МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

КУРСОВОЙ ПРОЕКТ  
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ст. преподаватель |  |  |  | Е. О. Шумова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ |
| Разработка приложения для организации взаимодействия объектов при заданных критериях |
| по дисциплине: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. | 4936 |  |  |  | С. В. Молчанов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

**Задание**

**на курсовой проект по дисциплине**

**«Объектно-ориентированное программирование»**

|  |  |  |  |
| --- | --- | --- | --- |
| СТУДЕНТУ | 4936 |  | С. В. Молчанов |
|  | № группы |  | инициалы, фамилия |

Тема «Разработка приложения для организации взаимодействия объектов при заданных критериях»

Исходные данные: «Система классов для работы каталога комиксов»

Проект должен содержать:

* анализ предметной области
* разработку классов
* разработку тестового приложения
* оформление пояснительной записки по результатам выполнения проекта
* создание презентации к проекту

|  |  |  |
| --- | --- | --- |
| Срок сдачи законченного проекта |  |  |
| Руководитель проекта | ст. преп. Е. О. Шумова |  |
|  |  |  |

Дата выдачи задания 01.09.2021 г.

Оглавление

[**Задание на курсовой проект** 4](#_Toc91076003)

[**Введение** 4](#_Toc91076004)

[**1.** **Постановка задачи** 5](#_Toc91076005)

[**1.1.** **Анализ предметной области** 5](#_Toc91076006)

[**1.2.** **Формулировка технического задания** 6](#_Toc91076007)

[**2.** **Проектирование классов** 7](#_Toc91076008)

[**2.1.** **Разработка классов** 7](#_Toc91076009)

[**2.2.** **Диаграмма классов** 9](#_Toc91076010)

[**3.** **Разработка приложения** 11](#_Toc91076011)

[**3.1.** **Разработка интерфейса приложения** 11](#_Toc91076012)

[**3.1.1.** **Вкладка «Регистрация и вход»** 11](#_Toc91076013)

[**3.1.2 Вкладка «Основное окно»** 12](#_Toc91076014)

[**3.1.3 Вкладка «Каталог»** 13](#_Toc91076015)

[**3.1.4 Вкладка «Функционал администратора»** 14](#_Toc91076016)

[**3.2.** **Реализация классов** 18](#_Toc91076017)

[**3.3.** **Использованные паттерны** 21](#_Toc91076018)

[**4.** **Тестирование** 23](#_Toc91076019)

[**Заключение** 32](#_Toc91076020)

[**Список использованных источников** 33](#_Toc91076021)

[**Приложение А** 34](#_Toc91076022)

### **Задание на курсовой проект**

В ходе курсового проекта необходимо разработать приложение, позволяющее протестировать взаимодействие объектов классов, спроектированных и реализованных студентом для решения конкретной задачи при заданных критериях.

При разработке программного обеспечения следует использовать шаблоны проектирования. Также предпочтение должно быть отдано графическому приложению.

Вариант Собственный (согласованный)

Разработка системы классов «Система классов для работы каталога комиксов»

### **Введение**

Объектно-ориентированное, или объектное, программирование (ООП) - парадигма программирования, в которой основными концепциями являются понятия объектов и классов. Данная концепция в настоящее время является ведущей в области программирования. Количество прикладных языков программирования, реализующих объектно-ориентированную парадигму, является наибольшим по отношению к другим парадигмам. Данный подход реализован в таких языках программирования как С, C++, Object Pascal, Java, С# и другие.

Информационная система — это организационно-техническая система, предназначенная для выполнения информационно-вычислительных работ или предоставления информационно-вычислительных услуг, удовлетворяющих потребности системы управления и ее пользователей - управленческого персонала, внешних пользователей.

В данном случае, была разработана автоматизированная информационная система «Для комиксов» на языке программирования С++ с использованием базы данных (MySQL). Эта простая программа обеспечивает стабильную работу системы, добавление и удаление комиксов, учёта комиксов по определённым параметрам.

### **Постановка задачи**

### **Анализ предметной области**

«Работа каталога комиксов» можно расценивать как работу сервиса в организации структуры комиксов. Данная система выполняет функции организации каталога, а также базы данных пользователей.

Словарь объектов предметной области:

* Комикс;
* Жанр;
* Автор;
* Издательство;

Субъектами предметной области являются:

* администратор, редактирующий каталог и его содержимое;
* авторизованные пользователи, просматривающие содержание каталога и использующие фильтры для выбора конкретных комиксов.

Основным функциональным требованием к разрабатываемому приложению является наличие графического пользовательского интерфейса. Второстепенной задачей является добавление возможности работы с базой данных MySQL. Главным критерием к графическому интерфейсу является простота в понимании и низкий порог вхождения для новых пользователей.

### **Формулировка технического задания**

В результате анализа предметной области можно сформулировать следующие функциональные требования:

Программа должна быть реализована на языке С++ в виде приложения с графическим интерфейсом. Должно быть реализовано меню пользователя в виде кнопок, по нажатию на которые пользователь будет переходить в категории: «Регистрация», «Вход», «Каталог», «Функции администратора» и «Фильтрация». Реализованы следующие функции для работы с комиксами: добавление, удаление записи, вывод на экран.

**Для категории «Комикс» должны быть реализованы следующие функции:**

* Добавление нового комикса
* Удаление комикса
* Вывод на экран всех комиксов
* Вывод на экран конкретных комиксов

**Для категории «Жанры» должны быть реализованы следующие функции:**

* Добавление нового жанра
* Вывод на экран комиксов жанра

**Для категории «Авторы» должны быть реализованы следующие функции:**

* Добавление нового автора
* Вывод на экран комиксов автора

**Для категории «Издательства» должны быть реализованы следующие функции:**

* Добавление нового издательства
* Вывод на экран комиксов издательства

**Для категории «Пользователи» должны быть реализованы следующие функции:**

* Добавление нового пользователя
* Вход в систему
* Просмотр списка пользователей (для администратора)

### **Проектирование классов**

### **Разработка классов**

Классы сущностей:

comix – класс сущности «комикс». Основное назначения класса хранить информацию сущности предметной области «комикс». Основные поля класса:

* Название комикса
* Число частей
* Автор комикса
* Жанр комикса
* Статус выхода комикса
* Перевод комикса
* Издательство комикса
* Описание комикса

author – класс сущности «автор». Основное назначения класса хранить информацию сущности предметной области «автор». Основные поля класса:

* Список комиксов автора

genre – класс сущности «жанр». Основное назначения класса хранить информацию сущности предметной области «жанр». Основные поля класса:

* Описание
* Список комиксов жанра

publisher – класс сущности «издательство». Основное назначения класса хранить информацию сущности предметной области «издательство». Основные поля класса:

* Список комиксов издательства

user – класс сущности «пользователь». Основное назначения класса хранить информацию сущности предметной области «пользователь». Основные поля класса:

* Логин
* Пароль
* E-mail

Интерфейсные классы:

MainWindow – класс главного окна. Класс отвечает за подключение БД к проекту.

Admin\_tools – класс для работы с функционалом администратора. Класс предназначен для отправки запросов в базу данных на выборку, вставку, удаление. Основные поля:

* Объекты класса QSqlQuery– связь с БД

catalog – класс для работы с каталогом. Содержит основную информацию по комиксам. Основные поля:

* Таблица комиксов
* Поле для фильтрации комиксов.

Reg\_log – класс для работы с регистрацией и входом. Выполняет сохранение новых пользователей, а также проверку данных во время входа.

Основные поля:

* Регистрация
* Вход.

Классы представления:

admin – класс, представляющий данные об администраторе системы. Наследуется от класса «user». Используется паттерн «одиночки».

### **Диаграмма классов**

На основе анализа предметной области и спроектированных классов была построена следующая диаграмма классов:

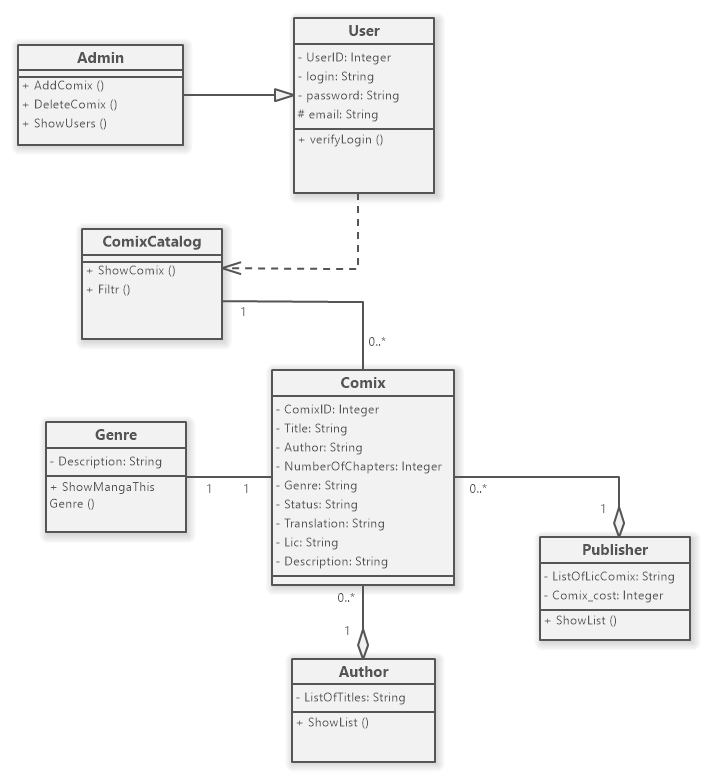


Рисунок 2.2.1

* 1. **Диаграмма базы данных**

На основе анализа предметной области и спроектированных классов была построена следующая диаграмма сущностей для базы данных:

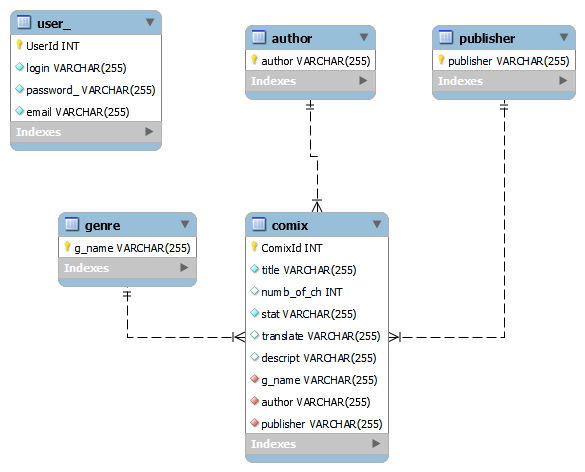


Рисунок 2.3.1

Данная диаграмма использована в реализации для тестового приложения. При возникновении необходимости подключения к базе данных с другой структурой таблиц придется переписать большую часть классов.

