МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное   
учреждение высшего образования

**"Южно-Уральский государственный университет**

**(национальный исследовательский университет)"**

**Высшая школа электроники и компьютерных наук**

**Кафедра системного программирования**

**ОТЧЕТ   
по учебной практике**

бакалавра направления 09.03.04 "Программная инженерия"

|  |  |
| --- | --- |
|  | Выполнил:\_\_\_\_\_\_\_\_\_\_\_\_ студент группы КЭ-203  Ращупкин Е.В.  Проверил:\_\_\_\_\_\_\_\_\_\_\_\_\_  Преподаватель кафедры СП  Глизница М.Н.  Дата:\_\_\_\_\_\_\_\_, Оценка:\_\_\_\_\_\_\_\_\_\_ |

Челябинск-2022

Министерство науки и высшего образования Российской Федерации

Южно-Уральский государственный университет

Кафедра системного программирования

УТВЕРЖДАЮ

Зав. кафедрой

системного программирования

\_\_\_\_\_\_\_\_\_\_\_\_\_ Л.Б. Соколинский

**ЗАДАНИЕ**

**по** **учебной практике**

1. **Цель работы**

Разработать GUI-приложение, работающее с входной информацией, вводимой пользователем с помощью управляемых элементов формы, либо из текстового файла.

1. **Исходные данные к работе**
2. База данных меломана. База групп и исполнителей: база песен, дисков с перечнем песен (в виде ссылок), поиск песен заданной группы, всех дисков, где встречается заданная песня.
3. Данные хранятся в файле в формате JSON.
4. Класс «Author» с полями «AuthorId», «AuthorName», «ListIdSong». Класс «Disk», с полями «DiskId», «DiskName», «ListIdSong». Класс «Song», с полями «SongId», «SongName», «ListIdAuthor», «ListIdDisk».
5. **Перечень подлежащих разработке вопросов**
6. Определение структуры приложения (по модулям), структур данных, используемых для хранения основной пользовательской информации.
7. Дизайн оконного интерфейса, анализ структуры входных данных и их защита от некорректного ввода информации.
8. Разработка основного функционала приложения: основных форм и механизмов получения информации из их компонентов и их файлов; основного алгоритма функционирования приложения; тестирование приложения.
9. Подготовка руководства пользователя и документации для программиста.
10. **Сроки**

Дата выдачи задания: "27" июня 2022 г.

Срок сдачи законченной работы: "23" июля 2022 г.

**Руководитель:**

Преподаватель. кафедры СП \_\_\_\_\_\_\_\_\_\_\_\_\_ Глизница М.Н.

должность, ученая степень подпись ФИО руководителя

**Задание принял к исполнению:**

\_\_\_\_\_\_\_\_\_\_\_\_\_ Ращупкин Е.В.

подпись ФИО студента

**ОГЛАВЛЕНИЕ**

[1. ПОСТАНОВКА ЗАДАЧИ 4](#_Toc1)

[2. ДИЗАЙН ОКОННОГО ИНТЕРФЕЙСА 11](#_Toc2)

[3. РАЗРАБОТКА ФУНКЦИОНАЛА ОСНОВНЫХ ФОРМ И МЕХАНИЗМОВ ПОЛУЧЕНИЯ ИНФОРМАЦИИ 22](#_Toc3)

[3.1. Главная форма 22](#_Toc4)

[4. РАЗРАБОТКА ОСНОВНОГО МЕХАНИЗМА ФУНКЦИОНИРОВАНИЯ ПРИЛОЖЕНИЯ 26](#_Toc5)

[5. ТЕСТИРОВАНИЕ 32](#_Toc6)

[5.1. Автономное тестирование 32](#_Toc7)

[5.2. Комплексное тестирование 35](#_Toc8)

[6. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 36](#_Toc9)

[ЗАКЛЮЧЕНИЕ 37](#_Toc10)

[ЛИТЕРАТУРА 38](#_Toc11)

[ПРИЛОЖЕНИЯ 39](#_Toc12)

# ПОСТАНОВКА ЗАДАЧИ

Требуется написать программу для работы с базой данных меломана. Формат записи базы данных: группы и исполнители, песни, диски с перечнем песен. Программа должна обеспечивать поиск по базе по заданным критериям, позволять редактировать и дополнять базу.

Записи базы данных будут иметь следующие типы (табл.1).

Табл. 1. Переменные, используемые для хранения записей базы данных

|  |  |  |
| --- | --- | --- |
| Переменная | Тип переменной | Содержательный смысл |
| AuthorId | int | Id Автора |
| AuthorName | std::string | Название группы |
| DiskId | int | Id Диска |
| DiskName | std::string | Название диска |
| ListIdSong | std::vector<int> | Вектор id песен диска |
| SongId | int | Id Песни |
| SongName | std::string | Название песни |
| ListIdAuthor | std::vector<int> | Вектор id авторов песни |
| ListIdDisk | std::vector<int> | Вектор id дисков песни |

Можно реализовать решение данной задачи с помощью трех двусвязных списков.

Каждый элемент первого двусвязного списка будет включать информацию о авторе песен, в частности “AuthorName”, “AuthorId”, указатели на следующий элемент и предыдущий. (признаком конца списка служит нулевой указатель), а также вектор песен автора включающий в себя их “Id”.

Каждый элемент второго двусвязного списка будет включать информацию о диске с песнями, в частности “DiskName”, “DiskId”, указатели на следующий элемент и предыдущий. (признаком конца списка служит нулевой указатель), а также вектор песен диска включающий в себя их “Id”.

Каждый элемент третьего двусвязного списка будет включать информацию о песне, в частности “SongName”, “SongId”, указатели на следующий элемент и предыдущий. (признаком конца списка служит нулевой указатель), а также два вектора, первый вектор авторов включает в себя их “Id”, второй вектор дисков включает в себя их “Id”.

Листинг 1. Пример класса автора

class Author

{

private:

int AuthorId; // Id Автора

std::string AuthorName; //Название группы

std::vector<int> ListIdSong; //Список песен

public:

Author()

{

AuthorId = 0;

AuthorName = "";

ListIdSong = {};

}

void ChangeAuthor(int \_AuthorId, std::string \_AuthorName, std::vector<int> \_ListIdSong);

void SetId(int \_AuthorId); //Изменить Id автора

int GetId(); //Получить Id автора

void SetName(std::string \_AuthorName); //Изменить название группы

std::string GetName(); //Получить название группы

void Show(); //Показать список песен автора

void AddSong(int songid); //Добавить песню автору

void RemoveSong(int songid); //Удалить песню у Автора

std::vector<int> GetSongs(); //Получить песни у Автора

};

Листинг 2. Пример класса диска

class Disk

{

private:

int DiskId; // Id диска

std::string DiskName; //Название диска

std::vector<int> ListIdSong; //Список песен

public:

Disk()

{

DiskId = 0;

DiskName = "";

ListIdSong = {};

}

void ChangeDisk(int \_DiskId, std::string \_DiskName, std::vector<int> \_ListIdSong);

void SetId(int \_DiskId); //Изменить Id диска

int GetId(); //Получить Id диска

void SetName(std::string \_DiskName); //Изменить название диска

std::string GetName(); //Получить название диска

void Show(); //Показать список песен диска

void AddSong(int songid); //Добавить песню диску

void RemoveSong(int songid); //Удалить песню у диска

std::vector<int> GetSongs(); //Получить песни у диска

};

Листинг 3. Пример класса песни

class Song

{

private:

int SongId; // Id песни

std::string SongName; // Название песни

std::vector<int> ListIdAuthor; // Список авторов песни

std::vector<int> ListIdDisk; // Список дисков песен

public:

Song() // Конструктор

{

SongId = 0;

SongName = "";

ListIdAuthor = {};

ListIdDisk = {};

}

// Установить все поля песни

void ChangeSong(int \_SongId, std::string \_SongName, std::vector<int> \_ListIdAuthor, std::vector<int> \_ListIdDisk);

void SetId(int \_SongId); // Изменить Id песни

int GetId(); // Получить Id песни

void SetName(std::string \_SongName); // Изменить название песни

std::string GetName(); // Получить название песни

void AddAuthor(int authorid); // Добавить автора песни

void RemoveAuthor(int authorid); // Удалить автора песни

void SetAuthors(std::vector<int> \_ListIdAuthor); //Установить авторов песни

std::vector<int> GetAuthors(); // Получить авторов песни

void AddDisk(int diskid); // Добавить диск песне

void RemoveDisk(int diskid); // Удалить диск песни

void SetDisks(std::vector<int> \_ListIdDisk); //Установить диски песни

std::vector<int> GetDisks(); // Получить диски песни

};

Разрабатываемое приложение состоит из трех оконных форм:

1. Окно справки (контактами разработчика и основным назначением программы).
2. Основное окно программы, на нём располагаются 4 вкладки, Author, Disk, Song, Display, они отвечают за ввод данных пользователем, кнопки вызова диалога выбора файлов для чтения/записи информации, кнопки поиска и вызова отчётов.
3. Окно отчёта (Представление данных в виде отчета.).

Каждой из разработанных оконных форм соответствует пара файлов (\*.h и \*.cpp), приведенные в таблице 2. В этой таблице, к тому же, указаны другие модули.

Табл. 2. Модули создаваемого проекта

| Имя файла | Описание информации, содержащейся в нем | Функциональное назначение | Файлы проекта, подключенные к текущему файлу посредством директивы #include |
| --- | --- | --- | --- |
| about.h | Окно справки | Отображение информации о программе. | Нет |
| author.h | Класс автора и его методы. | Позволяет использовать функционал класса автора. (удаление, добавление, изменение, поиск,). | Нет |
| author.cpp | Реализация методов класса Author | Позволяет использовать методы класса Author | author.h |
| disk.h | Класс диска и его методы. | Позволяет использовать функционал класса диска. (удаление, добавление, изменение, поиск,). | Нет |
| disk.cpp | Реализация методов класса Disk | Позволяет использовать методы класса Disk | disk.h |
| list.h | Работа связного списка. | Позволяет работать с объектами списка. | Нет |
| mainwindow.h | Класс главнго окна | Методы главного окна. | about.h  report.h |
| process.h | Функции работа с Json файлами, ввод, вывод, поиск по списку. | Позволяет работать с json файлом, производить поиск по id. | list.h  author.h  song.h  disk.h |
| report.h | Окно отчета | Методы окна отчёта | Нет |
| song.h | Класс песни и его методы. | Позволяет использовать функционал класса песни. (удаление, добавление, изменение, поиск,). | Нет |
| song.cpp | Реализация методов класса Song | Позволяет использовать методы класса Song | song.h |
| about.cpp | Методы окна помощи. | Функционал окна помощи. | about.h  ui\_about.h |
| main.cpp | Главный файл, вызывает все нужные функции для запуска. | Вызывает основные функции. | mainwindow.h |
| mainwindow.cpp | Методы основного рабочего окна приложения. Содержит логику всех элементов интерфейса. | Работа с основным окном приложения. Ввод и вывод данных от пользователя | mainwindow.h  ui\_mainwindow.h  process.h |
| report.cpp | Методы окна отчёта приложения. | Содержит логику элементов интерфейса. | report.h  ui\_report.h |

Используя информацию из табл.2 легко нарисовать структурную схему разрабатываемого приложения. (Рис. 1)

