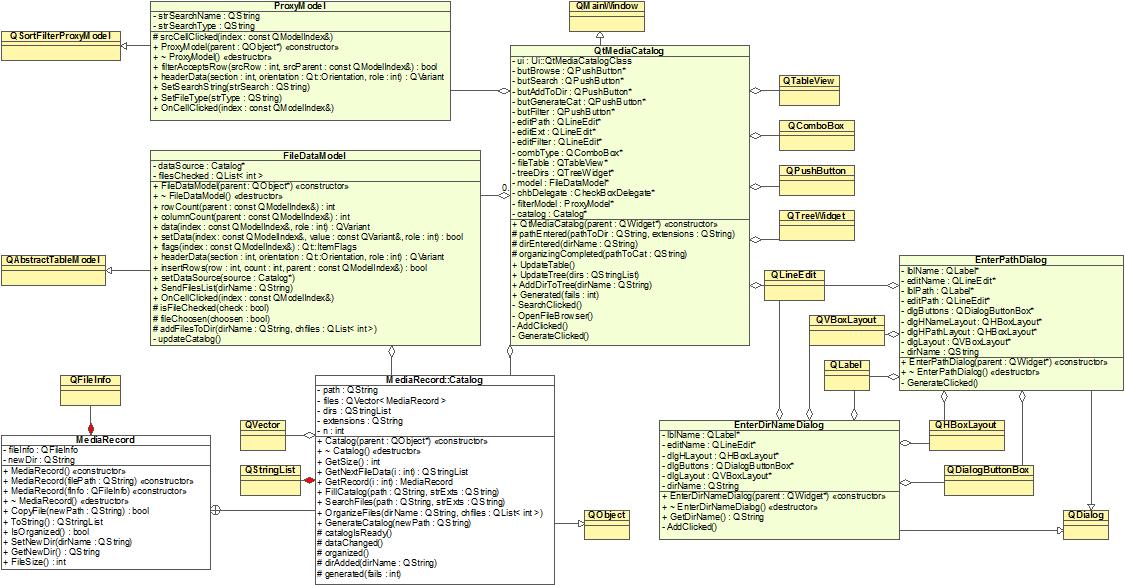
**

Рисунок 2.2 - Диаграмма классов UML

## Разработка графического интерфейса пользователя

Все элементы графического интерфейса были запрограммированы с помощью специальных классов Qt (т.н. виджетов). Все эти классы являются наследниками класса QWidget. Для их правильного расположения на форме главного окна использовались классы группировки виджетов (т.н. слои или layouts) - QGridLayout, QHBoxLayout и QVBoxLayout.

В интерфейс программы были добавлены следующие элементы:

Кнопки (класс QPushButton):

butBrowse – кнопка Browse – открывает файловый браузер, чтобы пользователь мог указатьв какой папке расположены его медиа-файлы.

butSearch – кнопка Search

butAddToDir – кнопка Add To Directory

butGenerateCat – кнопка Generate Media Catalog

butFilter – кнопка Filter

Поля ввода (класс QLineEdit):

editPath

editExt

Табличное представление (класс QTableView):

fileTable – таблица, в которое отображается информация о файлах, которые по не отсортированы пользователем в каталоги

Древовидный список (класс QTreeWidget):

treeDirs – список всех созданных пользователем директорий

Следующие классы были унаследованы от стандартных классов Qt. Для этих классов необходимо было написать собственную реализацию, чтобы учесть потребности нашего приложения.

Модели – объекты в Qt, необходимые для передачи данных представлению. Стандартная модель, предоставляемая Qt – QStandardItemModel – не обладает достаточным функционалом, поэтому мы реализуем собственный класс модели, унаследованный от QAbstractTableModel.

Также нам понадобится возможность отображать только часть данных. Для этого в Qt предназначен класс QSortFilterProxyModel, позволяющий фильтровать данные в таблице.

Основная модель:

класс FileDataModel : model

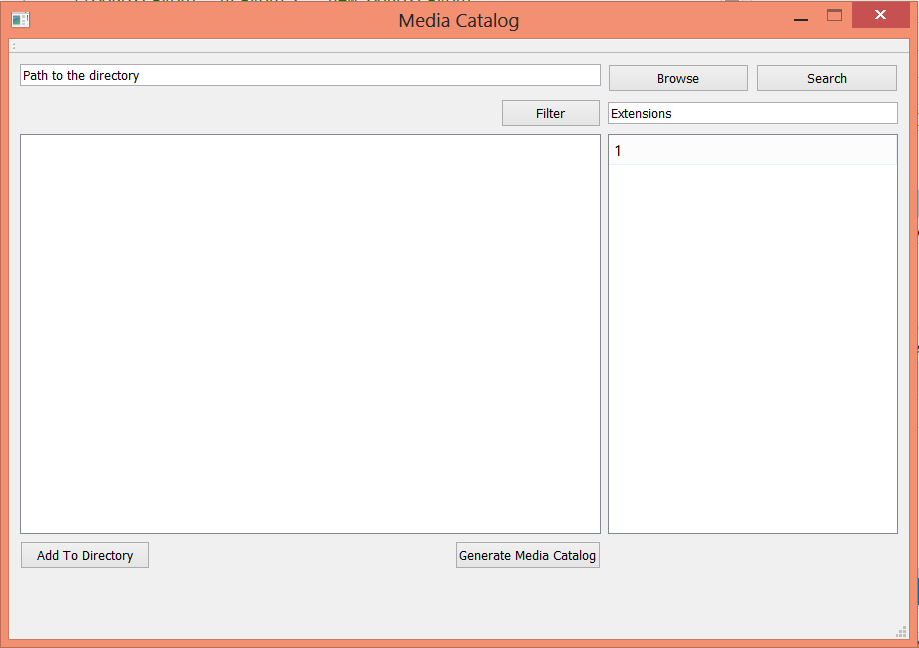
Модель для фильтрации данных:

класс ProxyModel : filerModel

Для того, чтобы принимать информацию от пользователей нам также понадобятся диалоговые окна. Создадим собственные классы, на базе имеющегося в Qt QDialog:

класс QEnterDirNameDialog – для ввода имени каталога, создаваемого пользователем. Он также будет выводить предупреждение, если пользователь ввел подкаталог, родительский каталог которого еще не создан, и контролировать правильности ввода имени.

класс QEnterPathDialog– для выбора пути генерации итоговой коллекции медиафайлов.

** Рисунок 2.3 - Интерфейс главного окна программы

Классы, предназначенные для хранения данных:   
класс MediaRecord – описывает запись в коллекции - т.е. аудио- или видео- файл, найденный в процессе поиска

– содержит информацию о самом файле (путь, размер, расширение и т.п.) и в какой каталог он помещен

класс Catalog catalog

– хранит данные о найденных файлах, список каталогов, организованных пользователем, является основным источником данных для модели FileDataModel.