### **Разработка приложения**

### **Разработка интерфейса приложения**

### **Вкладка «Регистрация и вход»**

В окне «Регистрация и вход» должны быть поля для ввода данных для создания новых объектов классов user.

В левой части окна расположены основные управляющие элементы регистрации, разделенные на блоки. Каждое поле ввода предназначено для ввода данных об новом пользователе.

В поле «Логин» вводится логин пользователя.

В поле «Пароль» вводится пароль пользователя.

В поле «E-mail» вводится email пользователя.

Кнопка «Регистрация» вызывает функцию добавления нового пользователя.

В правой части вкладки расположены основные элементы для входа в систему. Каждое поле ввода предназначено для ввода данных об пользователе, существующем в системе.

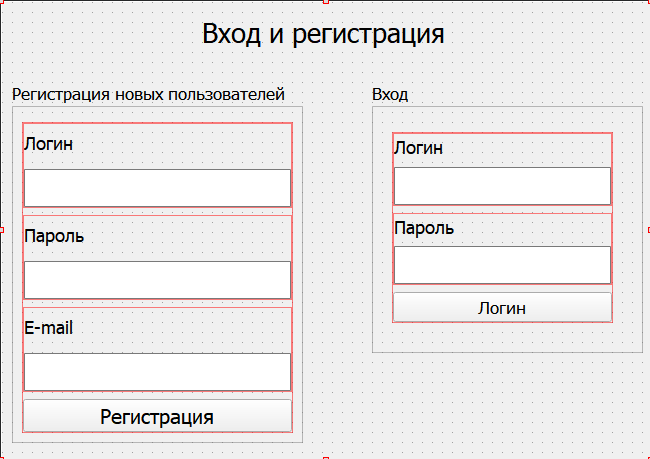


Рис. 3.1.1.1 Интерфейс окна «Регистрация и вход»

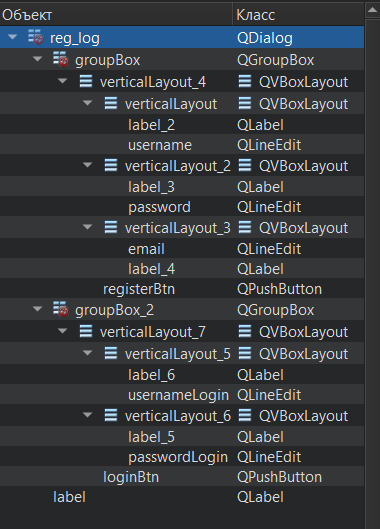


Рис. 3.1.1.2 Иерархия классов интерфейса окна «Регистрация и вход»

### **3.1.2 Вкладка «Основное окно»**

В окне «Основное окно» расположено поле, отображающее логин текущего авторизованного пользователя.

В нижней части окна расположены основные управляющие элементы. Под полем вывода логина располагаются кнопки для функционала администратора и вывода каталога комиксов.

Кнопка «Функции администратора» вызывает окно функционала администратора поверх текущего окна.

Кнопка «Вывод каталога комиксов» вызывает окно каталога комиксов поверх текущего окна.

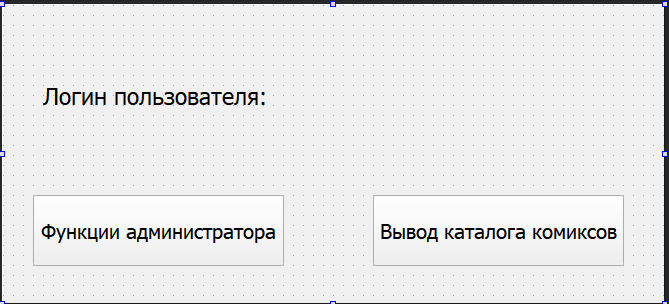


Рис. 3.1.2.1 Интерфейс окна «Основное окно»

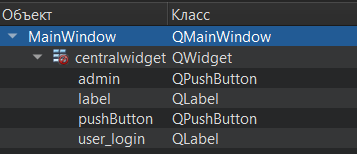


Рис. 3.1.2.2 Иерархия классов интерфейса окна «Основное окно»

### **3.1.3 Вкладка «Каталог»**

В окне «Каталог» содержится таблица комиксов и элементы фильтрации таблицы.

В нижней части окна расположены основные управляющие элементы.

Всплывающий список «Фильтрация каталога» необходим для выбора параметра сущности, по которому будет происходить фильтрация каталога.

Текстовое поле принимает ключ, по которому будут отобраны комиксы по выбранному фильтру.

Кнопка «Фильтр» вызывает функцию отбора комиксов по ключу в выбранном режиме фильтрации.

В верхней части вкладки расположена таблица, в которой выводится вся информация о комиксах.

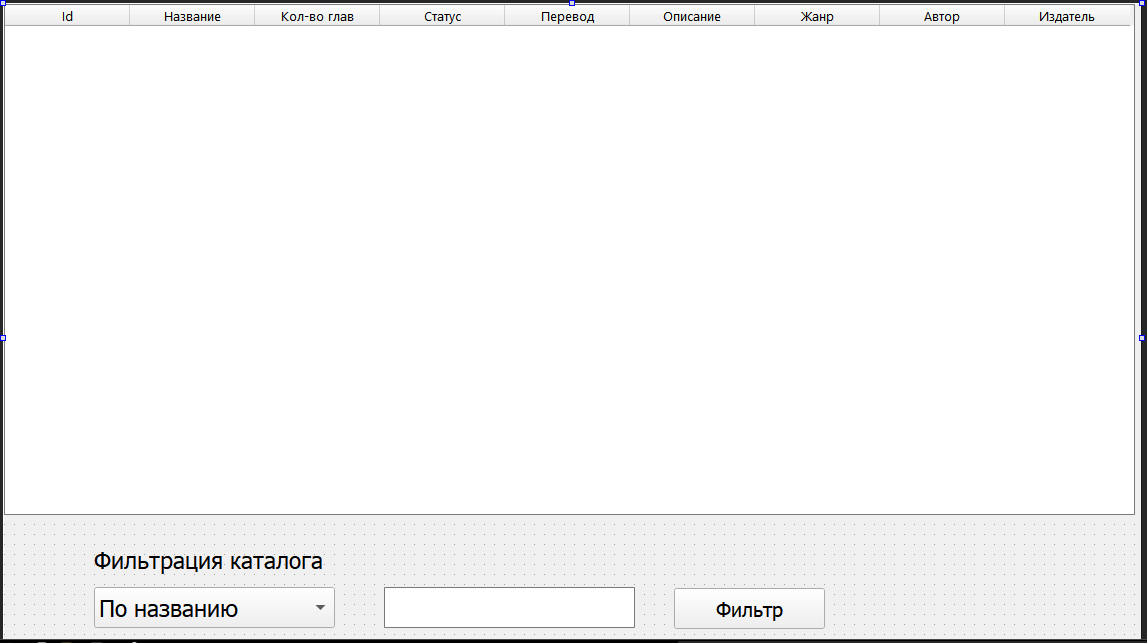


Рис. 3.1.3.1 Интерфейс окна «Каталог»

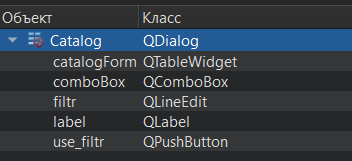


Рис. 3.1.3.2 Иерархия классов интерфейса окна «Каталог»

### **3.1.4 Вкладка «Функционал администратора»**

В окне «Функционал администратора» содержится три разные вкладки, отвечающие за разные функции администратора.

В верхней части окна расположены различные вкладки, содержащие основные управляющие элементы.

Вкладка «Вставка» содержит поля для ввода данных о комиксе и кнопку «Добавить», которая вызывает функцию добавления комикса.

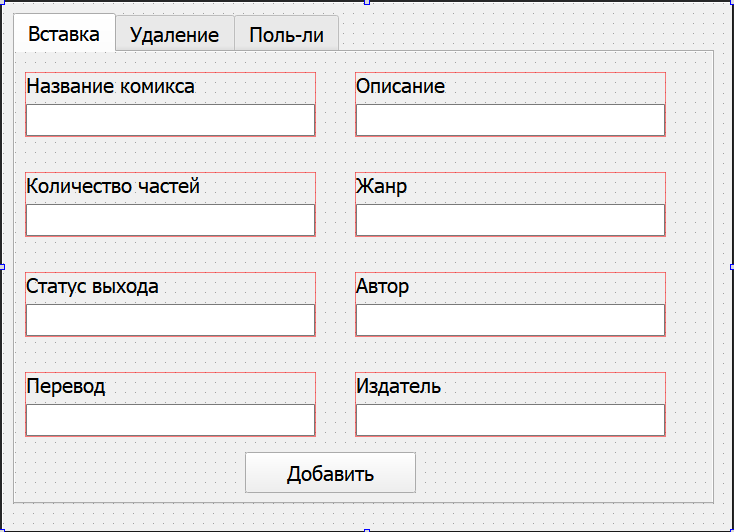


Рис. 3.1.4.1 Интерфейс вкладки «Вставка»

Вкладка «Удаление» содержит таблицу комиксов (для лучшего отображения показываются только id и название), поле для ввода id удаляемого комикса, а также кнопки «Обновление таблицы» для отображения текущих комиксов в каталоге и «Удалить комикс», которая вызывает функцию удаления комикса.