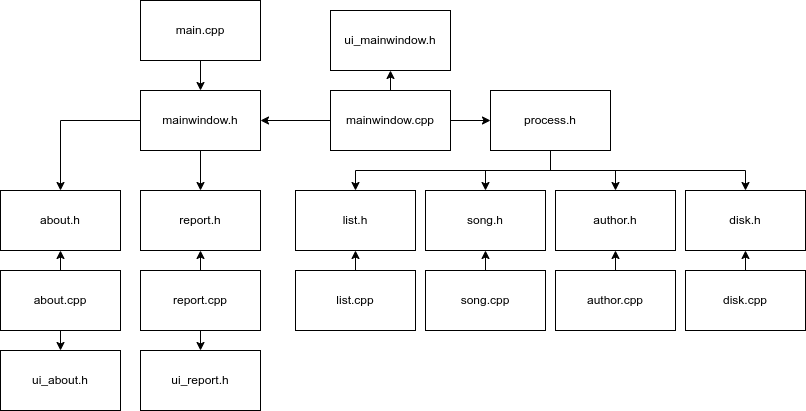


Рис. 1. структурная схема приложения

# ДИЗАЙН ОКОННОГО ИНТЕРФЕЙСА

Для реализации программы работы с данными базе данных меломана можно воспользоваться следующей формой, содержащей 4 вкладки.(Рис. 2-5)

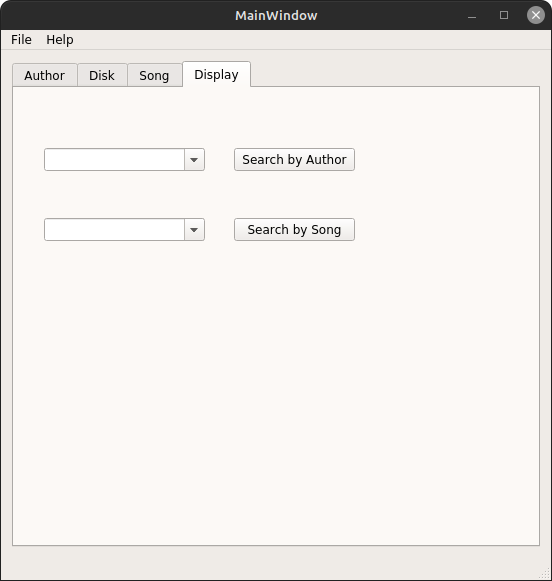


Рис. 2. Оформление формы основного окна приложения, вкладка «Display»

Во вкладке «Display» производится ввод данных для вывода отчёта по всем песням Автора и всех дисков, на котором есть указанная песня. (Рис. 2)

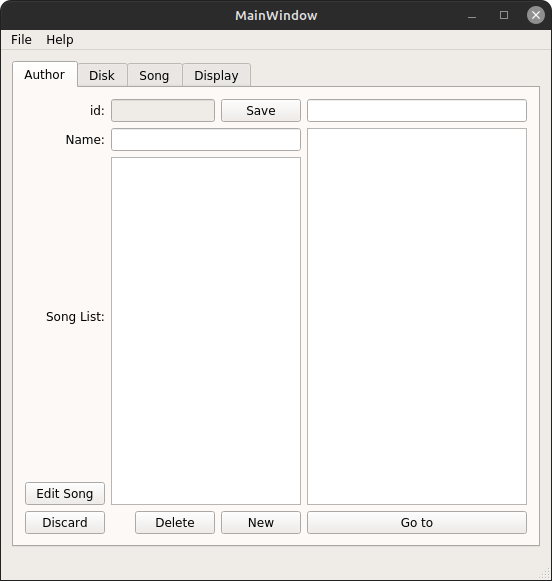


Рис. 3. Оформление формы основного окна приложения, вкладка «Author»

Во вкладке «Author» производится добавление, удаление, изменение записи Автора. Производится навигация по записям. А также переход к форме изменения песни «Song». (Рис. 3)

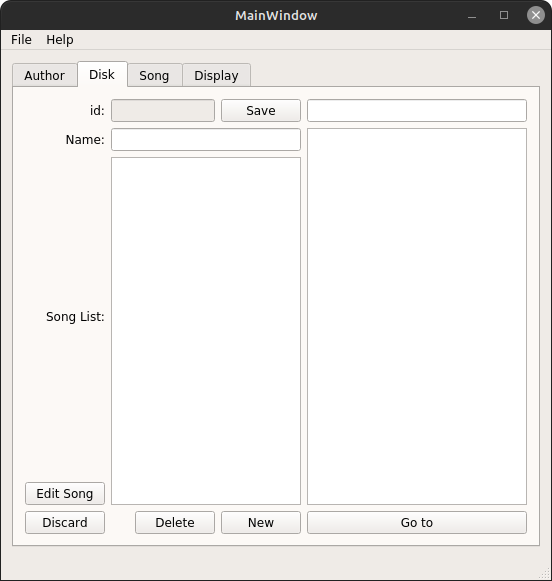


Рис. 4. Оформление формы основного окна приложения, вкладка «Disk»

Во вкладке «Disk» производится добавление, удаление, изменение записи Диска. Производится навигация по записям. А также переход к форме изменения песни «Song». (Рис. 4)

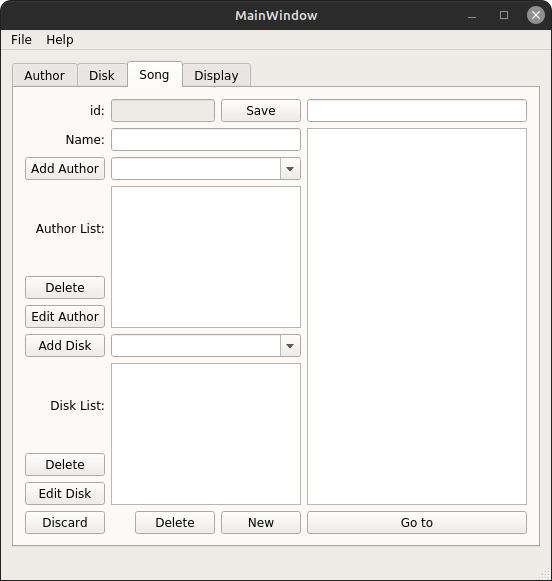


Рис. 5. Оформление формы основного окна приложения, вкладка «Disk»

Во вкладке «Song» производится добавление, удаление, изменение записи Диска. В данном окне устанавливаются связи Песня-Автор и Песня-Диск. Производится навигация по записям. А также переход к формам изменения автора «Author» и диска «Disk». (Рис. 5)

Размещенные на форме компоненты и перечень методов и событий, которые необходимо реализовать приведен в табл.3.

Таблица 3. Компоненты основного окна приложения

| Имя компоненты | Тип | Ограничения для ввода информации | Реализованные события | Функциональное назначение |
| --- | --- | --- | --- | --- |
| comboBoxDisplaySearchByAuthor | QComboBox | Значение из списка авторов | - | Выбор автора для отображения его песен. |
| pushButtonDisplaySearchByAuthor | QPushButton | - | clicked | По щелчку открывает форму «Report», в которой указаны песни автора. |
| comboBoxDisplaySearchBySong | QComboBox | Значение из списка песен | - | Выбор песни для отображения дисков, где она встречается. |
| pushButtonDisplaySearchBySong | QPushButton | - | clicked | По щелчку открывает форму «Report», в которой указаны диски, где встречается песня |
| lineEditAuthorId | QLineEdit | - | - | Выводит Id текущего автора |
| lineEditAuthorName | QLineEdit | Не пустое | - | Ввод имени автора с клавиатуры или автоматический, если запись существует. |
| lineEditAuthorSearch | QLineEdit | - | textChanged | Быстрый поиск среди авторов |
| listWidgetAuthorGoTo | QListWidget | - | doubleClicked | Выбор автора для перемещения |
| listWidgetAuthorSongList | QListWidget | - | doubleClicked | Выводит список песен автора.По двойному щелчку мыши, переходит на изменение песни. |
| pushButtonAuthorDelete | QPushButton | - | clicked | Удаление записи |
| pushButtonAuthorDiscard | QPushButton |  | clicked | Отмена изменений |
| pushButtonAuthorEditSong | QPushButton | - | clicked | Переход на изменение песни. |
| pushButtonAuthorGoTo | QPushButton | - | - | Переход на изменение песни из listWidgetAuthorGoTo |
| pushButtonAuthorNew | QPushButton | - | clicked | Переход к созданию нового автора |
| pushButtonAuthorSave | QPushButton | - | clicked | Сохраняет текущую запись |
| lineEditDiskId | QLineEdit | - | - | Выводит Id текущего диска |
| lineEditDiskName | QLineEdit | Не пустое | - | Ввод названия диска с клавиатуры или автоматический, если запись существует. |
| lineEditDiskSearch | QLineEdit | - | textChanged | Быстрый поиск среди дисков |
| listWidgetDiskGoTo | QListWidget | - | doubleClicked | Выбор диска для перемещения |
| listWidgetDiskSongList | QListWidget | - | activated  doubleClicked | Выводит список песен диска. По двойному щелчку мыши, переходит на изменение песни. |
| pushButtonDiskDelete | QPushButton | - | clicked | Удаление записи |
| pushButtonDiskDiscard | QPushButton | - | clicked | Отмена изменений |
| pushButtonDiskEditSong | QPushButton | - | clicked | Переход на изменение песни. |
| pushButtonDiskGoTo | QPushButton | - | clicked | Переход на изменение песни из listWidgetDiskGoTo |
| pushButtonDiskNew | QPushButton | - | clicked | Переход к созданию нового диска |
| pushButtonDiskSave | QPushButton | - | clicked | Сохраняет текущую запись |
| comboBoxSongAddAuthor | QComboBox | - | activated | Выбор автора для добавления к песне |
| comboBoxSongAddDisk | QComboBox |  | activated | Выбор диска для добавления к песне |
| lineEditSongId | QLineEdit | - | - | Выводит Id текущего автора |
| lineEditSongName | QLineEdit | Не пустое | - | Ввод имени автора с клавиатуры или автоматический, если запись существует. |
| lineEditSongSearch | QLineEdit | - | textChanged | Быстрый поиск среди песен |
| listWidgetSongAuthorList | QListWidget | - | activated  doubleClicked | Выводит список авторов песни. По двойному щелчку мыши, переходит на изменение автора. |
| listWidgetSongDiskList | QListWidget | - | activated  doubleClicked | Выводит список дисков, где встречается песня. По двойному щелчку мыши, переходит на изменение диска. |
| listWidgetSongGoTo | QListWidget | - | doubleClicked | Выбор песни для перемещения |
| pushButtonSongAddAuthor | QPushButton | - | clicked | Добавляет автора песни из comboBoxSongAddAuthor |
| pushButtonSongAddDisk | QPushButton | - | clicked | Добавляет песню в диск из comboBoxSongAddDisk |
| pushButtonSongDelete | QPushButton | - | clicked | Удаление записи |
| pushButtonSongDeleteAuthor | QPushButton | - | clicked | Удаляет связь автор-песня |
| pushButtonSongDeleteDisk | QPushButton | - | clicked | Удаляет связь диск-песня |
| pushButtonSongDiscard | QPushButton | - | clicked | Отмена изменений |
| pushButtonSongEditAuthor | QPushButton | - | clicked | Переход на изменение автора. |
| pushButtonSongEditDisk | QPushButton | - | clicked | Переход на изменение диска. |
| pushButtonSongGoTo | QPushButton | - | clicked | Переход на изменение песни из listWidgetSongGoTo |
| pushButtonSongNew | QPushButton | - | clicked | Переход к созданию новой песни |
| pushButtonSongSave | QPushButton | - | clicked | Сохраняет текущую запись |

