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

КУРСОВАЯ РАБОТА (ПРОЕКТ)
ЗАЩИЩЕНА 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. №	4134К		Н.А. Костяков
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2023
КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ.

Задание
на курсовой проект по дисциплине
«Объектно-ориентированное программирование»

Студенту группы 4134к Костякову Никите Андреевичу.

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

Исходные данные: Кинопрокат

Оглавление

1. Постановка задачи

1.1 Анализ предметной области

Данная курсовая работа по программированию направлена на разработку системы классов, которая будет описывать Кинопрокат. Основной целью проектирования является создание программного продукта, который позволит вести учет фильмов отданных в прокат и их возврат.

Предметной областью является управление магазина кинопроката. Основными сущностями предметной области являются фильмы, пользователи, операции аренды и возврата

1.2 Словарь предметной области:

- Фильм
- Автор
- Покупатель
- Аренда
- Возврат

1.3Функциональные требования:

- Ведение данных о наличии фильмов для проката
- Фиксация данных об аренде фильма
- Сохранение информации о фильме: автор, год, название
- Учет данных о покупателях, их билет для доступа
- Расчет стоимости проката с учетом скидок и наценок.
- Поиск и фильтрация данных о прокате и возврата.

2 Разработка классов

2.1 Для разработки иерархии классов мы начнем с выделения основных сущностей предметной области и определения классов, описывающих эти сущности. Затем мы определим управляющие классы и интерфейсные классы для организации взаимодействия между ними и с внешней средой. Ниже приведена детальная разработка иерархии классов.

2.2 Иерархия классов

2.2.1 Фильм

-id (Идентификатор фильма)
-name (название фильма)
-автор (автор фильма)
-год (год выпуска фильма)

Методы:

- Конструктор для создания

- Методы для получения и установки значений полей

- Метод для изменения статуса

```
class movie{
public:
   int number;
   int id, year;
    std::string author, name;
   bool state; //0 - на складе, 1 - выдан
   std::string front ;
   float cnt_rates = 1+rand()%9;
   float sum_rates = cnt_rates+rand()%25;
   float rating = sum_rates/cnt_rates ;
   void set_rate( int stars);
   movie* next= nullptr;

   movie(int num=0, int i=0, int y=0, std::string auth=0, std::string n=0, bool stat
=1){
       number=num;id=i; year = y; author = auth; name = n; state = stat;
       std::string rat = "";
       for (int j = 0; j<3; j++){
          rat+= std::to_string(this->rating)[j];
```

```
        }
        front= name +" "+std::to_string(year)+" "+" "+rat+" ☆";
    }

};
```

2.2.2 Юзер
-id (идентификатор пользователя)
-ФИО (Фамилия имя отчество полльзователя)

Методы:

- Конструктор для создания

- Методы для получения и установки значений полей

2.2.3 Аренда
-id (Идентификатор аренды)
-userId (идентификатор пользователя, который взял фильм)
-filmId (идентификатор фильма, который взяли в аренду)
-Date take (Дата взятия в аренду)
-Date return (дата, кода фильм должны вернуть)

Методы:

- Конструктор для создания

- Методы для получения и установки значений полей

- Метод для изменения статуса

2.3 Управляющие классы:
2.3.1. магазин (shop):

- Поля:

  - Список всех доступных фильмов

- Методы:

  - Методы для добавления и удаления фильмов

- Методы для поиска фильмов различным критериям (год, автор)

```
class shop
{
public:
    movies catalog;
    std::string name;

    void set_name(std::string name){
        this->name = name;
    }

    shop(std::string name){
        set_name(name);

    }


    std::string get_name(){
        return name;
    }
};
```

2.3.2. Прокат (rental):

- Поля:

  - Список всех операций проката

- Методы:

  - Методы для регистрации новой операции проката

  - Методы для отчетности по прокатам

```
class Rental {
private:
    IRentalStrategy* _rentalStrategy;

public:
    Rental(IRentalStrategy* rentalStrategy) : _rentalStrategy(rentalStrategy) {}
```

```
    double CalculateRentalCost(int rentalTime) {
        return _rentalStrategy->CalculateRentalCost(rentalTime);
    }
};
```

2.4 Интерфейсные классы:

Графический интерфейс (GUI):

- Методы:

- Методы для взаимодействия с пользователем, включая ввод и вывод данных.

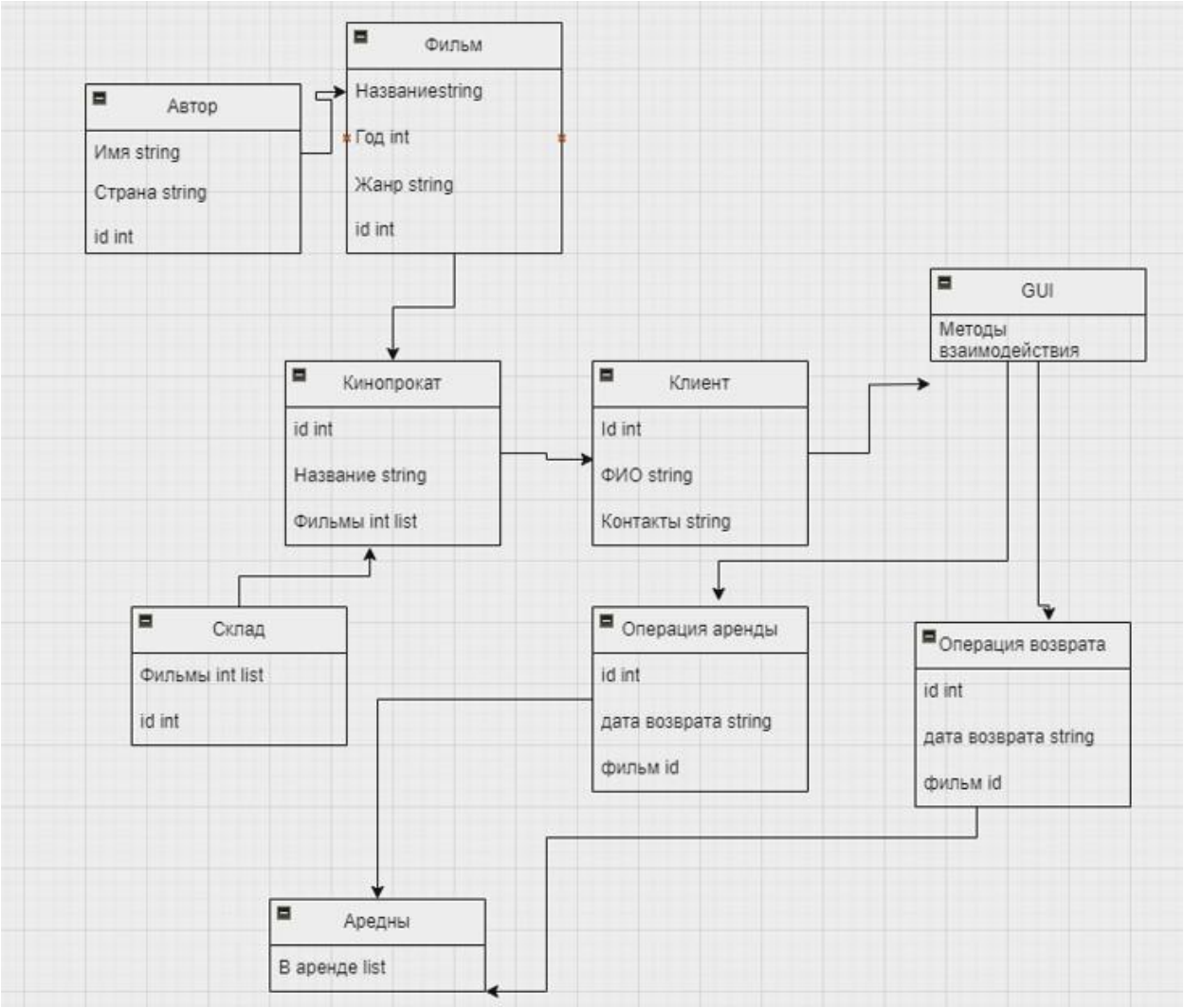
Обоснование проектных решений:

- Использование классов сущностей для описания базовых объектов предметной области позволяет удобно хранить и обрабатывать информацию о фильмах в наличии, авторах, покупателях и операциях проката.

- Управляющие классы, такие как Склад и Прокат, позволяют организовать управление данными и операциями, связанными с фильмами и прокатом

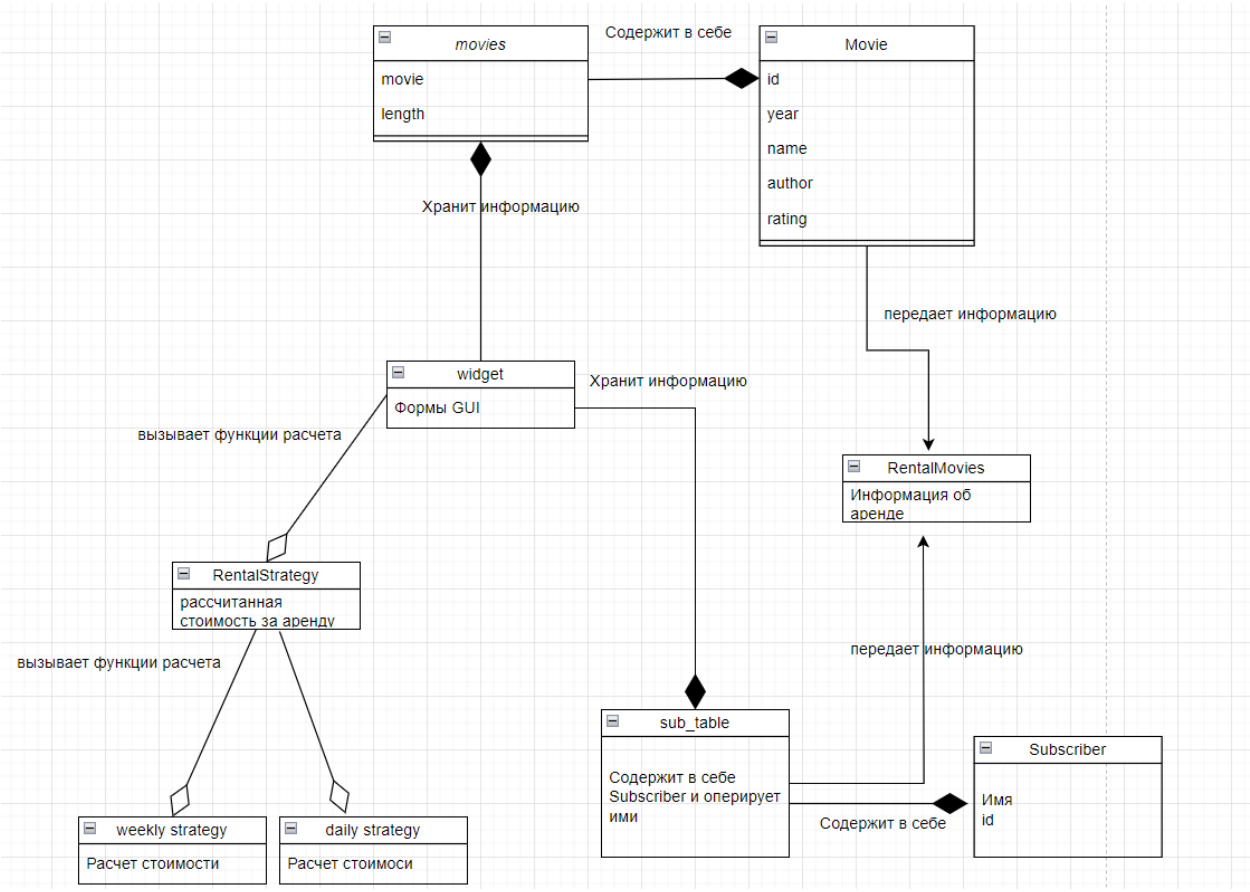
- Интерфейсный класс GUI обеспечивает взаимодействие пользователя с программой через графический интерфейс, что делает использование приложения более удобным для конечных пользователей.