Catalog содержит:

– массив QVector типа < MediaRecord >. Он заполняется, когда мы получаем путь до директории в которой лежат файлы.

– список QStringList. В нем хранится список всех добавленных пользователем каталогов.

## Описание бизнес-логики

## Выборка и отображение медиа-файлов

Первый этап создания программы – поиск медиа-файлов и их отображение в таблице.

1. Способ доступа к файловой системе, директориям и файлам
2. Выбор алгоритма поиска
3. Способ представление информации о файлах

Разберем подробно каждый пункт.

1. Доступ к файловой системе, директориям и файлам

Здесь удобно воспользоваться классами QFileDialog, QDir и QFileInfo.

QFileDialog позволит нам получить доступ к файловой системе. QDir – класс, с помощью которого можно узнать содержимое директории - содержащиеся в ней файлы и поддиректории.

QFileInfo – класс, дающий доступ к информации о файле.

Сигнал clicked() кнопки Browse соединяем со слотом главного окна BrowseClicked(). В нем вызываем статический метод класса QFileDialog для открытия файлового браузера. Получаем на выходе путь к файлам. Так мы гарантируем то, что папка на самом деле существует. Дополнительная проверка теперь не требуется. Путь отображается в поле editPath. Сигнал clicked() кнопки Search соединяем со слотом SearchClicked(). Берем значение из полей editPath и editExt и посылаем их через сигнал pathEntered() в catalog, в слот FillCatalog(). FillCatalog() содержит алгоритм обхода директорий. Получая сигнал pathEntered() вместе с путем к файлам, он открывает соответствующую директорию и начинает обход. Ищем файлы только с теми расширениями, которые мы получили из поля editExt. Когда каталог заполнен, посылается сигнал catalogIsReady().

1. Выбор алгоритма поиска

Для обхода директории и всех поддиректорий есть 2 способа:

- Рекурсивный обход директорий и

- Итерационный обход (с запоминанием точки входа)

Мы будем использовать первый вариант, т.к. итерационный подход в данном случае не дает никаких преимуществ, ни по времени, ни по потребляемой памяти, а реализовать его сложнее.

В методе FillCatalog(), получающем на вход dirPath, мы вызываем рекурсивный метод SearchFiles(). В нем создаем объект класса QDir. Передаем в него dirPath. Теперь в этом объекте содержится вся информация о директории. В том числе, EntryInfoList, из которого можно получить список всех файлов и поддиректорий в этой директории. Проходим в цикле по этому списку, добавляя файлы с соответствующими расширениями в массив files. А для тех элементов списка, которые сами являются директориями, снова вызываем метод SearchFiles(). Когда обход закончен, посылаем сигнал catalogIsReady() в модель FileDataModel.

1. Предоставление информации о файлах пользователю

Qt имеет несколько вариантов реализации таблицы – это классы QTableWidget и QTableView.

Решено было использовать QTableView с собственной моделью FileDataModel, наследованной от QAbstarctTableModel. При наличии собственной модели можно полностью отделить представление от данных. В дальнейшем, когда понадобиться добавить представлению новые функции, достаточно будет описать их в FileDataModel. QTableView + модель так же являются примером реализации MVC-архитектуры в Qt.

Далее необходимо решить каким методом модель будет извлекать данные из Catalog. Мы будем использовать указатель на экземпляр класса Catalog. При завершении поиска Catalog посылает сигнал catalogIsReady() и указатель на самого себя в слот модели dataSource(). Так как хранящиеся в классе Catalog данные инкапсулированы, мы не можем получить их напрямую. Поэтому добавляем ему метод GetNextFileData(), который будет получать индекс, а в ответ выдавать информацию о файле в виде QStringList. В модели также необходимо реимплементировать метод data(), который и будет вызывать GetNextFileData() у Catalog, а потом передавать информацию в соотвествующие поля таблицы.

## Сортировка файлов по директориям

Следующий этап разобъем на такие под этапы:

1. Фильтрация
2. Выбор файлов для сортировки
3. Создание каталогов
4. Обновление данных в табличном представлении
5. Фильтрация

Для отбора и сортировки данных в таблице используем еще один встроенный класс фреймворка – QSortFilterProxyModel. Класс QSortFilterProxyModel содержит в себе множество способов фильтрации элементов в таблице: setFilterWildcard, setFilterFixedString, setFilterRegularExpression, и др. Но наша главная цель использовать его для отображения только тех файлов, которые еще не организованы в каталоги. Поэтому необходимо написать на его основе свой класс (ProxyModel). С помощью метода setSourceModel() устанавливаем нашу модель FileDataModel в качестве источника, данные которого будут фильтроваться. И переключаем свойство setSortingEnabled() у представления QTableView. В перегруженном методе filterAcceptsRow() перечисляем условия, при которых должна отображаться строка в представлении.

1. Выбор файлов для сортировки

Для того, чтобы пользователь мог пометить файлы, которые он хочет положить в каталог, напротив каждого файла в таблице добавим элемент CheckBox. При установке флажка в чэкбокс соответствующий ему файл будет добавлен в список файлов на каталогизацию. Этот список мы будем хранить в модели, и, при нажатии кнопки AddToDir, передавать в Catalog.

Для того чтобы табличное представление реагировало на действия пользователя с чэкбоксом, необходимо проделать ряд действий.

В модели FileDataModel в методе flags() обозначим ячейки первого столбца как ItemIsUserCheckable. После этого в них появятся чекбоксы, но т.к. это не QCheckBox, а всего лишь иконки, они не реагируют на действия пользователя. Их состояние (checked/unchecked) можно менять из модели с помощью метода data(). Для того чтобы они изменялись по клику, воспользуемся сигналом clicked() идущим от QTableView. Он содержит в себе индекс ячейки таблицы, который мы будем передавать в модель.

Создадим в модели FileDataModel слот OnCellClicked(), принимающий на вход индекс ячейки. И такой же слот в ProxyModel (это необходимо в связи с несовпадением индексов между моделями - ProxyModel фильтрует строки, и индексы ячеек в таблице меняются). Для установки флажка в чекбокс дополнительно перегрузим в модели FileDataModel метод setData(). Он будет вызываться из метода OnCellClicked().

Теперь соединяем сигнал clicked() идущий от QTableView со слотом в ProxyModel OnCellClicked(). В этом слоте индекс ячейки, отфильтрованной ProxyModel, будет конвертироваться в индекс соответствующей ячейки FileDataModel. Далее новый индекс посылаем сигналом srcCellClicked() в модель FileDataModel в слот OnCellClicked(). Таким образом значение ячейки будет меняться, а вместе с ним будет меняться и состояние чекбокса.