Окно помощи кратко описывает функционал программы и выводит контакты автора (Рис. 6)

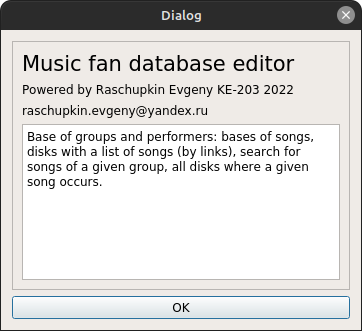


Рис. 6. Оформление формы окна помощи приложения

Размещенные на форме компоненты и перечень методов и событий, которые необходимо реализовать приведен в табл.4

Табл. 4. Компоненты окна помощи приложения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя компоненты | Тип | Ограничения для ввода информации | Реализованные события | Функциональное назначение |
| pushButtonAboutOk | QPushButton | - | clicked | Закрытие окна |

Окно отчёта выводит данные по запросу из главного окна. (Рис. 7)

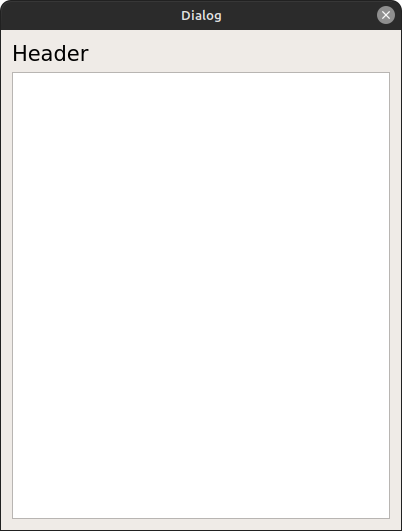


Рис. 7. Оформление формы окна отчёта приложения

Алгоритмы и программная реализация приведенных в табл. 3-4 событий и методов приведены в следующем разделе.

# РАЗРАБОТКА ФУНКЦИОНАЛА ОСНОВНЫХ ФОРМ И МЕХАНИЗМОВ ПОЛУЧЕНИЯ ИНФОРМАЦИИ

## Главная форма

Главная форма состоит из 4 вкладок, они имеют схожие связи между вводимыми/выводимыми данными. Для первых трех вкладок имеются схожие механизмы для получения наименования, id объекта, а также зависимостей. Например, на рис.8 указана структурная схема главного окна вкладки «Author». Также на этом рисунке показана структурная схема панели Меню.

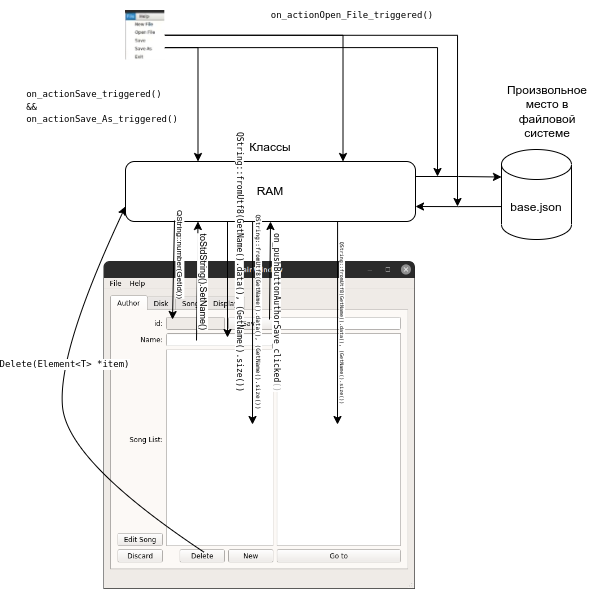


Рис. 8. Структурная схема вкладки «Author» и меню «File»

Аналогичную структурную схему имеет и вкладка «Disk».

Схема вкладки «Song» изображена на рис. 9.

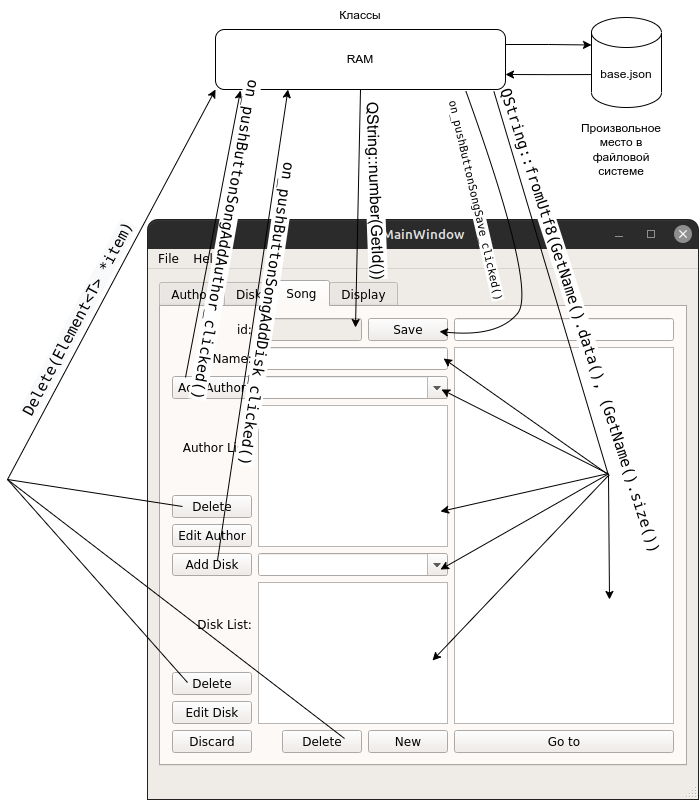


Рис. 9. Структурная схема вкладки «Song»

Пример реализации перехода к изменению Автора представлен в листинге 4. Для начала переменная OnLoad, обозначающия процесс программного изменения полей, переставляется на истину. Затем отчищается лист песен автора. После узел переставляется на узел, который указан во входных параметрах. Узел изменения указывается текущим. Получаем id, name и лист, устанавливаем в окно имени имя автора, в окно айди устанавливаем айди. Затем, пробегаемя по всем песням, добавляя их в список и устанавливаем видимость только тем, которые есть у автора.

Листинг 4. Пример Реализации перехода к изменению автора

void MainWindow::AuthorGoToEdit(int index) // Переход к изменению автора

{

OnLoad = true;

ui->listWidgetAuthorSongList->clear();

at = AuthorList.Move(index);

eat = at;

std::cout << "Author: Moving to " << index << std::endl;

ui->statusBar->showMessage("Author: Moving to " + QString::number(index));

AuthorId = at->data.GetId();

AuthorName = at->data.GetName();

AuthorListIdSong = at->data.GetSongs();

ui->lineEditAuthorId->setText(QString::number(AuthorId));

ui->lineEditAuthorName->setText(QString::fromUtf8((AuthorName).data(), (AuthorName).size()));

if (SongList.GetCount() != 0)

{

st = SongList.GetFirst();

QString currentSongName;

while (st)

{

// setHidden(true),

currentSongName = QString::fromUtf8((st->data.GetName()).data(), (st->data.GetName()).size());

QListWidgetItem \*item = new QListWidgetItem(currentSongName);

ui->listWidgetAuthorSongList->addItem(item);

item->setHidden(true);

for (int i = 0; i < AuthorListIdSong.size(); i++)

{

if (st->data.GetId() == AuthorListIdSong[i])

{

item->setHidden(false);

break;

}

}

st = st->next;

}

}

OnLoad = false;

AuthorEditActivated(true);

}

Аналогично работает и переход к диску, переход к песне осуществляется подобным образом за исключеним того, что заместо вывода списка песен мы выводим списки авторов и дисков, открывая необходимые.

Кнопки перехода на изменение, как и двойной щелчёк по элементам просто вызывают эту функцию, как, например, переход к автору по двойному щелчку ЛКМ. (см. листинг 5)

Листинг 5. Пример Реализации перехода к изменению автора по двойному щелчку

void MainWindow::on\_listWidgetAuthorGoTo\_doubleClicked(const QModelIndex &index) // Переход на позицию по двойному щелчку

{

AuthorGoToEdit(index.row());

}

Переходы из внутреннего списка работают аналогично за исключением того, что они дополнительно переключают вкладку.

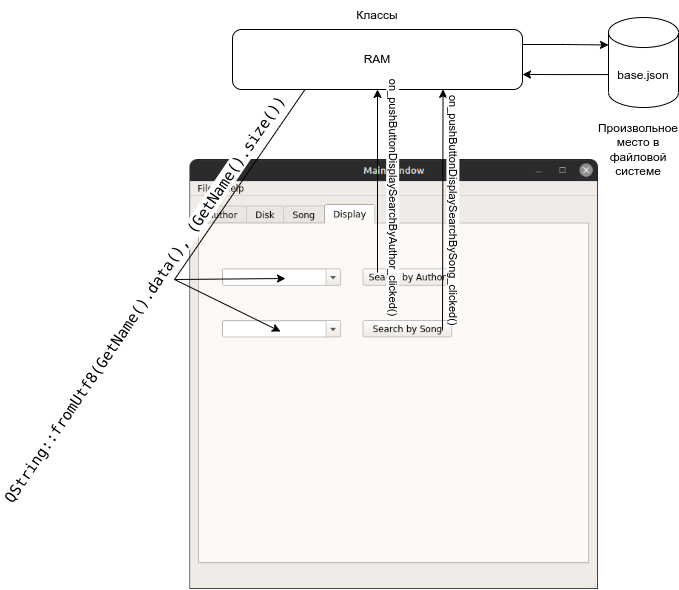


Рис. 10. Структурная схема вкладки «Display»

Функции вывода отчёта по Автору работает так. Поочерёдно считывают id песен автора указанного в comboBox, затем, проходя по списку песен, названия песен добавляются в отчёт. Аналогично и для отчёта по дискам.

Программная реализация этих и других функций приведена в приложении А.

# РАЗРАБОТКА ОСНОВНОГО МЕХАНИЗМА ФУНКЦИОНИРОВАНИЯ ПРИЛОЖЕНИЯ

Программные реализации функций, отвечающие за чтение файла в Json приведены в листингах 6-9. Открывается файл, если файл не открыт, выдаётся ошибка, иначе, файл конвертируется в json. Перевод в лист осуществляется последовательным добавлением текущего элемента json в переменную файла.

Листинг 6. Программная реализации чтения json из файла.

// Открытие файла запись в json

nlohmann::ordered\_json jsonFromFile(std::string FileNameInput)

{

std::ifstream ifs{FileNameInput}; //Открытие файла

if (!ifs.is\_open())

{

std::cerr << "Unable to open file\n";

throw 1;

}

nlohmann::ordered\_json file\_json = nlohmann::ordered\_json::parse(ifs); //Перевод в Json

ifs.close(); //Закрытие файла

return file\_json;

}

Листинг 7. Программная реализации перевода json в список авторов.

