ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | И.М. Лозоватский |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №7  **Создание двумерного пользовательского интерфейса в среде Qt Creator** |
| **по дисциплине:  Проектирование человеко-машинного интерфейса** |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. | 4134к |  |  |  | Костяков Н.А. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург

2024

**Цель работы:** освоение кросс-платформенной интегрированной среды разработки, изучение базовых возможностей разработки и отладки приложения с двумерным пользовательским интерфейсом с использованием библиотеки Qt и компилятора MinGW **Задание:** Разработать и отладить приложение на языке С++ с графическим пользовательским интерфейсом

**Среда разработки qt creator 4.0.3 (community)**

**Листинг программы**

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:**\n**id**\t**Название**\t\t\t\t\t**Автор**\t\t**год";

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*

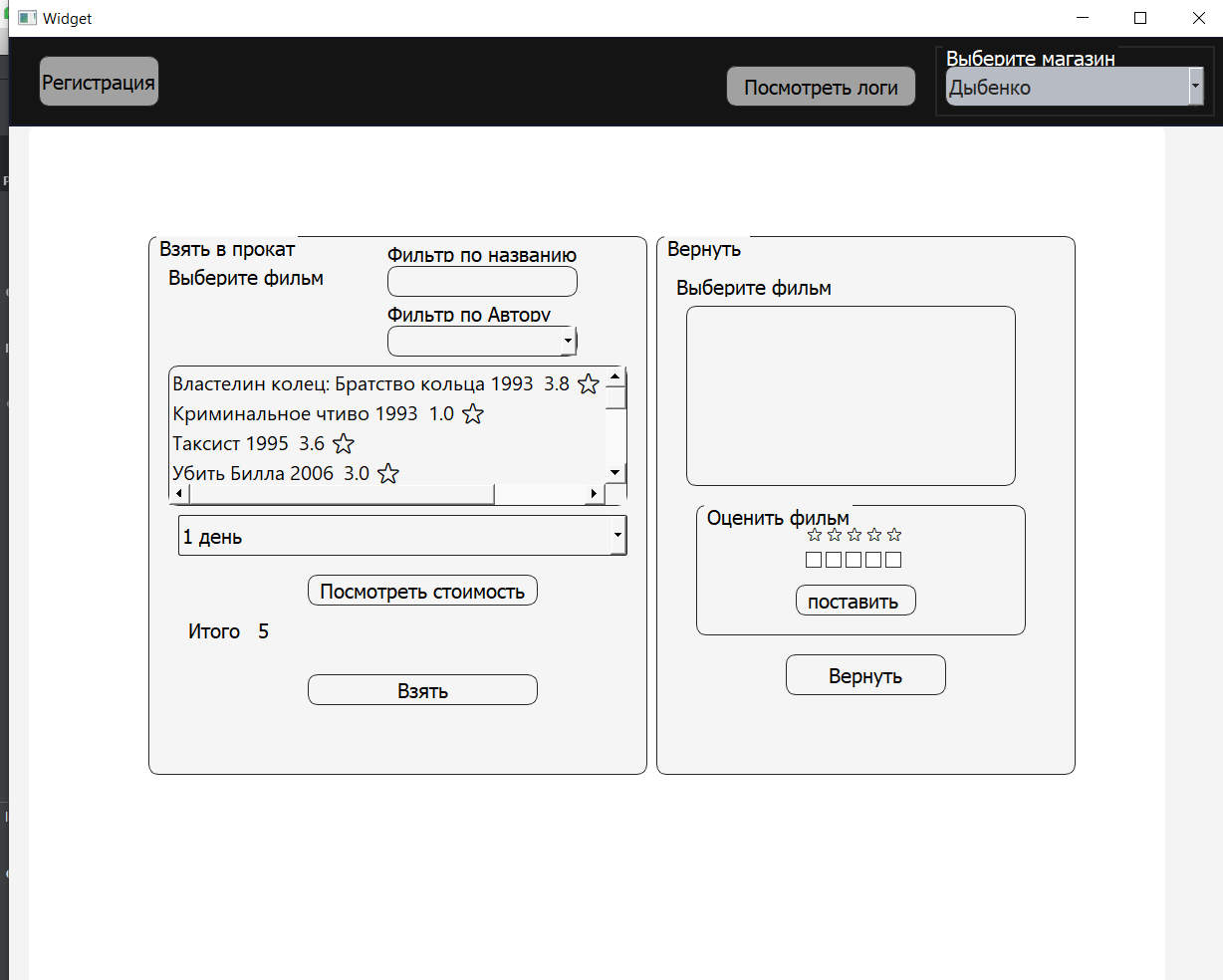
**Описание интерфейса**

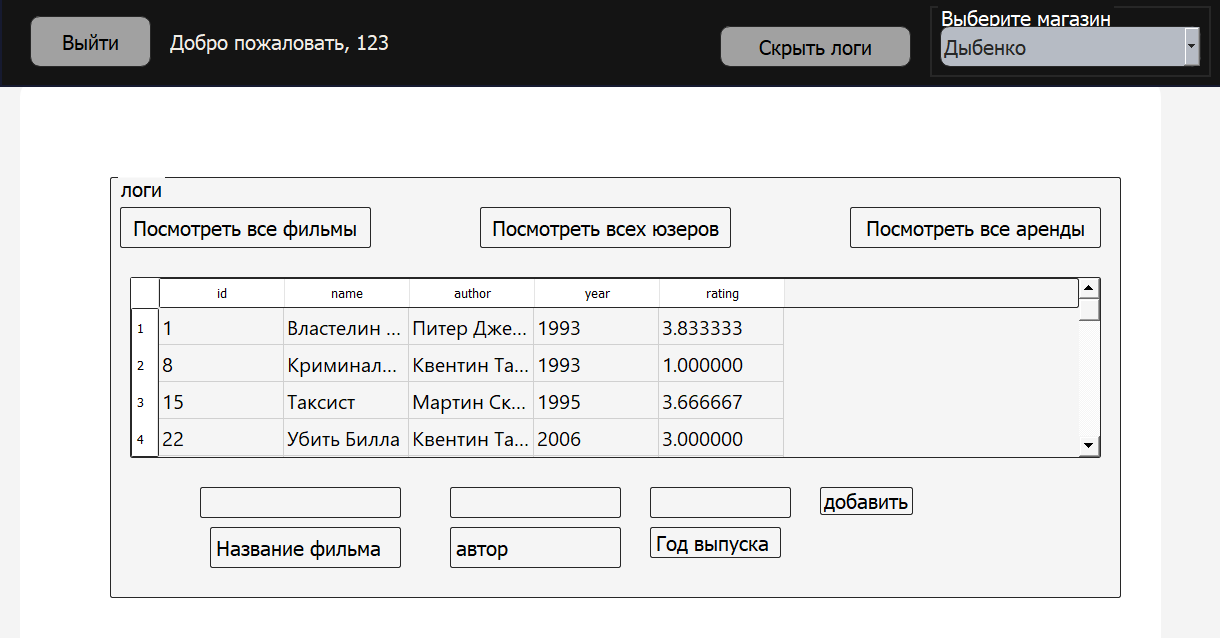
Интерфейс представляет из себя набор GroupBox которые становятся прозрачными при определенных событиях и непрозрачными при дргугих. Таким образом все действия происходят в одном окне и элементы взаимозаменяют друг друга.

ui->groupBox\_registration->setHidden(1);