1. Создание каталогов

По кнопке AddToDir нам нужно вызывать диалоговое окно, в котором вводится имя каталога.

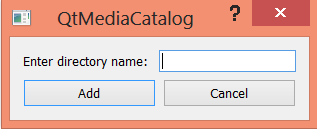


Рисунок 2.4. Диалоговое окно для ввода названия каталога

Для этой цели в Qt есть специальный класс – QDialog. В нашем случае используем наследованный от него EnterDirNameDialog. В нем организуем проверку вводимых значений и вывод предупреждающего сообщения. Для получения имени каталога добавляем метод GetDirName().

Соединяем сигнал clicked() кнопки AddToDir со слотом главного окна AddClicked(). В нем создаем переменную dirName и экземпляр класса EnterDirNameDialog. Запускаем диалог методом exec(), и проверяем какое значение он возвращает. Если “Accepted” , то это значит, что была нажата кнопка Ok, и мы можем принять значение, полученное от пользователя. Присваиваем переменной dirName результат вызова GetDirName(). Передаем имя каталога сигналом dirEntered() в слот модели SendFilesList(). Этот слот отправляет полученную строку и список файлов отмеченных галочками в слот OrganizeFiles() класса Catalog, а потом очищает список файлов.

В OrganizeFiles() записываем имя каталога в список сформированных каталогов dirs, в массиве files типа fileTodir всем помеченным файлам из списка приписываем каталог и посылаем сигнал organized().

1. Обновление данных в табличном представлении

Теперь из таблицы нужно убрать обработанные файлы. В модели filterModel есть слот invalidate() для обновления данных в таблице. Соединяем его с сигналом organized(), идущим от Catalog. В filterAcceptsRow() прописываем условие – если файлу назначен каталог, мы его не отображаем. Теперь как только файлы будут организованы в каталоги, они исчезнут из общего списка.

## Генерация коллекции

1. Копирование

Копирование файлов в новые директории будет осуществляться с помощью метода Copy() класса MediaRecord, который, по сути, является всего лишь удобной оберткой для одноименного метода в классе QFile. На вход ему поступает путь до корневой директории, а он сам уже приписывает относительный путь, который хранится в экземпляре MediaRecord.

Для того чтобы начать копирование файлов пользователь должен кликнуть по кнопке Generate Catalog и ввести путь, по которому будет сгенерирована коллекция.

Сигнал от кнопки ‘’ обрабатывается в главном окне программы и пересылается вместе с указанным путем в класс Catalog в слот GenerateCatalog(). В этом слоте для каждой записи массива files<MediaRecord> вызывается метод Copy(). Перед этим в классе Catalog проверяется наличие свободного места на диске, а также наличие у пользователя доступа к записи данных на него.

1. Сохранение информации о сортировке

В файл для бэкапа будет записываться основная информация из объекта catalog: директория, в которой производился поиск, массив files(), список каталогов dirs. Этих данных достаточно для восстановления сеанса. Но на случай изменений в файловой системе (переносе/переименовании файлов/папок) также, безусловно, потребуется возможность проверки загруженных из бэкапа данных.

ЗАКЛЮЧЕНИЕ

В ходе работы были изучены возможности языка программирования С++ с точки зрения объектно-ориентированного подхода. Также были рассмотрены различные методологии проектирования (IDEF0,UML) и построены соответствующие диаграммы.

Результатом данной работы стало создание приложения для каталогизации медиа-файлов. В процессе мы научились декомпозировать задачу и применять различные современные подходы и инструменты для разработки прикладного программного обеспечения.

На первом этапе были собраны и проанализированы сведения, которые могли понадобиться в процессе разработки. Далее в результате декомпозиции были выделены основные логические сущности и спроектированы описывающие их классы. Процесс разработки был разбит на отдельные последовательные этапы.

Удалось реализовать полностью работоспособное приложение, со всеми заявленными возможностями. Написанный код соответствует требованиям объектно-ориентированного подхода и является расширяемым и поддерживаемым. Все ошибки, выявленные в ходе тестирования и отладки приложения, устранены.

Архитектурный шаблон MVC удалось реализовать лишь частично – в рамках фреймворка Qt. Для полноценного встраивание концепции MVC в приложение потребовалось бы полностью отделить классы, представляющие данные, от классов Qt – сделать их независимыми от фреймворка. В масштабах данного проекта это не так существенно, но в случае его расширения желательно доработать этот момент.

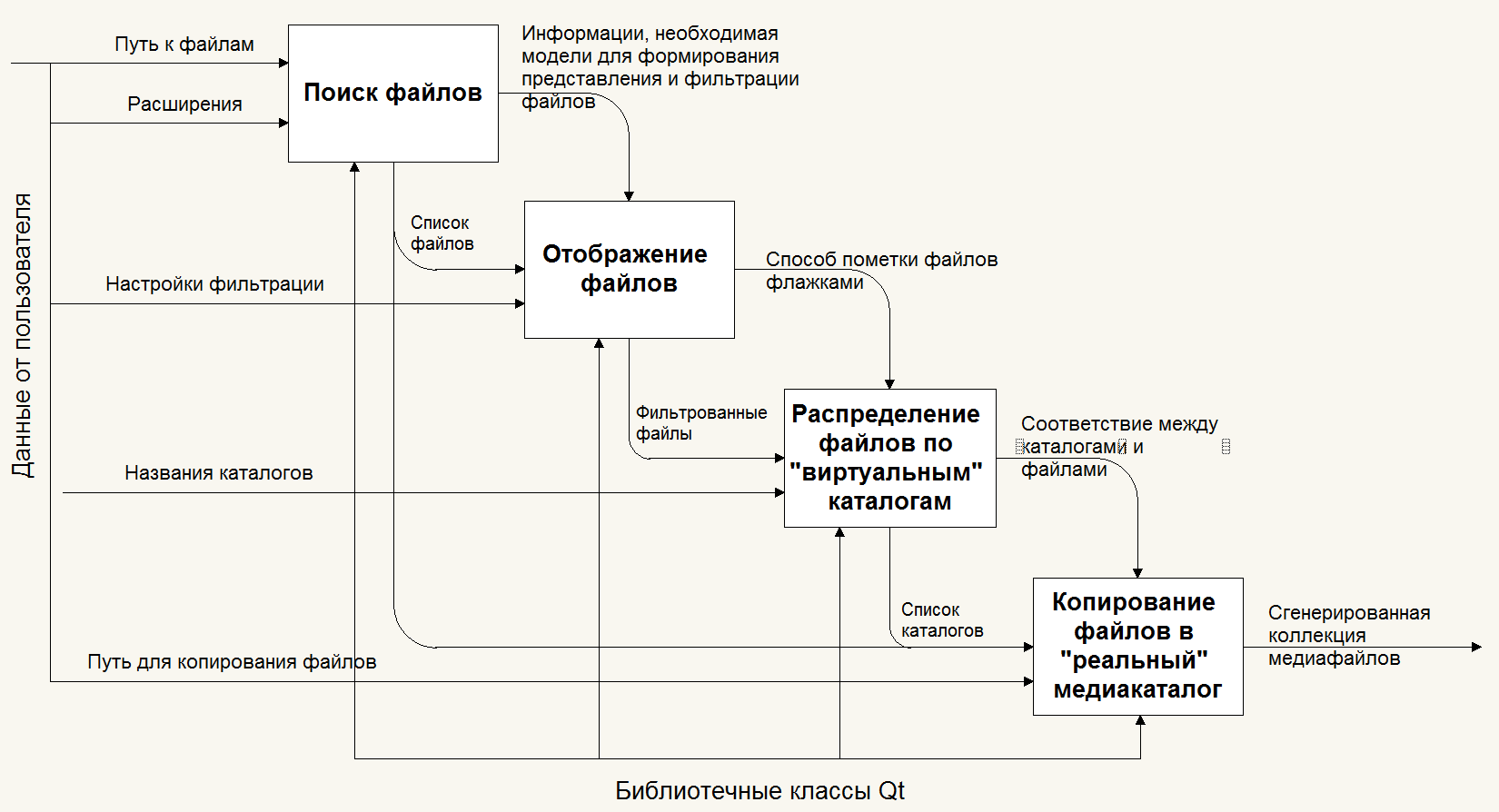
Таким образом, все поставленные задачи решены.