// Чтение Json в Лист

void readAuthors(nlohmann::ordered\_json file\_json, List<Author> &AuthorList)

{

int AuthorId;

std::string AuthorName;

std::vector<int> AuthorListIdSong;

Author currentAuthor;

for (nlohmann::ordered\_json::iterator it1 = file\_json.at("authors").begin(); it1 != file\_json.at("authors").end(); ++it1)

{

AuthorId = std::stoi(it1.key());

AuthorName = file\_json.at("authors").at(it1.key())["name"].get<std::string>();

AuthorListIdSong = file\_json.at("authors").at(it1.key())["songs"].get<std::vector<int>>();

currentAuthor.ChangeAuthor(AuthorId, AuthorName, AuthorListIdSong);

AuthorList.AddEnd(currentAuthor);

}

}

Листинг 8. Программная реализации перевода json в список дисков.

// Чтение Json в Лист

void readDisks(nlohmann::ordered\_json file\_json, List<Disk> &DiskList)

{

int DiskId;

std::string DiskName;

std::vector<int> DiskListIdSong;

Disk currentDisk;

for (nlohmann::ordered\_json::iterator it1 = file\_json.at("disks").begin(); it1 != file\_json.at("disks").end(); ++it1)

{

DiskId = std::stoi(it1.key());

DiskName = file\_json.at("disks").at(it1.key())["name"].get<std::string>();

DiskListIdSong = file\_json.at("disks").at(it1.key())["songs"].get<std::vector<int>>();

currentDisk.ChangeDisk(DiskId, DiskName, DiskListIdSong);

DiskList.AddEnd(currentDisk);

}

}

Листинг 9. Программная реализации перевода json в список песен.

// Чтение Json в Лист

void readSongs(nlohmann::ordered\_json file\_json, List<Song> &SongList)

{

int SongId;

std::string SongName;

std::vector<int> ListIdAuthor;

std::vector<int> ListIdDisk;

Song currentSong;

for (nlohmann::ordered\_json::iterator it1 = file\_json.at("songs").begin(); it1 != file\_json.at("songs").end(); ++it1)

{

SongId = std::stoi(it1.key());

SongName = file\_json.at("songs").at(it1.key())["name"].get<std::string>();

ListIdAuthor = file\_json.at("songs").at(it1.key())["authors"].get<std::vector<int>>();

ListIdDisk = file\_json.at("songs").at(it1.key())["disks"].get<std::vector<int>>();

currentSong.ChangeSong(SongId, SongName, ListIdAuthor, ListIdDisk);

SongList.AddEnd(currentSong);

}

}

Поиск элемента по id реализован с помощью перегруженной функции SearchElemetById его программная реализация приведена в листинге 10. Его логика проста. Пока мы не встречаем элемент с заданным id, мы переходим к следующему элементу. При встрече, возвращаем элемент.

Листинг 10. Программная реализации поиска элемента по id.

// Поиск элемента по id среди авторов

Element<Author> \*SearchElemetById(List<Author> AuthorList, int AuthorId)

{

Element<Author> \*t = AuthorList.GetFirst();

while (t)

{

if (AuthorId == t->data.GetId())

return t;

t = t->next;

}

return NULL;

}

// Поиск элемента по id среди дисков

Element<Disk> \*SearchElemetById(List<Disk> DiskList, int DiskId)

{

Element<Disk> \*t = DiskList.GetFirst();

while (t)

{

if (DiskId == t->data.GetId())

return t;

t = t->next;

}

return NULL;

}

// Поиск элемента по id среди песен

Element<Song> \*SearchElemetById(List<Song> SongList, int SongId)

{

Element<Song> \*t = SongList.GetFirst();

while (t)

{

if (SongId == t->data.GetId())

return t;

t = t->next;

}

return NULL;

}

Реализация удаления элемента состоит из 2х этапов, удаление элемента из списка и удаление связей. Удаление элемента из списка приведено в листинге 11. Реализация удаления связей приведена в листинге 12-14. Удаления связей рекурсивное, для каждого id в векторе связей ищется соответствующий элемент и у него удаляется связь с основным.

Листинг 11. Программная реализации удаления элемента из списка.

// Удалить текущий элемент

template <class T>

void List<T>::Delete(Element<T> \*item)

{

if (count == 0 || item == NULL) // Проверка, не пустой ли список

return;

Element<T> \*itemPrev = item->prev; // Предыдущий элемент

Element<T> \*itemNext = item->next; // Следующий элемент

if ((count > 1) && (itemPrev != nullptr)) // Проверка, удаляется ли не первый элемент списка

itemPrev->next = itemNext;

if ((count > 1) && (itemNext != nullptr)) // Проверка, удаляется ли не последний элемент списка

itemNext->prev = itemPrev;

if (item == head) // Если удаляется первый элемент

head = itemNext;

if (item == tail) // Если удаляется последний элемент

tail = itemPrev;

delete item; // Удалить элемент item

count--; // Уменьшить общее количество элементов

}

Листинг 12. Программная реализации удаления связей автора.

// Удаление связей Автора

void deleteLinks(Element<Author> \*at, List<Song> &SongList)

{

int AuthorId = at->data.GetId();

std::vector<int> AuthorListIdSong = at->data.GetSongs();

for (int i = 0; i < AuthorListIdSong.size(); i++)

{

Element<Song> \*st = SongList.GetFirst();

while (st)

{

if (st->data.GetId() == AuthorListIdSong[i])

{

st->data.RemoveAuthor(AuthorId);

break;

}

st = st->next;

}

}

}

Листинг 13. Программная реализации удаления связей диска.

}

// Удаление связей Диска

void deleteLinks(Element<Disk> \*dt, List<Song> &SongList)

{

int DiskId = dt->data.GetId();

std::vector<int> DiskListIdSong = dt->data.GetSongs();

for (int i = 0; i < DiskListIdSong.size(); i++)

{

Element<Song> \*st = SongList.GetFirst();

while (st)

{

if (st->data.GetId() == DiskListIdSong[i])

{

st->data.RemoveDisk(DiskId);

break;

}

st = st->next;

}

}

}

Листинг 14. Программная реализации удаления связей песни.

// Удаление связей Песни

void deleteLinks(Element<Song> \*st, List<Author> &AuthorList, List<Disk> &DiskList)

{

int SongId = st->data.GetId();

std::vector<int> ListIdAuthor = st->data.GetAuthors();

std::vector<int> ListIdDisk = st->data.GetDisks();

// Удаление связей с авторами

for (int i = 0; i < ListIdAuthor.size(); i++)

{

Element<Author> \*at = AuthorList.GetFirst();

while (at)

{

if (at->data.GetId() == ListIdAuthor[i])

{

at->data.RemoveSong(SongId);

break;

}

at = at->next;

}

}

// Удаление связей с дисками

for (int i = 0; i < ListIdDisk.size(); i++)

{

Element<Disk> \*dt = DiskList.GetFirst();

while (dt)

{

if (dt->data.GetId() == ListIdDisk[i])

{

dt->data.RemoveSong(SongId);

break;

}

dt = dt->next;

}

}

}

Реализация вставки в начало и конец приведена в листингах 15-16.

Листинг 15. Программная реализации вставки элемента в конец списка.

// Добавить элемент в конец списка

template <class T>

void List<T>::AddEnd(T \_data)

{

try

{

Element<T> \*t = new Element<T>(\_data); // Создать новый элемент с данными \_data

t->prev = tail; // установить указатель на предыдущий элемент

if (tail != nullptr) // Заполнить поле переставить указатель последнего элемента на новый

tail->next = t;

if (count == 0) // Проверка, есть ли в списке элементы

{

head = tail = t; // Если элементов нет, установить элемент на начало и конец

}

else

{

tail = t; // Если элементы в списке есть, то это конец списка

}

count++; // Увеличить общее количество элементов

}

catch (const std::bad\_alloc &e)

{

std::cerr << e.what(); // Если память не выделена, то вывести системное сообщение

}

}

Листинг 16. Программная реализации вставки элемента в начало списка.

// Добавить элемент в начало списка

template <class T>

void List<T>::AddBegin(T \_data)

{

try

{

Element<T> \*t = new Element<T>(\_data); // Создание нового элемента

t->next = head; // Следующий элемент указывает на предыдущий первый

if (count > 0) // Проверка, не пустой ли список

{

head->prev = t;

head = t;

}

else // если элементов нет, то устанавливаем элемент на начало и конец

head = tail = t;

count++; // Увеличение общего количества элементов

}

catch (const std::bad\_alloc &e) // если память не выделена, ошибка

{

std::cerr << e.what();

}

}

Реализация изменения элемента приведена в листинге 17.

Листинг 17. Программная реализации изменения элемента.

// Изменить значение элемента в текущей позиции

template <class T>

void List<T>::SetElement(T \_data, Element<T> \*t)

{

t->data = \_data;

}

# ТЕСТИРОВАНИЕ

Приведём результаты автономного и комплексного тестирования разработанной программы.

## Автономное тестирование

Автономное тестирование показывет, как работают все функции, которые мы рассмотрели в 4 главе.

Автономное тестирование функции чтения json из файла приведено в таблице 5.

Табл. 5. Тестирование операции чтения json из файла

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Верно ли?** |
| /home/evgen/Documents/base.json | nlohmann::ordered\_json file\_json | **Да** |
| /home/evgen/Documents/photo.png | nlohmann::detail::type\_error  Ошибка о невозможности парсинга в json | **Да** |
| /hooome/evgen/Documents/base.json | Unable to open file | **Да** |

Автономное тестирование функции перевода json в список авторов приведено в таблице 6.

Табл. 6. Тестирование операции перевода json в список авторов

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Верно ли?** |
| {"authors":{"1":{"name":"makerA","songs":[1,2,3]},"2":{"name":"makerB","songs":[1,2,5,6,7]}}} | Список авторов | **Да** |
| {"disks": {"1":{"name": "diskA","songs": [1,2,3]}}} | Ошибка об отсутсвии авторов | **Да** |

Автономное тестирование функции перевода json в список дисков приведено в таблице 7.

Табл. 7. Тестирование операции перевода json в список дисков

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Верно ли?** |
| {"disks": {"1":{"name": "diskA","songs": [1,2,3]}}} | Список дисков | **Да** |
| {"authors":{"1":{"name":"makerA","songs":[1,2,3]},"2":{"name":"makerB","songs":[1,2,5,6,7]}}} | Ошибка об отсутсвии дисков | **Да** |

Автономное тестирование функции перевода json в список песен приведено в таблице 8.

Табл. 8. Тестирование операции перевода json в список песен

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Верно ли?** |
| {"songs":{"1":{"authors":[1,2],"disks":[1],"name":"songA"}}} | Список дисков | **Да** |
| {"authors":{"1":{"name":"makerA","songs":[1,2,3]},"2":{"name":"makerB","songs":[1,2,5,6,7]}}} | Ошибка об отсутсвии песен | **Да** |

Автономное тестирование функции поиска элемента по id приведено в таблице 9.

Табл. 9. Тестирование операции поиска элемента по id

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Верно ли?** |
| Существующий id | Element<Author> \* | **Да** |
| Несуществующий id | NULL | **Да** |
| Пустой список | NULL | **Да** |

Автономное тестирование функции удаления элемента приведено в таблице 10.