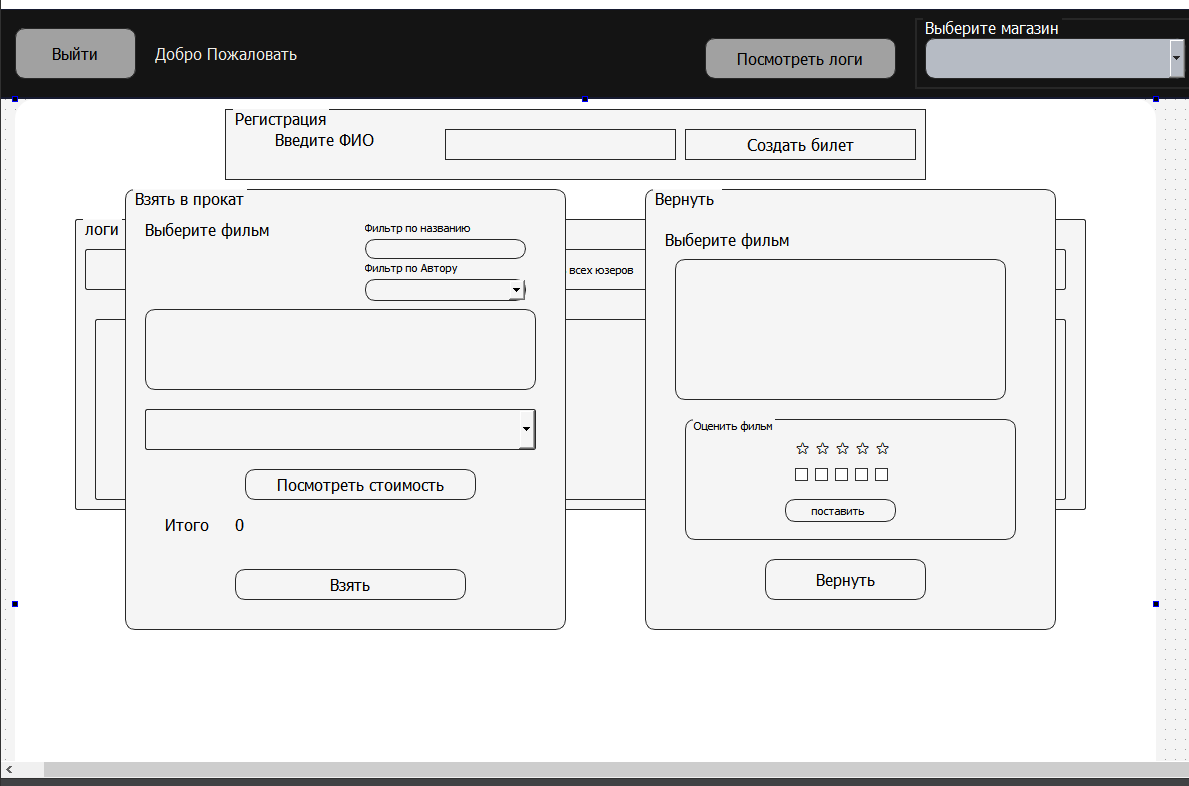
Разработались несколько форм для регистрации, которая состоит из нескольких полей ввода и кнопки для проверки, форма для того, чтобы выбрать фильм, которая наполняется в зависимости от выбранного магазина, форма возврата, аналогичная форме выбора фильма и навигационная панель, которая размещает на себе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