2.5 Рассмотрим полученную диаграмму сущностей:



Эта иерархия классов позволит эффективно управлять данными и операциями, связанными с кинопрокатом, а также обеспечит удобный интерфейс для пользователей.

2.6 Диаграмма классов


movies
movie
length
Содержит в себе
Movie
id
year
name
author
rating
Хранит информацию
передает информацию
widget
Формы GUI
Хранит информацию
вызывает функции расчета
RentalMovies
Информация об аренде
RentalStrategy
рассчитанная стоимость за аренду
вызывает функции расчета
передает информацию
sub_table
Содержит в себе Subscriber и оперирует ими
Subscriber
Имя
id
Содержит в себе
weekly strategy
Расчет стоимости
daily strategy
Расчет стоимоси

3. Разработка приложения

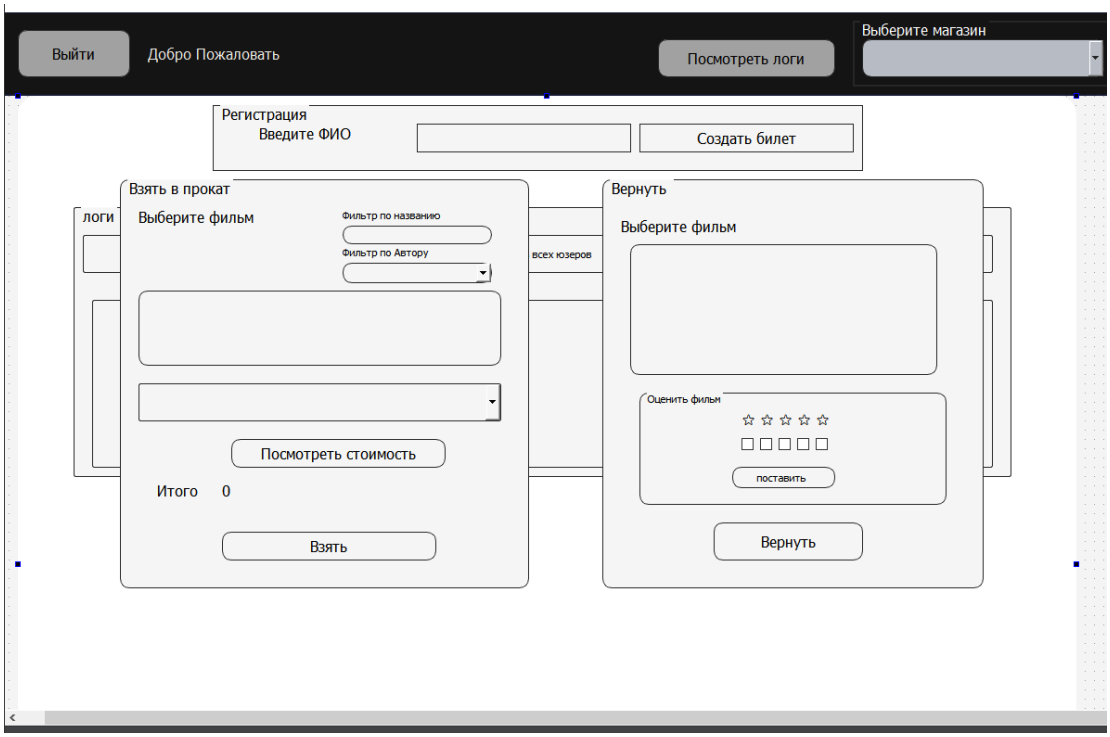
3.1 Описание интерфейсов

Интерфейс представляет из себя набор GroupBox которые становятся прозрачными при определенных событиях и непрозрачными при дргугих. Таким образом все действия происходят в одном окне и элементы взаимозаменяют друг друга.

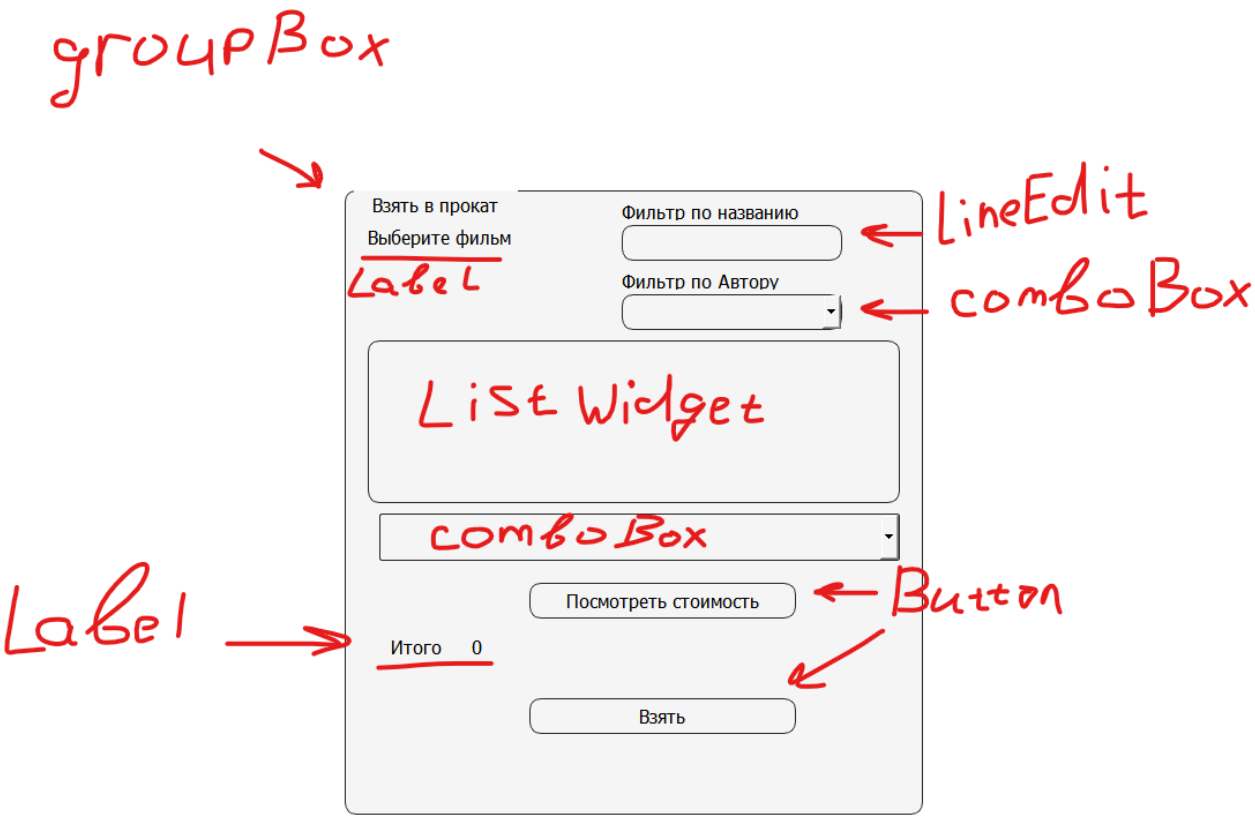
```
ui->groupBox_registration->setHidden(1);
```

Разработались несколько форм для регистрации, которая состоит из нескольких полей ввода и кнопки для проверки, форма для того, чтобы выбрать фильм, которая наполняется в зависимости от выбранного магазина, форма возврата, аналогичная форме выбора фильма и навигационная панель, которая размещает на себе дополнительный функционал, например регистрация, выбор магазина и просмотра логов

