Міністерство освіти та науки України

Запорізький національний технічний університет

кафедра програмних засобів

з/к № //ХХХХХХХХХХХ//

**Пояснювальна записка**

з дисципліни «Об’єктно-орієнтоване програмування»

до курсової роботи на тему

«Створення візуального інтерфейсу»

Виконав

ст. гр. КНТ-415 Д. А. Рязанов

ст. гр. КНТ-415 О. В. Сергеев

ст. гр. КНТ-415 О. А. Бірюков

Прийняв

професор Г.В. Табунщик

доцент Т.І. Каплієнко

ст.. викл. Н.О. Миронова

Запоріжжя

2016

# ЗМІСТ

[ЗМІСТ 2](#_Toc469852682)

[Завдання на курсову роботу 4](#_Toc469852683)

[РЕФЕРАТ 5](#_Toc469852693)

[Перелік умовних позначень, символів, одиниць, скорочень і термінів 6](#_Toc469852694)

[Вступ 7](#_Toc469852695)

[1 Опис предметної області 8](#_Toc469852696)

[1.1 Огляд існуючих методів вирішення завдання 8](#_Toc469852697)

[1.2 Огляд існуючих програмних засобів, що вирішують аналогічні завдання 8](#_Toc469852698)

[1.3 Постановка завдання роботи 11](#_Toc469852699)

[2 Аналіз програмних засобів 12](#_Toc469852700)

[2.1 Огляд особливостей мови програмування 12](#_Toc469852701)

[2.2 Огляд особливостей обраного компілятора 13](#_Toc469852702)

[2.3 Огляд шаблонного класу бібліотеки STL, що використовується в роботі 14](#_Toc469852703)

[2.4 Класи візуального інтерфейсу 15](#_Toc469852704)

[2.6 Висновки з розділу 2 16](#_Toc469852705)

[3 Основні рішення з реалізації компонентів системи 17](#_Toc469852706)

[3.1 Основні рішення щодо уявлення даних системи 17](#_Toc469852707)

[3.2 Основні розроблені алгоритми 18](#_Toc469852708)

[3.3 Основні рішення щодо модульного уявлення системи 19](#_Toc469852709)

[3.4 Особливості реалізації системи 24](#_Toc469852710)

[3.5 Результати тестування системи 25](#_Toc469852711)

[3.6 Висновки з розділу 3 25](#_Toc469852712)

[4 посібник програміста 25](#_Toc469852713)

[4.1 Призначення й умови застосування програми 25](#_Toc469852714)

[4.2 Характеристика програми 27](#_Toc469852715)

[4.3 Звертання до програми 26](#_Toc469852716)

[4.4 Початкові та вихідні дані 27](#_Toc469852717)

[4.5 Повідомлення 27](#_Toc469852718)

[5 Інструкція користувача 28](#_Toc469852719)

[5.1 Призначення та функції програми 28](#_Toc469852720)

[5.2 Умови виконання програми 30](#_Toc469852721)

[5.3 Виконання програми 30](#_Toc469852722)

[4.5 Повідомлення 38](#_Toc469852723)

[Висновки 39](#_Toc469852724)

[Список використаної літератури 40](#_Toc469852725)

[Додаток А Текст програми 42](#_Toc469852726)

[Модуль addscene.h 42](#_Toc469852727)

[Модуль dataforstatistic.h 42](#_Toc469852728)

[Модуль finalywindow.h 43](#_Toc469852729)

[Модуль mainwindow.h 43](#_Toc469852730)

[Модуль optionsforhall.h 45](#_Toc469852731)

[Модуль scene.h 46](#_Toc469852732)

[Модуль statistic.h 47](#_Toc469852733)

[Модуль addscene.cpp 47](#_Toc469852734)

[Модуль dataforstatistic.cpp 49](#_Toc469852735)

[Модуль finalywindow.cpp 50](#_Toc469852736)

[Модуль mainwindow.cpp 51](#_Toc469852737)

[Модуль optionsforhall.cpp 63](#_Toc469852738)

[Модуль scene.cpp 66](#_Toc469852739)

[Модуль statistic.cpp 68](#_Toc469852740)

[Додаток Б Інтерфейс програми 71](#_Toc469852741)

# Завдання на курсову роботу

Запорізький національний технічний університет

Кафедра *програмних засобів*

Дисципліна *Об’єктно-орієнтоване програмування*

Спеціальність *Програмне забезпечення автоматизованих систем*

Курс *2* Група *КНТ-415* Семестр *III*

**ЗАВДАННЯ**

**на курсову роботу студентам**

*Рязанову Данилу Андрійовичу*

*Сергєєву Олександру Володимировичу*

*Бірюкову Олександру Андрійовичу*

1. Тема проекту (роботи): *Система бронювання квитків театру*

2. Термін здачі студентом закінченого проекту (роботи):

3. Вихідні дані до проекту:  *Реалізувати систему бронювання квитків театру:*

*вхідні дані: інформація про вистави (назва, дата, час, ціна за кожний тип місця), інформація про місця(назва вистави, дата, час, тип місця, ряд, місце, стан(куплене/броньоване))*

*вихідні дані: програма повинна забезпечувати створення нових вистав, редагування існуючих вистав, видалення вистав, купівля, бронювання, повернення місць, можливість налаштування залу*

4. ЗМІСТ розрахунково-пояснювальної записки (перелік питань, що їх належить розробити*):*

*1 Аналіз предметної області*

*2 Аналіз програмних засобів*

*3 Основні рішення з реалізації компонентів системи*

*4 Посібник програміста*

*5 Інструкція користувача*

*Висновки*

*Додаток А Текст програми*

*Додаток Б Інтерфейс програми*

6. Дата видачі завдання: *09.09.2016*

# РЕФЕРАТ

Пояснювальна записка до курсової роботи містить: 77 сторінок, 39 рисунків та 2 додатка.

Об’єкт розробки – програмне забезпечення, для ефективної організації бронювання та продажі квитків в театр.

Дане програмне забезпечення повинно виконувати основні функції з керування продажом та бронюванням квитків: продаж, бронювання, відновлення квитків на виставу; редагування, додавання та видалення вистави.

Метою роботи є створення програмного засобу для ефективного прискорення та контролю бронювання та продажу квитків, основні вимоги до якого були викладені вище.

Для досягнення поставленої мети необхідно розв’язати такі задачі:

1. дослідження предметної області;
2. аналіз аналогів (виявлення переваг та недоліків);
3. вибір мови програмування та компілятору;
4. розробка архітектури системи та інтерфейсу;
5. програмна реалізація.

У результаті виконання курсової роботи було створено ПЗ для ефективного прискорення та контролю бронювання та продажу квитків.

QTFRAMEWORK, QTCREATOR, C++, SQLite, JSON, ООП, ПЗ

# Перелік умовних позначень, символів, одиниць, скорочень і термінів

МП – мова програмування;

ПП – програмний продукт;

ПЗ – програмне забезпечення;

БД – база даних;

СКБД – система керування базами даних;

Qt – кроссплатформена бібліотека розробки GUI на C++;

MinGW – Minimalist GNU for Windows;

STL (Standard Template Library) – бібліотека для C++, що містить набір узгоджених узагальнених алгоритмів, контейнерів, засобів доступу до їхнього вмісту і різних допоміжних функцій;

JSON (JavaScript Object Notation) – текстовий формат обміну даними, заснований на JavaScript;

SQLite (Structured Query Language Lite) – [реляційна система керування базами даних](https://uk.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BA%D0%B5%D1%80%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B1%D0%B0%D0%B7%D0%B0%D0%BC%D0%B8_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85).

# Вступ

Програма є аналогом UCS-Премьери – сервісу, що являє собою онлайн касу для придбання квитків. Призначена для зручного придбання квитків з любого місця де є інтернет.

Метою роботи є здобуття навичок створення програми за допомогою засобів ООП.

Задачею є розробка програми для продажу квитків в театр. Під час розробки необхідно реалізувати ергономічний інтерфейс, ефективні алгоритми, для роботи з програмою та реалізувати роботу з даними в форматах SQLite і JSON.

В процесі реалізації даної задачі використати основні властивості об’єктно-орієнтованого програмування. В якості інструментів для розробки було обрано мову програмування C++ в середовищі Qt. В процесі розробки необхідно виконати наступні завдання:

- провести аналіз предметної області;

- розробити відповідні структури даних;

- створити візуальний інтерфейс;

- розробити програму;

- провести тестування;

- оформити пояснювальну записку.

# 1 Опис предметної області

## Огляд існуючих методів вирішення завдання

У нашому сучасному світі існує багато театрів, покупка квитків в яких займає багато часу через отсутсвия відповідного прогрмманого забезпечення. Вибір даної предметної області обумовлений тим що для більшості театрів швидкість реалізації бронювання та продажу квитків є першорядним критерієм як обслуговування

У цю предметну область входять: театр, постановка, місце, оператор

## 1.2 Огляд існуючих програмних засобів, що вирішують аналогічні завдання

Ticketland (ticketland.ru), розробник - ТОВ «МДТЗК». Переваги: робить персоналізовані рекомендації користувачам, орієнтуючись на їхні інтереси, надаються спеціальні бонуси і подарунки. Недоліки: не дуже зручний інтерфейс, відсутність сортування по типу місць. Вікно виконання цієї програми представлене на рисунку 1.1.