Скриншоты программы

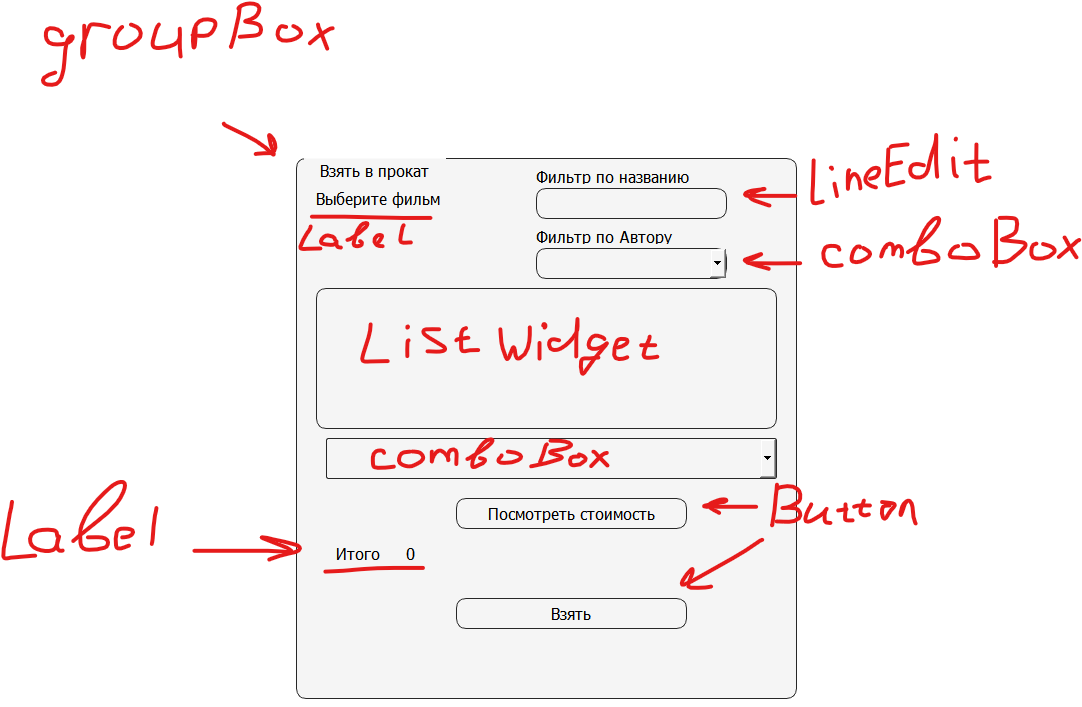




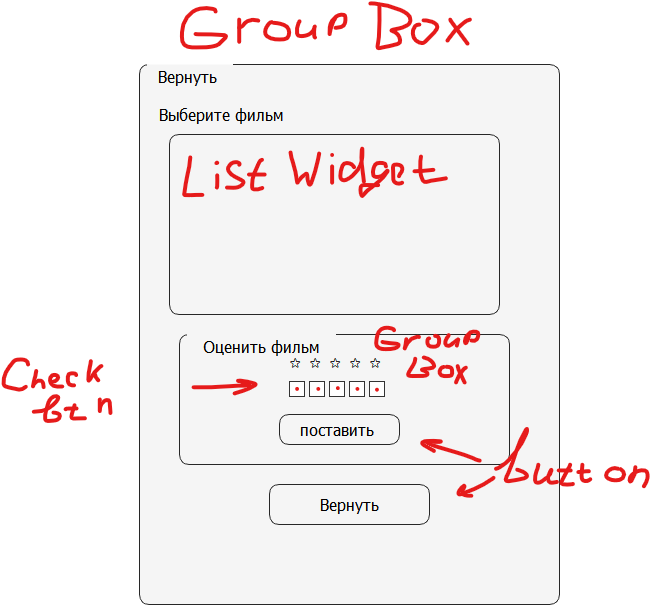
Общий вид формы из режима дизайнера



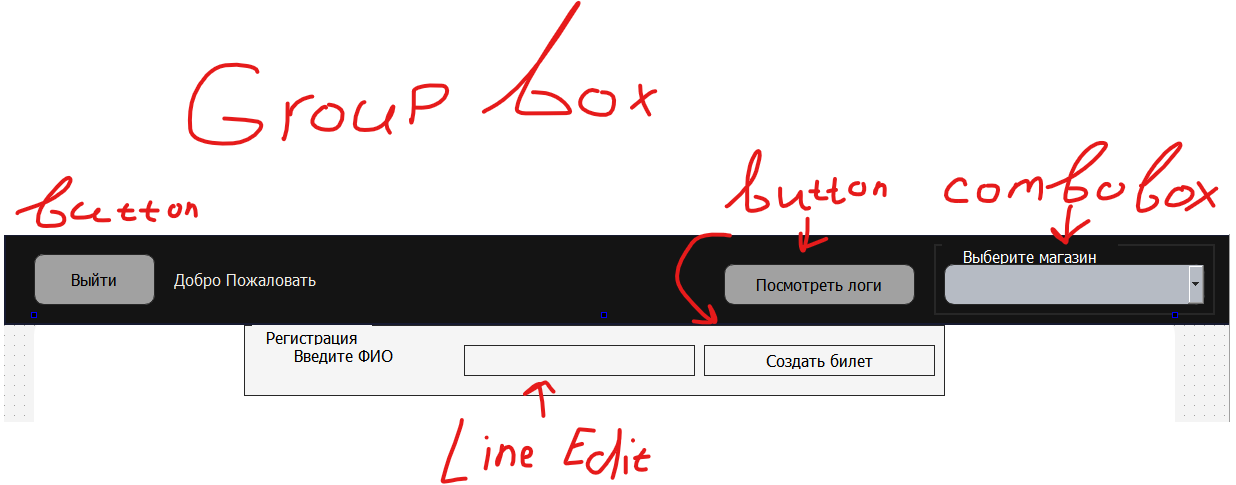
Форма для того, чтобы взять фильм состоит из нескольких компонентов. Процесс наполнения происходит динамически. Фильмы попадают в List Widget, авторы в Фильтр по автору.



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



**Выводы**

Я освоил Среду разработки QT для создания кроссплатформенных приложений