СПИСОК ЛИТЕРАТУРЫ

1. Буч Г. - Объектно-ориентированный анализ и проектирование с примерами приложений. — 3-е издание. — «Вильямс», 2010.- 718 c.
2. Марченко А.Л. C++ Бархатный путь. - 2-е изд. Издательство: М.: Горячая линия — телеком, 2005 г. - 399 c.
3. Павловская Т.А. C/C++. Процедурное и объектно-ориентированное программирование. Учебник для вузов. Стандарт 3-го поколения. СПБ: Питер, 2015 г. - 495 с.
4. Шлее М.- Qt 5.3. Профессиональное программирование на C++.. BHV, 2015 г. - 928 c.
5. Документация QtFramework - <http://doc.qt.io/>
6. Лекции по технологии ООП с дистанционного курса на сайте http://dl.spbstu.ru/
7. Раздел по программированию на Qt форума http://stackoverflow.com/

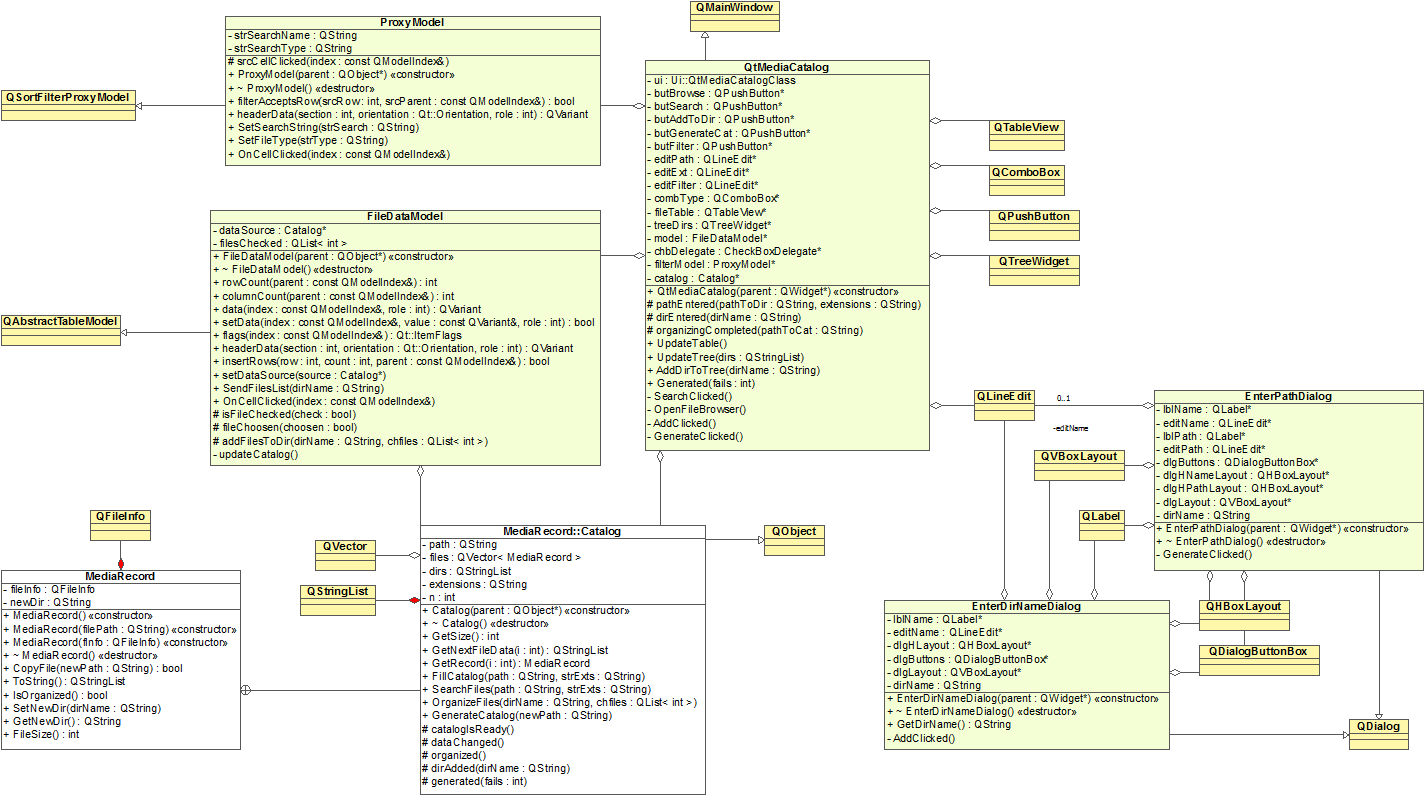
ПРИЛОЖЕНИЕ 1

**Диаграмма IDEF0**



ПРИЛОЖЕНИЕ 2

**Диаграмма классов UML**



# ПРИЛОЖЕНИЕ 3

**Листинг программы**

#include <QObject>

#include <QFileInfo>

#include <QFile>

#include <QDir>

#pragma once

class MediaRecord

{

private:

QFileInfo fileInfo;

QString newDir;

public:

MediaRecord();

MediaRecord(QString filePath);

MediaRecord(QFileInfo fInfo);

bool CopyFile(QString newPath);

QStringList ToString(); //returns array with file info

bool IsOrganized();

void SetNewDir(QString dirName);

QString GetNewDir();

int FileSize() { return fileInfo.size(); };

#include "MediaRecord.h"

#include <qDebug>

#pragma once

MediaRecord::MediaRecord() {}

MediaRecord::MediaRecord(QString filePath)

{

fileInfo.setFile((filePath));

newDir = "";

}

MediaRecord::MediaRecord(QFileInfo fInfo)

{

fileInfo = fInfo;

newDir = "";

}

bool MediaRecord::CopyFile(QString newPath)

{

if (!QFile::copy(fileInfo.absoluteFilePath(), newPath + "/" + newDir+"/" + fileInfo.fileName())) {

qDebug() << "Copying file "<<fileInfo.absoluteFilePath()<<" failed!"