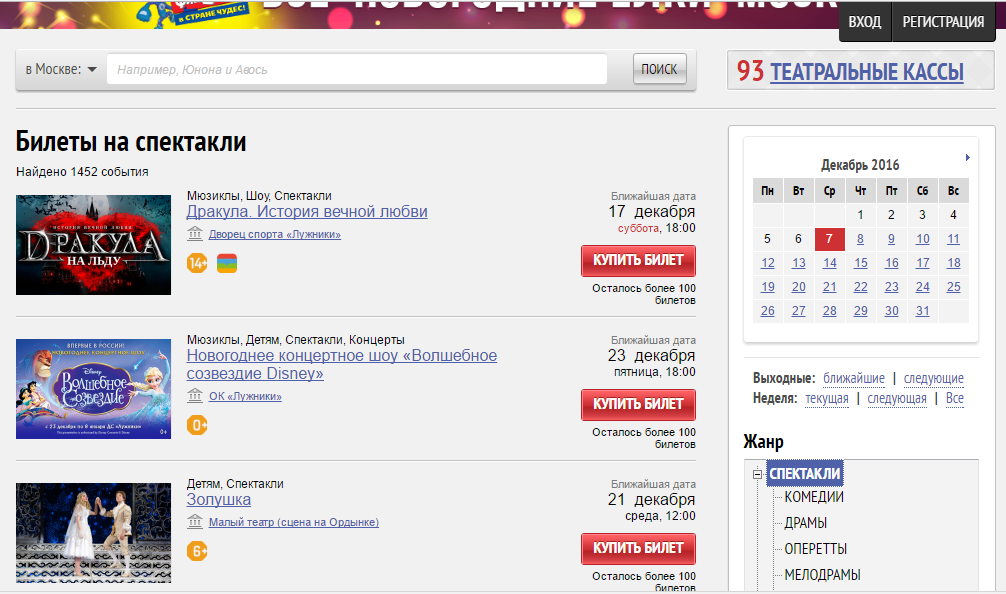


Рисунок 1.1 – Вікно виконання програми Ticketland

UCS-Премьера, розробники – UCS Original Software. Переваги: зручний інтерфейс, мультіплатформеність. Недоліки: відсутність багатомовності, скудні тона. Вікно виконання цієї програми представлене на рисунку 1.2.

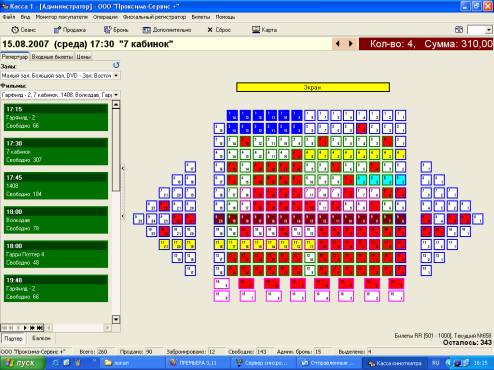


Рисунок 1.2 – Вікно виконання програми UCS-Премьера

Multiplex (multiplex.ua) розробники - INVOCO. Переваги: Переваги: кроссплатформенность, зручна навігація по сайту. Недоліки: . Вікно виконання цієї програми представлене на рисунку 1.3.

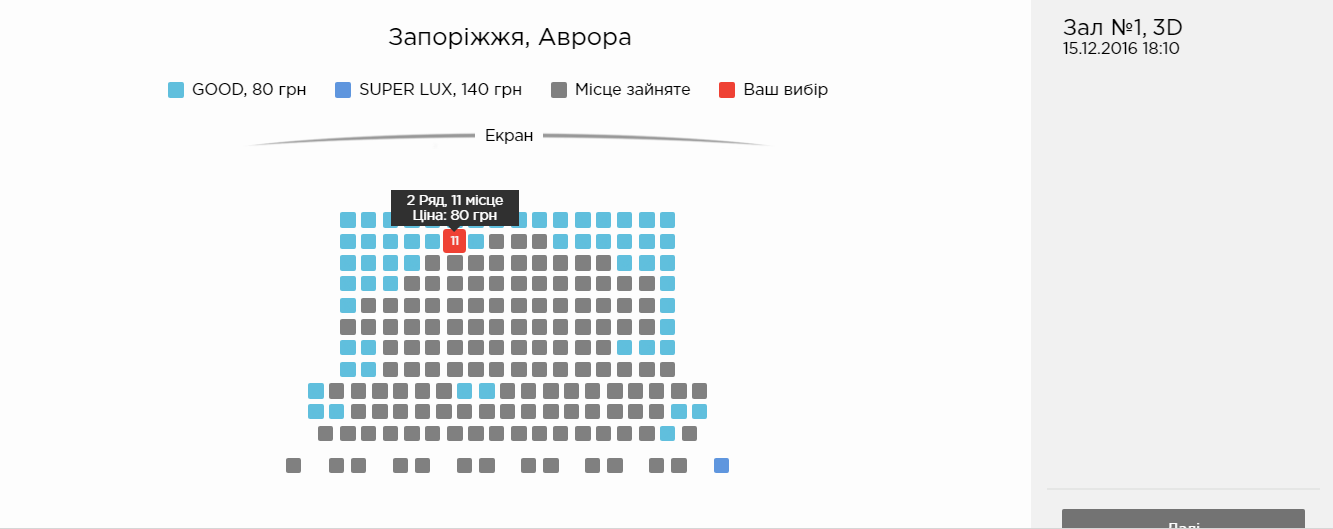


Рисунок 1.3 – Вікно виконання програми Multiplex

Kassir (kassir.ru), розробники - ТОВ «Кассир». Переваги: сортування за датою. Недоліки: незручний інтерфейс відображення місць. Вікно виконання цієї програми представлене на рисунку 1.4.



Рисунок 1.4 – Вікно виконання програми Kassir

Bileter (bileter.ru), розробники – ТОВ «Билетёр». Переваги: наявність пошуку театру по мапі. Недоліки: не ергономічний інтерфейс . Вікно виконання цієї програми представлене на рисунку 1.5.

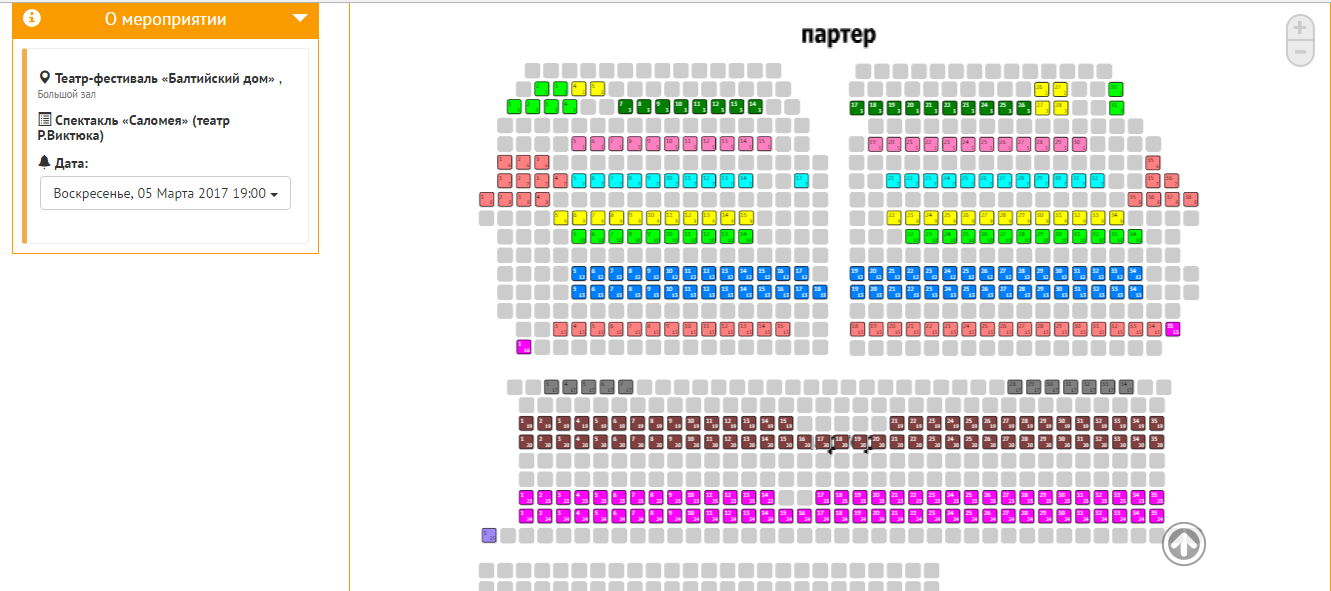


Рисунок 1.5 – Вікно виконання програми Bileter

## 1.3 Постановка завдання роботи

Створити програмних продукт основною задачею якого є купівля і бронювання квитків на вистави в театр. Він дозволяє оператору заощадити час на обслуговування покупця.

Система повинна забезпечувати:

1. бронювання та купівля квитківж

2. зручний доступ до даних кожного спектаклю;

3. інструмент добавлення нової вистави;

4. редагування та видалення вистави;

5. сортування записів за датою;

6. надання актуальної статистики;

7. налаштування розмірів залу.