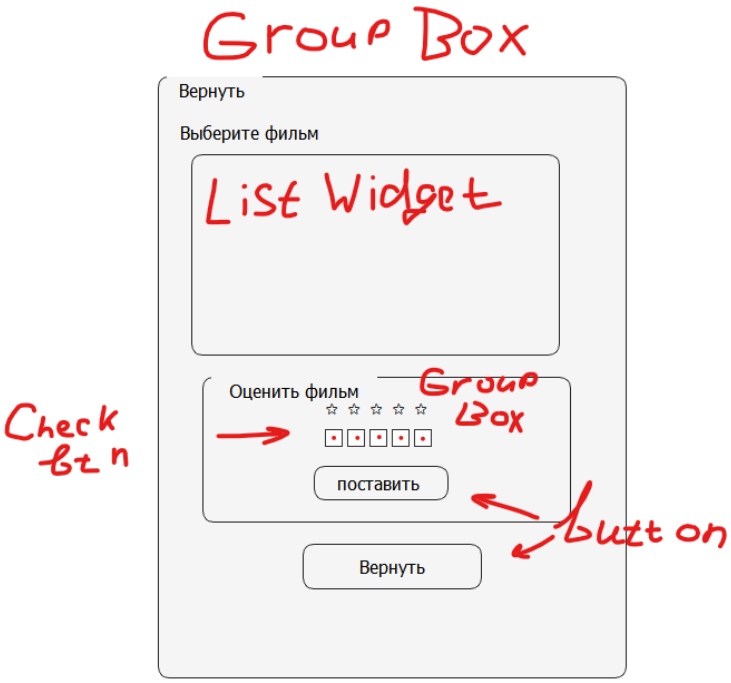
Общий вид формы из режима дизайнера



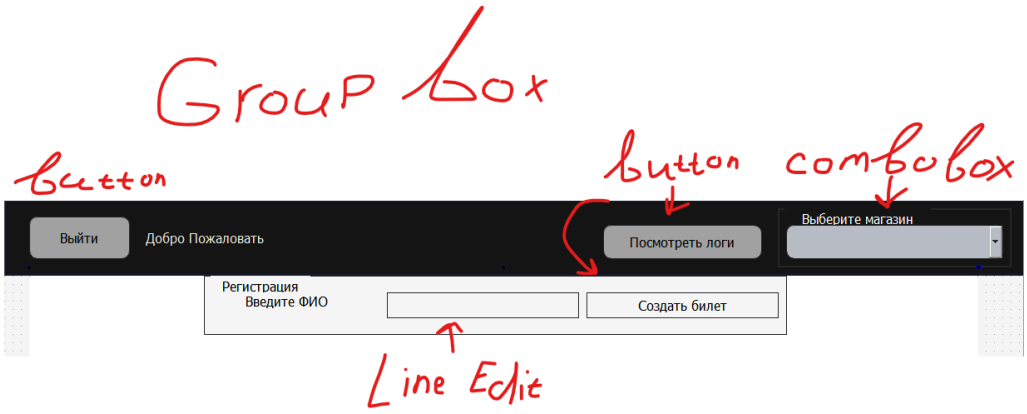
Форма для того, чтобы взять фильм состоит из нескольких компонентов. Процесс наполнения происходит динамически. Фильмы попадают в List Widget, авторы в Фильтр по автору.



Форма для возврата фильма устроена похожим образом. Ее отличие в том, что list widget этой группы наполняется только тогда, когда пользователь взял фильм. Еще одно отличие в том, что фильму можно поставить рейтинг в звездочках, которые выполнены в виде checkbox



Группа навигационной панели и регистрации. Здесь расположены кнопки, которые позволяют перемещаться по интерфейсу. Список магазинов формируется на старте программы и при выборе определенного меняется набор фильмов доступных для просмотра



Логи для программы реализованы в отдельном GroupBox. Здесь информация о базах данных выводится в TableWidget , которые контролируются тремя кнопками переключения таблицы. Ниже на форме можно добавить фильм, чтобы тот был доступен внутри приложения

3.2 Разработка методов классов

Логикой управляют несколько классов, которые объединяют в себе общие методы для работы программы

Так класс shop, который представляет физический магазин, в себе имеет класс movies, который играет роль каталога. Так каждый такой магазин может сам для себя определить набор фильмов, которыми торгует

Rental strategy определяет, по какой цене будет выдан фильм покупателю, так исходя из сценариев, в программе видно, сколько стоит прокат.

Subscribers – таблица пользователей, которые взаимодействуют с ПО

Rental movies – адаптер для класса rental, который собирает нужную информацию о фильме и пользователе и покует ее в таблицу взятых в аренду фильмов

Вот подробное описание методов для каждого класса

1. «WeeklyRentalStrategy» class

Методы:

1.1. «CalculateRentalCost»
- Описание: Метод реализует стратегию расчета стоимости аренды на неделю.
- Параметры:
  - «int rentalTime»: Количество недель аренды.
- Возвращаемое значение: Стоимость аренды на указанное количество недель.

2. «IRentalStrategy» class (интерфейс)

Методы:

2.1. «CalculateRentalCost»

- Описание: Виртуальный метод, который должен быть реализован в каждой конкретной стратегии аренды.
- Параметры:
  - «int rentalTime»: Время аренды в единицах, соответствующих конкретной стратегии.
- Возвращаемое значение: Стоимость аренды, рассчитанная согласно конкретной стратегии.

3. «movie» class

Методы:

3.1. «set_rate»
- Описание: Метод устанавливает рейтинг фильма и обновляет соответствующие поля.
- Параметры:
  - «int stars»: Оценка фильма в виде количества звезд.
- Возвращаемое значение: Отсутствует.

4. «movies» class

Методы:

4.1. «add»
- Описание: Метод добавляет новый фильм в коллекцию.
- Параметры:
  - «int y»: Год выпуска фильма.
  - «std::string auth»: Автор фильма.
  - «std::string n»: Название фильма.
- Возвращаемое значение: Отсутствует.

4.2. «get»
- Описание: Метод возвращает указатель на фильм по индексу.
- Параметры:
  - «int index»: Индекс фильма в коллекции.
- Возвращаемое значение: Указатель на объект класса «movie».

4.3. «get_length»
- Описание: Метод возвращает текущую длину коллекции фильмов.
- Параметры: Отсутствуют.
- Возвращаемое значение: Длина коллекции.

4.4. «get_rating»
- Описание: Метод возвращает рейтинг фильма по индексу.
- Параметры:

- «int index»: Индекс фильма в коллекции.
- Возвращаемое значение: Рейтинг фильма.

4.5. «get_by_name»
- Описание: Метод возвращает указатель на фильм по названию.
- Параметры:
  - «std::string name»: Название фильма.
- Возвращаемое значение: Указатель на объект класса «movie».

5. «Rental» class

Методы:

5.1. «CalculateRentalCost»
- Описание: Метод использует текущую стратегию аренды для расчета стоимости аренды.
- Параметры:
  - «int rentalTime»: Время аренды в единицах, соответствующих текущей стратегии.
- Возвращаемое значение: Стоимость аренды, рассчитанная согласно текущей стратегии.

6. «RentalMovies» class

Методы:

6.1. «add»
- Описание: Метод добавляет информацию об аренде фильма в коллекцию.
- Параметры:
  - «info *input»: Указатель на объект информации об аренде.
- Возвращаемое значение: Отсутствует.

6.2. «find»
- Описание: Метод ищет информацию об аренде по названию фильма и ID арендатора.
- Параметры:
  - «std::string name»: Название фильма.
  - «int sub_id»: ID арендатора.
- Возвращаемое значение: Указатель на объект информации об аренде или «nullptr», если информация не найдена.

6.3. «return_film»
- Описание: Метод удаляет информацию об аренде при возврате фильма.
- Параметры:
  - «std::string name»: Название фильма.

- «subscriber *sub»: Указатель на объект арендатора.
- Возвращаемое значение: Отсутствует.

6.4. «show»
- Описание: Метод возвращает строку с информацией обо всех арендах.
- Параметры: Отсутствуют.
- Возвращаемое значение: Строка с информацией об арендах.

6.5. «get»
- Описание: Метод возвращает указатель на объект информации об аренде по индексу.
- Параметры:
  - «int index»: Индекс информации об аренде.
- Возвращаемое значение: Указатель на объект информации об аренде.

7. «info» class

Методы:

7.1. Конструктор «info»
- Описание: Конструктор класса, инициализирующий объект информации об аренде.
- Параметры:
  - «movie *mov»: Указатель на объект фильма.
  - «subscriber *sub»: Указатель на объект арендатора.

8. «shop» class

Методы:

8.1. «set_name»

- Описание: Метод устанавливает название магазина.
- Параметры:
  - «std::string name»: Название магазина.
- Возвращаемое значение: Отсутствует.

8.2. Конструктор «shop»
- Описание: Конструктор класса, инициализирующий объект магазина.
- Параметры:
  - «std::string name»: Название магазина.

8.3. «get_name»
- Описание: Метод возвращает название магазина.

- Параметры: Отсутствуют.
- Возвращаемое значение: Название магазина.

9. «subscriber» class

Методы:

9.1. Конструктор «subscriber»
- Описание: Конструктор класса, инициализирующий объект абонента.
- Параметры:
  - «std::string fio»: ФИО абонента.
  - «int number»: Номер абонента.
  - «int id»: Идентификатор абонента.

10. «sub_table» class

Методы:

10.1. «get_length»
- Описание: Метод возвращает текущую длину таблицы абонентов.
- Параметры: Отсутствуют.
- Возвращаемое значение: Длина таблицы абонентов.

10.2. «show»
- Описание: Метод возвращает строку с информацией обо всех абонентах.
- Параметры: Отсутствуют.
- Возвращаемое значение: Строка с информацией об абонентах.

10.3. «registers»
- Описание: Метод регистрирует нового абонента в таблице.
- Параметры:
  - «std::string name»: Имя абонента.
- Возвращаемое значение: Отсутствует.

10.4. «find»
- Описание: Метод ищет абонента по имени в таблице.
- Параметры:
  - «std::string name»: Имя абонента.
- Возвращаемое значение: Указатель на объект абонента или «nullptr», если абонент не найден.

10.5. «get»
- Описание: Метод возвращает указатель на объект абонента по индексу.
- Параметры:
  - «int index»: Индекс абонента в таблице.

- Возвращаемое значение: Указатель на объект абонента.

11. «DailyRentalStrategy» class

Методы:

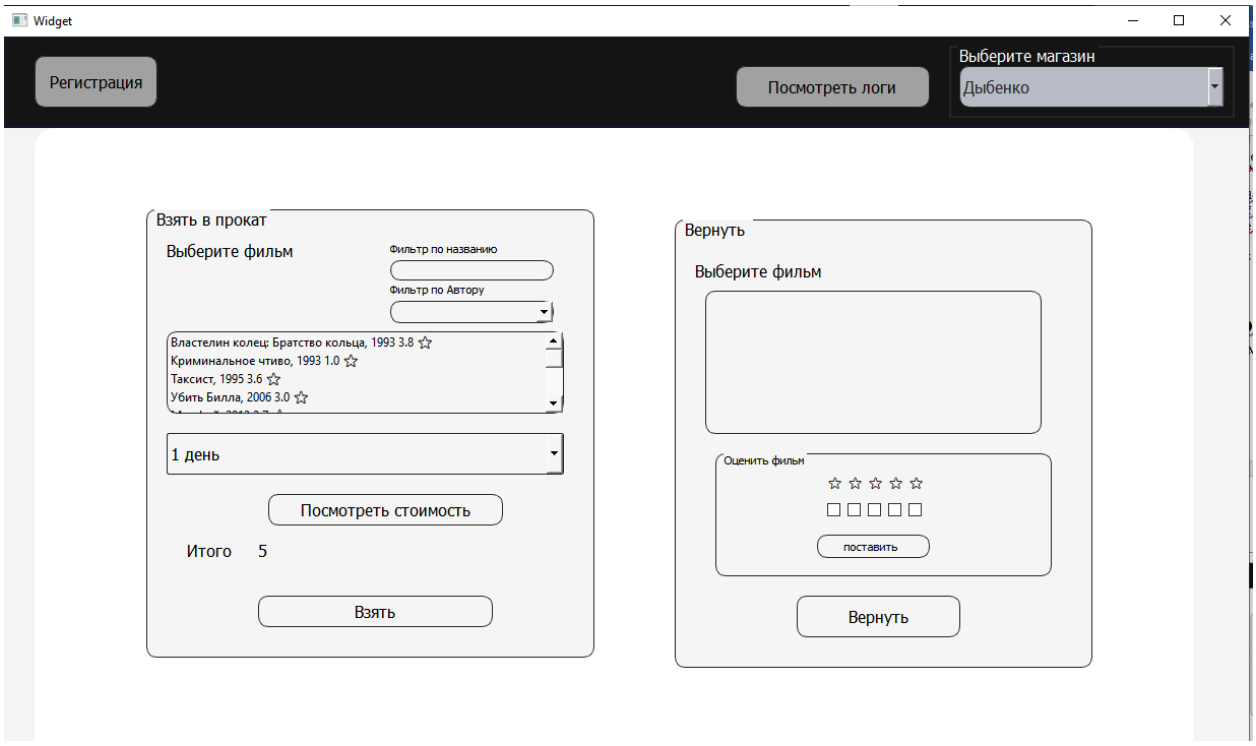
11.1 «CalculateRentalCost»

- Описание: Метод реализует стратегию расчета стоимости ежедневной аренды.