;

qDebug() << newPath + "/" + newDir + "/" + fileInfo.fileName();

return false;

}

else

{

qDebug() << fileInfo.absoluteFilePath() << " is copied to:";

qDebug() << newPath + "/" + newDir + "/" + fileInfo.fileName();

return true;

}

}

QStringList MediaRecord::ToString()

{

return (QStringList() << fileInfo.fileName()

<< fileInfo.suffix())

<<

(((float)fileInfo.size() / 1024 / 1024) > 1023 ?

QString::number((float)fileInfo.size() / 1024 / 1024 / 1024, 'f', 2) + " Gb"

: QString::number((float)fileInfo.size() / 1024 / 1024, 'f', 2) + " Mb");

}

bool MediaRecord::IsOrganized()

{

if (newDir == "") {

return false;

}

return true;

}

void MediaRecord::SetNewDir(QString dirName)

{

newDir = dirName;

qDebug() << "File " << fileInfo.fileName() << " was put into " << newDir;

}

QString MediaRecord::GetNewDir()

{

return newDir;

}

#include <QObject>

#include <QVector>

#include <QList>

#include "MediaRecord.h"

#pragma once

class Catalog : public QObject

{

Q\_OBJECT

private:

QString path;

QVector <MediaRecord> files;

QStringList dirs;

QString extensions;

int n;

public:

Catalog(QObject \*parent = Q\_NULLPTR);

int GetSize() { return n; };

QStringList GetNextFileData(int i);

MediaRecord GetRecord(int i) { return files.at(i); };

public slots:

void FillCatalog(QString path, QString strExts = "");

void SearchFiles(QString path, QString strExts);

void OrganizeFiles(QString dirName, QList<int> chfiles);

void GenerateCatalog(QString newPath);

signals:

void catalogIsReady();

void dataChanged();

void organized();

void dirAdded(QString dirName);

void generated(int fails);

};

#include "Catalog.h"

#include <QDir>

#include <QStorageInfo>

#include <qDebug>

#include <QRegularExpression>

#pragma once

Catalog::Catalog(QObject \*parent)

: QObject(parent)

{

n = 0;

extensions = "\*.mp3 \*.avi \*.mkv \*.mp4 ";

}

void Catalog::SearchFiles(QString path, QString strExts)

{

QDir dir(path); qDebug() << "FILLING CATALOG" << dir.absolutePath();

QStringList exts(strExts.split(" "));

for (QFileInfo i : dir.entryInfoList(exts,QDir::Files|QDir::AllDirs)) {

if (i.absolutePath() == dir.absolutePath())

{

if (i.isDir())

{

FillCatalog(i.absolutePath() + "/" + i.baseName());

}

else

{

files.push\_back(MediaRecord(i));

qDebug() << "ADDING " << i.fileName();

n += 1;

}

};

};

}

void Catalog::FillCatalog(QString path, QString strExts)

{

if(extensions != strExts)

extensions += strExts;

SearchFiles(path, extensions);

qDebug() << n << " FILES ADDED";

emit catalogIsReady();

}

QStringList Catalog::GetNextFileData(int i)

{

if ((i<GetSize())&&(i>=0))

{

MediaRecord temp(files.at(i));

return temp.ToString()<<temp.GetNewDir()<<QString::number(i);

}

return QStringList();

}

void Catalog::OrganizeFiles(QString dirName, QList<int> chfiles)

{

qDebug() << "Directory " << dirName;

QRegularExpression re("\\+");

QString dir2N = dirName.replace(re, "/");

qDebug() << "Directory " << dir2N;

dirName.replace("\\", "/");

dirs.append(dirName);

for each (int i in chfiles)

{

files[i].SetNewDir(dirName);

}

emit dirAdded(dirName);

emit organized();

}

void Catalog::GenerateCatalog(QString newPath)

{

QStorageInfo storage(newPath);

if (storage.isReadOnly())

qDebug() << "isReadOnly:" << storage.isReadOnly();

qDebug() << "name:" << storage.name();

qDebug() << "filesystem type:" << storage.fileSystemType();

qDebug() << "size:" << storage.bytesTotal() / 1000 / 1000 << "Mb";

qDebug() << "free space:" << storage.bytesAvailable() / 1000 / 1000 << "Mb";

qint64 totalSize = 0;

for each (MediaRecord var in files)

{

if (var.IsOrganized()) { totalSize += var.FileSize(); }

}

qDebug() << totalSize<< "bytes";

qDebug() << "free space:" << storage.bytesAvailable();

qDebug() << "Enter GenerateCatalog()";

if((totalSize<storage.bytesAvailable()) && !storage.isReadOnly())

{

//creating directories

QDir root(newPath);

for each (QString str in dirs)

{

root.mkpath(newPath + "/" + str);

}

//copying files

int fails = 0;

for each (MediaRecord var in files)

{

if (var.IsOrganized() && !var.CopyFile(newPath)) { fails++; }

}

qDebug() << "MEDIACATALOG IS READY"<<"\nFAILED TO COPY: "<<fails;

emit generated(fails);

}

}

#pragma once

#include <QAbstractTableModel>

#include "Catalog.h"

class FileDataModel : public QAbstractTableModel

{

Q\_OBJECT

public:

FileDataModel(QObject \*parent = Q\_NULLPTR);

~FileDataModel();

int rowCount(const QModelIndex &parent = QModelIndex()) const override { return dataSource->GetSize(); };

int columnCount(const QModelIndex &parent = QModelIndex()) const override { return 6;};

QVariant data(const QModelIndex &index, int role) const override;

bool setData(const QModelIndex &index, const QVariant &value, int role = Qt::EditRole) override;

Qt::ItemFlags flags(const QModelIndex & index) const override;

QVariant headerData(int section, Qt::Orientation orientation, int role) const override;

void setDataSource(Catalog \*source);

private slots:

public slots :

void SendFilesList(QString dirName);

void OnCellClicked(const QModelIndex &index);

signals:

void isFileChecked(bool check);

void fileChoosen(bool choosen);

void addFilesToDir(QString dirName, QList<int> chfiles);

private:

Catalog \*dataSource;

QList<int> filesChecked;//

void updateCatalog();

};

#include "FileDataModel.h"

#include <qDebug>

#include <QCheckBox>

#pragma once

FileDataModel::FileDataModel(QObject \*parent)

: QAbstractTableModel(parent) {}

FileDataModel::~FileDataModel() {}

QVariant FileDataModel::data(const QModelIndex & index, int role) const

{

if (!index.isValid())

return QVariant();

else if (index.row() >= dataSource->GetSize() || index.row() < 0)

return QVariant();

else

{

const QStringList curr = dataSource->GetNextFileData(index.row());

if ((role == Qt::CheckStateRole) && (index.column() == 0)) {

if (filesChecked.contains(curr.at(4).toInt()))

{

return Qt::Checked;

}

else

return Qt::Unchecked;

}

if (role == Qt::DisplayRole) {

if ((index.column() - 1 <= curr.length()) && (index.column() != 0)) {

return curr.at(index.column() - 1);

}

}

}

return QVariant();

}

bool FileDataModel::setData(const QModelIndex & index, const QVariant & value, int role)

{

if (role == Qt::EditRole)

{

//check value from editor

if (value.toBool()) {

// add file if it was checked

filesChecked.append(index.row());

emit fileChoosen(true);

qDebug() << "LIST LENGTH+: " << filesChecked.length();

}

else {

//remove file if it was unchecked

int i = filesChecked.indexOf(index.row());

filesChecked.removeAt(i);

if (filesChecked.isEmpty()) emit fileChoosen(false);

qDebug() << "LIST LENGTH-: " << filesChecked.length();

}

}

return true;

}

void FileDataModel::OnCellClicked(const QModelIndex &index)

{

QModelIndex ind = index;

if (index.isValid() && (index.column() == 0)) {

bool isChecked = data(index, Qt::CheckStateRole).toBool();

setData(index, !isChecked, Qt::EditRole);

}

}

Qt::ItemFlags FileDataModel::flags(const QModelIndex & index) const

{

if (index.column() == 0)

return Qt::ItemIsUserCheckable | QAbstractTableModel::flags(index) | Qt::ItemIsEnabled;

else

return QAbstractTableModel::flags(index);

}

QVariant FileDataModel::headerData(int section, Qt::Orientation orientation, int role) const

{

if (orientation == Qt::Horizontal)

{

if (role == Qt::DisplayRole) {

switch (section)

{

case 0:

return " ";

case 1:

return "File name";

case 2:

return "Type";

case 3:

return "Size";

case 4:

return "Catalog";

case 5:

return "Record index";

default:

break;

}

}

else if (role == Qt::SizeHintRole)

{

switch (section)

{

case 0:

return QSize(10,10);

case 2:

return QSize(20, 10);

case 3:

return QSize(30, 10);

default:

break;

}

}

}

else if (orientation == Qt::Vertical) {

return QVariant();

}

return QAbstractTableModel::headerData(section, orientation, role);

}

void FileDataModel::setDataSource(Catalog \* source)

{

dataSource = source;

}

void FileDataModel::SendFilesList(QString dirName)

{

emit addFilesToDir(dirName, filesChecked);

filesChecked.clear();

qDebug() << "LIST IS CLEARED!!!";

}

void FileDataModel::updateCatalog()

{

beginResetModel(); resetInternalData(); endResetModel();

emit layoutChanged();

}

#pragma once

#include <QSortFilterProxyModel>

class ProxyModel : public QSortFilterProxyModel

{

Q\_OBJECT

private:

QString strSearchName;

QString strSearchType;

signals:

void srcCellClicked(const QModelIndex &index);

public:

ProxyModel(QObject \*parent);

~ProxyModel();

bool filterAcceptsRow(int srcRow, const QModelIndex &srcParent) const override;

QVariant headerData(int section, Qt::Orientation orientation, int role) const override;

public slots:

void SetSearchString(QString strSearch);

void SetFileType(QString strType);

void OnCellClicked(const QModelIndex &index);

};

#include "ProxyModel.h"

#include <QRegularExpression>

#include <qDebug>

#pragma once

ProxyModel::ProxyModel(QObject \*parent)

: QSortFilterProxyModel(parent)

{

strSearchName = "";

strSearchType = "";

}

ProxyModel::~ProxyModel()

{

}

bool ProxyModel::filterAcceptsRow(int srcRow, const QModelIndex & srcParent) const

{

QModelIndex index = sourceModel()->index(srcRow, 4, srcParent);

QModelIndex indexName = sourceModel()->index(srcRow, 1, srcParent);

QModelIndex indexType = sourceModel()->index(srcRow, 2, srcParent);

if (!sourceModel()->data(index).toString().isEmpty())

{

return false;

}

QRegExp re("\*" + strSearchName + "\*",Qt::CaseInsensitive, QRegExp::Wildcard);

qDebug() << strSearchName;

QString str = sourceModel()->data(indexName).toString();

if (!re.exactMatch(str))

{

return false;

}

return true;

}

QVariant ProxyModel::headerData(int section, Qt::Orientation orientation, int role) const

{

return sourceModel()->headerData(section, orientation, role);

}

void ProxyModel::SetSearchString(QString str)

{

if (str != strSearchName)

strSearchName = str;

invalidateFilter();

}

void ProxyModel::SetFileType(QString type)

{

if (type != strSearchType)

strSearchType = type;

invalidateFilter();

}

void ProxyModel::OnCellClicked(const QModelIndex & index)

{

QModelIndex srcIndex = this->mapToSource(index);

emit srcCellClicked(srcIndex);

}

#pragma once

#include <QDialog>

#include <QDialogButtonBox>

#include <QLabel>

#include <QLineEdit>

#include <QRegExp>

#include <QRegExpValidator>

#include <QPushButton>

#include <QHBoxLayout>

#include <QMessageBox>

class EnterDirNameDialog : public QDialog

{

Q\_OBJECT

private:

QLabel \*lblName;

QLineEdit \*editName;

QHBoxLayout \*dlgHLayout;

QDialogButtonBox \*dlgButtons;

QVBoxLayout \*dlgLayout;

QString dirName;

public:

EnterDirNameDialog(QWidget \*parent);

~EnterDirNameDialog();

QString GetDirName() { return dirName;};

private slots:

void AddClicked();

};

#include "EnterDirNameDialog.h"

#include <qDebug>

EnterDirNameDialog::EnterDirNameDialog(QWidget \*parent)

: QDialog(parent)

{

//Widgets

lblName = new QLabel("Enter directory name: ", this);

editName = new QLineEdit(this);

editName->setValidator(new QRegExpValidator(QRegExp("[^<>:\"|!?\*]+(\|/)")));

dlgButtons = new QDialogButtonBox(QDialogButtonBox::Ok | QDialogButtonBox::Cancel, Qt::Horizontal, this);

dlgButtons->button(QDialogButtonBox::Ok)->setText("Add");