# 2 Аналіз програмних засобів

## 2.1 Огляд особливостей мови програмування

Для написання програми було обрано мову програмування С++. Мова С++ - це мова програмування загального призначення, мета якого – зробити роботу програміста більш зручною. Вона надає програмісту інструмент для опису, що підлягають виконанню дій і набір концепцій, якими оперую програміст, обдумуючи, що можна зробити.

С++ підтримує такі парадигми програмування, як процедурне програмування, об'єктно-орієнтоване програмування, узагальнене програмування, забезпечує модульність, роздільну компіляцію, обробку винятків, абстракцію даних, оголошення типів (класів) об'єктів, віртуальні функції. Стандартна бібліотека включає, в тому числі, загальновживані контейнери і алгоритми.

C ++ поєднує властивості як високорівневих, так і низькорівневих мов. Найбільшу увагу приділено підтримці об'єктно-орієнтованого і узагальненого програмування.

Для розробки програми було обрано середовище Qt. Qt - кросплатформовий інструментарій розробки ПЗ мовою програмування C++. Дозволяє запускати написане за його допомогою ПЗ на більшості сучасних операційних систем шляхом простої компіляції тексту програми для кожної ОС без зміни початкового коду. Включає всі основні класи, які можуть бути потрібні при розробці прикладного програмного забезпечення, починаючи з елементів графічного інтерфейсу і закінчуючи класами для роботи з мережею, базами даних, OpenGL, SVG і XML. Бібліотека дозволяє керувати потоками, працювати з мережею, і забезпечує кросплатформовий доступ до файлів.

## 2.2 Огляд особливостей обраного компілятора

У ході розробки використовувався компілятор MinGW версії 4.1.9, що є безкоштовним компілятором та представляє собою прямий програмний порт GNU Compiler Collection (GCC) під ОС Windows. Даний компілятор дозволяє розробляти рідні (native) додатки Windows. Увесь процес компіляції такий: препроцессінг, трансляція, лінкування. В результаті отримуємо готовий бінарний файл. Більшість мов, які підтримує GCC, також підтримуються і у MinGW. Це C, C++, Objective-C, Fortran, Ada.

MinGW за замовчуванням виконує компонування з бібліотекою компонентів MSVCRT, яка є бібліотекою мови C, створеної у 1998 році. Проект підтримує і поширює кілька різних ключових компонентів і додаткових пакетів, включаючи різні порти GNU toolchain, такі як GCC і Binutils, переведені в еквіваленті пакети. Ці утиліти можуть бути використані з командного рядка вікна або інтегровані в IDE. В доповнення, компонент MinGW, відомий як MSYS (Minimal SYStem) надає win32-порти легкої Unix-подібної shell оболонки, що включає rxvt і набір інструментів POSIX, достатній для запуску скриптів Autoconf. Реалізації заголовних файлів Win32 і бібліотек імпорту Win32 для зв'язування під час виконання програми від початку до її завершення мають пермісивну ліцензію, а порти GNU доступні під GNU General Public License. Бінарні збірки повного пакету MSYS та окремих MinGW GNU утиліт доступні для завантаження на сайті MinGW.

MinGW використовує виклики Win32 API. Внаслідок цього, він не може скомпілювати деякі програми Unix, що вимагають підтримку POSIX API. Зокрема, це відноситься до додатків, які вимагають такої функціональності POSIX, як fork(), mmap() або ioctl() і допускають запуск в середовищі POSIX.

У зв'язку з тим, що в рамках початкового проекту MinGW не обіцяв, що в його кодову базу будуть вноситися оновлення пов'язані з додаванням деяких нових ключових елементів Win32 API, а також найбільш необхідної підтримки 64-бітної архітектури, був створений проект MinGW-w64 (включений до останніх версій проекту MinGW). Він є новою чистою реалізацією порту GNU Compiler Collection (GCC) під Microsoft Windows здійсненої спочатку компанією OneVision та переданої у 2008 році в громадське користування (Public Domain).

MinGW-w64 забезпечує більш повну реалізацію Win32 API та включає:

* кращу підтримку стандарту C99;
* кращу підтримку Pthreads (включаючи можливість залучити функціонал стандарту C++11 бібліотеки libstdc++ компілятора GCC);
* GCC multilib;
* точки входу в програму з підтримкою Unicode (wmain / wWinMain);
* DDK (з проекту ReactOS);
* DirectX (з проекту WINE);
* підтримка великих файлів;
* підтримка 64-розрядних ОС Windows.

## 2.3 Огляд шаблонного класу бібліотеки STL, що використовується в роботі

В програмі використано шаблони класів QVector та Vector. Клас QVector - це шаблоний клас в середовищі Qt, що представляє динамічний масив.

QVector є одним із загальних контейнерних класів Qt. Він зберігає свої елементи в суміжних комірках пам’яті та забезпечує швидкий доступ за індексом. QVector рекомендовано до використання, коли елементи є більшими, за розміром, ніж вказівник, для того щоб уникнути накладних витрат на індивідуальне виділення для них місця в "купі", під час вставки нового елемента.

QVector надає основні функції для додавання, переміщення та видалення елементів: insert(), replace(), remove(), prepend(), append(). Звернення за індексом відбувається за допомогою функцій at(), value() та перевантеженого оператора [].

В програмі, що розробляється, необхідний швидкий доступ до елементу контейнера за індексом. В контейнер поміщується обьект створенного классу, який суттєво більший за розміром, ніж вказівник. Саме через це, в якості контейнера було обрано шаблоний клас QVector.

Клас vector розташований в заголовку <vector>. Як і всі стандартні компоненти, він розташований в просторі імен std. Даний інтерфейс емулює роботу стандартного масиву C (наприклад, швидкий довільний доступ до елементів), а також деякі додаткові можливості, начебто автоматичного зміни розміру вектора при вставці або видаленні елементів.

Всі елементи вектора повинні належати одному типу. Наприклад, не можна спільно зберігати дані типів char і int в одному екземплярі вектора. Клас vector має стандартний набір методів для доступу до елементів, додавання і видалення елементів, а також отримання кількості збережених елементів.

## 2.4 Класи візуального інтерфейсу

Особливість Qt — використання Meta Object Compiller (MOC) — попередньої системи обробки початкового коду. MOC дозволяє в багато разів збільшити потужність бібліотек, вводячи такі поняття, як слоти (slots) і сигнали (signals). Qt комплектується графічним середовищем розробки графічного інтерфейсу QTDesigner, що дозволяює створювати діалоги і форми. Ідеологія створення форм у Qt базується на використанні менеджерів розташування, котрі надають «гумовий» дизайн, при якому розмір і розташування елементів форм визначаються автоматично, що значно прискорює розробку графічного інтерфейсу. В поставці Qt є «Qt Linguist» — могутня графічна утиліта, що дозволяє спростити локалізацію й переклад програми багатьма мовами, та «Qt Assistant» — довідкова система Qt, що спрощує роботу з документацією для бібліотек і дозволяє створювати крос-платформову довідку для ПЗ, розробленого на основі Qt.

Графічна бібліотека Qt розділена на декілька модулів:

QtDesigner — класи створення розширень QtDesigner'а для своїх власних віджетів;

QtGui — компоненти графічного інтерфейсу;

QtWidgets — містить класи для класичних застосувань на основі віджетів, модуль виділений з QtGui в Qt 5.

Ці модулі були використанні для створення інтерфейсу програми.

Інтерфейс розроблено за макетом «Центральна сцена». Тобто візуально в центрі робочого вікна програми будуть розміщені основні функціональні елементи інтерфейсу користувача.

Інтерфейс побудовано на основі Головного вікна програми з використанням додаткових діалогових вікон за необхідності. На Головному вікні розміщені: Панель меню, Рядок стану, Панель інструментів, різні варіанти списків, поля вводу, кнопки, таблиці. Всі елементі доповнені відповідними підписами. Є підтримка комбінацій гарячих клавіш та табуляції.

Додаткові діалогові вікна можуть бути функціональними або вікнами-повідомленнями з відповідними їм графічними елементами.

## 2.6 Висновки з розділу 2

Мовою програмування було обрано C++ з Qt Framework за зручність створення GUI, потужність та гнучкість мови, широкий спектр можливостей.

В якості компілятора було обрано MinGW 4.8.1, який є доступним, зручним, налаштованим під ОС Windows.

Інтерфейс було розроблено засобами Qt Designer за макетом «Центральна сцена».

# 3 Основні рішення з реалізації компонентів системи

## 3.1 Основні рішення щодо уявлення даних системи

Дані розробленої програми можна абстрактно розділити на 4 групи: дані для заповнення місць, дані, які використовуються для представлення інформації про вистави, дані, які використовуються для відображення статистики по придбаним місцям, та дані, які відповідають за налаштування залу.

Дані для заповнення місць є одними з основних для роботи з алгоритмами програми, дані зберігаються в БД та представленні у формі таблиці яка зберігає в собі всю необхідну інформацію для зручного та швидкого відображення даних в інтерфейсі програми. Вона складається з наступних полів: поле з кодом (використовується для швидкого знаходження броньованих місць), поле з назвою вистави, поле з часом вистави, поле з датою вистави, поле яке відповідає за тип місця, поле яке відповідає за статус місця (куплене або броньоване), а також основні поля які зберігають в собі значення ряду та місця в залі.