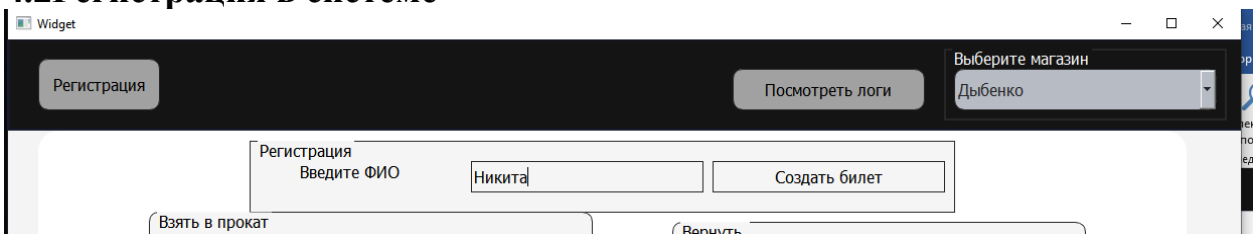
- Параметры:
  - «int rentalTime»: Количество дней аренды.
- Возвращаемое значение: Стоимость аренды на указанное количество дней.

4 .Тестирование программы

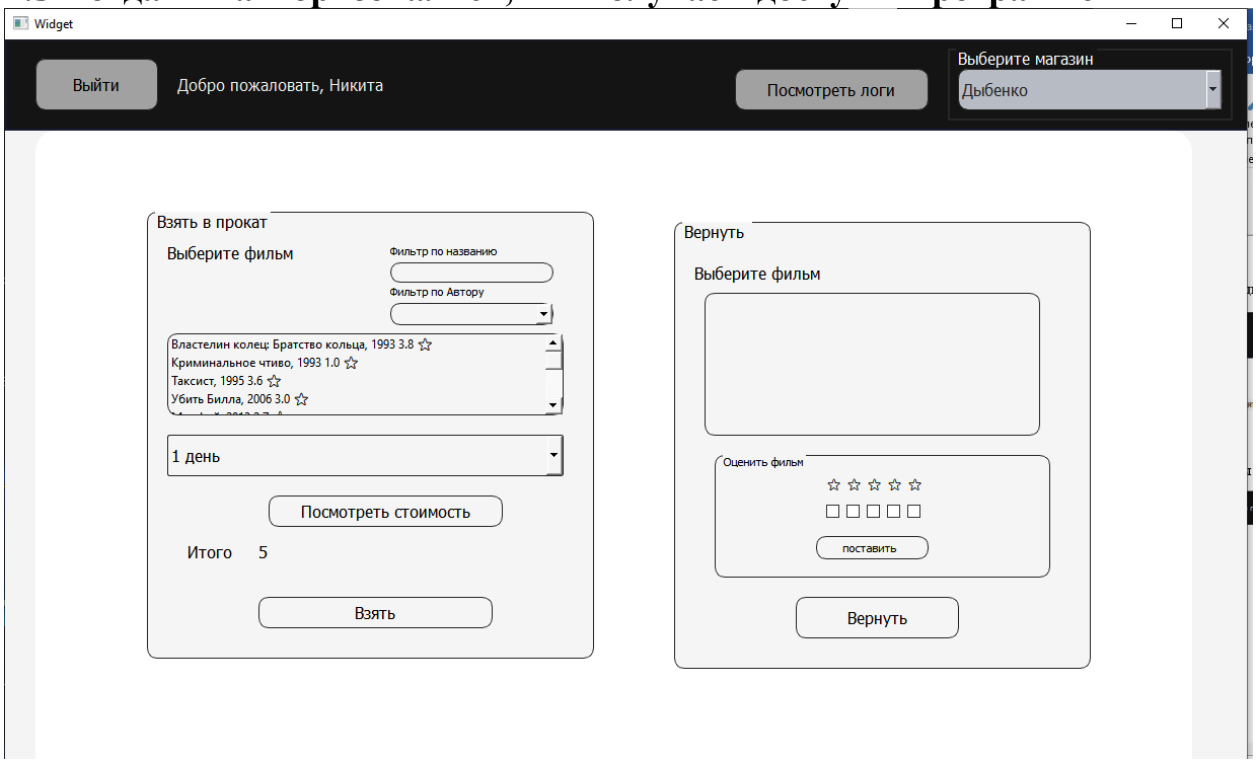
4.1 Главное меню



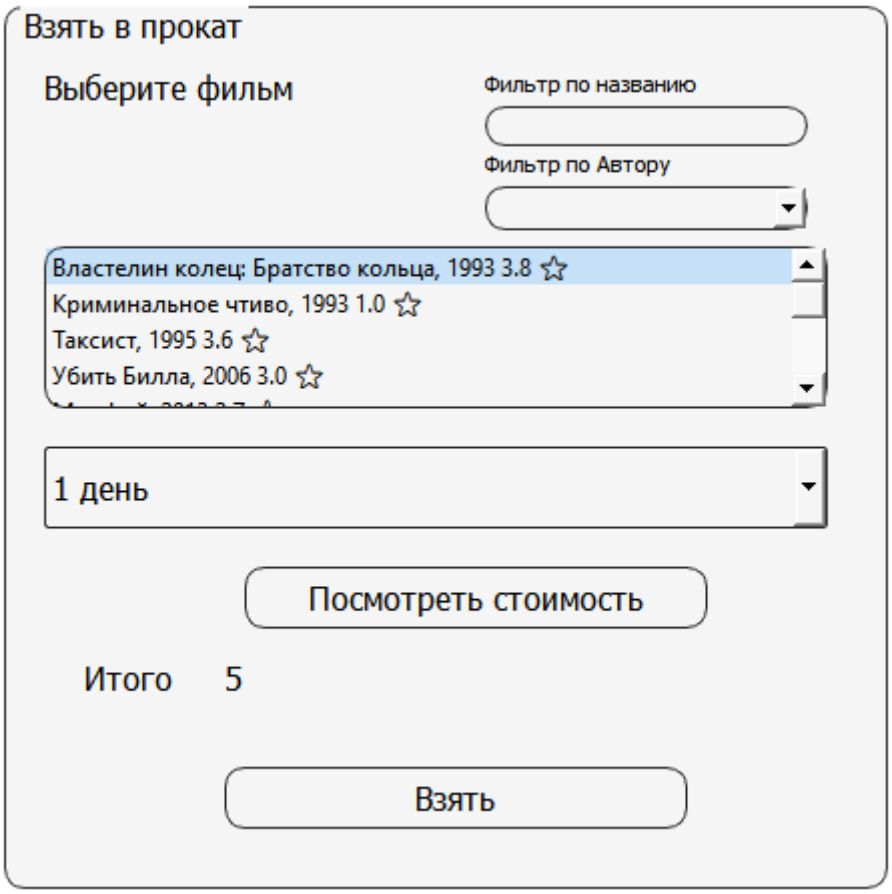
4.2Регистрация в системе



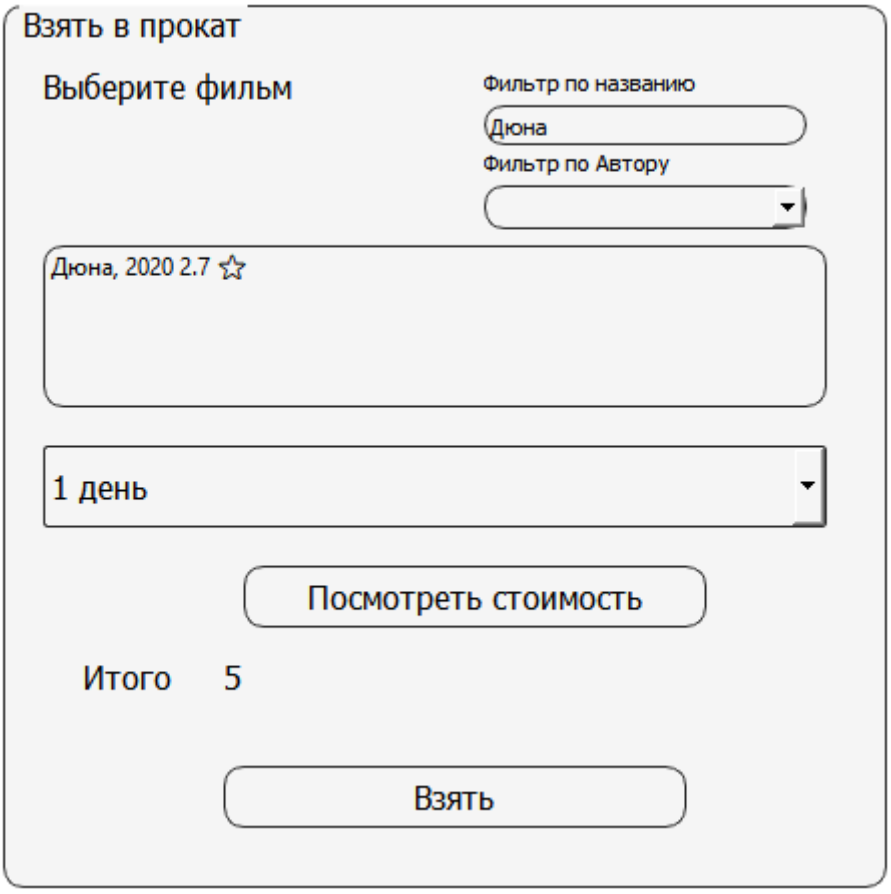
4.3 Когда мы авторизовались, мы получаем доступ к программе