// Layouts

dlgHLayout = new QHBoxLayout;

dlgHLayout->addWidget(lblName);

dlgHLayout->addWidget(editName);

dlgLayout = new QVBoxLayout;

dlgLayout->addLayout(dlgHLayout);

dlgLayout->addWidget(dlgButtons);

setLayout(dlgLayout);

//Connecting signals and slots

connect(dlgButtons, SIGNAL(accepted()), this, SLOT(AddClicked()));

connect(dlgButtons, &QDialogButtonBox::rejected, this, &QDialog::reject);

dirName = "";

}

EnterDirNameDialog::~EnterDirNameDialog()

{

}

void EnterDirNameDialog::AddClicked()

{

QRegularExpression rx(".(\\|/)+.");

dirName = editName->text();

if ((editName->text().contains(("\\")))||(editName->text().contains(("/"))))

{

QMessageBox \*msgBox = new QMessageBox(QMessageBox::NoIcon, "Add sub directory",

"If parent directory doesn't exist it will be created automatically. Accept?",

QMessageBox::Ok | QMessageBox::Cancel, this);

qDebug() << editName->text();

if (msgBox->exec()== QMessageBox::Ok) { accept(); }

}

else

{ accept(); }

}

#pragma once

#include <QtWidgets/QMainWindow>

#include "ui\_QtMediaCatalog.h"

#include "FileDataModel.h"

#include "ProxyModel.h"

#include <QStandardItemModel>

#include "qlayout.h"

#include <QLabel>

#include <QLineEdit>

#include <QPushButton>

#include <QComboBox>

#include "qtreewidget.h"

#include "qtableview.h"

#include "qlist.h"

#include "Catalog.h"

class QtMediaCatalog : public QMainWindow

{

Q\_OBJECT

public:

QtMediaCatalog(QWidget \*parent = Q\_NULLPTR);

private:

Ui::QtMediaCatalogClass ui;

QPushButton \*butBrowse;

QPushButton \*butSearch;

QPushButton \*butAddToDir;

QPushButton \*butGenerateCat;

QPushButton \*butFilter;

QLineEdit \*editPath;

QLineEdit \*editExt;

QLineEdit \*editFilter;

QComboBox \*combType;

QTableView \*fileTable;

QTreeWidget \*treeDirs;

FileDataModel \*model;

ProxyModel \*filterModel;

Catalog\* catalog;

signals:

void pathEntered(QString pathToDir, QString extensions = "");

void dirEntered(QString dirName);

void organizingCompleted(QString pathToCat);

public slots:

void UpdateTable();

void UpdateTree(QStringList dirs);

void AddDirToTree(QString dirName);

void Generated(int fails);

private slots :

void SearchClicked();

void OpenFileBrowser();

void AddClicked();

void GenerateClicked();

};

#include "QtMediaCatalog.h"

#include "EnterPathDialog.h"

#include "EnterDirNameDialog.h"

#include <QHeaderView>

#include <QInputDialog>

#include <QRegExp>

#include <QFileDialog>

#include <QDir>

#include <qDebug>

#pragma once

QtMediaCatalog::QtMediaCatalog(QWidget \*parent)

: QMainWindow(parent)

{

ui.setupUi(this);

QWidget \*mainWdgt = new QWidget(this);

catalog = new Catalog();

//Widgets

butBrowse = new QPushButton(tr("Browse"), mainWdgt);//?tr

butSearch = new QPushButton("Search", mainWdgt);

butAddToDir = new QPushButton("Add To Directory", mainWdgt);

butAddToDir->setEnabled(false);

butGenerateCat = new QPushButton("Generate Media Catalog", mainWdgt);

butGenerateCat->setEnabled(false);

butFilter = new QPushButton("Filter", mainWdgt);

editFilter = new QLineEdit(mainWdgt);

editFilter->setPlaceholderText("Enter pattern for search");

combType = new QComboBox(mainWdgt);

combType->insertItems(0, QStringList() << "Audio" << "Video");

editPath = new QLineEdit(mainWdgt);

editPath->setPlaceholderText("Path to the directory (click \"Browse\"\)");

editPath->setReadOnly(true);

editExt = new QLineEdit(mainWdgt);

editExt->setPlaceholderText("Extensions");

editExt->setValidator(new QRegExpValidator(QRegExp("([A-Z][a-z]{2,4} )+", Qt::CaseInsensitive)));

editExt->setText("\*.mp3 \*.avi \*.mkv \*.mp4 ");

treeDirs = new QTreeWidget(mainWdgt);

treeDirs->setHeaderHidden(true);

fileTable = new QTableView(mainWdgt);

model = Q\_NULLPTR;

filterModel = new ProxyModel(this);

fileTable->setSortingEnabled(true);

fileTable->setFixedHeight(400);

fileTable->verticalHeader()->hide();

fileTable->horizontalHeader()->setSectionResizeMode(QHeaderView::Stretch);

fileTable->setSelectionBehavior(QAbstractItemView::SelectRows);

//Layout and widgets positioning

QGridLayout \*mainLayout = new QGridLayout;

mainLayout->addWidget(editPath, 0, 0, 1, 4, Qt::AlignVCenter);

mainLayout->addWidget(butBrowse,0,4);

mainLayout->addWidget(butSearch,0,5);

mainLayout->addWidget(editFilter, 2, 0, 1, 2, Qt::AlignVCenter);

mainLayout->addWidget(combType, 2, 2, Qt::AlignJustify);

mainLayout->addWidget(butFilter, 2, 3, Qt::AlignRight);

combType->setGeometry(butFilter->geometry());

butFilter->setFixedWidth(100);

mainLayout->addWidget(editExt, 2, 4, 1, 2);

mainLayout->addWidget(fileTable, 3, 0, 16, 4);

mainLayout->addWidget(treeDirs, 3, 4, 16, 2);

mainLayout->addWidget(butAddToDir,20,0);

butAddToDir->setFixedWidth(130);

mainLayout->addWidget(butGenerateCat,20,3);

mainLayout->setAlignment(Qt::AlignTop);

mainWdgt->setLayout(mainLayout);

setCentralWidget(mainWdgt);

setWindowTitle("Media Catalog");

setFixedSize(900, 600);