Дані, які використовуються для представлення інформації про вистави, зберігаються також у БД. Всі дані з цієї таблиці потрапляють в клас Scene, який за допомогою методів достає значення та виводить їх на панель вистав. Таблиця з виставами зберігає наступні поля: назва, дата, час, та ціна за кожний тип місць на цю виставу.

Дані які відповідають за відображення статистики проданих місць за певний проміжок часу. Зберігаються в БД, та за допомогою методів класу DataforStatistic потрапляють на графік у вікні статистики. Таблиця зберігає поля з датою, типом місця та лічильником проданих місць.

З усіма даними які є у нас в програмі ми працюємо через класи, які не прив’язані до інтерфейсу. Дані для налаштувань залу зберігаються в файлі формату JSON, Зберігає 3 об’єкти, в кожному з яких зберігається інформація про кількість рядів та місць в ряді для конкретного типу місця. Для роботи з даними ми використовуємо СКБД SQLite.

## 3.2 Основні розроблені алгоритми

Для функціонування програми було реалізовано алгоритми додавання нової вистави, видалення та редагування існуючої вистави, пошуку існуючих вистав на певний день, бронювання місця разом з генерацією коду, купівлі повернення вибраних місць, зміни налаштувань залу, пошуку місць за даним кодом бронювання, а також будування графіків.

При додаванні нової вистави в першу чергу програма перевіряє наявність вже існуючої вистави на цей день, час та з тією ж назвою. Якщо така вистава відсутня, то програма заносить у БД дані про нову.

Після зміни поточної дати програма здійснює пошук вистав які відповідають поточній даті, та виводить їх у таблицю вистав. Після кліку по назві вистави відкриється контекстне меню з двома пунктами: «Редагувати» та «Видалити». Якщо кристувач обрав перший пункт, то відкриється вікно редагування, в якому вже будуть завантажені дані цієї вистави з БД. Користувач може їх змінити та зберегти, після чого БД оновить дані. Якщо ж він обере пункт «Видалити», то програма видалить усі дані що були пов'язані з цією виставою з БД.

Якщо користувач вибере виставу, то з БД завантажаться дані про неї і оновиться таблиця місць, де кожен рядок таблиці відповідає ряду, а кожен стопвець ­­– місцю у ряді. Якщо натиснути на будь-яку клітинку, то вона стане виділеною.

При натисканні на кнопку «Бронювати» усі виділені місця, якщо вони не є вже заброньованими або купленими, стають заброньваними і генерується спеціальний код. Якщо його ввести у відповідне поле, та натиснути кнопку «Пошук», програма сама знайде ті місця, яким відповідає даний код та зробить їх виділеними.

При натискані на кнопку «Купити» дані про усі виділені місця, якщо вони не є вже купленими, заносяться до таблиці діалогового вікна, де користувач перевіряє що вибрав саме ті місця, що йому потрібні, та купує їх.

При натискані на кнопку «Повернути» дані про усі місця, які є заброньованими або купленими видаляються з БД і місце стає вільним.

При натискані на пункт «Налаштування залу» відкріється нове вікно, де користувач може виставити нову кількість рядів та місць у ньому для кожного типу місць.

Дані про усі продані білети заносяться у відповудну таблицю БД. Коли користувач відкріває вікно статистики то програма завантажує їх і будує графік, який відображає число білетів, які було продано у певний день для кожного з типів місць.

## 3.3 Основні рішення щодо модульного уявлення системи

Програма складається з таких файлів:

Заголовні файли

**About.h** містить оголошення класу About, який успадковується від класу QDialog. Клас містить макрос Q\_OBJECT.

Поле **private** містить наступні дані:

Ui::About \*ui – вказівник на об’єкт форми About;

Поле **public** містить наступні дані:

explicit About(QWidget \*parent = 0) – конструктор класу About;

~About() – деструктор класу About;

**AddScene.h** містить оголошення класу AddScene який успадковується від класу QDialog. Клас містить макрос Q\_OBJECT.

Поле **private** містить наступні дані:

Ui::AddScene \*ui – вказівник на об’єкт форми AddScene;

QSqlDatabase mydb – змінна типу QSqlDatabase;

Поле **public** містить наступні дані:

explicit AddScene(QWidget \*parent = 0) – конструктор класу AddScene;

explicit AddScene(bool edit, Scene \*temp, QWidget \*parent = 0) – конструкторкласу ddScene;

~AddScene() – деструктор класу AddScene;

bool Editing – змінна типу bool;

QString PriviosName

PriviosTime,

PriviosDate; – змінні типу QString,

Поле **private slots** містить наступні дані:

void on\_ButtonAdd\_clicked() – метод типу void;

**MainWindow.h** містить оголошення класу MainWindow, який успадковується від класу QDialog. Клас містить макрос Q\_OBJECT.

Поле **private** містить наступні дані:

Ui::MainWindow \*ui – вказівник на об’єкт форми MainWindow;

QSqlDatabase mydb –змінна типу QSqlDatabase;

void places\_fill() – метод типу void;

void cleasing\_places() – метод типу void;

void create\_a\_MainTable() – метод типу void;

void places\_overwrite(int row, int column) – метод типу void;

void pix\_close(int row, int column) – метод типу void;

void pix\_checking(int row, int column) – метод типу void;

void pix\_reserv(int row, int column) – метод типу void;

void pix\_standart(int row, int column) – метод типу void;

void customizeTableInf() – метод типу void;

Поле **public** містить наступні дані:

explicit MainWindow(QWidget \*parent = 0) – конструктор класу MainWindow;

~*MainWindow*() – деструктор класу MainWindow;

Scene \*CurScene – вказівник на клас Scene;

QDateTime dt – змінна типу QDateTime;

int row\_height – змінна типу int;

int column\_width – змінна типу int;

bool \*\*coordinates\_of\_places – вказівник на вказівник типу bool;

void coordinates\_of\_places\_cleaning(int temp) – метод типу void;

std::vector <short> SelectedPlacesRow – змінна типу vector <short>;

std::vector <short> SelectedPlacesCol – змінна типу vector <short>;

int PreviousIndex – змінна типу int;

int CountRow – змінна типу int;

int CountColumn – змінна типу int;

int CountPurchased – змінна типу int;

int CountBooked – змінна типу int;

Поле **private slots** містить наступні дані:

void on\_informationAbout\_triggered() – метод типу void;

void on\_action\_triggered() – метод типу void;

void on\_tableWidget\_cellClicked(int row, int column) – метод типу void;

void on\_comboBox\_currentIndexChanged(int index) – метод типу void;

void on\_tableSeans\_cellClicked(int row, int column) – метод типу void;

void on\_dateEdit\_dateChanged(const QDate &date) – метод типу void;

void on\_pushButton\_clicked() – метод типу void;

void on\_pushButton\_2\_clicked() – метод типу void;

void on\_action\_exit\_triggered() – метод типу void;

void on\_action\_statistic\_sale\_triggered() – метод типу void;

void on\_pushButton\_3\_clicked() – метод типу void;

void on\_options\_room\_triggered() – метод типу void;

void on\_action\_addScene\_triggered() – метод типу void;

void on\_action\_hovered() – метод типу void;

void on\_tableSeans\_customContextMenuRequested(const QPoint &pos) – метод типу void;

void slotEditRecord() – метод типу void;

**OptionsForHall.h** містить оголошення класу OptionsForHall, який успадковується від класу QDialog. Клас містить макрос Q\_OBJECT.

Поле **private** містить наступні дані:

void parseJSON(QJsonDocument& jsonDoc) – метод типу void;

Ui::OptionsForHall \*ui – вказівник на об’єкт форми OptionsForHall;

QSqlDatabase mydb – змінна типу QSqlDatabase;

void DrawTable() – змінна типу void;

Поле **public** містить наступні дані:

explicit OptionsForHall(QWidget \*parent = 0) – конструктор класу OptionsForHall;

~*OptionsForHall*() – деструктор класу OptionsForHall;

double dataJsonFile[3][2] – змінна типу double;

QString type\_placeJsonFile[3] – змінна типу QString;

Поле **private slots** містить наступні дані:

void on\_spinBox\_Row\_valueChanged(int arg1) – метод типу void;

void on\_spinBox\_Column\_valueChanged(int arg1) – метод типу void;

void on\_ComboBoxType\_currentIndexChanged(int index) – метод типу void;

void on\_pushButtonOK\_clicked() – метод типу void;

**Scene.h** містить оголошення класу Scene, який успадковується від класу QDialog. Клас містить макрос Q\_OBJECT.

Поле **public** містить наступні дані:

Scene() – конструктор класу Scene;

void set\_name(QString arg) – метод типу void;

void set\_date(QString arg) – метод типу void;

void set\_time(QString arg) – метод типу void;

void set\_cost() – метод типу void;

void SetArrayCountPlaces() – метод типу void;

void SetDataToTables() – метод типу void;

void DeleteTables() – метод типу void;

void InsertTablesToDataBase(std::vector<short> SelectedPlacesRow, std::vector<short> SelectedPlacesCol, int Index, int operation) – метод типу void;

int ArrayCountPlaces[3][2] – змінна типу int;

int \*\*\*TablesPlaces – вказівник на вказівник на вказівник типу int;

int CountOfTypesPlaces – зміна типу int;

QString name, date, time –змінна типу QString;

double cost\_parter, cost\_benuar, cost\_beletaj – змінні типу double;

QString nameofplace[3] – змінна типу QString;

**Statistic.h** містить оголошення класу Statistic, який успадковується від класу QDialog. Клас містить макрос Q\_OBJECT.