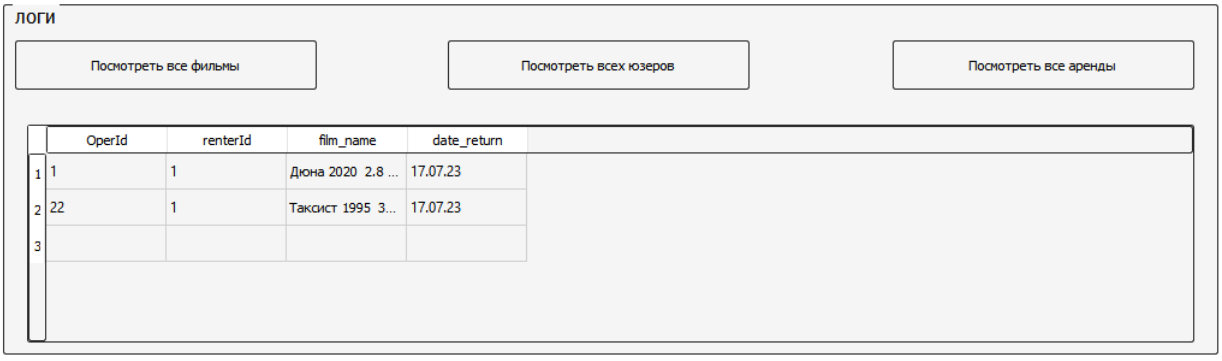
4.5 Выбираю себе фильм



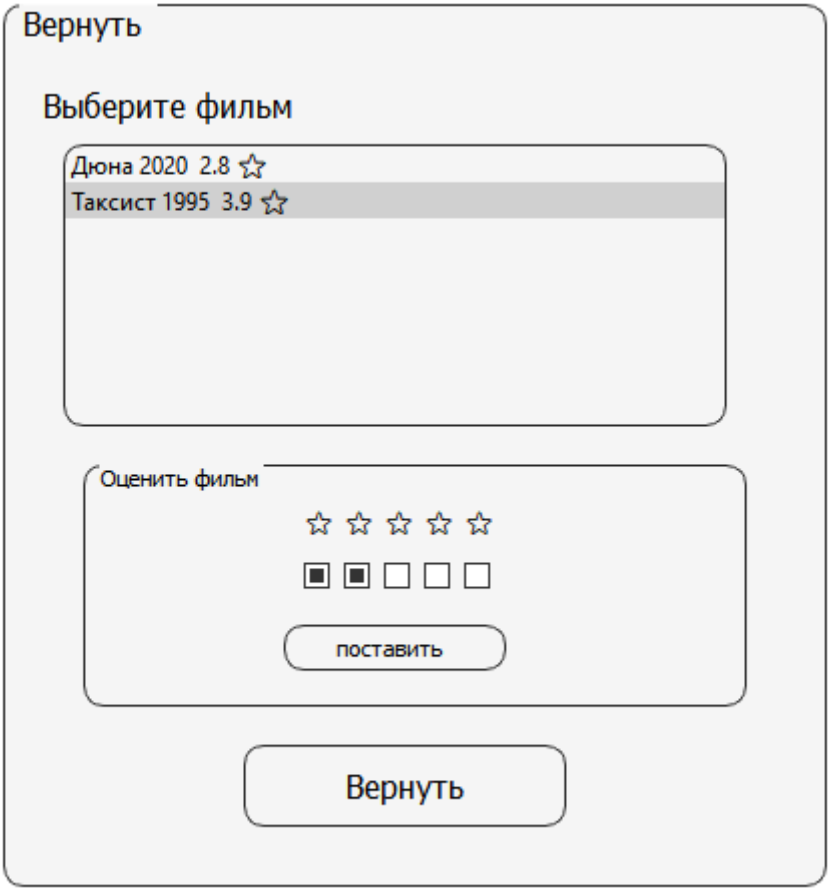
4.6 Ищу фильм Дюна



4.7 В логах появилась запись

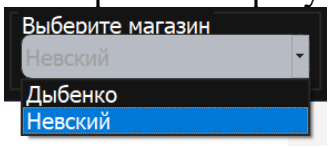


4.8 Также фильм можно вернуть и поставить ему оценку

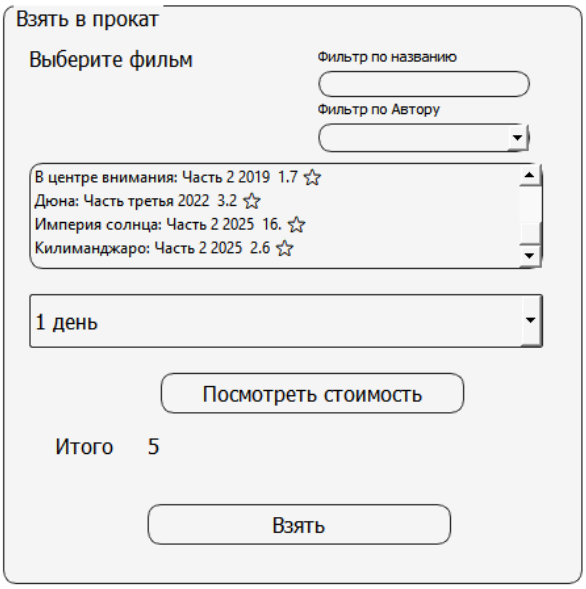


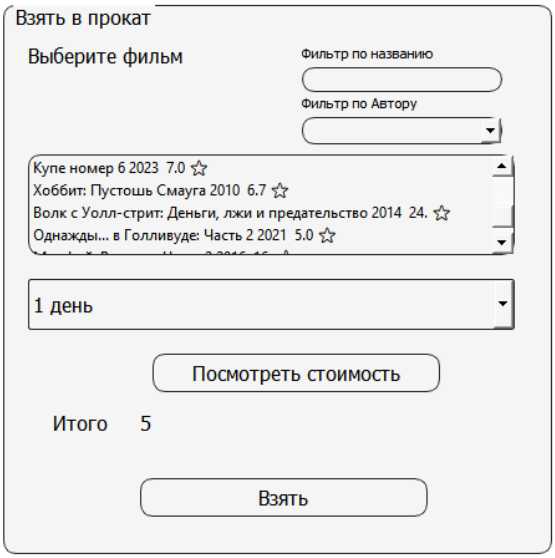
4.9 В разных магазинах разные фильмы:

Выбираем интересующий

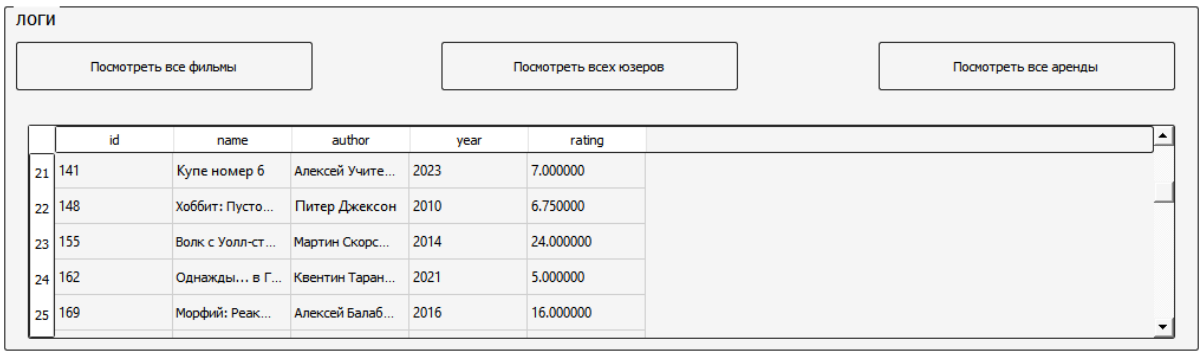


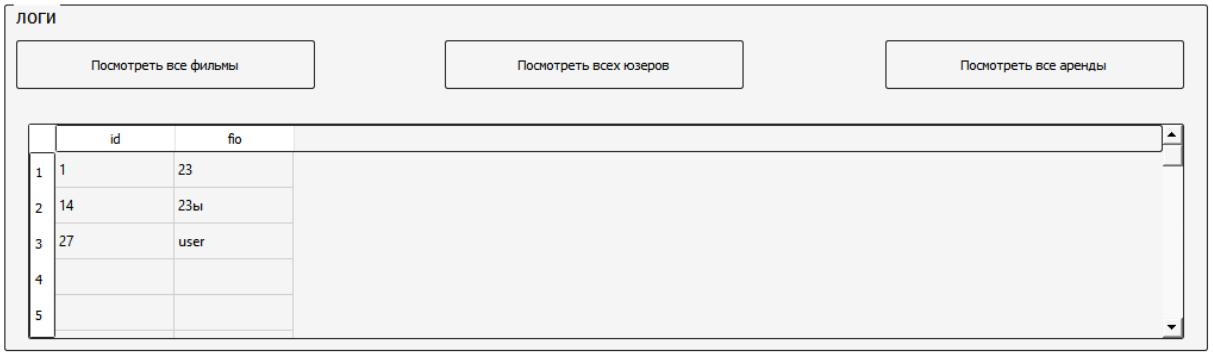
И смотрим на каталог



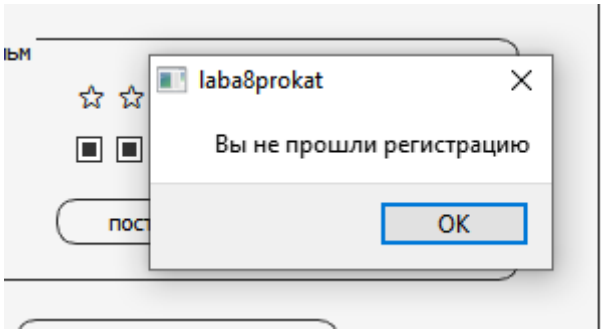


4.10 В логах тоже можно посмотреть на информацию

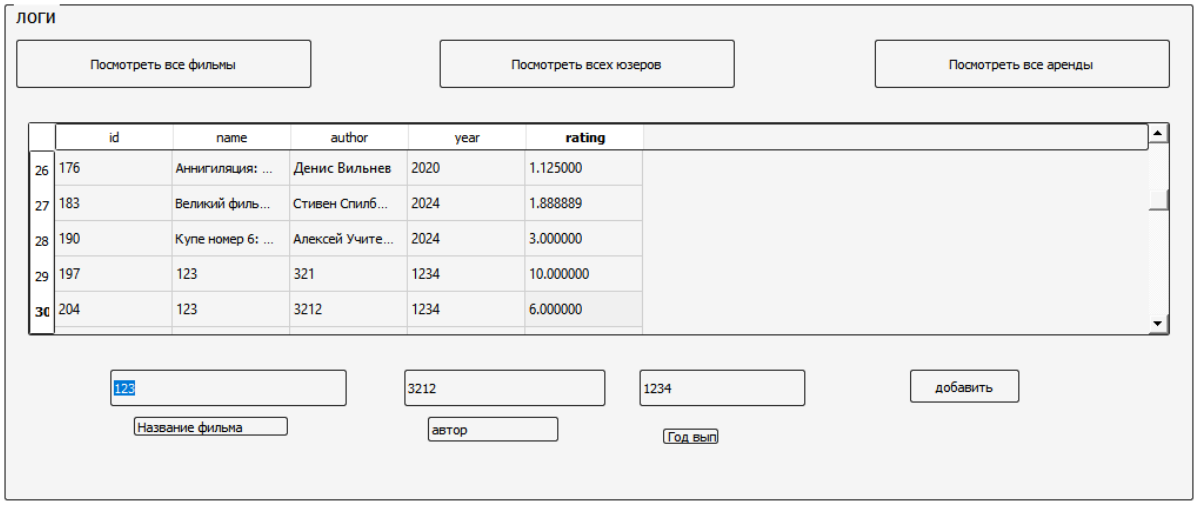




4.11 Если неавторизованный пользователь попробует что-то сделать, он получит сообщение



4.12 Фильмы также можно и добавлять



Приложение

dailyrentalstrategy.h

```
ifndef DAILYRENTALSTRATEGY_H
define DAILYRENTALSTRATEGY_H
include "rentalstrategy.h"
class DailyRentalStrategy : public IRentalStrategy {
public:
    double CalculateRentalCost(int rentalTime) override {
        return rentalTime * 5.0;
    }
};
endif // DAILYRENTALSTRATEGY_H
```

dailyrentalstrategyh.cpp

```
include "dailyrentalstrategyh.h"

dailyrentalstrategyh::dailyrentalstrategyh()
{

}
```

main.cpp

```
include "widget.h"
include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Widget w;
    w.show();

    return a.exec();
}
```

movies.h

```
ifndef MOVIES_H
define MOVIES_H
include <string>


class movie{
public:
    int number;
    int id, year;
     std::string author, name;
    bool state; //0 - на складе, 1 - выдан
    std::string front ;
    float cnt_rates = 1+rand()%9;
    float sum_rates = cnt_rates+rand()%25;
    float rating = sum_rates/cnt_rates ;
    void set_rate( int stars);
    movie* next= nullptr;

    movie(int num=0, int i=0, int y=0, std::string auth=0, std::string n=0, bool stat
=1){
        number=num;id=i; year = y; author = auth; name = n; state = stat;
        std::string rat = "";
        for (int j = 0; j<3; j++){
            rat+= std::to_string(this->rating)[j];
```

```
        }
        front= name +" "+std::to_string(year)+" "+" "+rat+" ☆";
    }

};

class movies{
private:
    int count;
    movie* head = nullptr;

public:
    void add(int y, std::string auth, std::string n);
    movie* get(int index);
    int get_length(){return count;}

    float get_rating(int index);
    movie* get_by_name(std::string name);
    movies(){

    }
};

movie* movies::get_by_name(std::string name)
{

    movie* current = head;
    int i = 0;
    while (i<count){
        if(current->front==name)return current;
        current=current->next;
        i++;
    }
    return nullptr;
}

movie* movies::get(int index){
    if(count<index) return nullptr;
    movie* current = head;
    int i = 0;
    while (i<index){
        current=current->next;
        i++;
    }
```

```
    return current;
}

void movie::set_rate(int stars){

    this->cnt_rates++;
    this->sum_rates+=stars;
    this->rating =  this->sum_rates/ this->cnt_rates;
    std::string rat = "";
    for (int j = 0; j<3; j++){
        rat+= std::to_string(this->rating)[j];

    }
    this-> front= name +" "+std::to_string(year)+" "+" "+rat+" ☆";

}
float movies::get_rating(int index ){
    movie* trg = get(index);
    return trg->rating;

}

void movies::add(int y, std::string auth, std::string n){
    int number = get_length();
    int id =1+ number*7;
    movie* sub = new movie(number, id, y, auth, n ,1);

    if(head==nullptr){
        head = sub;
        count++;
        return;
    }
    movie* current = head;
    while(current->next!=nullptr){
        current = current->next;
    }
    current->next = sub;
    count++;
    return;
}

endif // MOVIES_H
```

rental.h

```
ifndef RENTAL_H
```

```
define RENTAL_H
include "rentalstrategy.h"
include "weeklystrategy.h"
include "dailyrentalstrategy.h"



class Rental {
private:
    IRentalStrategy* _rentalStrategy;

public:
    Rental(IRentalStrategy* rentalStrategy) : _rentalStrategy(rentalStrategy) {}

    double CalculateRentalCost(int rentalTime) {
        return _rentalStrategy->CalculateRentalCost(rentalTime);
    }
};

//int main() {
    //auto dailyRental = new Rental(new DailyRentalStrategy());
    //std::cout << "Daily rental cost for 3 days: " << dailyRental-
>CalculateRentalCost(3) << std::endl;

   // auto weeklyRental = new Rental(new WeeklyRentalStrategy());
   // std::cout << "Weekly rental cost for 2 weeks: " << weeklyRental-
>CalculateRentalCost(2) << std::endl;

   // delete dailyRental;
   // delete weeklyRental;

   // return 0;
//}
endif // RENTAL_H
```

rentalmovieadapter.h

```
ifndef RENTALMOVIEADAPTER_H
define RENTALMOVIEADAPTER_H
include "rentalmovies.h"
include "movies.h"
include "subscriber.h"

class RentalMoviesAdapter : public RentalMovies {
private:
    movie movieInfo;
```

```
    subscriber customerInfo;

public:
    RentalMoviesAdapter(movie movieInfo, subscriber customerInfo) {
        this->movieInfo = movieInfo;
        this->customerInfo = customerInfo;
    }

    void rentMovie(string movieName) override {
            }

    void returnMovie(string movieName) override {
        cout << "The movie " << movieName << " has been returned by " <<
customerInfo.name << "." << endl;
    }
};

endif // RENTALMOVIEADAPTER_H
```

rentalmovies.h

```
ifndef ARENDA_H
define ARENDA_H
include <string>
include "movies.h"
include "subscriber.h"
include <iostream>
class info{
public:
    int renterId;
    int filmId;
    int OperId;
    std::string date_take;
    std::string date_return;
    std::string film_name;
    info* next = nullptr;

    info(movie* mov, subscriber* sub){
        renterId = sub->id;
        filmId = mov->id;
        film_name = mov->front;
        date_take = "16.07.23";
        date_return = "17.07.23";
    }
};
```

```
class RentalMovies {
private:
    int count;


public:
    info* head= nullptr;
    int get_length(){return count;}
    std::string show();
    void add(info *input);
    info* find(std::string name, int sub_id);
    void return_film(std::string name, subscriber* sub);
    std::string* get_all_from_sub(subscriber* sub);
    info* get(int index);
};

info* RentalMovies::get(int index){

        if(count<index) return nullptr;
        info* current = head;
        std::cout<<current->film_name;
        int i = 0;
        while (i<index){
            current=current->next;
            i++;
            std::cout<<current->film_name;
        }

        return current;

}

void RentalMovies::return_film(std::string name, subscriber* sub){
    info* to_remove = find(name, sub->id);
    info* current = head;
    if(current == to_remove){
        head = head->next;
        count--;
        return;

    }

    while(current->next!= to_remove){
        current = current->next;
    }
```

```
    current->next=current->next->next;
    count--;
}

info* RentalMovies::find(std::string name, int sub_id){
    info* current = head;
    while (current!=nullptr){
        if(current->film_name==name && current->renterId == sub_id) return
current;
        current= current->next;
    }
    return nullptr;
}

std::string RentalMovies::show(){
    std::string res = "";
    info* current = head;
    while(current!=nullptr){
        res += std::to_string(current->OperId)+": "+" "+current->film_name+", ";
        current = current->next;
    }
    return res;
}

void RentalMovies::add(info* input){
    if(!input) return;
    int number = get_length();
    int id = 1+number*21;
    input->OperId= id;
    info* sub = input;

    if(head==nullptr){
        head = sub;
        count++;
        return;
    }
    info* current = head;
    while(current->next!=nullptr){
        current = current->next;
    }
    current->next = sub;
    count++;
    return;
}
endif
```

rentalstrategy.h

```
ifndef RENTALSTRATEGY_H
define RENTALSTRATEGY_H

class IRentalStrategy {
public:
   virtual double CalculateRentalCost(int rentalTime) = 0;
};

endif // RENTALSTRATEGY_H
```

shop.cpp

```
include "shop.h"
include "movies.h"

shop::shop()
{
   public:
   movies catalog;
}
```

shop.h

```
ifndef SHOP_H
define SHOP_H
include "string.h"

class shop
{
public:
   movies catalog;
   std::string name;

   void set_name(std::string name){
      this->name = name;
   }

   shop(std::string name){
      set_name(name);

   }


   std::string get_name(){
      return name;
```

```
    }
};

endif // SHOP_H
```

subscriber.h

```
ifndef SUBSCRIBER_H
define SUBSCRIBER_H
include <string>
class subscriber{
public:
    int number;
    int id;
    std::string fio;

    subscriber* next = nullptr;
    subscriber(std::string fio, int number, int id){
        this->fio = fio;
        this->id = id;
        this->number = number;
    }

};

class sub_table{
  private:
    subscriber* head = nullptr;
    int count = 0;

  public:

    int get_length();
    std::string show();
    void registers(std::string name);
    subscriber* find(std::string name);
    subscriber* get(int index);
};

subscriber* sub_table::get(int index){
    subscriber* current = head;
    for(int i = 0; i<index; i++){
        current=current->next;
    }
    return current;
```

```
}

std::string sub_table::show(){

    std::string res= "";
    subscriber* current = head;
    while(current!=nullptr){
        res = res + current->fio+", ";
        current = current->next;
    }
    return res;
}


subscriber* sub_table::find(std::string name){

    if(name == "")return nullptr;
    subscriber* current = head;
    while(current!=nullptr){
        if (current->fio == name){
            return current;
        }
        current = current->next;
    }
    return nullptr;
}

void sub_table::registers(std::string name){

    int number = get_length();
    int id = 1+ number*13;
    subscriber* sub = new subscriber(name, number, id);

    if(head==nullptr){
        head = sub;
        count++;
        return;
    }
    subscriber* current = head;
    while(current->next!=nullptr){
        if  (current->fio==name){
            return;
        }
        current = current->next;
```

```
    }
    if  (current->fio==name){
        return;
    }
    current->next = sub;
    count++;
    return;

}

int sub_table::get_length(){
    return count;
}

endif // SUBSCRIBER_H
```