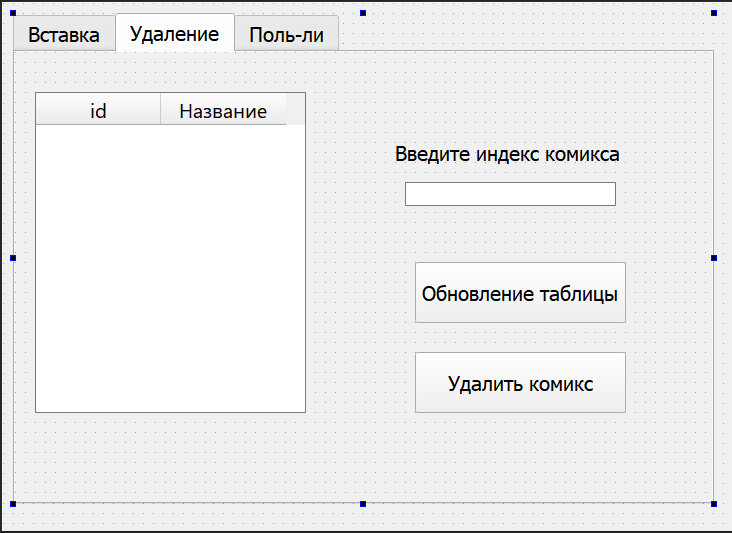


Рис. 3.1.4.2 Интерфейс вкладки «Удаление»

Вкладка «Пользователи» содержит таблицу всех пользователей, зарегистрированных в системе, а также кнопку «Вывод пользователей», которая обновляет таблицу, выводя всех пользователей.

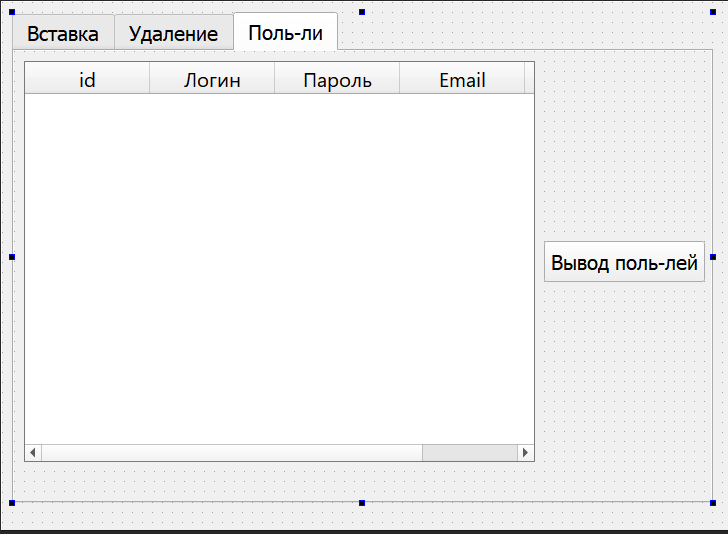
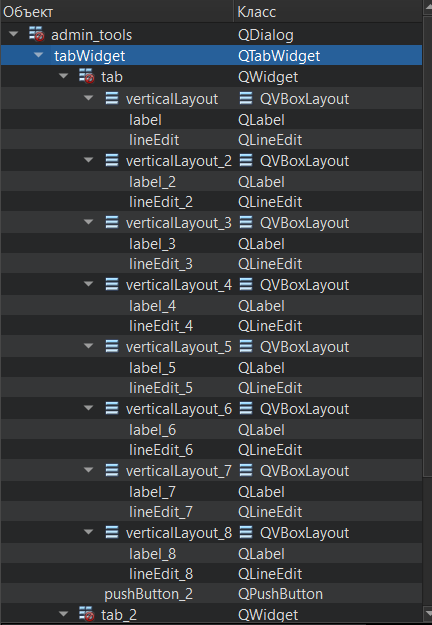


Рис. 3.1.4.3 Интерфейс вкладки «Пользователи»



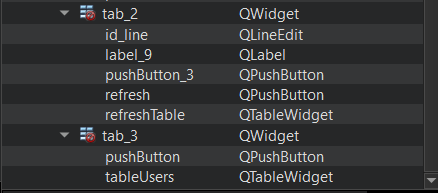


Рис. 3.1.4.4 Иерархия классов интерфейса окна «Функционал администратора»

### **Реализация классов**

* + 1. **Интерфейс класса сущности «Автор»:**

*class* **author**: *public* QObject

{

*public*:

*//конструкторы ...*

//сеттеры и геттеры *...*

*private*:

QVector<comix\*> TitleList;

};

* + 1. **Интерфейс класса сущности «Комикс»:**

*class* **comix**: *public* QObject

{

*public*:

*//конструкторы ...*

*//сеттеры и геттеры ...*

*//добавление и удаления объектов предметной области ...*

*private*:

int ComixId;

QString title;

QString author;

int numb\_of\_ch;

QString genre;

QString status;

QString translate;

QString lic;

QString description;

};

* + 1. **Реализация класса сущности «Жанр»:**

*class* **genre**: *public* QObject

{

*public*:

*//конструкторы ...*

//сеттеры и геттеры *...*

*//добавление и удаления объектов предметной области ...*

*private*:

QString descript;

};

* + 1. **Интерфейс класса сущности «Издатель»:**

*class* **publisher**: *public* QObject

{

*public*:

*//конструкторы ...*

*//сеттеры и геттеры ...*

*//добавление и удаления объектов предметной области ...*

*private*:

QVector<comix\*> ComList;

};

* + 1. **Интерфейс класса сущности «Пользователь»:**

*class* **user***: public* QObject

*{*

*public:*

*//конструкторы ...*

*//сеттеры и геттеры ...*

*//добавление и удаления объектов предметной области ...*

*private:*

QString login;

QString pass;

QString email;

*};*

* + 1. **Реализация класса представления «Администратор»:**

*class* **Admin***: public* user

*{*

*public:*

*//конструкторы ...*

//сеттеры и геттеры *...*

*//Функция* *возвращает* *указатель* *на* *одиночку*

*static* Admin\* **getAdmin**();

QString current\_login;

*private:*

*//Указатель* *на* *класс* *(одиночку)*

*static* Admin\* adm;*}*

*;*

### **Использованные паттерны**

* + 1. **Model-View-Controller**

MVC - Cхема разделения данных приложения, и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

Включает в себя три основные части:

* ***Модель*** (*Model*) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.
* ***Представление*** (*View*) отвечает за отображение данных модели пользователю, реагируя на изменения модели.
* ***Контроллер*** (*Controller*) интерпретирует действия пользователя, оповещая модель о необходимости изменений.



Рисунок 3.3.1

Основная цель применения этой концепции состоит в отделении бизнес-логики (*модели*) от её визуализации (*представления*, *вида*). За счёт такого разделения повышается возможность повторного использования кода. Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах и/или с различных точек зрения.

В данном случае паттерн используется для отображения модальных окон.

* + 1. **Singleton**

Одиночка — это порождающий паттерн, который гарантирует существование только одного объекта определённого класса, а также позволяет достучаться до этого объекта из любого места программы.

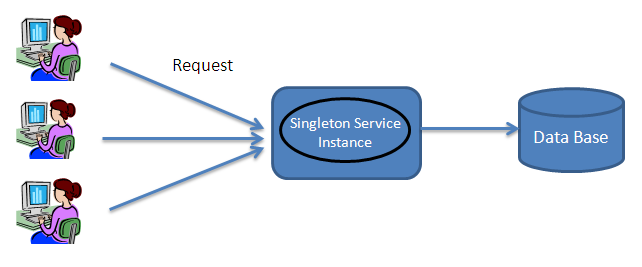


Рисунок 3.3.2

У класса есть только один экземпляр, и он предоставляет к нему глобальную точку доступа. При попытке создания данного объекта он создаётся только в том случае, если ещё не существует, в противном случае возвращается ссылка на уже существующий экземпляр и нового выделения памяти не происходит. Существенно то, что можно пользоваться именно *экземпляром* класса, так как при этом во многих случаях становится доступной более широкая функциональность. Например, к описанным компонентам класса можно обращаться через интерфейс, если такая возможность поддерживается языком.