Поле **private** містить наступні дані:

Ui::Statistic \*ui – вказівник на об’єкт форми Statistic;

Поле **public** містить наступні дані:

explicit Statistic(QDateTime datenow, QWidget \*parent = 0) – конструктор класу Statistic;

~*Statistic*() – деструктор класу Statistic;

Файли реалізації:

Statistic.cpp – містить реалізацію методів класу Statistic;

Scene.cpp – містить реалізацію методів класу;

OptionsForHall.cpp – містить реалізацію методів класу Options;

MainWindow.cpp – містить реалізацію методів класу MainWindow;

main.cpp – являє собою точку запуску програми;

FinalyWindow.cpp – містить реалізацію методів класу FinalyWindow;

DataForStatistic.cpp – містить реалізацію методів класу DataForStatistic;

AddScene.cpp – містить реалізацію методів класу AddScene;

About.cpp – містить реалізацію методів класу About.

## 3.4 Особливості реалізації системи

Під час реалізації програми, стояла мета найбільш глибоко використати функціонал Qt, як для інтерфейсних рішень, так і для реалізації алгоритмів та роботи з даними. Для підвищення зручності реалізації програми також були використанні такі засоби ООП: реалізація класів та створення об’єктів класів, спадкування, віртуальна функція, шаблонний клас STL.

Особливості реалізації системи логічно розглядати за категоріями: особливості реалізації інтерфейсу, особливості реалізації алгоритмів, особливості реалізації роботи з даними.

До першої категорії відноситься макет інтерфейсу «Центральна сцена», згідно з яким головна робоча область знаходиться, візуально, у центрі інтерфейсу. Саме у напрямку від центру формується візуальній потік, що проходить за годинниковою стрілкою по всім основним елементам керування. Відповідно до потоку та макету інтерфейсу, візуальна ієрархія його компонентів починається від центру. Додатково до головного вікна існує ще чотири, що виконують службові функції (редагування даних, повідомлення, відображення даних).

Щодо особливостей реалізації алгоритмів, то тут слід відзначити, що основним класом є клас Scene, що зберігає всю поточну інформацію про ту виставу, що обрав користувач. Після того як користувач купує, бронює, повертає місце, змінює час, назву або ціни білетів, то перш за все зміни відбуваються у цьму класі і вже потім заносяться у БД.

Розглядаючи особливості реалізації роботи за даними, варто зазначити можливість збереження даних у форматах SQLite і JSON. Залежно від обраного методу запису у файл або зчитування з нього, дані кодуються та декодуються відповідним чином. Завдяки цьому було реалізовано два зручних способи збереження інформації.

## 3.5 Результати тестування системи

Під час тестування програми, знайдені недоліки було виправлено. Під час фінального тестування недоліків виявлено не було. Програма функціонує правильно, та стабільно.

## 3.6 Висновки з розділу 3

В результаті виконання курсової роботи було розроблено комплекс класів з використанням засобів Qt, який повністю реалізує поставлену задачу. Були використані основні методи об’єктно - орієнтованого програмування.

Було проведено тестування програми. Програма працює коректно та містить весь, запланований до реалізалізації, функціонал.

# 4 посібник програміста

## 4.1 Призначення й умови застосування програми

Даний програмний засіб представляє собою програмне забезпечення для ефективної роботи касира в театрі

В програмі реалізовані такі основні функції як: додавання вистави, редагування вистави, придбання, бронювання та повернення квитів, генерація унікального коду, пошук за унікальним кодом, видалення вистав з БД якщо за датою вони застаріли, налаштування кількості міст в залі а також відображення актуальної статистики.

Далі представлені основні вимоги до системи, необхідні для стабільної роботи і функціональності VTeatre. В характеристиках вказані мінімальні параметри.

Системні вимоги:

1. процесор з тактовою частотою 800 MHz;
2. 128 Мб та більше DDR2 RAM;
3. вільне місце на ПЗУ: 40 Мб;
4. архітектура системи з розрядністю 32- або 64-біт;
5. операційна система: Windows (з версії ХР та вище);
6. стандартні пристрої вводу-виводу (клавіатура, миша, монітор);

## 4.2 Характеристика програми

Часові рамки коректної роботи програми обмежені 2026 роком, це пов’язано з реалізацією системного календаря в більшості операційних систем.

В програмі реалізовано єдиний режим роботи – в якості оператора.

Контроль правильності виконання і самовідновлення програми реалізовано за допомогою обробки виключних ситуацій.

## 4.3 Звертання до програми

Виклик програми проводиться за допомогою стандартних засобів операційної системи з exe файлу.

Внесення даних у програми може відбуватися як в ручному режимі, так і з SQLite чи JSON файлу.

## 4.4 Початкові та вихідні дані

Початкові дані містять інформацію про вистави, придбані, броньовані місця, значення для налаштувань залу, значення для відображення статистики.

Вихідні дані полягають у забезпеченні можливості створення нових вистав, редагування існуючих вистав, видалення вистав, також забезпечують купівлю, бронювання та повернення місця, змін налаштувань залу.

Початкові та вихідні дані можуть бути завантажені та збережені в XML та JSON файлах.

## 4.5 Повідомлення

Повідомлення:

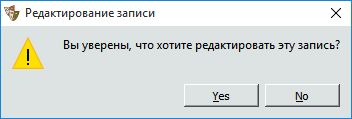


Рисунок 4.1 – Повідомлення при натисканні редагування вистави

Це повідомлення виникає перед користувачем коли він намагається редагувати вистави. Реагувати на це повідомлення потрібно залежно від ваших намірів

Попередження:

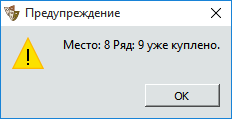


Рисунок 4.2 – Попереджує про змогу купити куплене місце

Це попередження з’являється перед користувачем у випадках, коли оператор хоче придбати місце яке вже було до цього придбано. Ця помилка не додасть дане місце в фінальне вікно оплати.

Повідомлення про помилку:

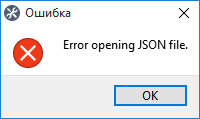


Рисунок 4.3 – Повідомлення про помилку відкриття файлу за даними

За випадків коли файл з даними відсутній в кореневому каталозі, виникає така помилка.

# 5 Інструкція користувача

## 5.1 Призначення та функції програми

Програмний засіб призначений для оператора театру. Він має на меті допомогти оператору прискорити роботу по бронюванню та продажі квитків на вистави.

Використовувати VTeatre може абсолютно будь-яка людина з мінімальними навичками володіння комп’ютером, а ергономічний інтерфейс спрощує пристосування до нового програмного засобу. Програма має такі функції:

1. бронювання та придбання квитка на будь-яку виставу театру;
2. налаштування залу, а саме: кількість рядів та місць;
3. редагування, додавання, видалення будь-якої вистави;
4. відображення статистики по продажам квитків.

## 5.2 Умови виконання програми

Далі представлені основні вимоги до системи, необхідні для стабільної роботи і функціональності VTeatre. В характеристиках вказані мінімальні параметри.

Системні вимоги:

1. процесор з тактовою частотою 800 MHz;
2. 128 Мб та більше DDR2 RAM;
3. вільне місце на ПЗУ: 40 Мб;
4. архітектура системи з розрядністю 32- або 64-біт;
5. операційна система: Windows (з версії ХР та вище);
6. стандартні пристрої вводу-виводу (клавіатура, миша, монітор).

## 5.3 Виконання програми

Для запуску програми необхідно виконати наступні кроки:

1. упевнитися у відповідності характеристик ПК, на яку встановлюється програма, до мінімальних системних вимог програми;
2. упевнитися у справності всіх компонентів ПК та електронного носія, з якого здійснюватиметься запуск програми;
3. у каталозі із завантаженою програмою знайти виконуваний файл VTeatre.exe та здійснити його запуск.

Подальша робота з програмою здійснюється за допомогою графічного інтерфейсу в режимі WYSIWYN. Для виконання певних функцій необхідно взаємодіяти з відповідними компонентами GUI. Інтерфейс головного вікна програми представлений на рисунку 5.1.