Табл. 10. Тестирование операции удаления элемента

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Верно ли?** |
| Существующий элемент непустого списка | Удаление | **Да** |
| Несуществующий элемент | Ничего | **Да** |
| Пустой список | Ничего | **Да** |
| Неполные связи существующего элемента | Удаление связей доступных со стороны удаляемого элемента | **Да** |

Автономное тестирование функции вставки элемента приведено в таблице 11.

Табл. 11. Тестирование операции удаления элемента

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Верно ли?** |
| Вставка в начало в непустом списке | Список с новым элементом в начале | **Да** |
| Вставка в конец в непустом списке | Список с новым элементом в конце | **Да** |
| Вставка в начало в пустом списке | Список с единственныи элементом | **Да** |
| Вставка в конец в пустом списке | Список с единственныи элементом | **Да** |

Автономное тестирование функции изменения элемента приведено в таблице 12.

Табл. 12. Тестирование операции изменения элемента

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Верно ли?** |
| Изменение элемента на непустой элемент с данными | Элемент непустой | **Да** |
| Изменение элемента на пустой элемент без данный | Элемент пустой | **Да** |

## Комплексное тестирование

Комплексное тестирование показывет взаимодействия компонентов программы.

Табл.5. Тестирование головного модуля (примеры заполнения таблицы).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Описание ситуации | Входные данные | Выходные данные | Тест пройден? |
| 1. | Добавление/Удаление/Изменение/Вывод данных | Данные полей названий и списки связей | Добавление/Удаление/Изменение/Вывод данных | Да |
| 2. | Пользователь ввёл пустое название имени Автора/Диска/Песни | Пустое поле имени | Предупреждение с сообщением о необходимости заполнить поле имени. | Да |
| 3. | Пользователь закрыл окно системного диалога с Открытием/Сохранением файла | - | Прерывание операции | Да |
| 4. | Отсутвие данных в файле о Авторах/Дисков/Песен | Json файл | Предупреждение об отсутсвии списков, продолжение работы | Да |
| 5. | Открытие повреждённого файла | Повреждённый файл | Предупреждение об ошибке парсинга с json, прерывание операции | Да |
| 6. | Неверный путь сохранения | Неверный путь сохранения | Предупреждение об ошибке сохранения, прерываение операции | Да |

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Данный раздел следует разбить на следующие пункты:

1. Запуск приложения, приветственные экраны, какие ресурсы необходимы для выполнения программы.
2. Различные способы ввода и вывода информации.
3. Описание всех возможностей программы.
4. Перечень ошибок и способов их устранения.

Следует отметить, что пользователь может быть не знаком с языками программирования и объектно-ориентированным подходом. Пользователь – это человек, который запускает исполняемый файл и работает с ним. От пользователя следует скрывать все детали реализации, поскольку эта информация для него не актуальна. Чтобы было легче написать данный раздел представьте, что вы рассказываете о функционировании своей программы потенциальному покупателю и показываете ему все возможности вашей разработки.

# ЗАКЛЮЧЕНИЕ

Данная работа посвящена разработке оконного приложения «База данных меломана» с использованием языка C++. Для хранения данных использованы объекты классов «Author», «Disk» и «Song», которые помещены в шаблонный класс List. При организации функционирования класса была создана перегруженная функция SearchElemetById, используемая для поиска элементов списка по id в разных списках.

В ходе выполнения работы были пройдены все основные этапы разработки программного обеспечения: анализ, написание спецификации, проектирование, разработка алгоритмов, кодирование, тестирование и сопровождение. Проведено автономное тестирование основных методов разработанного класса и комплексное тестирование программы в целом. В результате тестирования ошибок не обнаружено.

Работоспособность программы можно улучшить, оптимизировав обновление полей. Можно также добавить отображение отчёта по песням диска. Доработать программу можно, создав поля даты создания песни и краткой информации о авторе.

# ЛИТЕРАТУРА

1. Qt cross-platform software development framework : Qt Documentation. — URL: https://doc.qt.io/ (дата обращения: 18.07.2022).

# ПРИЛОЖЕНИЯ

Приложение A. Программная реализация событий главного окна

Листинг 18 Программная реализация событий главного окна

#include <QApplication>

#include <QFileDialog>

#include <QMessageBox>

#include <QDebug>

#include <iostream>

#include <vector>

#include <fstream>

#include <nlohmann/json.hpp>

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "process.h"

// Авторы

List<Author> AuthorList;

Element<Author> \*at; // Элемент для вывода всех значений

Element<Author> \*eat; // Изменямый в данный момент элемент

Author currentAuthor;

int AuthorId;

std::string AuthorName;

std::vector<int> AuthorListIdSong;

// Диски

List<Disk> DiskList;

Element<Disk> \*dt; // Элемент для вывода всех значений

Element<Disk> \*edt; // Изменямый в данный момент элемент

Disk currentDisk;

int DiskId;

std::string DiskName;

std::vector<int> DiskListIdSong;

// Песни

List<Song> SongList;

Element<Song> \*st; // Элемент для вывода всех значений

Element<Song> \*est; // Изменямый в данный момент элемент

Song currentSong;

int SongId;

std::string SongName;

std::vector<int> ListIdAuthor;

std::vector<int> ListIdDisk;

// Файлы

QString fileNameInput;

QString fileNameOutput;

QString fileNameDefaultOutput;

// Состояния

bool AuthorEdited = false;

bool DiskEdited = false;

bool SongEdited = false;

bool DataChanged = false;

bool OnLoad = false;

void MainWindow::refresh() // Обновление виджетов со cвязями

{

OnLoad = true;

ui->listWidgetAuthorGoTo->clear();

ui->comboBoxSongAddAuthor->clear();

ui->comboBoxDisplaySearchByAuthor->clear();

ui->listWidgetDiskGoTo->clear();

ui->comboBoxSongAddDisk->clear();

ui->comboBoxDisplaySearchBySong->clear();

ui->listWidgetSongGoTo->clear();

at = AuthorList.GetFirst();

while (at) // Обновление списков авторов

{

QString QAuthorName = QString::fromUtf8((at->data.GetName()).data(), (at->data.GetName()).size());

ui->listWidgetAuthorGoTo->addItem(QAuthorName);

ui->comboBoxSongAddAuthor->addItem(QAuthorName);

ui->comboBoxDisplaySearchByAuthor->addItem(QAuthorName);

at = at->next;

}

dt = DiskList.GetFirst();

while (dt) // Обновление списков дисков

{

QString QDiskName = QString::fromUtf8((dt->data.GetName()).data(), (dt->data.GetName()).size());

ui->listWidgetDiskGoTo->addItem(QDiskName);

ui->comboBoxSongAddDisk->addItem(QDiskName);

dt = dt->next;

}

st = SongList.GetFirst();

while (st) // Обновление списков песен

{

QString QSongName = QString::fromUtf8((st->data.GetName()).data(), (st->data.GetName()).size());

ui->listWidgetSongGoTo->addItem(QSongName);

ui->comboBoxDisplaySearchBySong->addItem(QSongName);

st = st->next;

}

OnLoad = false;

}

void MainWindow::refreshAuthor() // Обновление окна изменения автора

{

OnLoad = true;

ui->lineEditAuthorId->clear();

ui->lineEditAuthorName->clear();

ui->listWidgetAuthorSongList->clear();

OnLoad = false;

}

void MainWindow::refreshDisk() // Обновление окна изменения диска

{

OnLoad = true;

ui->lineEditDiskId->clear();

ui->lineEditDiskName->clear();

ui->listWidgetDiskSongList->clear();

OnLoad = false;

}

void MainWindow::refreshSong() // Обновление окна изменения песни

{

OnLoad = true;

ui->lineEditSongId->clear();

ui->lineEditSongName->clear();

ui->listWidgetSongAuthorList->clear();

ui->listWidgetSongDiskList->clear();

OnLoad = false;

}

void MainWindow::AuthorOnEdit(bool option) // Блокирует все поля, кроме изменяемых у Автора

{

AuthorEdited = option;

ui->tabDisk->setEnabled(not option);

ui->tabSong->setEnabled(not option);

ui->tabDisplay->setEnabled(not option);

ui->lineEditAuthorSearch->setEnabled(not option);

ui->listWidgetAuthorGoTo->setEnabled(not option);

ui->pushButtonAuthorSave->setEnabled(option);

ui->listWidgetAuthorSongList->setEnabled(false);

ui->pushButtonAuthorDelete->setEnabled(false);

ui->pushButtonAuthorDiscard->setEnabled(option);

ui->pushButtonAuthorEditSong->setEnabled(false);

ui->pushButtonAuthorGoTo->setEnabled(not option);

ui->pushButtonAuthorNew->setEnabled(not option);

ui->menuBar->setEnabled(not option);

if (not option)

ui->lineEditAuthorName->setEnabled(option);

}

void MainWindow::AuthorEditActivated(bool option) // Открывает поля для изменения

{

ui->lineEditAuthorName->setEnabled(option);

ui->listWidgetAuthorSongList->setEnabled(option);

ui->pushButtonAuthorDelete->setEnabled(option);

ui->pushButtonAuthorEditSong->setEnabled(option);

}

void MainWindow::DiskOnEdit(bool option) // Блокирует все поля, кроме изменяемых у Диска

{

DiskEdited = option;

ui->tabAuthor->setEnabled(not option);

ui->tabSong->setEnabled(not option);

ui->tabDisplay->setEnabled(not option);

ui->lineEditDiskSearch->setEnabled(not option);

ui->listWidgetDiskGoTo->setEnabled(not option);

ui->pushButtonDiskSave->setEnabled(option);

ui->listWidgetDiskSongList->setEnabled(false);

ui->pushButtonDiskDelete->setEnabled(false);

ui->pushButtonDiskDiscard->setEnabled(option);

ui->pushButtonDiskEditSong->setEnabled(false);

ui->pushButtonDiskGoTo->setEnabled(not option);

ui->pushButtonDiskNew->setEnabled(not option);

ui->menuBar->setEnabled(not option);

if (not option)

ui->lineEditDiskName->setEnabled(option);

}

void MainWindow::DiskEditActivated(bool option) // Открывает поля для изменения

{

ui->lineEditDiskName->setEnabled(option);

ui->listWidgetDiskSongList->setEnabled(option);

ui->pushButtonDiskDelete->setEnabled(option);

ui->pushButtonDiskEditSong->setEnabled(option);

}

void MainWindow::SongOnEdit(bool option) // Блокирует все поля, кроме изменяемых у песни

{

SongEdited = option;

ui->tabAuthor->setEnabled(not option);

ui->tabDisk->setEnabled(not option);

ui->tabDisplay->setEnabled(not option);

ui->lineEditSongSearch->setEnabled(not option);

ui->listWidgetSongGoTo->setEnabled(not option);

ui->pushButtonSongSave->setEnabled(option);

ui->listWidgetSongAuthorList->setEnabled(true);

ui->pushButtonSongDelete->setEnabled(false);

ui->pushButtonSongDiscard->setEnabled(option);

ui->pushButtonSongEditAuthor->setEnabled(false);

ui->pushButtonSongEditDisk->setEnabled(false);

ui->pushButtonSongGoTo->setEnabled(not option);

ui->pushButtonSongNew->setEnabled(not option);

ui->menuBar->setEnabled(not option);

if (not option)