weeklystrategy.h

```
ifndef WEEKLYRENTALSTRATEGY_H
define WEEKLYRENTALSTRATEGY_H
include "rentalstrategy.h"
class WeeklyRentalStrategy : public IRentalStrategy {
public:
    double CalculateRentalCost(int rentalTime) override {
        return rentalTime * 20.0;
    }
};
endif // WEEKLYRENTALSTRATEGY_H
```

widget.cpp

```
include "widget.h"
include "ui_widget.h"
include "subscriber.h"
include "movies.h"
include "rental.h"
include "rentalmovies.h"
include "shop.h"
include "string.h"
include <QMessageBox>
sub_table subs;

RentalMovies RM;
shop Dibenko("Дыбенко");
shop Nevskiy("Невский");
std::string cur_user="";
```

```
int mark_stars(Ui::Widget *ui, int st){
    QCheckBox *stars[] = {ui->checkBox, ui->checkBox_2, ui->checkBox_3, ui->checkBox_4, ui->checkBox_5};
   for (int i =0;i<5 ; i++){
      stars[i]->setCheckState(Qt::CheckState(0));
   }
   for (int i =0;i<st ; i++){
      stars[i]->setCheckState(Qt::CheckState(1));
   }
   return st;

}
int last_stared ;

void update_authors(Ui::Widget *ui){
   ui->listWidget_catalog->clear();

   shop * current;
   if(ui->comboBox__shop_select->currentIndex() == 0){
    current =&Dibenko;
   }
   else{
      current =&Nevskiy;
   }
   ui->comboBox_authors->clear();
   ui->comboBox_authors->addItem(QString::fromStdString(""));
   for (int i = 0; i< current->catalog.get_length(); i++){
      QStringList itemsInComboBox;
      for (int index = 0; index < ui->comboBox_authors->count(); index++)
         itemsInComboBox <<ui->comboBox_authors->itemText(index);
      if(!itemsInComboBox.contains(QString::fromStdString( current->catalog.get(i)->author))) {
        ui->comboBox_authors->addItem(QString::fromStdString( current->catalog.get(i)->author));
      }

   }
}

void update_films(Ui::Widget *ui){
   ui->listWidget_catalog->clear();

   shop * current;
   if(ui->comboBox__shop_select->currentIndex() == 0){
    current =&Dibenko;
```

```
    }
    else{
        current =&Nevskiy;
    }



    std::string filter = ui->lineEdit_search->text().toStdString();
    std::string author = ui->comboBox_authors->currentText().toStdString();
    if(filter==""){
        for(int i = 0; i< current->catalog.get_length(); i++){
            std::string rat = "";
            for (int j = 0; j<3; j++){
                rat+= std::to_string(current->catalog.get(i)->rating)[j];
            }
            if(author==current->catalog.get(i)->author || author==""){
                std::string name = current->catalog.get(i)->front  ;
                ui->listWidget_catalog->addItem(QString::fromStdString(name));
            }
            }
    }
     else{
            for(int i = 0; i< current->catalog.get_length(); i++){
                std::string rat = "";
                for (int j = 0; j<3; j++){
                    rat+= std::to_string(current->catalog.get(i)->rating)[j];
                }
                std::string name = current->catalog.get(i)->front;
                bool flag = 0;
                int counter = 0;

                for (int j = 0; j<name.length(); j++){
                    if(name[j]==filter[counter]){
                        counter++;
                    }else{
                        counter =0;
                    }
                    if (counter == filter.length()){
                        if(author==current->catalog.get(i)->author || author==""){
                            std::string name = current->catalog.get(i)->front;
                            ui->listWidget_catalog-
>addItem(QString::fromStdString(name));
                        }
```

```
            }
        }
    }
}


Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);

     ui->comboBox__shop_select->addItem("Дыбенко");
     ui->comboBox__shop_select->addItem("Невский");



    ui->comboBox_period->addItem("1 день");
    ui->comboBox_period->addItem("2 дня");
    ui->comboBox_period->addItem("1 неделя");


    Dibenko.catalog.add(1993, "Питер Джексон", "Властелин колец: Братство кольца" );
    Dibenko.catalog.add(1993, "Квентин Тарантино", "Криминальное чтиво" );
    Dibenko.catalog.add(1995, "Мартин Скорсезе", "Таксист"                    );
    Dibenko.catalog.add(2006, "Квентин Тарантино", "Убить Билла");
    Dibenko.catalog.add(2013, "Алексей Балабанов", "Морфий");
    Dibenko.catalog.add(2018, "Денис Вильнев", "Бегущий в лабиринте");
    Dibenko.catalog.add(2000, "Стивен Спилберг", "Список Шиндлера");
    Dibenko.catalog.add(2014, "Алексей Учитель", "Кислород");


    Nevskiy.catalog.add(1997, "Фрэнк Дарабонт", "Темная башня"     );
    Nevskiy.catalog.add(1990, "Дэвид Кроненберг", "История насилия");
    Nevskiy.catalog.add(1999,"Найт Шьямалан", "Шестое чувство" );
    Nevskiy.catalog.add(2006, "Квентин Тарантино", "Убить Билла");
    Nevskiy.catalog.add(2013, "Алексей Балабанов", "Морфий");
    Nevskiy.catalog.add(2018, "Денис Вильнев", "Бегущий в лабиринте");

    Dibenko.catalog.add(2003, "Питер Джексон", "Властелин колец: Возвращение короля" );
    Dibenko.catalog.add(2005, "Мартин Скорсезе", "Отступники"                );
```

```
    Dibenko.catalog.add(2010, "Квентин Тарантино", "Начало" );
    Dibenko.catalog.add(2020, "Денис Вильнев", "Дюна" );
    Dibenko.catalog.add(2022, "Стивен Спилберг", "Вестсайдская история"
);
    Dibenko.catalog.add(2022, "Алексей Учитель", "Летят журавли" );

    Nevskiy.catalog.add(2002, "Фрэнк Дарабонт", "Истинное детективное
агентство" );
    Nevskiy.catalog.add(2008, "Дэвид Кроненберг", "Опасные методы" );
    Nevskiy.catalog.add(2011, "Найт Шьямалан", "Девушка из табора" );
    Nevskiy.catalog.add(2017, "Денис Вильнев", "Блейд Раннер 2049" );
    Nevskiy.catalog.add(2020, "Стивен Спилберг", "Не смотрите наверх"
);
    Nevskiy.catalog.add(2022, "Алексей Учитель", "Плохие парни 3" );
    Dibenko.catalog.add(2008, "Питер Джексон", "Хоббит: Нежданное
путешествие" );
    Dibenko.catalog.add(2012, "Мартин Скорсезе", "Волк с Уолл-стрит" );
    Dibenko.catalog.add(2019, "Квентин Тарантино", "Однажды... в Голливуде"
);
    Dibenko.catalog.add(2007, "Алексей Балабанов", "Морфий: Реактор"
);
    Dibenko.catalog.add(2019, "Денис Вильнев", "Аннигиляция" );
    Dibenko.catalog.add(2023, "Стивен Спилберг", "Великий фильм" );
    Dibenko.catalog.add(2023, "Алексей Учитель", "Купе номер 6" );

    Nevskiy.catalog.add(2009, "Фрэнк Дарабонт", "Ходячие мертвецы" );
    Nevskiy.catalog.add(2013, "Дэвид Кроненберг", "Космическая одиссея 2001
года" );
    Nevskiy.catalog.add(2015, "Найт Шьямалан", "В центре внимания" );
    Nevskiy.catalog.add(2021, "Денис Вильнев", "Дюна: Часть вторая" );
    Nevskiy.catalog.add(2023, "Стивен Спилберг", "Империя солнца"
);
    Nevskiy.catalog.add(2023, "Алексей Учитель", "Килиманджаро" );
    Dibenko.catalog.add(2010, "Питер Джексон", "Хоббит: Пустошь Смауга" );
    Dibenko.catalog.add(2014, "Мартин Скорсезе", "Волк с Уолл-стрит: Деньги,
лжи и предательство" );
    Dibenko.catalog.add(2021, "Квентин Тарантино", "Однажды... в Голливуде:
Часть 2" );
    Dibenko.catalog.add(2016, "Алексей Балабанов", "Морфий: Реактор: Часть
2" );
    Dibenko.catalog.add(2020, "Денис Вильнев", "Аннигиляция: Часть 2" );
    Dibenko.catalog.add(2024, "Стивен Спилберг", "Великий фильм: Часть 2" );
    Dibenko.catalog.add(2024, "Алексей Учитель", "Купе номер 6: Часть 2" );

    Nevskiy.catalog.add(2011, "Фрэнк Дарабонт", "Ходячие мертвецы: Пима" );
```

```
    Nevskiy.catalog.add(2017, "Дэвид Кроненберг", "Космическая одиссея 2001 года: Часть 2" );
    Nevskiy.catalog.add(2019, "Найт Шьямалан", "В центре внимания: Часть 2" );
    Nevskiy.catalog.add(2022, "Денис Вильнев", "Дюна: Часть третья" );
    Nevskiy.catalog.add(2025, "Стивен Спилберг", "Империя солнца: Часть 2" );
    Nevskiy.catalog.add(2025, "Алексей Учитель", "Килиманджаро: Часть 2" );


    ui->label_welcome->setHidden(1);
    ui->groupBox_logs->setHidden(1);
    ui->groupBox_registration->setHidden(1);
    ui->pushButton_quit->setHidden(1);
    update_films(ui);
    update_authors(ui);
    ui->label_cost->setText(QString::fromStdString("5"));



}



Widget::~Widget()
{
    delete ui;
}



void Widget::on_pushButton_check_cost_clicked()
{
    double cost;
    auto rent = new Rental(new DailyRentalStrategy());
    int period = ui->comboBox_period->currentIndex();
    switch (period) {
    case 0:
        rent = new Rental(new DailyRentalStrategy());
        cost = rent->CalculateRentalCost(1);
        break;
    case 1:
         rent = new Rental(new DailyRentalStrategy());
        cost = rent->CalculateRentalCost(2);
        break;
    case 2:
```

```
        rent = new Rental(new WeeklyRentalStrategy());
        cost = rent->CalculateRentalCost(1);
        break;
    default:
        break;
    }

    ui->label_cost->setText(QString::number(cost));
}

void Widget::on_pushButton_take_clicked()
{
    shop * current;
    if(ui->comboBox__shop_select->currentIndex() == 0){
      current =&Dibenko;
    }
    else{
        current =&Nevskiy;
    }
    std::string name =cur_user;
    if(name=="") {
        QMessageBox msgBox;
        msgBox.setText("Вы не прошли регистрацию");
        msgBox.exec();
        return;
    }


    subscriber* sub = subs.find(name);
    movie* mov = current->catalog.get_by_name(ui->listWidget_catalog-
>currentItem()->text().toStdString());
    info * input = new info(mov, sub);
    RM.add(input);

    ui->listWidget_film_return->clear();

    sub = subs.find(name);
    if (sub==nullptr) return;

    info* cur = RM.head;
    std::string res ;
    while (cur){
        if(cur->renterId == sub->id)
        {res= cur->film_name;
        ui->listWidget_film_return->addItem(QString::fromStdString(res));}
```

```
        cur = cur->next;
    }


}



void Widget::on_pushButton_return_clicked()
{
    std::string name = cur_user;
    if(name=="") {
        QMessageBox msgBox;
        msgBox.setText("Вы не прошли регистрацию");
        msgBox.exec();
        return;
    }
    subscriber* sub = subs.find(name);
    std::string mov = ui->listWidget_film_return->currentItem()-
>text().toStdString();


    RM.return_film(mov, sub);

    info* cur = RM.head;
    std::string res ;
    ui->listWidget_film_return->clear();
    while (cur){
        if(cur->renterId == sub->id)
        {res= cur->film_name;
        ui->listWidget_film_return->addItem(QString::fromStdString(res));}
        cur = cur->next;
    }


}

void Widget::on_pushButton_check_clicked()
{

}

void Widget::on_pushButton_show_films_clicked()

{
```

```
    std::string outp = "Класс Фильмов:\nid\tНазвание\t\t\t\tАвтор\t\tгод";

    shop * current;
    if(ui->comboBox__shop_select->currentIndex() == 0){
      current =&Dibenko;
    }
    else{
        current =&Nevskiy;
    }
    ui->tableWidget_logs->clear();
    ui->tableWidget_logs->setColumnCount(5);
    QStringList headers;
    headers<<"id" << "name" << "author"<<"year" <<"rating";
    ui->tableWidget_logs->setHorizontalHeaderLabels(headers);
    for(int i = 0; i<current->catalog.get_length();i++){
         ui->tableWidget_logs->insertRow(i);
            QTableWidgetItem *item=new QTableWidgetItem
(QString::fromStdString(current->catalog.get(i)->name));
            QTableWidgetItem *item2=new QTableWidgetItem
(QString::fromStdString(current->catalog.get(i)->author));
            QTableWidgetItem *item3=new QTableWidgetItem
(QString::fromStdString(std::to_string(current->catalog.get(i)->year)));
            QTableWidgetItem *item4=new QTableWidgetItem
(QString::fromStdString(std::to_string(current->catalog.get(i)->rating)));
            QTableWidgetItem *item5=new QTableWidgetItem
(QString::fromStdString(std::to_string(current->catalog.get(i)->id)));

            ui->tableWidget_logs->setItem(i, 0, item5);
            ui->tableWidget_logs->setItem(i, 1, item);
            ui->tableWidget_logs->setItem(i, 2, item2);
            ui->tableWidget_logs->setItem(i, 3, item3);
            ui->tableWidget_logs->setItem(i, 4, item4);



    }

}

void Widget::on_pushButton_show_films_2_clicked()
{
```

```
    ui->tableWidget_logs->clear();
    ui->tableWidget_logs->setColumnCount(2);
    QStringList headers;
    headers << "id" << "fio" ;
    ui->tableWidget_logs->setHorizontalHeaderLabels(headers);
    for(int i = 0; i< subs.get_length();i++){
        ui->tableWidget_logs->insertRow(i);
          QTableWidgetItem *item=new QTableWidgetItem
(QString::fromStdString(std::to_string(subs.get(i)->id)));
          QTableWidgetItem *item2=new QTableWidgetItem
(QString::fromStdString(subs.get(i)->fio));

          ui->tableWidget_logs->setItem(i, 0, item);
          ui->tableWidget_logs->setItem(i, 1, item2);


    }

}

void Widget::on_pushButton_show_films_3_clicked()
{

    ui->tableWidget_logs->clear();
    ui->tableWidget_logs->setColumnCount(4);
    QStringList headers;
    headers << "OperId" << "renterId" << "film_name" << "date_return";
    ui->tableWidget_logs->setHorizontalHeaderLabels(headers);

    for(int i = 0; i< RM.get_length();i++){
        ui->tableWidget_logs->insertRow(i);
          QTableWidgetItem *item=new QTableWidgetItem
(QString::fromStdString(std::to_string(RM.get(i)->OperId)));
          QTableWidgetItem *item2=new QTableWidgetItem
(QString::fromStdString(std::to_string(RM.get(i)->renterId)));
          QTableWidgetItem *item3=new QTableWidgetItem
(QString::fromStdString(RM.get(i)->film_name));
          QTableWidgetItem *item4=new QTableWidgetItem
(QString::fromStdString(RM.get(i)->date_return));

          ui->tableWidget_logs->setItem(i, 0, item);
          ui->tableWidget_logs->setItem(i, 1, item2);
          ui->tableWidget_logs->setItem(i, 2, item3);
          ui->tableWidget_logs->setItem(i, 3, item4);
```

```
    }


}

void Widget::on_pushButton_clicked()
{
    ui->groupBox_registration->setHidden(0);

}

void Widget::on_pushButton_registrate_clicked()
{
    std::string name = ui->lineEdit_registration->text().toStdString();
    subs.registers(name);
    ui->groupBox_registration->setHidden(1);
    ui->label_welcome->setText(QString::fromStdString("Добро пожаловать,
"+name));
    ui->label_welcome->setHidden(0);
    ui->pushButton->setHidden(1);
    cur_user = name;
    ui->pushButton_quit->setHidden(0);
}

void Widget::on_pushButton_2_clicked()
{
    if (ui->groupBox_logs->isHidden()){
        ui->groupBox_logs->setHidden(0);
        ui->groupBox_ArendaTake->setHidden(1);
        ui->groupBox_ArendaReturn->setHidden(1);
        ui->pushButton_2->setText(QString::fromStdString("Скрыть логи"));
    }
    else{
        ui->groupBox_logs->setHidden(1);
        ui->pushButton_2->setText(QString::fromStdString("Показать логи"));
        ui->groupBox_ArendaTake->setHidden(0);
        ui->groupBox_ArendaReturn->setHidden(0);
    }

}

void Widget::on_comboBox__shop_select_currentIndexChanged(int index)
{
```

```
    update_films(ui);
    update_authors(ui);
}

void Widget::on_pushButton_quit_clicked()
{
    cur_user="";
    ui->pushButton_quit->setHidden(1);
    ui->pushButton->setHidden(0);
    ui->label_welcome->setHidden(1);
}

void Widget::on_pushButton_3_clicked()
{
    std::string name = ui->lineEdit_film_name->text().toStdString();
    std::string author = ui->lineEdit_film_author->text().toStdString();
    std::string year = ui->lineEdit_film_year->text().toStdString();
    shop * current;
    if(ui->comboBox__shop_select->currentIndex() == 0){
      current =&Dibenko;
    }
    else{
        current =&Nevskiy;
    }
    current->catalog.add(std::stoi(year), author, name);
}

void Widget::on_checkBox_5_clicked()
{

   last_stared= mark_stars(ui, 5);
}

void Widget::on_checkBox_4_clicked()
{
    last_stared=mark_stars(ui, 4);
}

void Widget::on_checkBox_3_clicked()
{
   last_stared= mark_stars(ui, 3);
}

void Widget::on_checkBox_2_clicked()
{
```

```
    last_stared=mark_stars(ui, 2);
}

void Widget::on_checkBox_clicked()
{
  last_stared= mark_stars(ui, 1);
}

void Widget::on_pushButton_4_clicked()
{
    shop * current;
    if(ui->comboBox__shop_select->currentIndex() == 0){
      current =&Dibenko;
    }
    else{
        current =&Nevskiy;
    }
    std::string name = ui->listWidget_film_return->currentItem()-
>text().toStdString();
    current->catalog.get_by_name(name)->set_rate(last_stared);
    update_films(ui);
    ui->pushButton_4->setEnabled(0);
}

void Widget::on_lineEdit_search_textChanged(const QString &arg1)
{
    update_films(ui);
}

void Widget::on_comboBox_authors_currentIndexChanged(int index)
{
    update_films(ui);

}

void Widget::on_listWidget_film_return_clicked(const QModelIndex &index)
{
    ui->pushButton_4->setEnabled(1);
}
```