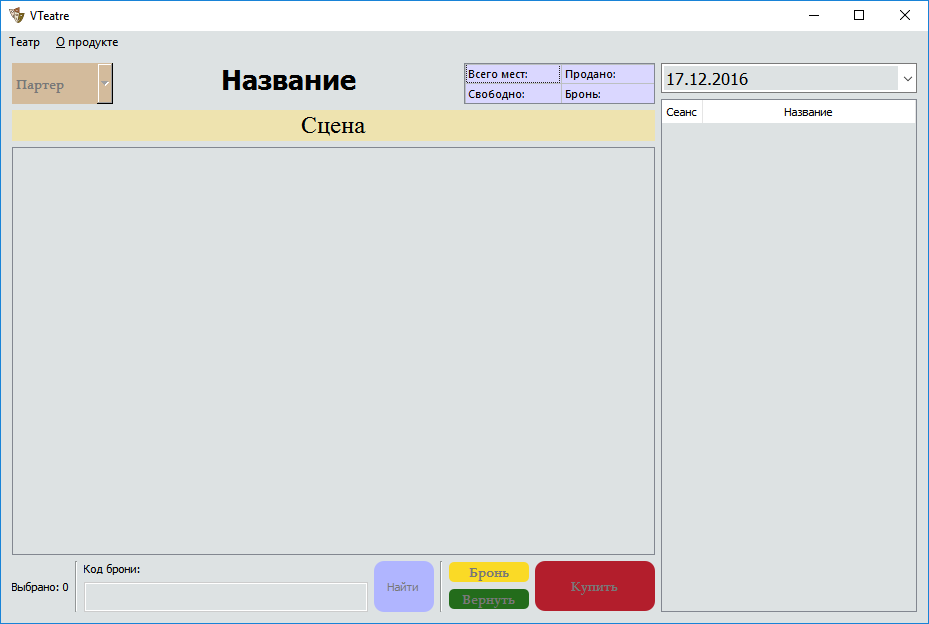


Рисунок 5.1 – Інтерфейс головного вікна програми

Для того щоб продивитися інформації о місцях вистави, треба натиснути на поле с датою, вибрати необхідну дату (рис. 5.2), натиснути на потрібну назву вистави в відповідній таблиці (рис. 5.3).

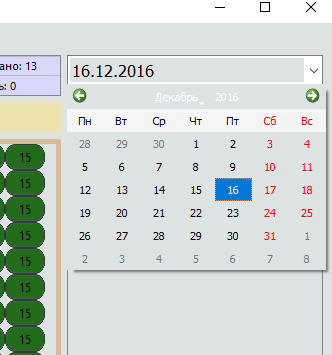


Рисунок 5.2 – Поле з датою

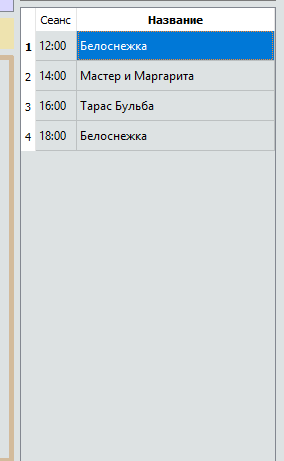


Рисунок 5.3 – Таблиця з виставами

Кожна вистава має декілька типів місць. Для того щоб продивитися інформацію по місцям якогось типу треба змінити цей тип в випадаючому списку (рис 5.4).

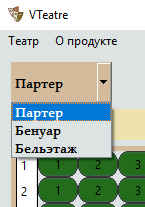


Рисунок 5.4 – Випадаючий список з типами місць

Для того щоб забронювати чи продати місце, оператор повинен один раз натиснути на відповідне місце в таблиці, місце перефарбується в блакитний колір (рис. 5.5). Щоб придбати, забронювати чи відновити вибрані місця потрібно натиснути на кнопку «Купить», «Бронь» или «Вернуть» відповідно (рис. 5.6). Кількість натиснутих місць відображається у вигляді надпису в лівій нижній частині екрану (рис. 5.7.1), основна інформація про таблицю місць відображається в верхній правій частині екрану у вигляді таблиці з 4 комірками (рис. 5.7.2): «Всего мест», «Продано», «Свободно», «Бронь».



Рисунок 5.5 – Вибір місць



Рисунок 5.6 – Функціональні кнопки



Рисунок 5.7.1 – Відображення кількості вибраних місць



Рисунок 5.7.2 – Таблиця з інформацією про таблицю місць

Якщо оператор вибрав місце та натиснув на «Купить», відкривається нове вікно для кінцевого підтвердження покупки. Вікно покупки складається з таблиці з детальною інформацією про вибране місце, надпису з загальною вартістю та функціональної кнопки (рис 5.8).

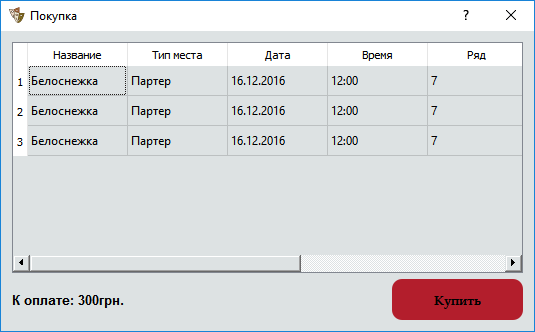


Рисунок 5.8 – Вікно покупки

Якщо оператор вибрав місце та натиснув на «Бронь», викликається вікно з кодом броні (рис. 5.9), для того щоб в майбутньому придбати заброньовані місця, покупець повинен вказати оператору цей код. Для того щоб виділити місця, які забронював покупець, оператор має можливість ввести код броні в відповідне поле та натиснути кнопку «Найти» (рис 5.10).

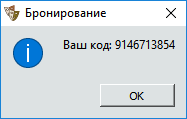


Рисунок 5.9 – Код броні

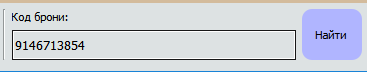


Рисунок 5.10 – Поле для вводу коду броні та функціональна кнопка

Для того щоб перейти до налаштувань залу, додати нову виставу, чи проглянути статистику, потрібно натиснути на відповідний пункт в меню «Театр» (рис. 5.11).

Вікно налаштувань складається з випадаючого списку типів місць, двох лічильників, відповідающих за кількість рядів та місць, таблиці для демонстрації зовнішнього вигляду в залежності від параметрів лічильників, кнопки «Принять» (рис. 5.12).

Кнопка «Принять» фіксує нові розміри таблиці місць. Після фіксації можна закривати вікно налаштувань.

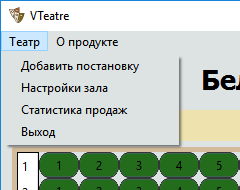


Рисунок 5.11 – Пункти меню «Театр»

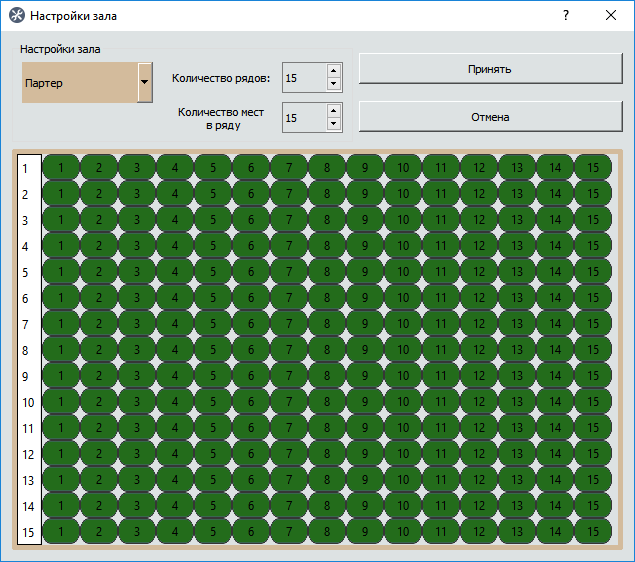


Рисунок 5.12 – Вікно налаштувань залу

Вікно додавання вистави складається з поля з назвою вистави, двох часових полів, трьох лічильників цін для кожного типу місць, кнопки «Добавить» (рис 5.13).

Щоб додати нову виставу, потрібно заповнити всі поля та натиснути кнопку «Добавить», після цього можна закрити вікно, чи продовжити додавання вистав.

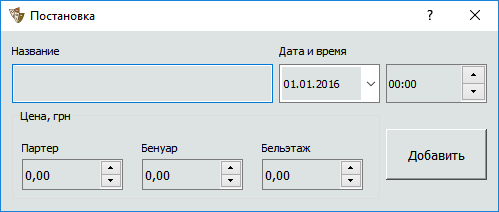


Рисунок 5.13 – Вікно додання вистав

Вікно статистики складається з трьох кнопок, кожна з якої відповідає за відображення статистики за різний проміжок часу, та координатної площини, де вісь Х відповідає за дату, а вісь Y за кількість придбаних місць (рис 5.14). На площині відображаються три графіка, кожен з яких відповідає за тип місця. Легенда відображає назви цих графіків.

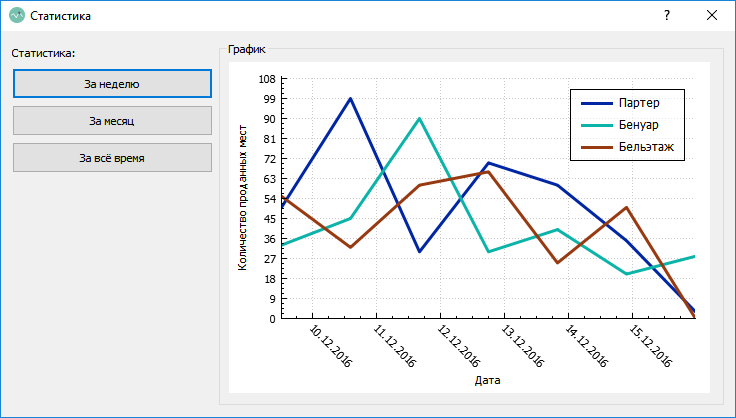


Рисунок 5.14 – Вікно статистики

При натисненні правою кнопкою на вистави, з’явиться контекстне мені, в якому можна здійснити такі операції як редагування вистави та її видалення (рис 5.15).