{

ui->lineEditSongName->setEnabled(option);

ui->comboBoxSongAddAuthor->setEnabled(option);

ui->pushButtonSongAddAuthor->setEnabled(option);

ui->listWidgetSongAuthorList->setEnabled(option);

ui->pushButtonSongDeleteAuthor->setEnabled(option);

ui->pushButtonSongAddDisk->setEnabled(option);

ui->comboBoxSongAddDisk->setEnabled(option);

ui->listWidgetSongDiskList->setEnabled(option);

ui->pushButtonSongDeleteDisk->setEnabled(option);

}

}

void MainWindow::SongEditActivated(bool option) // Открывает поля для изменения

{

ui->lineEditSongName->setEnabled(option);

ui->comboBoxSongAddAuthor->setEnabled(option);

ui->pushButtonSongAddAuthor->setEnabled(option);

ui->listWidgetSongAuthorList->setEnabled(option);

ui->pushButtonSongDeleteAuthor->setEnabled(option);

ui->pushButtonSongEditAuthor->setEnabled(option);

ui->listWidgetSongAuthorList->setEnabled(option);

ui->pushButtonSongAddDisk->setEnabled(option);

ui->comboBoxSongAddDisk->setEnabled(option);

ui->pushButtonSongDeleteDisk->setEnabled(option);

ui->pushButtonSongEditDisk->setEnabled(option);

ui->listWidgetSongDiskList->setEnabled(option);

ui->pushButtonSongDelete->setEnabled(option);

}

MainWindow::MainWindow(QWidget \*parent) : QMainWindow(parent), ui(new Ui::MainWindow)

{

ui->setupUi(this);

}

MainWindow::~MainWindow() // Деструктор

{

delete ui;

}

void MainWindow::on\_actionAbout\_triggered() // Вызов справки

{

about = new About(this);

about->setModal(true);

about->show();

}

void MainWindow::on\_pushButtonDisplaySearchByAuthor\_clicked() // Поиск песен заданного автора

{

if (ui->comboBoxDisplaySearchByAuthor->currentIndex() != -1)

{

report = new Report(this); // Создание окна

report->setModal(true);

report->show();

at = AuthorList.Move(ui->comboBoxDisplaySearchByAuthor->currentIndex());

AuthorId = at->data.GetId();

AuthorName = at->data.GetName();

AuthorListIdSong = at->data.GetSongs();

// Устанавливаем заголовок

QString Header = QString::fromUtf8(AuthorName.data(), AuthorName.size()) + " songs:";

connect(this, &MainWindow::signalDisplaySearchSetHeader, report, &Report::slotDisplaySearchSetHeader);

emit signalDisplaySearchSetHeader(Header); // Отправка сигнала заголовка

// Добавляем песни

QString item;

connect(this, &MainWindow::signalDisplaySearchAddToList, report, &Report::slotDisplaySearchAddToList);

for (int i = 0; i < AuthorListIdSong.size(); i++) // Проходим по списку песен автора

{

st = Process::SearchElemetById(SongList, AuthorListIdSong[i]);

SongName = " - " + st->data.GetName();

ListIdAuthor = st->data.GetAuthors();

std::string itemAuhors = Process::SearchElemetById(AuthorList, ListIdAuthor[0])->data.GetName();

for (int j = 1; j < ListIdAuthor.size(); j++)

{

itemAuhors = itemAuhors + ", " + Process::SearchElemetById(AuthorList, ListIdAuthor[j])->data.GetName();

}

itemAuhors = itemAuhors + SongName;

item = QString::fromUtf8(itemAuhors.data(), itemAuhors.size());

emit signalDisplaySearchAddToList(item); // Отправка сигнала небходимой песни с авторами

}

qDebug() << "Display: author songs report created";

ui->statusBar->showMessage("Display: author songs report created");

}

}

void MainWindow::on\_pushButtonDisplaySearchBySong\_clicked() // Поиск дисков, где встречается заданная песня

{

if (ui->comboBoxDisplaySearchBySong->currentIndex() != -1)

{

report = new Report(this); // Создание окна

report->setModal(true);

report->show();

st = SongList.Move(ui->comboBoxDisplaySearchBySong->currentIndex());

SongId = st->data.GetId();

SongName = st->data.GetName();

ListIdDisk = st->data.GetDisks();

// Устанавливаем заголовок

QString Header = QString::fromUtf8(SongName.data(), SongName.size()) + " disks:";

connect(this, &MainWindow::signalDisplaySearchSetHeader, report, &Report::slotDisplaySearchSetHeader);

emit signalDisplaySearchSetHeader(Header); // Отправка сигнала с заголовком

// Добавляем диски

QString item;

connect(this, &MainWindow::signalDisplaySearchAddToList, report, &Report::slotDisplaySearchAddToList);

for (int i = 0; i < ListIdDisk.size(); i++) // Проходим по всем элементам

{

dt = Process::SearchElemetById(DiskList, ListIdDisk[i]);

DiskName = dt->data.GetName();

item = QString::fromUtf8(DiskName.data(), DiskName.size());

emit signalDisplaySearchAddToList(item); // Отправка сигнала текущего диска

}

qDebug() << "Display: song disks report created";

ui->statusBar->showMessage("Display: song disks report created");

}

}

void MainWindow::on\_actionOpen\_File\_triggered() // Открытие файла

{

if (DataChanged) // На случай, если изменения не сохранены

{

QMessageBox::StandardButton reply;

reply = QMessageBox::question(this, "Open", "Do you really want to open new file? Unsaved data will be lost.", QMessageBox::Yes | QMessageBox::No);

if (reply != QMessageBox::Yes)

{

qDebug() << "Opening aborted.";

ui->statusBar->showMessage("Opening aborted");

return;

}

}

fileNameInput = "";

fileNameInput = QFileDialog::getOpenFileName(this, tr("Open base"), "" ,tr("Image Files (\*.json)"));

if (fileNameInput != "") // Если системный диалог закрыли

{

nlohmann::ordered\_json file\_json;

try // Открытие

{

qDebug() << "Opening" << fileNameInput << "...";

ui->statusBar->showMessage("Opening \"" + fileNameInput + "\"...");

std::string FileNameInput = fileNameInput.toStdString();

file\_json = Process::jsonFromFile(FileNameInput);

}

catch (const nlohmann::detail::parse\_error &e) // Невозможность парсинга

{

qDebug() << e.what();

ui->statusBar->showMessage(e.what());

QMessageBox::warning(this, "Warning", "Error while parsing json");

return;

}

catch (...) // Остальные ошибки

{

qDebug() << "Error while opening";

ui->statusBar->showMessage("Cant open file");

QMessageBox::warning(this, "Warning", "Cant open file");

return;

}

try // Чтение в лист

{

std::cout << file\_json << std::endl;

//Если список не пустой, чистим

if (AuthorList.GetCount() != 0)

AuthorList.Clear();

if (DiskList.GetCount() != 0)

DiskList.Clear();

if (SongList.GetCount() != 0)

SongList.Clear();

//Обновление полей

refresh();

refreshAuthor();

refreshDisk();

refreshSong();

AuthorEditActivated(false);

DiskEditActivated(false);

SongEditActivated(false);

try // Чтение Авторов

{

std::cout << "Author reading..." << std::endl;

Process::readAuthors(file\_json, AuthorList);

std::cout << "Author reading completed." << std::endl;

}

catch (...) // Ошибка чтения в лист

{

qDebug() << "Cant read Authors";

ui->statusBar->showMessage("Cant read Authors");

QMessageBox::warning(this, "Warning", "Cant read Authors");

}

try // Чтение Дисков

{

std::cout << "Disk reading..." << std::endl;

Process::readDisks(file\_json, DiskList);

std::cout << "Disk reading completed." << std::endl;

}

catch (...) // Ошибка чтения в лист

{

qDebug() << "Cant read Disks";

ui->statusBar->showMessage("Cant read Disks");

QMessageBox::warning(this, "Warning", "Cant read Disks");

}

try // Чтение Песен

{

std::cout << "Song reading..." << std::endl;

Process::readSongs(file\_json, SongList);

std::cout << "Song reading completed." << std::endl;

}

catch (...) // Ошибка чтения в лист

{

qDebug() << "Cant read Songs";

ui->statusBar->showMessage("Cant read Songs");

QMessageBox::warning(this, "Warning", "Cant read Songs");

}

qDebug() << "Opening completed";

ui->statusBar->showMessage("Opening completed");

DataChanged = false;

MainWindow::refresh();

}

catch (const nlohmann::detail::type\_error &e) // Ошибка типизации

{

qDebug() << e.what();

ui->statusBar->showMessage(e.what());

QMessageBox::warning(this, "Warning", "Error while converting from json");

return;

}

catch (...) // Остальные ошибки

{

qDebug() << "The file is damaged";

ui->statusBar->showMessage("The file is damaged");

QMessageBox::warning(this, "Warning", "The file is damaged");

return;

}

fileNameDefaultOutput = fileNameInput;

}

else

{

qDebug() << "Opening aborted.";

ui->statusBar->showMessage("Opening aborted");

}

}

void MainWindow::saving(const QString fileNameOutput) // Сохранение

{

if (fileNameOutput != "") // На случай, если подаётся пустое имя файла

{

nlohmann::ordered\_json output\_file\_json{};

try

{

ui->statusBar->showMessage("Saving as \"" + fileNameOutput + "\"...");

qDebug() << "Saving as " << fileNameOutput;

std::cout << "Author writing..." << std::endl;

Process::writeAuthors(output\_file\_json, AuthorList);

std::cout << "Author writing completed." << std::endl;

std::cout << "Disk writing..." << std::endl;

Process::writeDisks(output\_file\_json, DiskList);

std::cout << "Disk writing completed." << std::endl;

std::cout << "Song writing..." << std::endl;

Process::writeSongs(output\_file\_json, SongList);

std::cout << "Song writing completed." << std::endl;

}

catch (...)

{

qDebug() << "Error while converting to json";

ui->statusBar->showMessage("Error while converting to json");

QMessageBox::warning(this, "Warning", "Error while converting to json");

return;

}

try

{

//Вывод в файл

std::string Output = fileNameOutput.toStdString();

Process::jsonToFile(output\_file\_json, Output);

DataChanged = false;

ui->statusBar->showMessage("Saved");

qDebug() << "Saved";

fileNameDefaultOutput = fileNameOutput;

}

catch (...)