### **Тестирование**

*Вкладка «Регистрация и вход»:*

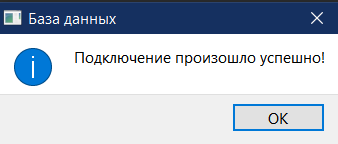


Рис. 4.1 Подтверждение подключения бд

* 1. **Добавление нового пользователя:**

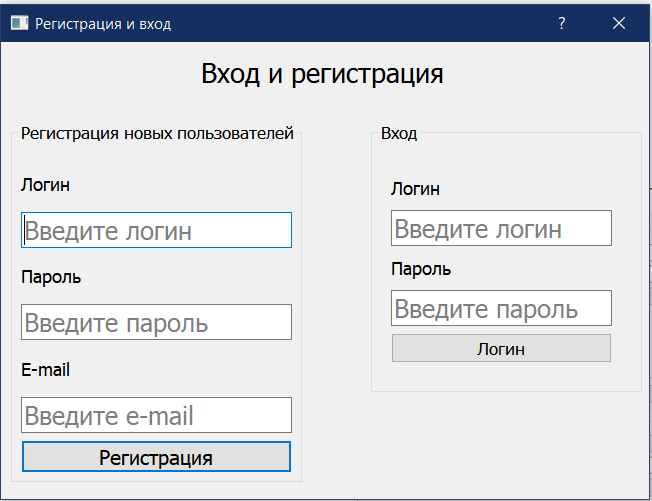


Рис. 4.1.1 До добавления нового пользователя

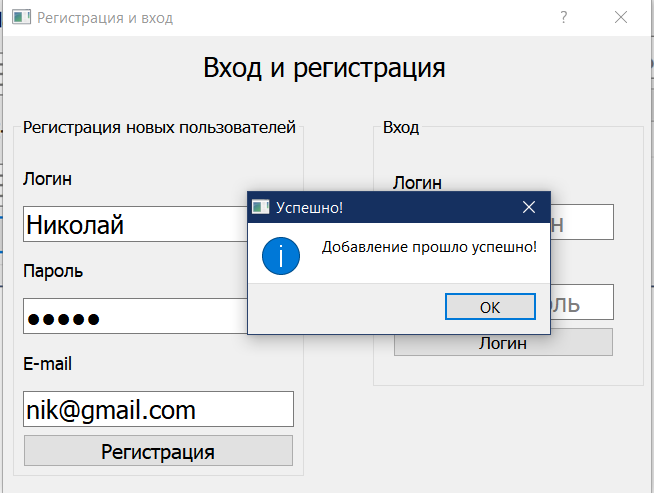


Рис. 4.1.2 После добавления пользователя

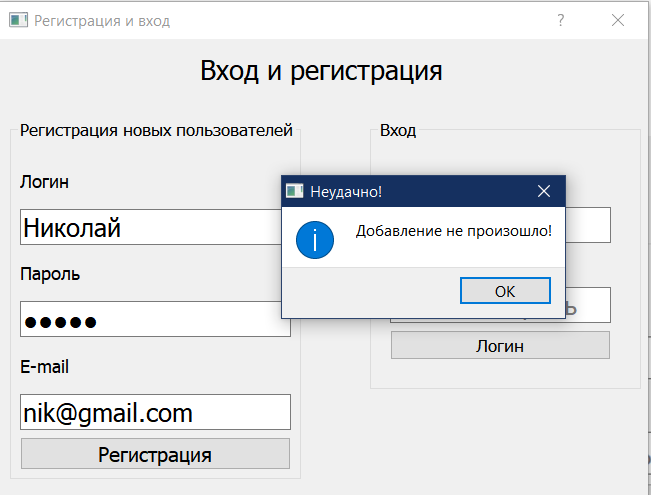


Рис. 4.1.3 Добавление пользователя с существующим логином

* 1. **Авторизация:**

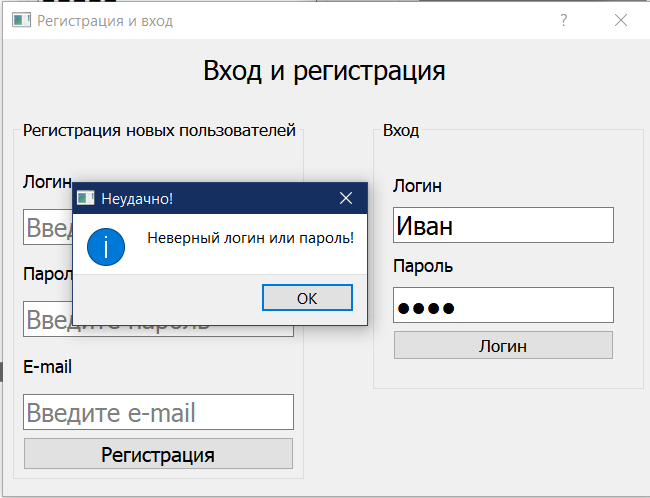


Рис. 4.2.1 Вход по неверным данным

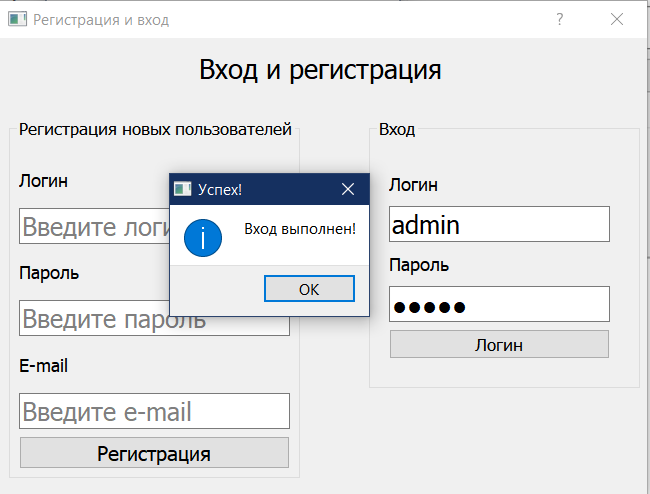


Рис. 4.2.1 Вход, используя зарегистрированного пользователя

* 1. **Просмотр каталога:**

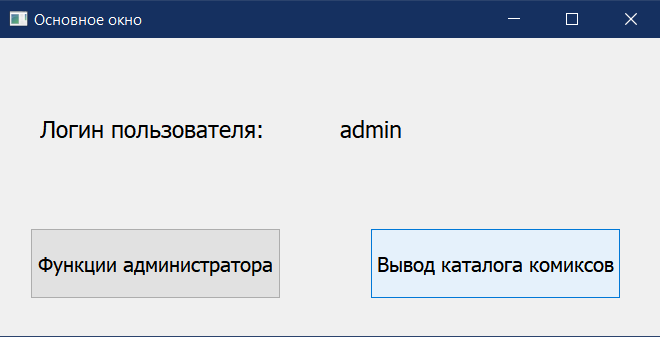


Рис. 4.3.1 Переход из основного окна

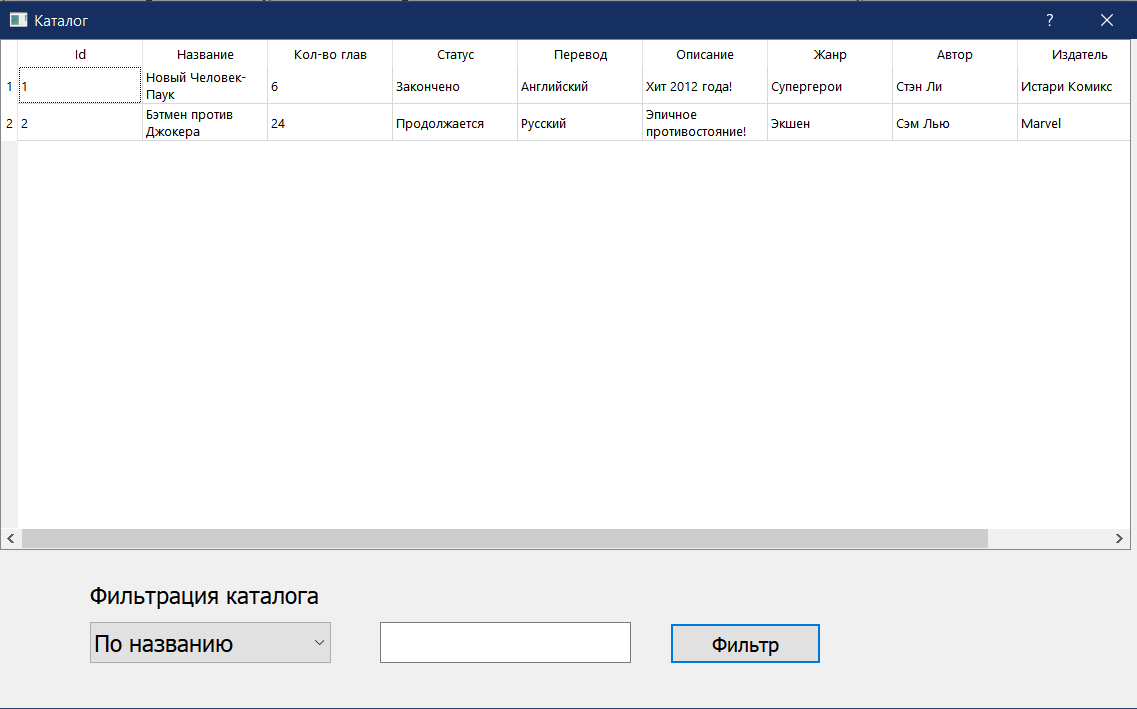


Рис. 4.3.2 Каталог без применения фильтров

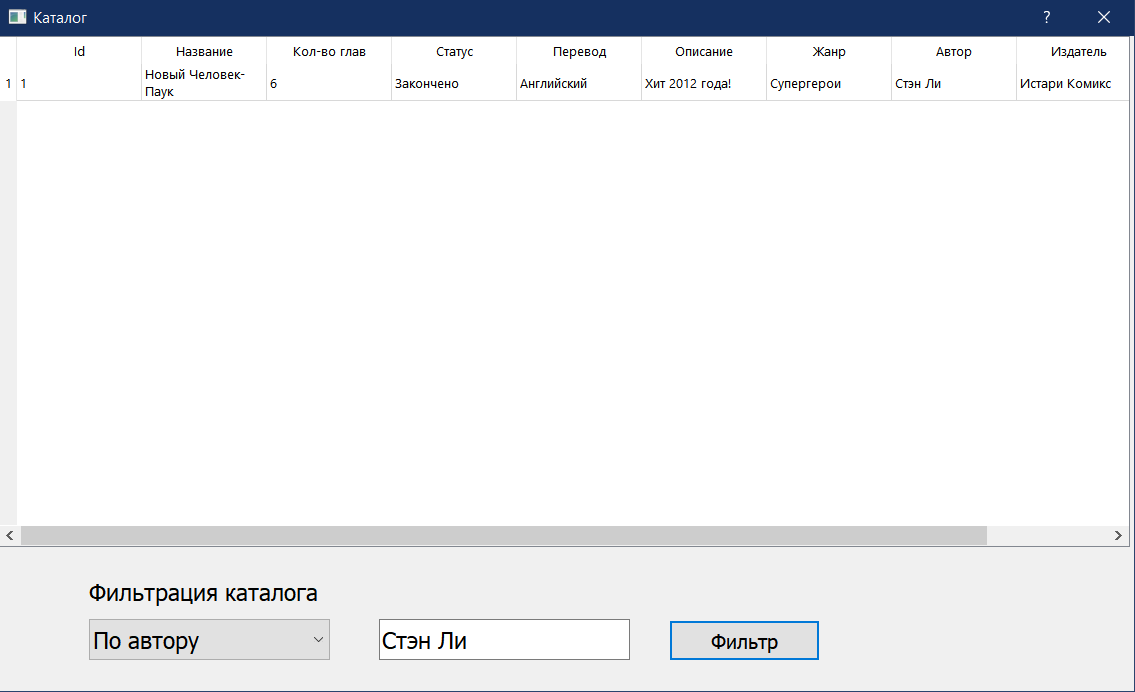


Рис. 4.3.3 Каталог с применением фильтра

* 1. **Вставка комикса:**

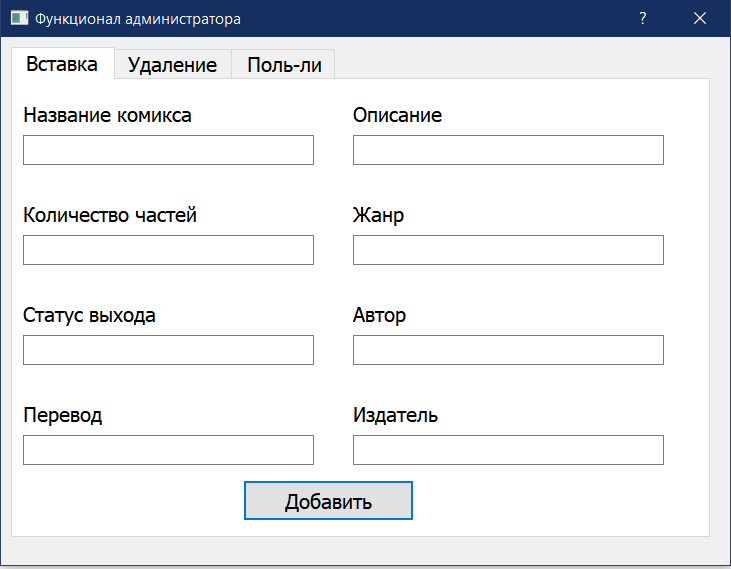


Рис. 4.4.1 До вставки комикса

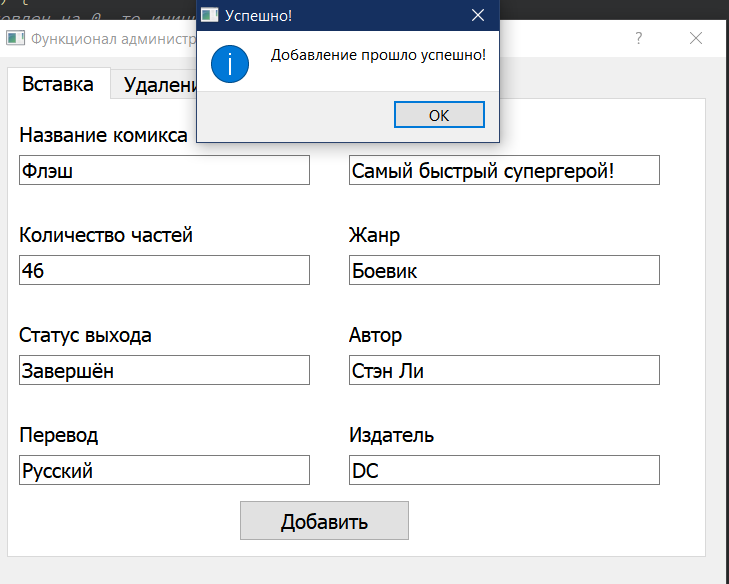


Рис. 4.4.2 Окно подтверждения вставки

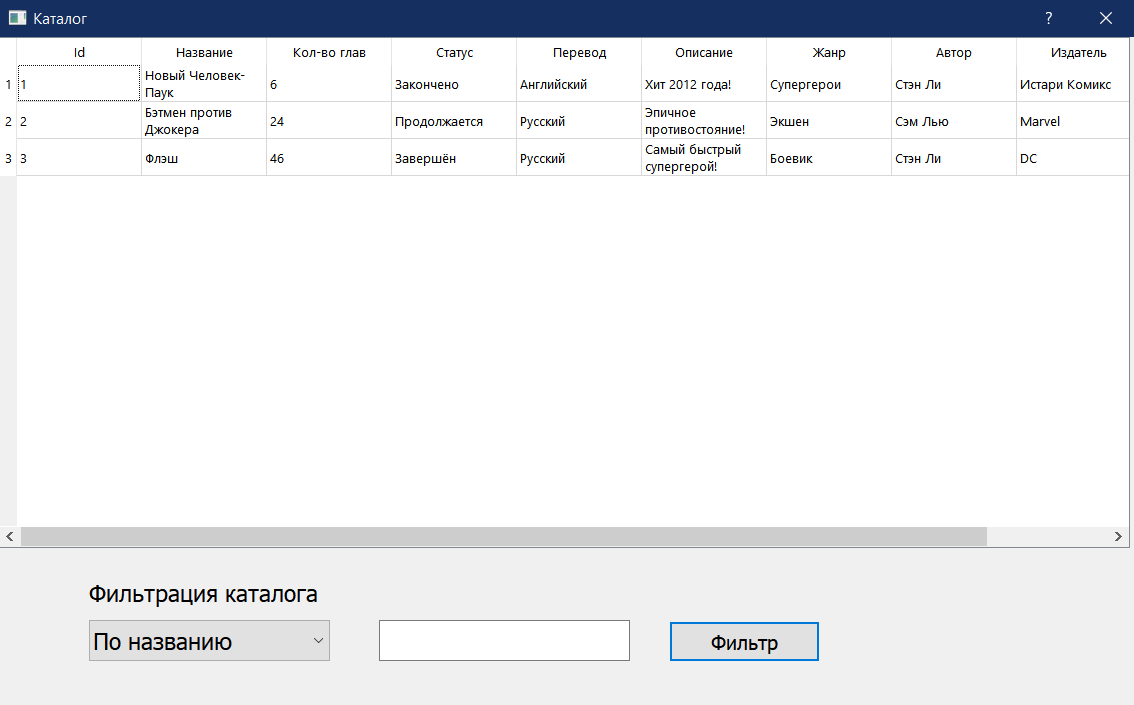


Рис. 4.4.3 Подтверждение вставки

* 1. **Удаление комикса:**

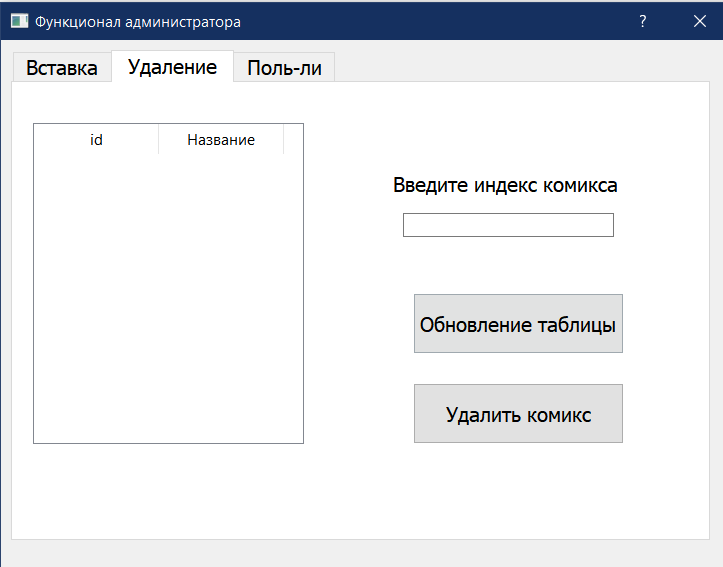


Рис. 4.5.1 Вкладка при открытии

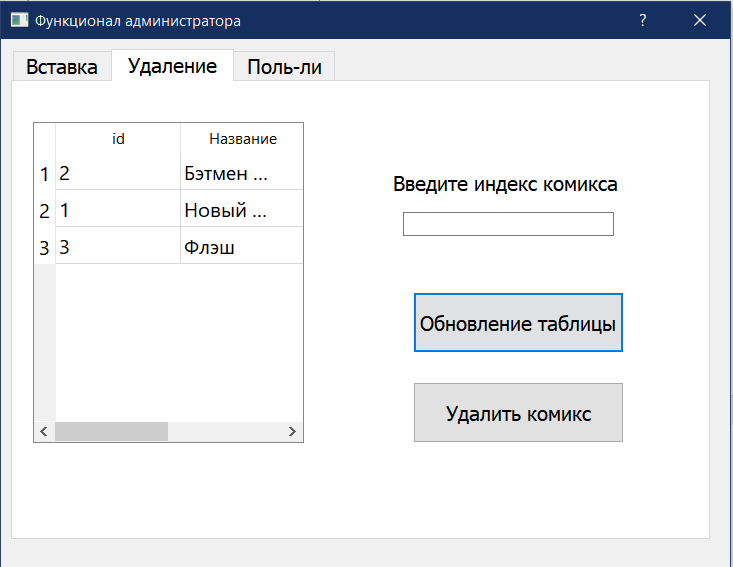


Рис. 4.5.2 Обновление таблицы

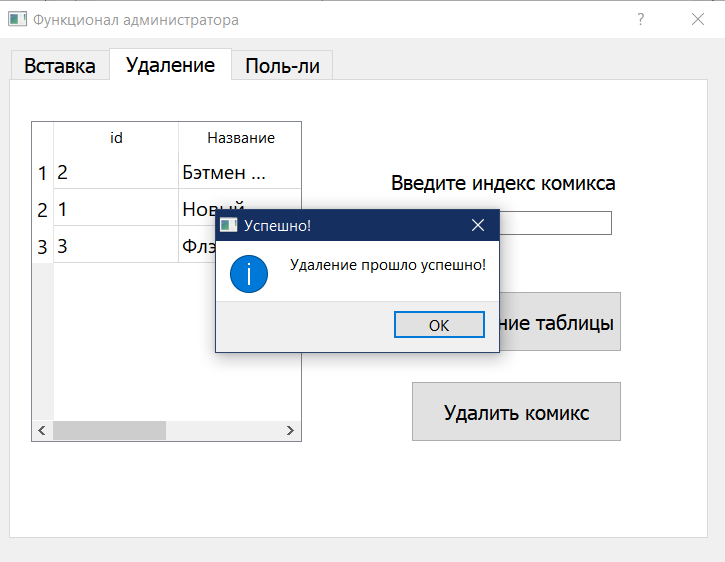


Рис. 4.5.3 Сообщение об удалении

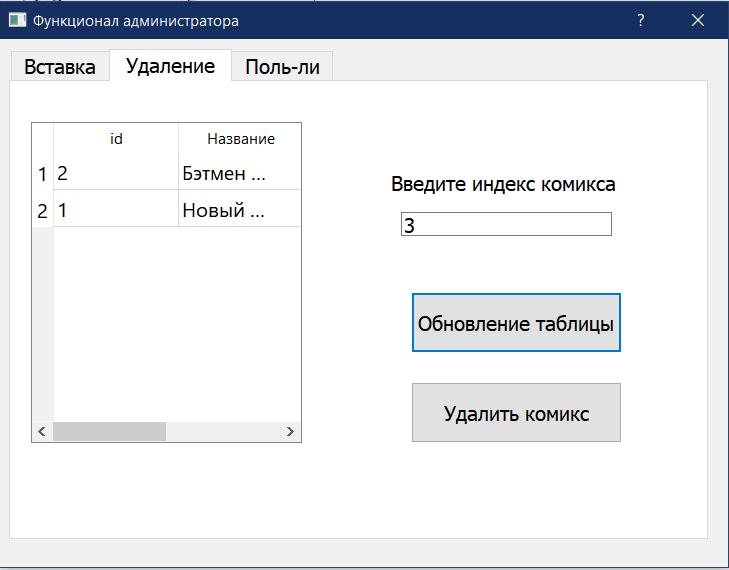


Рис. 4.5.4 Подтверждение удаления

* 1. **Вывод пользователей:**

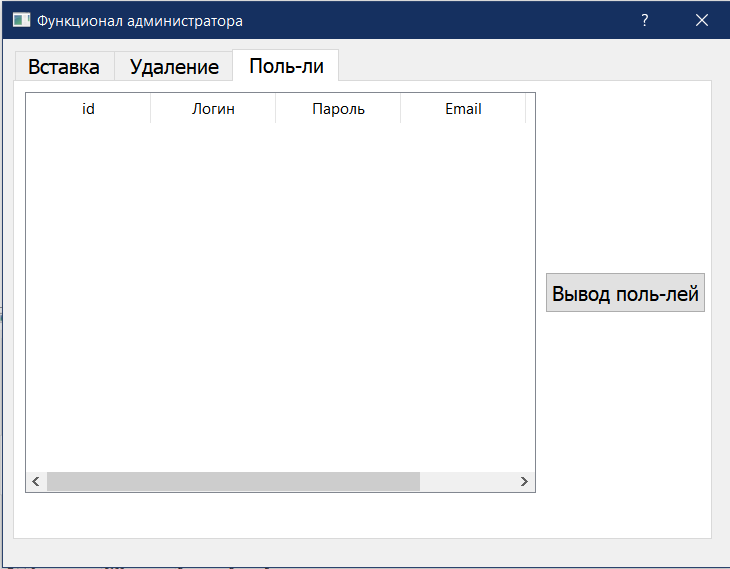


Рис. 4.6.1 До обновления таблицы

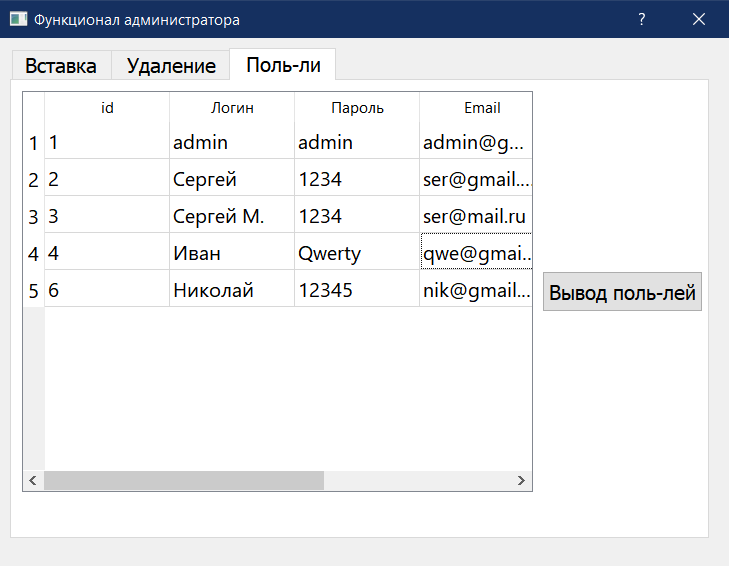


Рис. 4.6.2 После обновления таблицы

### **Заключение**

Подводя итоги, можно выделить преимущества и недостатки разработанного приложения. К достоинствам можно отнести:

* в программе присутствует система авторизации;
* данная программа проста в использовании;
* многие поля ввода содержат подсказки в виде надписей;
* данные в программе представлены в удобном для понимания представлении;
* принимает латиницу и кириллицу для ввода данных;
* хранение данных осуществляется в БД MySQL;
* лёгкость интерпретации на другую предметную область.