widget.h

```
ifndef WIDGET_H
define WIDGET_H

include <QWidget>
```

```
include <string>

namespace Ui {
class Widget;
}

class Widget : public QWidget
{
   Q_OBJECT


public:

   explicit Widget(QWidget *parent = 0);
   ~Widget();

private slots:
   void on_pushButton_registrate_clicked();

   void on_pushButton_check_cost_clicked();

   void on_pushButton_take_clicked();



   void on_pushButton_return_clicked();

   void on_pushButton_check_clicked();

   void on_pushButton_show_films_clicked();

   void on_pushButton_show_films_2_clicked();

   void on_pushButton_show_films_3_clicked();

   void on_pushButton_clicked();

   void on_pushButton_2_clicked();

   void on_comboBox__shop_select_currentIndexChanged(int index);

   void on_pushButton_quit_clicked();

   void on_pushButton_3_clicked();
```

```
    void on_checkBox_5_clicked();

    void on_checkBox_4_clicked();

    void on_checkBox_3_clicked();

    void on_checkBox_2_clicked();

    void on_checkBox_clicked();

    void on_pushButton_4_clicked();

    void on_lineEdit_search_textChanged(const QString &arg1);

    void on_comboBox_authors_currentIndexChanged(int index);

    void on_listWidget_film_return_clicked(const QModelIndex &index);

private:
    Ui::Widget *ui;
};

endif // WIDGET_H
```