{

qDebug() << "Error while saving";

ui->statusBar->showMessage("Error while saving");

QMessageBox::warning(this, "Warning", "Error while saving");

return;

}

}

else

{

qDebug() << "Saving aborted";

ui->statusBar->showMessage("Saving aborted");

}

}

void MainWindow::on\_actionSave\_As\_triggered() // Сохранить в произвольное место

{

fileNameOutput = "";

fileNameOutput = QFileDialog::getSaveFileName(this, tr("Save base as"), "SPMFD.json", tr("Image Files (\*.json)"));

saving(fileNameOutput);

}

void MainWindow::on\_actionSave\_triggered() // Сохранить в исходный файл

{

if (fileNameDefaultOutput != "")

{

saving(fileNameDefaultOutput);

}

else // В случе, если нету стандартного файла для записи

{

on\_actionSave\_As\_triggered();

}

}

void MainWindow::on\_actionExit\_triggered() // Выход

{

QMessageBox::StandardButton reply;

if (DataChanged) // На случай, если изменения не сохранены

{

reply = QMessageBox::question(this, "Exit", "Do you really want to Quit? Unsaved data will be lost.", QMessageBox::Yes | QMessageBox::No);

if (reply != QMessageBox::Yes)

{

qDebug() << "Quit aborted";

ui->statusBar->showMessage("Quit aborted");

return;

}

}

ui->statusBar->showMessage("Quit");

qDebug() << "Quit";

QApplication::quit();

}

//Вкладка Авторы

void MainWindow::AuthorGoToEdit(int index) // Переход к изменению автора

{

OnLoad = true;

ui->listWidgetAuthorSongList->clear();

at = AuthorList.Move(index);

eat = at;

std::cout << "Author: Moving to " << index << std::endl;

ui->statusBar->showMessage("Author: Moving to " + QString::number(index));

// Запись данных автора в переменные

AuthorId = at->data.GetId();

AuthorName = at->data.GetName();

AuthorListIdSong = at->data.GetSongs();

ui->lineEditAuthorId->setText(QString::number(AuthorId));

ui->lineEditAuthorName->setText(QString::fromUtf8((AuthorName).data(), (AuthorName).size()));

if (SongList.GetCount() != 0)

{

st = SongList.GetFirst();

QString currentSongName;

while (st) // Вывод списка песен автора

{

currentSongName = QString::fromUtf8((st->data.GetName()).data(), (st->data.GetName()).size());

QListWidgetItem \*item = new QListWidgetItem(currentSongName);

ui->listWidgetAuthorSongList->addItem(item);

item->setHidden(true);

for (int i = 0; i < AuthorListIdSong.size(); i++)

{

if (st->data.GetId() == AuthorListIdSong[i])

{

item->setHidden(false); // Видимы лишь те, что принадлежат автору

break;

}

}

st = st->next;

}

}

OnLoad = false;

AuthorEditActivated(true);

}

void MainWindow::on\_listWidgetAuthorGoTo\_doubleClicked(const QModelIndex &index) // Переход на позицию по двойному щелчку

{

AuthorGoToEdit(index.row());

}

void MainWindow::on\_pushButtonAuthorGoTo\_clicked() // Кнопка перейти на

{

if (ui->listWidgetAuthorGoTo->currentIndex().row() != -1)

on\_listWidgetAuthorGoTo\_doubleClicked(ui->listWidgetAuthorGoTo->currentIndex());

}

void MainWindow::on\_lineEditAuthorSearch\_textChanged(const QString &arg1) // Быстрый поиск

{

for (int i = 0; i < ui->listWidgetAuthorGoTo->count(); i++)

{

QListWidgetItem \*item = ui->listWidgetAuthorGoTo->item(i);

if (not(item->text().contains(ui->lineEditAuthorSearch->text())))

item->setHidden(true);

else

item->setHidden(false);

}

}

void MainWindow::on\_listWidgetAuthorSongList\_doubleClicked(const QModelIndex &index) // Переход из внутреннего листа

{

SongGoToEdit(index.row());

ui->tabWidget->setCurrentWidget(ui->tabWidget->widget(2));

}

void MainWindow::on\_pushButtonAuthorEditSong\_clicked() // Кнопка перехода из внутреннего списка

{

if (ui->listWidgetAuthorSongList->currentIndex().row() != -1)

{

SongGoToEdit(ui->listWidgetAuthorSongList->currentIndex().row());

ui->tabWidget->setCurrentWidget(ui->tabWidget->widget(2));

}

}

//Вкладка Диски

void MainWindow::DiskGoToEdit(int index) // Переход к изменению диска

{

OnLoad = true;

ui->listWidgetDiskSongList->clear();

dt = DiskList.Move(index);

edt = dt;

std::cout << "Disk: Moving to " << index << std::endl;

ui->statusBar->showMessage("Disk: Moving to " + QString::number(index));

// Запись данных диска в переменные

DiskId = dt->data.GetId();

DiskName = dt->data.GetName();

DiskListIdSong = dt->data.GetSongs();

ui->lineEditDiskId->setText(QString::number(DiskId));

ui->lineEditDiskName->setText(QString::fromUtf8((DiskName).data(), (DiskName).size()));

if (SongList.GetCount() != 0)

{

st = SongList.GetFirst();

QString currentSongName;

while (st) // Вывод списка песен диска

{

currentSongName = QString::fromUtf8((st->data.GetName()).data(), (st->data.GetName()).size());

QListWidgetItem \*item = new QListWidgetItem(currentSongName);

ui->listWidgetDiskSongList->addItem(item);

item->setHidden(true);

for (int i = 0; i < DiskListIdSong.size(); i++)

{

if (st->data.GetId() == DiskListIdSong[i])

{

item->setHidden(false); // Видимы лишь те, что принадлежат диску

break;

}

}

st = st->next;

}

}

OnLoad = false;

DiskEditActivated(true);

}

void MainWindow::on\_listWidgetDiskGoTo\_doubleClicked(const QModelIndex &index) // Переход на позицию по двойному щелчку

{

DiskGoToEdit(index.row());

}

void MainWindow::on\_pushButtonDiskGoTo\_clicked() // Кнопка перейти на

{

if (ui->listWidgetDiskGoTo->currentIndex().row() != -1)

DiskGoToEdit(ui->listWidgetDiskGoTo->currentIndex().row());

}

void MainWindow::on\_lineEditDiskSearch\_textChanged(const QString &arg1) // Быстрый поиск

{

for (int i = 0; i < ui->listWidgetDiskGoTo->count(); i++)

{

QListWidgetItem \*item = ui->listWidgetDiskGoTo->item(i);

if (not(item->text().contains(ui->lineEditDiskSearch->text())))

item->setHidden(true);

else

item->setHidden(false);

}

}

void MainWindow::on\_listWidgetDiskSongList\_doubleClicked(const QModelIndex &index) // Переход из внутреннего листа

{

SongGoToEdit(index.row());

ui->tabWidget->setCurrentWidget(ui->tabWidget->widget(2));

}

void MainWindow::on\_pushButtonDiskEditSong\_clicked() // Кнопка перехода из внутреннего списка

{

if (ui->listWidgetDiskSongList->currentIndex().row() != -1)

{

SongGoToEdit(ui->listWidgetDiskSongList->currentIndex().row());

ui->tabWidget->setCurrentWidget(ui->tabWidget->widget(2));

}

}

//Вкладка Песни

void MainWindow::SongGoToEdit(int index) // Перейти на песню для изменения

{

OnLoad = true;

ui->listWidgetSongAuthorList->clear();

ui->listWidgetSongDiskList->clear();

st = SongList.Move(index);

est = st;

std::cout << "Song: Moving to " << index << std::endl;

ui->statusBar->showMessage("Song: Moving to " + QString::number(index));

// Запись данных песни в переменные

SongId = st->data.GetId();

SongName = st->data.GetName();

ListIdAuthor = st->data.GetAuthors();

ListIdDisk = st->data.GetDisks();

currentSong.ChangeSong(SongId, SongName, ListIdAuthor, ListIdDisk);

ui->lineEditSongId->setText(QString::number(SongId));

ui->lineEditSongName->setText(QString::fromUtf8((SongName).data(), (SongName).size()));

// Обновление полей Авторов

QString currentAuthorName;

if (AuthorList.GetCount() != 0)

{

at = AuthorList.GetFirst();

while (at)

{

currentAuthorName = QString::fromUtf8((at->data.GetName()).data(), (at->data.GetName()).size());

QListWidgetItem \*item = new QListWidgetItem(currentAuthorName);

ui->listWidgetSongAuthorList->addItem(item);

item->setHidden(true);

for (int i = 0; i < ListIdAuthor.size(); i++)

{

if (at->data.GetId() == ListIdAuthor[i])

{

item->setHidden(false); // Видимы лишь те, что принадлежат песни

break;

}

}

at = at->next;

}

}

// Обновление полей Дисков

QString currentDiskName;

if (DiskList.GetCount() != 0)

{

dt = DiskList.GetFirst();

while (dt)

{

currentDiskName = QString::fromUtf8((dt->data.GetName()).data(), (dt->data.GetName()).size());

QListWidgetItem \*item = new QListWidgetItem(currentDiskName);

ui->listWidgetSongDiskList->addItem(item);

item->setHidden(true);

for (int i = 0; i < ListIdDisk.size(); i++)

{

if (dt->data.GetId() == ListIdDisk[i])

{

item->setHidden(false); // Видимы лишь те, что принадлежат песне

break;

}

}

dt = dt->next;

}

}

OnLoad = false;

SongEditActivated(true);

}

void MainWindow::on\_listWidgetSongGoTo\_doubleClicked(const QModelIndex &index) // Переход на позицию

{

SongGoToEdit(index.row());

}

void MainWindow::on\_pushButtonSongGoTo\_clicked() // Кнопка перейти на

{

if (ui->listWidgetSongGoTo->currentIndex().row() != -1)

SongGoToEdit(ui->listWidgetSongGoTo->currentIndex().row());

}

void MainWindow::on\_lineEditSongSearch\_textChanged(const QString &arg1) // Быстрый поиск

{

for (int i = 0; i < ui->listWidgetSongGoTo->count(); i++)

{

QListWidgetItem \*item = ui->listWidgetSongGoTo->item(i);

if (not(item->text().contains(ui->lineEditSongSearch->text())))

item->setHidden(true);

else

item->setHidden(false);

}

}

void MainWindow::on\_listWidgetSongAuthorList\_doubleClicked(const QModelIndex &index) // Переход из внутреннего списка авторов

{

if (not SongEdited)

{

AuthorGoToEdit(index.row());

ui->tabWidget->setCurrentWidget(ui->tabWidget->widget(0));

}

}

void MainWindow::on\_pushButtonSongEditAuthor\_clicked() // Кнопка перехода из внутреннего списка авторов

{

if (ui->listWidgetSongAuthorList->currentIndex().row() != -1)

{

AuthorGoToEdit(ui->listWidgetSongAuthorList->currentIndex().row());

ui->tabWidget->setCurrentWidget(ui->tabWidget->widget(0));

}

}

void MainWindow::on\_listWidgetSongDiskList\_doubleClicked(const QModelIndex &index) // Переход из внутреннего списка дисков

{

if (not SongEdited)

{

DiskGoToEdit(index.row());

ui->tabWidget->setCurrentWidget(ui->tabWidget->widget(1));

}

}

void MainWindow::on\_pushButtonSongEditDisk\_clicked() // Кнопка перехода из внутреннего списка дисков

{

if (ui->listWidgetSongDiskList->currentIndex().row() != -1)

{

DiskGoToEdit(ui->listWidgetSongDiskList->currentIndex().row());

ui->tabWidget->setCurrentWidget(ui->tabWidget->widget(1));

}

}

// Изменение данных автора

void MainWindow::on\_lineEditAuthorName\_textChanged(const QString &arg1) // Переключение в режим изменения, если изменилось название

{

if (not OnLoad)

AuthorOnEdit(true);

}

void MainWindow::on\_pushButtonAuthorSave\_clicked() // Сохранение изменений

{

if (AuthorEdited = true)