К недостаткам можно отнести:

* отсутствует функция редактирования для хранимых данных;
* жесткая привязка к конкретной схеме базы данных;
* сложное добавление новых классов-сущностей, следует из пункта выше.

### **Список использованных источников**

1. **Шлее М., Qt 5.10. Профессиональное программирование на С++. – СПб.: БХВ-Петербург, 2018. – 1072 с.**
2. **Ларман К. Применение UML 2.0 и шаблонов проектирования. Введение в объектно-ориентированный анализ, проектирование и итеративную разработку: пер. с англ. — М.: ИД «Вильямс», 2013. — 736 с.: ил.**
3. *Бьёрн Страуструп.* Программирование: принципы и практика с использованием С++. — 2016.
4. **Qt Documentation [Электронный ресурс]. – The Qt Company Ltd, 2021. – URL:** <https://doc.qt.io> **(дата обращения: 13.12.2021)**
5. Е.О. Шумова, ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ Методические указания к выполнению курсового проекта. – ФГАОУ ВО СПбГУАП, 2021. – 17 с.

### **Приложение А**

Код приложения

admin.h

#ifndef ADMIN\_H

#define ADMIN\_H

#include <QObject>

#include <QtSql/QSql>

#include <QtSql/QSqlDatabase>

#include <QtSql/QSqlError>

#include <QtSql/QSqlQuery>

#include <QMessageBox>

#include "user.h"

*class* **Admin** : *public* user

{

Q\_OBJECT

*public*:

*//Функция* *возвращает* *указатель* *на* *одиночку*

*static* Admin\* **getAdmin**();

QString current\_login;

*private*:

*//Указатель* *на* *класс* *(одиночку)*

*static* Admin\* adm;

*explicit* **Admin**(QObject \*parent = *nullptr*);

signals:

};

#endif *//* *ADMIN\_H*

admin.cpp

#include "admin.h"

Admin::**Admin**(QObject \*parent) {}

Admin\* Admin::adm = 0;

Admin\* Admin::**getAdmin**() {

*//* *Если* *указаетль* *установлен* *на* *0,* *то* *инициализируем* *класс*

*if* (adm == 0)

adm = *new* Admin(*nullptr*);

*return* adm;

}

admin\_tools.h

#ifndef ADMIN\_TOOLS\_H

#define ADMIN\_TOOLS\_H

#include <QDialog>

#include <QtSql/QSql>

#include <QtSql/QSqlDatabase>

#include <QtSql/QSqlError>

#include <QtSql/QSqlQuery>

#include <QMessageBox>

#include "comix.h"

*namespace* **Ui** {

*class* **admin\_tools**;

}

*class* **admin\_tools** : *public* QDialog

{

Q\_OBJECT

*public*:

*explicit* **admin\_tools**(QWidget \*parent = *nullptr*);