Якщо обрати вибрати пункт «Редактировать» то з’явиться вікно на аналог вікна створення вистави (рис 5.13) тільки з вже з заповненими даними про виставу. По завершенню редагування необхідних даних обов’язково натиснути кнопку «Редактировать» (рис 5.16) та закрити вікно

Якщо обрати пункт «Удалить» то дані про виставу назавжди зникнуть з програми і БД.

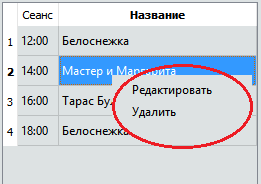


Рисунок 5.15 – Контекстне меню

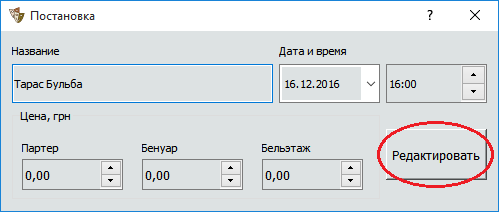


Рисунок 5.16 – Вікно редагування вистави

## 4.5 Повідомлення

Повідомлення:

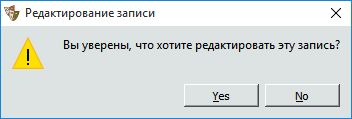


Рисунок 4.1 – Повідомлення при натисканні редагування вистави

Це повідомлення виникає перед користувачем коли він намагається редагувати вистави. Реагувати на це повідомлення потрібно залежно від ваших намірів

Попередження:

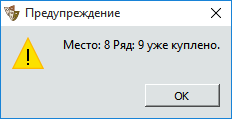


Рисунок 4.2 – Попереджує про змогу купити куплене місце

Це попередження з’являється перед користувачем у випадках, коли оператор хоче придбати місце яке вже було до цього придбано. Ця помилка не додасть дане місце в фінальне вікно оплати.

Повідомлення про помилку:

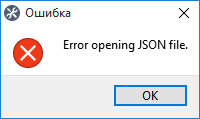


Рисунок 4.3 – Повідомлення про помилку відкриття файлу за даними

За випадків коли файл з даними відсутній в кореневому каталозі, виникає така помилка.

# Висновки

Під час розробки курсової роботи було створено програмне забезпечення, яке виконує функції бронювання та купівлі квитків в театр. Реалізований такий функціонал програми:

1. бронювання та придбання квитка на будь-яку виставу театру;
2. налаштування залу, а саме: кількість рядів та місць;
3. редагування, додавання, видалення будь-якої вистави;
4. відображення статистики по продажам квитків.

Програма розроблена на мові програмування С++ в середовищі Qt. Під час розробки програми, були використані засоби Qt та ООП.

За результатами тестування програми, всі її функції працюють коректно.

Було оформлено пояснювальну записку до курсової роботи, в якій було проведено аналіз програмної області та програмних засобів, описано основні рішення з реалізації компонентів системи, написано посібник програміста та інструкцію користувача.

# Список використаної літератури

1. Методичні вказівки до виконання курсової роботи з дисципліни “Об’єктно-орієнтоване програмування” для студентів напряму підготовки 6.050103 «Програмна інженерія» всіх форм навчання. Частина 1. Організаційна частина / Г.В. Табунщик, Г.В. Неласа, Н.О. Миронова – Запоріжжя: ЗНТУ, 2010. – 22 с.

2. Лафоре Р. Объектно-ориентированное программирование в C++ / Роберт Лафоре. – СПБ: Питер, 2004. – 924 с.

3. Шлее М. Qt 5.3. Профессиональное программирование на C++ / Макс Шлее. – СПБ: БХВ-Петербург, 2015. – 928 с.:ил.

4. Qt Documentation [Електронний ресурс] – Режим доступу до ресурсу: <http://doc.qt.io/>.

5. CyberForum [Електронний ресурс] – Режим доступу до ресурсу: <http://www.cyberforum.ru/>.

6. Хабрахабр [Електронний ресурс] – Режим доступу до ресурсу:

<https://habrahabr.ru/>.

# Додаток а Текст програми

## Модуль addscene.h

#ifndef ADDSCENE\_H

#define ADDSCENE\_H

#include <QDialog>

#include <QtSql>

#include <QList>

#include <scene.h>

namespace Ui {

class AddScene;

}

class AddScene : public QDialog

{

Q\_OBJECT

public:

explicit AddScene(QWidget \*parent = 0);

explicit AddScene(bool edit, Scene \*temp, QWidget \*parent = 0);

~*AddScene*();

QList <Scene> Scenes;

bool Editing;

QString PriviosName,

PriviosTime,

PriviosDate;

private slots:

void on\_ButtonAdd\_clicked();

private:

Ui::AddScene \*ui;

QSqlDatabase mydb;

};

#endif // ADDSCENE\_H

## Модуль dataforstatistic.h

#ifndef DATAFORSTATISTIC\_H

#define DATAFORSTATISTIC\_H

#include <QDate>

#include <QtSql>

#include <vector>

class DataforStatistic

{

int CountDays;

public:

DataforStatistic(int Days);

void InsertData();

QString \*\*\*DataOfPlaces;

};

#endif // DATAFORSTATISTIC\_H

## Модуль finalywindow.h

#ifndef FINALYWINDOW\_H

#define FINALYWINDOW\_H

#include <QDialog>

#include <scene.h>

namespace Ui {

class FinalyWindow;

}

class FinalyWindow : public QDialog

{

Q\_OBJECT

public:

explicit FinalyWindow(std::vector <short> SelectedPlacesRow, std::vector <short> SelectedPlacesCol, Scene \*CurScene, QString CurHoll, int CurIndex, bool \*verification, QWidget \*parent = 0);

~*FinalyWindow*();

private slots:

void on\_pushButtonOk\_clicked();

private:

Ui::FinalyWindow \*ui;

void CreateTable(std::vector <short> SelectedPlacesRow, std::vector <short> SelectedPlacesCol, Scene \*CurScene, QString CurHoll, int CurIndex);

double Cost;

bool \*verific;

};

#endif // FINALYWINDOW\_H

## Модуль mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QtSql>

#include <QMainWindow>

#include <QDate>

#include <QPixmap>

#include <QGraphicsScene>

#include "about.h"

#include "vector"

#include "string"

#include <QtDebug>

#include <QFileInfo>

#include <QTableView>

#include "statistic.h"

#include "optionsforhall.h"

#include "addscene.h"

#include "scene.h"

#include "finalywindow.h"

#include <vector>

#include <QMessageBox>

#include <ctime>

#include <QtSvg>

namespace Ui {

class MainWindow;

}

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = 0);

~MainWindow();

Scene \*CurScene;

QDateTime dt; // переменная для записи текущей даты

int row\_height; // высота строки

int column\_width; // ширина столбца

bool \*\*coordinates\_of\_places;//запоминаю координаты столбика и строчки

void coordinates\_of\_places\_cleaning(int temp);

//\*\*\*ПЕРЕМЕННЫЕ ОТ САНИ\*\*\*

std::vector <short> SelectedPlacesRow;

std::vector <short> SelectedPlacesCol;

int PreviousIndex;

int CountRow; //Количество строк

int CountColumn; //Количество столбцов

int CountPurchased; //количество купленных

int CountBooked; //количество забронированных

//\*\*\*ВОТ НА ЭТОМ МОИ ПЕРЕМЕННЫЕ ВСЬО\*\*\*

private slots:

void on\_informationAbout\_triggered();

void on\_tableWidget\_cellClicked(int row, int column);

void on\_comboBox\_currentIndexChanged(int index);

void on\_tableSeans\_cellClicked(int row, int column);

void on\_dateEdit\_dateChanged(const QDate &date);

void on\_pushButton\_clicked();

void on\_pushButton\_2\_clicked();

void on\_action\_exit\_triggered();

void on\_action\_statistic\_sale\_triggered();

void on\_pushButton\_3\_clicked();

void on\_options\_room\_triggered();

void on\_action\_addScene\_triggered();

void on\_action\_hovered();

void on\_tableSeans\_customContextMenuRequested(const QPoint &pos);

void slotEditRecord();

void slotDelRecord();

void on\_pushButtonCode\_clicked();

void main\_window\_resize(QResizeEvent \*event);

private:

Ui::MainWindow \*ui;

QSqlDatabase mydb;

void places\_fill();//заполняет места

void create\_a\_MainTable();

void places\_overwrite(int row, int column);//удаляет дубликаты в базе

void pix\_close(int row, int column);//красит в красный

void pix\_checking(int row, int column);//красит в синий

void pix\_reserv(int row, int column);//красит в желтый

void pix\_standart(int row, int column);//красит в зелёный

void customizeTableInf();

void DeleteBooked(std::vector <short> PurRow, std::vector<short>PurCol);

void CustomizePrice();

virtual void resizeEvent(QResizeEvent \*event);

signals:

void MyWindowReSize(QResizeEvent \*event);

};