//Connecting signals and slots

connect(this, SIGNAL(pathEntered(QString, QString)), catalog, SLOT(FillCatalog(QString, QString)));

connect(catalog, SIGNAL(catalogIsReady()), treeDirs, SLOT(clear()));

connect(catalog, SIGNAL(catalogIsReady()), this, SLOT(UpdateTable()));

connect(butBrowse, SIGNAL(clicked()), this, SLOT(OpenFileBrowser()));

connect(butSearch, SIGNAL(clicked()), this, SLOT(SearchClicked()));

connect(butAddToDir, SIGNAL(clicked()), this, SLOT(AddClicked()));

connect(butGenerateCat, SIGNAL(clicked()), this, SLOT(GenerateClicked()));

connect(catalog, SIGNAL(dirAdded(QString)), this, SLOT(AddDirToTree(QString)));

connect(this, SIGNAL(organizingCompleted(QString)), catalog, SLOT(GenerateCatalog(QString)));

connect(catalog, SIGNAL(generated(int)), this, SLOT(Generated(int)));

}

void QtMediaCatalog::SearchClicked() {

qDebug() << "PATH ENTERED"<< editPath->text();

emit pathEntered("E:/Video", editExt->text());

}

void QtMediaCatalog::UpdateTable()

{

if (model != Q\_NULLPTR) { delete model; }

qDebug() << "INITIALIZING MODEL";

model = new FileDataModel(this);

model->setDataSource(catalog);

filterModel->setSourceModel(model);

fileTable->setModel(filterModel);

fileTable->hideColumn(4);

fileTable->hideColumn(5);

fileTable->horizontalHeader()->setSectionResizeMode(0, QHeaderView::Fixed);

fileTable->horizontalHeader()->setSectionResizeMode(2, QHeaderView::Fixed);

fileTable->horizontalHeader()->setSectionResizeMode(3, QHeaderView::Fixed);

fileTable->horizontalHeader()->resizeSection(0,10);

fileTable->horizontalHeader()->resizeSection(2, 50);

fileTable->horizontalHeader()->resizeSection(3, 100);

fileTable->horizontalHeader()->setSectionResizeMode(1, QHeaderView::Interactive);

fileTable->setVerticalScrollMode(QAbstractItemView::ScrollPerPixel);

connect(model, SIGNAL(fileChoosen(bool)), butAddToDir, SLOT(setEnabled(bool)));

connect(this, SIGNAL(dirEntered(QString)), model, SLOT(SendFilesList(QString)));

connect(model, SIGNAL(addFilesToDir(QString, QList<int>)), catalog, SLOT(OrganizeFiles(QString, QList<int>)));

//calling invalidate() to refresh filter settings

//we need this when adding files to directory

//so they should be removed from the view

connect(catalog, SIGNAL(organized()), filterModel, SLOT(invalidate()));

connect(editFilter, SIGNAL(textChanged(QString)), filterModel, SLOT(SetSearchString(QString)));

connect(combType, SIGNAL(currentTextChanged(QString)), filterModel, SLOT(SetFileType(QString)));

connect(fileTable, SIGNAL(clicked(const QModelIndex &)), filterModel, SLOT(OnCellClicked(const QModelIndex &)));

connect(filterModel, SIGNAL(srcCellClicked(const QModelIndex &)), model, SLOT(OnCellClicked(const QModelIndex &)));

}

void QtMediaCatalog::UpdateTree(QStringList dirs)

{

// for full update

qDebug() << "UPDATING TREE";

QList<QTreeWidgetItem\*> dirsTmp;

for each (QString str in dirs)

{

QTreeWidgetItem \*item = new QTreeWidgetItem();

item->setText(0, str);

dirsTmp.append(item);

}

treeDirs->addTopLevelItems(dirsTmp);

}

void QtMediaCatalog::AddDirToTree(QString dirName)

{

qDebug() << "ADDING DIR TO TREE";

dirName.replace("\\", "/");

QStringList dirsTmp = dirName.split("/");

// if it's a top level item - add to tree

QTreeWidgetItem\* parentItem = Q\_NULLPTR;

if (treeDirs->findItems(dirsTmp[0], Qt::MatchFixedString | Qt::MatchCaseSensitive).isEmpty())

{

QTreeWidgetItem\* temp;

temp = new QTreeWidgetItem();

temp->setText(0, dirsTmp[0]);

treeDirs->addTopLevelItem(temp);

treeDirs->expandItem(temp);

qDebug() << "ROOT DIR ADDED" << dirsTmp[0];

}

//if not, we should find it's root node

QList<QTreeWidgetItem\*> rootItems = treeDirs->findItems(dirsTmp[0], Qt::MatchFixedString | Qt::MatchCaseSensitive);

for(int j = 0; j<rootItems.size(); j++){

if(rootItems[j]->text(0)== dirsTmp[0]) {

parentItem = rootItems[j];

break;

}

}

for(int i=1; i<dirsTmp.size();i++)

{

bool parentFound = false;

for (int j = 0; j<parentItem->childCount(); j++) {

if (parentItem->child(j)->text(0) == dirsTmp[i])

{

parentItem = parentItem->child(j);

parentFound = true;

break;

}

}

if(!parentFound){

QTreeWidgetItem\* temp;

temp = new QTreeWidgetItem();

temp->setText(0, dirsTmp[i]);

parentItem->addChild(temp);

treeDirs->expandItem(temp);

parentItem = temp;

qDebug() << "DIR ADDED";

}

}

//where to place this: (maybe separate slot so it depends on number of dirs)

//not working if new search is done

if(treeDirs->topLevelItemCount()!=0)

butGenerateCat->setEnabled(true);

else

butGenerateCat->setEnabled(false);

}

void QtMediaCatalog::Generated(int fails)

{

QMessageBox \*msgBox = new QMessageBox(QMessageBox::NoIcon, "Finished",

QString("Mediacatalog is generated. \n Fail to copy %1 files.").arg(QString::number(fails)),

QMessageBox::Ok, this);

msgBox->exec();

}

void QtMediaCatalog::OpenFileBrowser() {

qDebug() << "OPEN FILE DIALOG";

editPath->setText(QFileDialog::getExistingDirectory(this, tr("Choose Directory"),

"C:/",

QFileDialog::ShowDirsOnly

| QFileDialog::DontResolveSymlinks));//native file dialog

}

void QtMediaCatalog::AddClicked()

{

EnterDirNameDialog\* inputDlg = new EnterDirNameDialog(this);

if(inputDlg->exec()==QDialog::Accepted)

{

QString dirName = inputDlg->GetDirName();

if (!dirName.isEmpty()) emit dirEntered(dirName);

}

}

void QtMediaCatalog::GenerateClicked()

{

//заменить на кастомный диалог с проверкой вводимой директории

// и вводом имени каталога

QString catalogPath = QFileDialog::getExistingDirectory(this, tr("Choose directory or create new"), "C:/",

QFileDialog::ShowDirsOnly

| QFileDialog::DontResolveSymlinks);

if (!catalogPath.isEmpty()) {

emit organizingCompleted(catalogPath);

qDebug() << "SIGNAL EMITTED";

}

}