~***admin\_tools***();

*private* slots:

void **on\_pushButton\_clicked**();

void **on\_pushButton\_2\_clicked**();

void **on\_pushButton\_3\_clicked**();

void **on\_refresh\_clicked**();

*private*:

Ui::admin\_tools \*ui;

};

#endif *//* *ADMIN\_TOOLS\_H*

admin\_tools.cpp

#include "admin\_tools.h"

#include "ui\_admin\_tools.h"

admin\_tools::**admin\_tools**(QWidget \*parent) :

QDialog(*parent*),

ui(*new* Ui::admin\_tools)

{

ui->setupUi(*this*);

}

admin\_tools::~***admin\_tools***()

{

*delete* ui;

}

void admin\_tools::**on\_pushButton\_clicked**()

{

QSqlQuery users;

users.exec("select \* from user\_");

int i = 0;

int size = 0;

*if*(users.last()) {

size = users.at() + 1;

users.first();

users.previous();

}

ui->tableUsers->setRowCount(size);

*while*(users.next()){

ui->tableUsers->setItem(i,0,*new* QTableWidgetItem*(users.value(*"UserId"*).toString())*);

ui->tableUsers->setItem(i,1,*new* QTableWidgetItem*(users.value(*"login"*).toString())*);

ui->tableUsers->setItem(i,2,*new* QTableWidgetItem*(users.value(*"password\_"*).toString())*);

ui->tableUsers->setItem(i,3,*new* QTableWidgetItem*(users.value(*"email"*).toString())*);

i++;

}

}

void admin\_tools::**on\_pushButton\_2\_clicked**()

{

QSqlQuery qry\_log(QSqlDatabase::database("kursov"));

QString genre\_ = ui->lineEdit\_6->text();

QString author\_ = ui->lineEdit\_7->text();

QString publisher\_ = ui->lineEdit\_8->text();

qry\_log.prepare(QString("select g\_name from comix where g\_name = :g\_name"));

qry\_log.bindValue(":g\_name", genre\_);

qry\_log.exec();

int size = 0;

*if*(qry\_log.last()) {

size = qry\_log.at() + 1;

qry\_log.first();

qry\_log.previous();

}

*if*(size == 0) {

qry\_log.prepare(QString("insert into genre(g\_name) values (:g\_name)"));

qry\_log.bindValue(":g\_name", genre\_);

qry\_log.exec();

}

qry\_log.prepare(QString("select author from comix where author = :author"));

qry\_log.bindValue(":author", author\_);

qry\_log.exec();

size = 0;

*if*(qry\_log.last()) {

size = qry\_log.at() + 1;

qry\_log.first();

qry\_log.previous();

}

*if*(size == 0) {

qry\_log.prepare(QString("insert into author(author) values (:author)"));

qry\_log.bindValue(":author", author\_);

qry\_log.exec();