{

std::string newAuthorName = ui->lineEditAuthorName->text().toStdString();

if (newAuthorName == "")

QMessageBox::warning(this, "Warning", "Fill the field Author Name");

else

{

eat->data.SetName(newAuthorName); // Установка новго имени

OnLoad = true;

refreshDisk();

refreshSong();

refresh();

AuthorOnEdit(false);

AuthorEditActivated(true);

DiskEditActivated(false);

SongEditActivated(false);

OnLoad = false;

DataChanged = true;

qDebug() << "Author: saved";

ui->statusBar->showMessage("Author: saved");

}

}

}

void MainWindow::on\_pushButtonAuthorDiscard\_clicked() // Отмена изменений

{

refreshAuthor();

AuthorOnEdit(false);

qDebug() << "Author: edit discarded";

ui->statusBar->showMessage("Author: edit discarded");

}

void MainWindow::on\_pushButtonAuthorNew\_clicked() // Создание нового автора

{

if (AuthorList.GetCount() == 0) // Если элементов нет, устанавливаем первый айди

AuthorId = 1;

else // Установка следующего, после последнего id

AuthorId = AuthorList.GetLast()->data.GetId() + 1;

AuthorName = "Author" + std::to\_string(AuthorId);

AuthorListIdSong = {};

currentAuthor.ChangeAuthor(AuthorId, AuthorName, AuthorListIdSong);

AuthorList.AddEnd(currentAuthor);

refresh(); // Обновление полей

refreshAuthor();

refreshDisk();

refreshSong();

DiskEditActivated(false);

SongEditActivated(false);

DataChanged = true;

AuthorGoToEdit(AuthorList.GetCount() - 1);

qDebug() << "Author: Author" << AuthorId << " added";

ui->statusBar->showMessage("Author: Author" + QString::number(AuthorId) + " added");

}

void MainWindow::on\_pushButtonAuthorDelete\_clicked() // Удаление текущего автора

{

AuthorName = eat->data.GetName();

Process::deleteLinks(eat, SongList);

AuthorList.Delete(eat);

// Обновление виджетов

refresh();

refreshAuthor();

refreshDisk();

refreshSong();

AuthorOnEdit(false);

DiskOnEdit(false);

SongOnEdit(false);

DataChanged = true;

qDebug() << "Author: " << QString::fromUtf8(AuthorName.data(), AuthorName.size()) << " deleted";

ui->statusBar->showMessage("Author: " + QString::fromUtf8(AuthorName.data(), AuthorName.size()) + " deleted");

}

// Изменение данных диска

void MainWindow::on\_lineEditDiskName\_textChanged(const QString &arg1) // Переключение в режим изменения, если изменилось название

{

if (not OnLoad)

DiskOnEdit(true);

}

void MainWindow::on\_pushButtonDiskSave\_clicked() // Сохранение изменений

{

if (DiskEdited = true)

{

std::string newDiskName = ui->lineEditDiskName->text().toStdString();

if (newDiskName == "")

QMessageBox::warning(this, "Warning", "Fill the field Disk Name");

else

{

edt->data.SetName(newDiskName);

OnLoad = true;

refreshAuthor();

refreshSong();

refresh();

DiskOnEdit(false);

DiskEditActivated(true);

AuthorEditActivated(false);

SongEditActivated(false);

DataChanged = true;

qDebug() << "Disk: saved";

ui->statusBar->showMessage("Disk: saved");

}

}

}

void MainWindow::on\_pushButtonDiskDiscard\_clicked() // Отмена изменений

{

refreshDisk();

DiskOnEdit(false);

qDebug() << "Disk: edit discarded";

ui->statusBar->showMessage("Disk: edit discarded");

}

void MainWindow::on\_pushButtonDiskNew\_clicked() // Создание нового диска

{

if (DiskList.GetCount() == 0) // Если элементов нет, устанавливаем первый айди

DiskId = 1;

else // Установка следующего, после последнего id

DiskId = DiskList.GetLast()->data.GetId() + 1;

DiskName = "Disk" + std::to\_string(DiskId);

DiskListIdSong = {};

currentDisk.ChangeDisk(DiskId, DiskName, DiskListIdSong);

DiskList.AddEnd(currentDisk);

refresh(); // Обновление полей

refreshAuthor();

refreshDisk();

refreshSong();

AuthorEditActivated(false);

SongEditActivated(false);

DataChanged = true;

DiskGoToEdit(DiskList.GetCount() - 1);

qDebug() << "Disk: Disk" << DiskId << " added";

ui->statusBar->showMessage("Disk: Disk" + QString::number(DiskId) + " added");

}

void MainWindow::on\_pushButtonDiskDelete\_clicked() // Удаление текущего диска

{

DiskName = edt->data.GetName();

Process::deleteLinks(edt, SongList);

DiskList.Delete(edt);

refresh(); // Обновление полей.

refreshAuthor();

refreshDisk();

refreshSong();

AuthorOnEdit(false);

DiskOnEdit(false);

SongOnEdit(false);

DataChanged = true;

qDebug() << "Disk: " << QString::fromUtf8(DiskName.data(), DiskName.size()) << " deleted";

ui->statusBar->showMessage("Disk: " + QString::fromUtf8(DiskName.data(), DiskName.size()) + " deleted");

}

// Изменение данных песни

void MainWindow::on\_lineEditSongName\_textChanged(const QString &arg1) // Переключение в режим изменения, если изменилось название

{

if (not OnLoad)

SongOnEdit(true);

}

void MainWindow::on\_pushButtonSongSave\_clicked() // Сохранение изменений

{

if (SongEdited = true)

{

std::string newSongName = ui->lineEditSongName->text().toStdString();

if (newSongName == "")

QMessageBox::warning(this, "Warning", "Fill the field Song Name");

else

{

// Обновление связей Песня-Автор

Process::updateLinksSongAuthors(est, currentSong.GetAuthors(), est->data.GetAuthors(), AuthorList);

// Обновление связей Песня-Диск

Process::updateLinksSongDisks(est, currentSong.GetDisks(), est->data.GetDisks(), DiskList);

est->data.SetName(newSongName);

est->data.SetAuthors(currentSong.GetAuthors());

est->data.SetDisks(currentSong.GetDisks());

refreshAuthor();

refreshDisk();

refresh();

SongOnEdit(false);

AuthorEditActivated(false);

DiskEditActivated(false);

SongEditActivated(true);

DataChanged = true;

qDebug() << "Song: saved";

ui->statusBar->showMessage("Song: saved");

}

}

}

void MainWindow::on\_pushButtonSongDiscard\_clicked() // Отмена изменений

{

refreshSong();

SongOnEdit(false);

qDebug() << "Song: edit discarded";

ui->statusBar->showMessage("Song: edit discarded");

}

void MainWindow::on\_pushButtonSongNew\_clicked() // Создание новой песни

{

if (SongList.GetCount() == 0) // Если элементов нет, устанавливаем первый айди

SongId = 1;

else // Установка следующего, после последнего id

SongId = SongList.GetLast()->data.GetId() + 1;

SongName = "Song" + std::to\_string(SongId);

ListIdAuthor = {};

ListIdDisk = {};

currentSong.ChangeSong(SongId, SongName, ListIdAuthor, ListIdDisk);

SongList.AddEnd(currentSong);

refresh(); // Обновление полей

refreshAuthor();

refreshDisk();

refreshSong();

AuthorOnEdit(false);

DiskOnEdit(false);

DataChanged = true;

SongGoToEdit(SongList.GetCount() - 1);

qDebug() << "Song: Song" << SongId << " added";

ui->statusBar->showMessage("Song: Song" + QString::number(SongId) + " added");

}

void MainWindow::on\_pushButtonSongDelete\_clicked() // Удаление текущей песни

{

SongName = est->data.GetName();

Process::deleteLinks(est, AuthorList, DiskList);

SongList.Delete(est);

// Обновление полей

refresh();

refreshAuthor();

refreshDisk();

refreshSong();

AuthorOnEdit(false);

DiskOnEdit(false);

SongOnEdit(false);

DataChanged = true;

qDebug() << "Song: " << QString::fromUtf8(SongName.data(), SongName.size()) << " deleted";

ui->statusBar->showMessage("Song: " + QString::fromUtf8(SongName.data(), SongName.size()) + " deleted");

}

void MainWindow::on\_pushButtonSongAddAuthor\_clicked() // Добавление связи Автор-Песня

{

if (ui->comboBoxSongAddAuthor->currentIndex() != -1)

{

SongOnEdit(true);

currentSong.AddAuthor(AuthorList.Move(ui->comboBoxSongAddAuthor->currentIndex())->data.GetId());

ui->listWidgetSongAuthorList->item(ui->comboBoxSongAddAuthor->currentIndex())->setHidden(false);

}

}

void MainWindow::on\_pushButtonSongAddDisk\_clicked() // Добавление связи Диск-Песня

{

if (ui->comboBoxSongAddDisk->currentIndex() != -1)

{

SongOnEdit(true);

currentSong.AddDisk(DiskList.Move(ui->comboBoxSongAddDisk->currentIndex())->data.GetId());

ui->listWidgetSongDiskList->item(ui->comboBoxSongAddDisk->currentIndex())->setHidden(false);

}

}

void MainWindow::on\_pushButtonSongDeleteAuthor\_clicked() // Удаление связи Автор-Песня

{

if (ui->listWidgetSongAuthorList->currentIndex().row() != -1)

{

SongOnEdit(true);

currentSong.RemoveAuthor(AuthorList.Move(ui->listWidgetSongAuthorList->currentIndex().row())->data.GetId());

ui->listWidgetSongAuthorList->item(ui->listWidgetSongAuthorList->currentIndex().row())->setHidden(true);

}

}

void MainWindow::on\_pushButtonSongDeleteDisk\_clicked() // Удаление связи Диск-Песня

{

if (ui->listWidgetSongDiskList->currentIndex().row() != -1)

{

SongOnEdit(true);

currentSong.RemoveDisk(DiskList.Move(ui->listWidgetSongDiskList->currentIndex().row())->data.GetId());

ui->listWidgetSongDiskList->item(ui->listWidgetSongDiskList->currentIndex().row())->setHidden(true);

}

}

void MainWindow::on\_actionNew\_File\_triggered() // Создание нового проекта

{

qDebug() << "Creating new base..";

ui->statusBar->showMessage("Creating new base...");

if (DataChanged) // Если что-то в списках изменилось

{

QMessageBox::StandardButton reply;

reply = QMessageBox::question(this, "Open", "Do you really want create new file? Unsaved data will be lost.", QMessageBox::Yes | QMessageBox::No);

if (reply != QMessageBox::Yes)

{

qDebug() << "Creating aborted.";

ui->statusBar->showMessage("Creating aborted");

return;

}

}

//Если список не пустой, чистим

if (AuthorList.GetCount() != 0)

AuthorList.Clear();

if (DiskList.GetCount() != 0)

DiskList.Clear();

if (SongList.GetCount() != 0)

SongList.Clear();

//Обновление полей

refresh();

refreshAuthor();

refreshDisk();

refreshSong();

AuthorEditActivated(false);

DiskEditActivated(false);

SongEditActivated(false);

DataChanged = false;

qDebug() << "New base created";

ui->statusBar->showMessage("New base created");

}