}

qry\_log.prepare(QString("select publisher from comix where publisher = :publisher"));

qry\_log.bindValue(":publisher", publisher\_);

qry\_log.exec();

size = 0;

*if*(qry\_log.last()) {

size = qry\_log.at() + 1;

qry\_log.first();

qry\_log.previous();

}

*if*(size == 0){

qry\_log.prepare(QString("insert into publisher(publisher) values (:publisher)"));

qry\_log.bindValue(":publisher", publisher\_);

qry\_log.exec();

}

QString title\_ = ui->lineEdit->text();

int numb\_of\_ch\_ = ui->lineEdit\_2->text().toInt();

QString stat\_ = ui->lineEdit\_3->text();

QString translate\_ = ui->lineEdit\_4->text();

QString descript\_ = ui->lineEdit\_5->text();

qry\_log.prepare(QString("insert into comix(title, numb\_of\_ch, stat, translate, descript, g\_name, author, publisher) values (:title, :numb\_of\_ch, :stat, :translate, :descript, :g\_name, :author, :publisher)"));

qry\_log.bindValue(":title", title\_);

qry\_log.bindValue(":numb\_of\_ch", numb\_of\_ch\_);

qry\_log.bindValue(":stat", stat\_);

qry\_log.bindValue(":translate", translate\_);

qry\_log.bindValue(":descript", descript\_);

qry\_log.bindValue(":g\_name", genre\_);

qry\_log.bindValue(":author", author\_);

qry\_log.bindValue(":publisher", publisher\_);

qry\_log.exec();

QMessageBox::information(*this*, "Успешно!", "Добавление прошло успешно!");

}

void admin\_tools::**on\_pushButton\_3\_clicked**()

{

QSqlQuery qry\_del(QSqlDatabase::database("kursov"));

int id\_del = ui->id\_line->text().toInt();

qry\_del.prepare(QString("delete from comix where ComixId = :id"));

qry\_del.bindValue(":id", id\_del);

qry\_del.exec();

QMessageBox::information(*this*, "Успешно!", "Удаление прошло успешно!");

}

void admin\_tools::**on\_refresh\_clicked**()

{

QSqlQuery refr;

refr.exec("select ComixId, title from comix");

int i = 0;

int size = 0;

*if*(refr.last()) {

size = refr.at() + 1;

refr.first();

refr.previous();

}

ui->refreshTable->setRowCount(size);

*while*(refr.next()){

ui->refreshTable->setItem(i,0,*new* QTableWidgetItem*(refr.value(*"ComixId"*).toString())*);

ui->refreshTable->setItem(i,1,*new* QTableWidgetItem*(refr.value(*"title"*).toString())*);

i++;

}

}

author.h

#ifndef AUTHOR\_H

#define AUTHOR\_H

#include <QObject>

#include "comix.h"

*class* **author** : *public* QObject

{

Q\_OBJECT

*public*:

*explicit* **author**(QVector<comix\*> TitleList, QObject \*parent = *nullptr*);

*explicit* **author**(QObject \*parent = *nullptr*);

QVector<comix\*> **Get\_Lst**();

void **setLst**(QVector<comix\*> TitleList);

*private*:

QVector<comix\*> TitleList;

signals:

};

#endif *//* *AUTHOR\_H*

author.cpp

#include "author.h"

author::**author**(QObject \*parent) : QObject(*parent*)

{

}

author::**author**(QVector<comix\*> TitleList, QObject \*parent) : QObject(*parent*){

*this*->TitleList = TitleList;

}

QVector<comix\*> author::**Get\_Lst**() {

*return* *this*->TitleList;

};

void author::**setLst**(QVector<comix \*> TitleList) {

*this*->TitleList = TitleList;

}

catalog.h

#ifndef CATALOG\_H

#define CATALOG\_H

#include <QDialog>

#include <QtSql/QSql>

#include <QtSql/QSqlDatabase>

#include <QtSql/QSqlError>

#include <QtSql/QSqlQuery>

#include <QMessageBox>

#include "comix.h"

*namespace* **Ui** {

*class* **Catalog**;

}

*class* **Catalog** : *public* QDialog

{

Q\_OBJECT

*public*:

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

void **ShowCatalog**();

~***Catalog***();

*private*:

Ui::Catalog \*ui;

signals:

*private* slots:

void **on\_use\_filtr\_clicked**();

};

#endif *//* *CATALOG\_H*

catalog.cpp

#include "catalog.h"

#include "ui\_catalog.h"

Catalog::**Catalog**(QWidget \*parent) :

QDialog(*parent*), ui(*new* Ui::Catalog) {

ui->setupUi(*this*);

ShowCatalog();

}

Catalog::~***Catalog***()

{

*delete* ui;

}

void Catalog::**ShowCatalog**() {

QSqlQuery catalog;

catalog.exec("select \* from comix");

int i = 0;

int size = 0;

*if*(catalog.last()) {

size = catalog.at() + 1;

catalog.first();

catalog.previous();

}

ui->catalogForm->setRowCount(size);

*while*(catalog.next()){

ui->catalogForm->setItem(i,0,*new* QTableWidgetItem*(catalog.value(*"ComixId"*).toString())*);

ui->catalogForm->setItem(i,1,*new* QTableWidgetItem*(catalog.value(*"title"*).toString())*);

ui->catalogForm->setItem(i,2,*new* QTableWidgetItem*(catalog.value(*"numb\_of\_ch"*).toString())*);

ui->catalogForm->setItem(i,3,*new* QTableWidgetItem*(catalog.value(*"stat"*).toString())*);

ui->catalogForm->setItem(i,4,*new* QTableWidgetItem*(catalog.value(*"translate"*).toString())*);

ui->catalogForm->setItem(i,5,*new* QTableWidgetItem*(catalog.value(*"descript"*).toString())*);

ui->catalogForm->setItem(i,6,*new* QTableWidgetItem*(catalog.value(*"g\_name"*).toString())*);

ui->catalogForm->setItem(i,7,*new* QTableWidgetItem*(catalog.value(*"author"*).toString())*);

ui->catalogForm->setItem(i,8,*new* QTableWidgetItem*(catalog.value(*"publisher"*).toString())*);

i++;

}

}

void Catalog::**on\_use\_filtr\_clicked**()

{

QSqlQuery filtr\_ch(QSqlDatabase::database("kursov"));

int curNum = ui->comboBox->currentIndex();

QString check = ui->filtr->text();

*switch* (curNum) {

*case* 0: {

filtr\_ch.prepare(QString("select \* from comix where title = :check"));

filtr\_ch.bindValue(":check", check);

*break*;

}

*case* 1: {

filtr\_ch.prepare(QString("select \* from comix where g\_name = :check"));

filtr\_ch.bindValue(":check", check);

*break*;

}

*case* 2: {

filtr\_ch.prepare(QString("select \* from comix where author = :check"));

filtr\_ch.bindValue(":check", check);

*break*;

}

*case* 3: {

filtr\_ch.prepare(QString("select \* from comix where publisher = :check"));

filtr\_ch.bindValue(":check", check);

*break*;

}

}

filtr\_ch.exec();

int i = 0;

int size = 0;

*if*(filtr\_ch.last()) {

size = filtr\_ch.at() + 1;

filtr\_ch.first();

filtr\_ch.previous();

}

ui->catalogForm->setRowCount(size);

*while*(filtr\_ch.next()){

ui->catalogForm->setItem(i,0,*new* QTableWidgetItem*(filtr\_ch.value(*"ComixId"*).toString())*);

ui->catalogForm->setItem(i,1,*new* QTableWidgetItem*(filtr\_ch.value(*"title"*).toString())*);

ui->catalogForm->setItem(i,2,*new* QTableWidgetItem*(filtr\_ch.value(*"numb\_of\_ch"*).toString())*);

ui->catalogForm->setItem(i,3,*new* QTableWidgetItem*(filtr\_ch.value(*"stat"*).toString())*);

ui->catalogForm->setItem(i,4,*new* QTableWidgetItem*(filtr\_ch.value(*"translate"*).toString())*);

ui->catalogForm->setItem(i,5,*new* QTableWidgetItem*(filtr\_ch.value(*"descript"*).toString())*);

ui->catalogForm->setItem(i,6,*new* QTableWidgetItem*(filtr\_ch.value(*"g\_name"*).toString())*);

ui->catalogForm->setItem(i,7,*new* QTableWidgetItem*(filtr\_ch.value(*"author"*).toString())*);

ui->catalogForm->setItem(i,8,*new* QTableWidgetItem*(filtr\_ch.value(*"publisher"*).toString())*);

i++;

}

}

comix.h

#ifndef COMIX\_H

#define COMIX\_H

#include <QObject>

*class* **comix** : *public* QObject

{

Q\_OBJECT

*public*:

*explicit* **comix**(QString title, QString author, int numb\_of\_ch, QString genre, QString status, QString translate, QString lic, QString description, QObject \*parent = *nullptr*);

*explicit* **comix**(QObject \*parent = *nullptr*);

QString **Get\_title**();

int **Get\_numb**();

QString **Get\_author**();

QString **Get\_genre**();

QString **Get\_status**();

QString **Get\_translate**();

QString **Get\_lic**();

QString **Get\_description**();

void **setTitle**(QString title);

void **setAuthor**(QString author);

void **setNumbCh**(int numb\_of\_ch);

void **setGenre**(QString genre);

void **setStatus**(QString status);

void **setTranslate**(QString translate);

void **setLic**(QString lic);

void **setDescript**(QString description);

*private*:

int ComixId;

QString title;

QString author;

int numb\_of\_ch;

QString genre;

QString status;

QString translate;

QString lic;

QString description;

signals:

};

#endif *//* *COMIX\_H*

comix.cpp

#include "comix.h"

comix::**comix**(QObject \*parent) : QObject(*parent*){}

comix::**comix**(QString title, QString author, int numb\_of\_ch, QString genre, QString status, QString translate, QString lic, QString description, QObject \*parent) : QObject(*parent*){

*this*->title = title;

*this*->author = author;

*this*->numb\_of\_ch = numb\_of\_ch;

*this*->genre = genre;

*this*->status = status;

*this*->translate = translate;

*this*->lic = lic;

*this*->description = description;

}

QString comix::**Get\_title**() {

*return* *this*->title;

}

QString comix::**Get\_author**() {

*return* *this*->author;

}

int comix::**Get\_numb**() {

*return* *this*->numb\_of\_ch;

}

QString comix::**Get\_genre**() {

*return* *this*->genre;

}

QString comix::**Get\_status**() {

*return* *this*->status;

}

QString comix::**Get\_translate**() {

*return* *this*->translate;

}

QString comix::**Get\_lic**() {

*return* *this*->lic;

}

QString comix::**Get\_description**() {

*return* *this*->description;

}

void comix::**setTitle**(QString title) {

*this*->title = title;

}

void comix::**setAuthor**(QString author) {

*this*->author = author;

}

void comix::**setNumbCh**(int numb\_of\_ch) {

*this*->numb\_of\_ch = numb\_of\_ch;

}

void comix::**setGenre**(QString genre) {

*this*->genre = genre;

}

void comix::**setStatus**(QString status) {

*this*->status = status;

}

void comix::**setTranslate**(QString translate) {

*this*->translate = translate;

}

void comix::**setLic**(QString lic) {

*this*->lic = lic;

}

void comix::**setDescript**(QString description) {

*this*->description = description;

}

genre.h

#ifndef GENRE\_H

#define GENRE\_H

#include <QObject>

*class* **genre** : *public* QObject

{

Q\_OBJECT

*public*:

*explicit* **genre**(QString descript, QObject \*parent = *nullptr*);

*explicit* **genre**(QObject \*parent = *nullptr*);

QString **Get\_descript**();

void **setDesc**(QString descript);

*private*:

QString descript;

signals:

};

#endif *//* *GENRE\_H*

genre.cpp

#include "genre.h"

genre::**genre**(QObject \*parent) : QObject(*parent*){}

genre::**genre**(QString descript, QObject \*parent) : QObject(*parent*){

*this*->descript = descript;

}

QString genre::**Get\_descript**() {

*return* *this*->descript;

};

void genre::**setDesc**(QString descript) {

*this*->descript = descript;

}

mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QtSql/QSql>

#include <QtSql/QSqlDatabase>

#include <QtSql/QSqlError>

#include <QtSql/QSqlQuery>

#include <QMessageBox>

#include "reg\_log.h"

#include "catalog.h"

#include "admin.h"

#include "admin\_tools.h"

#include "author.h"

#include "comix.h"

#include "genre.h"

#include "publisher.h"

#include "user.h"

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

*class* **MainWindow** : *public* QMainWindow

{

Q\_OBJECT

*public*:

**MainWindow**(QWidget \*parent = *nullptr*);

~***MainWindow***();

*public* slots:

*//void* *openreg\_log();*

*private* slots:

void **on\_pushButton\_clicked**();

void **on\_admin\_clicked**();

*private*:

Ui::MainWindow \*ui;

QSqlDatabase db;

QString login;

QString pass;

};

#endif *//* *MAINWINDOW\_H*

mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(*parent*)

, ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

*//Подключение* *бд*

db = QSqlDatabase::addDatabase("QODBC");

db.setHostName("127.0.0.1");

db.setPort(3306);

db.setDatabaseName("kursov");

db.setUserName("root");

db.setPassword("1234");

bool ok = db.open();

*if* (ok){

*//* *Если* *бд* *подключилась* *выводим* *сообщение* *об* *успехе*

QMessageBox::information(*this*, "База данных", "Подключение произошло успешно!");

}

*else* {

*//* *Иначе,* *выводим* *сообщение* *о* *неудаче* *и* *блокируем* *кнопки* *сохранения* *и* *загрузки*

QMessageBox::warning(*this*, "База данных", "Не удалось подключиться к базе данных!");

}

reg\_log dialog(*this*, *&login*, *&pass*);

dialog.setModal(*true*);

dialog.*exec*();

Admin \*admin = Admin::getAdmin();

ui->user\_login->setText(admin->current\_login);

}

MainWindow::~***MainWindow***()

{

*delete* ui;

}

void MainWindow::**on\_pushButton\_clicked**()

{

Catalog cal(*this*);

cal.setModal(*true*);

cal.*exec*();

}

void MainWindow::**on\_admin\_clicked**()

{

Admin \*admin = Admin::getAdmin();

*if*(admin->current\_login == "admin") {

admin\_tools adm(*this*);

adm.setModal(*true*);

adm.*exec*();

} *else* {

QMessageBox::warning(*this*, "Администрация", "Вы не имеете права администратора!");

}

}

publisher.h

#ifndef PUBLISHER\_H

#define PUBLISHER\_H

#include <QObject>

#include "comix.h"

*class* **publisher** : *public* QObject

{

Q\_OBJECT

*public*:

*explicit* **publisher**(QVector<comix\*> ComList, QObject \*parent = *nullptr*);

*explicit* **publisher**(QObject \*parent = *nullptr*);

QVector<comix\*> **Get\_L**();

void **setComixList**(QVector<comix\*> ComList);

*private*:

QVector<comix\*> ComList;

signals:

};

#endif *//* *PUBLISHER\_H*

publisher.cpp

#include "publisher.h"

publisher::**publisher**(QObject \*parent) : QObject(*parent*)

{

}

publisher::**publisher**(QVector<comix\*> ComList, QObject \*parent) : QObject(*parent*){

*this*->ComList = ComList;

}

QVector<comix\*> publisher::**Get\_L**() {

*return* *this*->ComList;

}

void publisher::**setComixList**(QVector<comix \*> ComList) {

*this*->ComList = ComList;

}

reg\_log.h

#ifndef REG\_LOG\_H

#define REG\_LOG\_H

#include <QDialog>

#include <QtSql/QSql>

#include <QtSql/QSqlDatabase>

#include <QtSql/QSqlError>

#include <QtSql/QSqlQuery>

#include <QMessageBox>

#include "admin.h"

*namespace* **Ui** {

*class* **reg\_log**;

}

*class* **reg\_log** : *public* QDialog

{

Q\_OBJECT

*public*:

*explicit* **reg\_log**(QWidget \*parent = *nullptr*, QString \*login = *nullptr*, QString \*pass = *nullptr*);

~***reg\_log***();

*private* slots:

void **on\_registerBtn\_clicked**();

void **on\_loginBtn\_clicked**();

*private*:

Ui::reg\_log \*ui;

QString login;

QString pass;

};

#endif *//* *REG\_LOG\_H*

reg\_log.cpp

#include "reg\_log.h"

#include "ui\_reg\_log.h"

reg\_log::**reg\_log**(QWidget \*parent, QString \*login, QString \*pass) :

QDialog(*parent*),

ui(*new* Ui::reg\_log)

{

*this*->login = \*login;

*this*->pass = \*pass;

ui->setupUi(*this*);

ui->username->setPlaceholderText("Введите логин");

ui->password->setPlaceholderText("Введите пароль");

ui->email->setPlaceholderText("Введите e-mail");

ui->usernameLogin->setPlaceholderText("Введите логин");

ui->passwordLogin->setPlaceholderText("Введите пароль");

}

reg\_log::~***reg\_log***()

{

*delete* ui;

}

void reg\_log::**on\_registerBtn\_clicked**()

{

QSqlQuery qry;

QString username = ui->username->text();

QString password = ui->password->text();

QString email = ui->email->text();

qry.clear();

qry.prepare("insert into user\_(login, password\_, email) values (:username, :password, :email)");

qry.bindValue(":username", username);

qry.bindValue(":password", password);

qry.bindValue(":email", email);

*if*(qry.exec()) {

QMessageBox::information(*this*, "Успешно!", "Добавление прошло успешно!");

} *else* {

QMessageBox::information(*this*, "Неудачно!", "Добавление не произошло!");

}

}

void reg\_log::**on\_loginBtn\_clicked**()

{

QSqlQuery qrylog(QSqlDatabase::database("kursov"));

login = ui->usernameLogin->text();

pass = ui->passwordLogin->text();

qrylog.prepare(QString("select \* from user\_ where login = :login and password\_ = :pass"));

qrylog.bindValue(":login", login);

qrylog.bindValue(":pass", pass);

qrylog.exec();

int size = 0;

*if*(qrylog.last()) {

size = qrylog.at() + 1;

qrylog.first();

qrylog.previous();

}

*if*(size == 0) {

QMessageBox::information(*this*, "Неудачно!", "Неверный логин или пароль!");

} *else* {

*while*(qrylog.next()) {

QMessageBox::information(*this*, "Успех!", "Вход выполнен!");

Admin \*admin = Admin::getAdmin();

admin->current\_login = login;

close();

}

}

}

user.h

#ifndef USER\_H

#define USER\_H

#include <QObject>

#include <QVector>

#include "comix.h"

*class* **user** : *public* QObject

{

Q\_OBJECT

*public*:

*explicit* **user**(QString login, QString pass, QString email, QObject \*parent = *nullptr*);

*explicit* **user**(QObject \*parent = *nullptr*);

QString **Get\_login**();

QString **Get\_pass**();

QString **Get\_email**();

void **setLogin**(QString login);

void **setPass**(QString pass);

void **setEmail**(QString email);

*private*:

QString login;

QString pass;

QString email;

signals:

};

#endif *//* *USER\_H*

user.cpp

#include "user.h"

user::**user**(QObject \*parent) : QObject(*parent*)

{

}

user::**user**(QString login, QString pass, QString email, QObject \*parent) : QObject(*parent*){

*this*->login = login;

*this*->pass = pass;

*this*->email = email;

}

QString user::**Get\_login**() {

*return* *this*->login;

}

QString user::**Get\_pass**() {

*return* *this*->pass;

}

QString user::**Get\_email**() {

*return* *this*->email;

}

void user::**setLogin**(QString login) {

*this*->login = login;

}

void user::**setPass**(QString pass) {

*this*->pass = pass;

}

void user::**setEmail**(QString email) {

*this*->email = email;

}

main.cpp

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(*argc*, *argv*);

MainWindow w;

w.show();

*return* a.exec();

}