#endif // MAINWINDOW\_H

## Модуль optionsforhall.h

#ifndef OPTIONSFORHALL\_H

#define OPTIONSFORHALL\_H

#include <QDialog>

#include <QtSql>

#include <QtDebug>

#include <QFileInfo>

#include <QTableView>

#include <QtXml>

namespace Ui {

class OptionsForHall;

}

class OptionsForHall : public QDialog

{

Q\_OBJECT

public:

explicit OptionsForHall(QWidget \*parent = 0);

~*OptionsForHall*();

double dataJsonFile[3][2];

QString type\_placeJsonFile[3];

/\*int CountRow;

int CountColumn;

int TypeOfPlace; // 1 - Партер 2 - Бенуар 3 - Бельэтаж

int Array[3][2];\*/

private slots:

void on\_spinBox\_Row\_valueChanged(int arg1);

void on\_spinBox\_Column\_valueChanged(int arg1);

void on\_ComboBoxType\_currentIndexChanged(int index);

void on\_pushButtonOK\_clicked();

void on\_pushButtonCancel\_clicked();

private:

void parseJSON(QJsonDocument& jsonDoc);

Ui::OptionsForHall \*ui;

QSqlDatabase mydb;

void DrawTable();

};

#endif // OPTIONSFORHALL\_H

## Модуль scene.h

#ifndef SCENE\_H

#define SCENE\_H

#include <QString>

#include<optionsforhall.h>

#include <QtSql>

#include <vector>

class Scene

{

public:

Scene();

void set\_name(QString arg);

void set\_date(QString arg);

void set\_time(QString arg);

void set\_cost();

void set\_cost(int par, int ben, int bel);

void SetArrayCountPlaces();

void SetDataToTables();

void DeleteTables();

void InsertTablesToDataBase(std::vector<short> SelectedPlacesRow, std::vector<short> SelectedPlacesCol, int Index, int operation, QString Code = ""); // Обновляем базу данных

int get\_name();

int get\_date();

int get\_time();

double get\_cost();

int ArrayCountPlaces[3][2]; // Хранит количество мест для каждого типа зала

int \*\*\*TablesPlaces;

int CountOfTypesPlaces;

QString name,

date,

time;

double cost\_parter,

cost\_benuar,

cost\_beletaj;

QString nameofplace[3];

std::vector<double>Cost;

};

#endif // SCENE\_H

## Модуль statistic.h

#ifndef STATISTIC\_H

#define STATISTIC\_H

#include <QDialog>

#include "dataforstatistic.h"

namespace Ui {

class Statistic;

}

class Statistic : public QDialog

{

Q\_OBJECT

public:

explicit Statistic(QWidget \*parent = 0);

~*Statistic*();

private slots:

void on\_pushButton\_clicked();

void on\_pushButton\_2\_clicked();

private:

Ui::Statistic \*ui;

void CreateGraph(DataforStatistic \*data, int CountDays);

};

#endif // STATISTIC\_H

## Модуль addscene.cpp

#include "addscene.h"

#include "ui\_addscene.h"

AddScene::AddScene(QWidget \*parent) :

QDialog(parent),

ui(new Ui::AddScene)

{

ui->setupUi(this);

mydb = QSqlDatabase::addDatabase("QSQLITE"); // Подключение базы данных

QString path = qApp->applicationDirPath() + "/VTeatre.sqlite";

mydb.setDatabaseName("./VTeatre.sqlite");

mydb.open();

Editing = false; //Флаг редактирования

}

AddScene::AddScene(bool Edit,Scene \*temp, QWidget \*parent) :

QDialog(parent),

ui(new Ui::AddScene)

{

ui->setupUi(this);

Editing = Edit; //Флаг редактирования

PriviosName= temp->name; //Предыдущее имя

PriviosDate = temp->date; //Предыдущая дата

PriviosTime = temp->time; //Предыдущее время

ui->NameLineEdit->setText(temp->name); //Устанавливаем необходимое название постановки в поле

ui->timeEdit->setTime(QTime::fromString(temp->time)); //а так же дату и время и цены

ui->dateEdit->setDate(QDate::fromString(temp->date, "dd.MM.yyyy"));

ui->ParterPrice->setValue(temp->Cost[0]);

ui->BenuarPrice->setValue(temp->Cost[1]);

ui->BeletazPrice->setValue(temp->Cost[2]);

ui->ButtonAdd->setText("Редактировать");

}

AddScene::~AddScene()

{

delete ui;

}

void AddScene::on\_ButtonAdd\_clicked() // Нажатие на кнопку "Добавить"

{

if(Editing)

{

QString name = ui->NameLineEdit->text(),

time = ui->timeEdit->text(),

date = ui->dateEdit->text();

double cost\_parter = ui->ParterPrice->value(),

cost\_benuar = ui->BenuarPrice->value(),

cost\_beletaj = ui->BeletazPrice->value();

QSqlQuery qry\_update1("update Postanovka set name='" +name+ "', time\_seansa='" + time + "', date\_seansa='" +date+ "', cost\_parter = "+QString::number(cost\_parter)+", cost\_benuar = "+QString::number(cost\_benuar)+ ", cost\_beletaj = " +QString::number(cost\_beletaj)+ " where name='" +PriviosName+ "' and time\_seansa='" +PriviosTime+ "' and date\_seansa='" +PriviosDate+ "'");

QSqlQuery qry\_update2("update Employed\_place set name\_seansa='" +name+ "', time\_seansa='" + time + "', date\_seansa='" +date+ "' where name\_seansa='" +PriviosName+ "' and time\_seansa='" +PriviosTime+ "' and date\_seansa='" +PriviosDate+ "'");

Editing = false;

}

else {

QSqlQuery qry\_select;

if(!qry\_select.exec("select \* from Postanovka where date\_seansa='" +ui->dateEdit->text()+ "' and time\_seansa='"+ui->timeEdit->text()+"'"))

{

qDebug()<<qry\_select.lastError().databaseText();

}

qry\_select.first();

QSqlQuery qry\_insert;

if(qry\_select.value(2).toString()!=ui->dateEdit->text() && qry\_select.value(1).toString()!=ui->timeEdit->text())

{

qry\_insert.exec("insert into Postanovka(name, time\_seansa, date\_seansa, cost\_parter, cost\_benuar, cost\_beletaj) values('" + ui->NameLineEdit->text()+

"', '" + ui->timeEdit->text()+

"', '"+ui->dateEdit->text()+

"', "+QString::number(ui->ParterPrice->value())+

", "+QString::number(ui->BenuarPrice->value())+

", "+QString::number(ui->BeletazPrice->value())+")");

}

else {qDebug()<<"1";}

//QSqlQuery qry\_insert1;

QSqlQuery qry\_select1;

if(!qry\_select1.exec("select \* from DataforStatistic where date='" +ui->dateEdit->text()+ "'"))

{

qDebug()<<qry\_select1.lastError().databaseText();

}

qry\_select1.first();

if(qry\_select1.value(3).toString()!=ui->dateEdit->text())

{

if(!qry\_insert.exec("insert into DataforStatistic(type\_place, date, count\_income, count\_place) values ('Партер', '" +ui->dateEdit->text()+ "', 0, 0)"))

qDebug()<<qry\_insert.lastError().databaseText();

if(!qry\_insert.exec("insert into DataforStatistic(type\_place, date, count\_income, count\_place) values ('Бенуар', '" +ui->dateEdit->text()+ "', 0, 0)"))

qDebug()<<qry\_insert.lastError().databaseText();

if(!qry\_insert.exec("insert into DataforStatistic(type\_place, date, count\_income, count\_place) values ('Бельэтаж', '" +ui->dateEdit->text()+ "', 0, 0)"))

qDebug()<<qry\_insert.lastError().databaseText();

}

}

}

## Модуль dataforstatistic.cpp

#include "dataforstatistic.h"

DataforStatistic::DataforStatistic(int Days)

{

CountDays = Days; //Устанавливаем количество дней

DataOfPlaces = new QString \*\*[Days]; //Выделяем память для хранения даты и

for(int i = 0; i < CountDays; i++){ //количества купленных мест за определенный

DataOfPlaces[i] = new QString \* [3]; //промежуток времени

for(int j = 0; j < 3; j++)

DataOfPlaces[i][j] = new QString [2];

}

}

void DataforStatistic::InsertData(){

QSqlQuery qry("select \* from DataforStatistic order by date desc"); //Подключаю БД

for(int i = 0; i < CountDays; i++){ //Заполняю массив значениями из БД

for(int j = 0 ; j < 3 && qry.next(); j++){

DataOfPlaces[i][j][0] = qry.value(1).toString(); //Здесь количество мест в этот день

DataOfPlaces[i][j][1] = qry.value("date").toString(); //Здесь дата

}

}

}

## Модуль finalywindow.cpp

#include "finalywindow.h"

#include "ui\_finalywindow.h"

FinalyWindow::FinalyWindow(std::vector <short> SelectedPlacesRow, std::vector <short> SelectedPlacesCol, Scene \*CurScene, QString CurHoll, int CurIndex, bool \*verification, QWidget \*parent) : // конструктор с 2 параметрами

QDialog(parent),

ui(new Ui::FinalyWindow)

{

ui->setupUi(this);

verific = verification; //Флаг, указывающий на то, подтвердилли ли мы покупку

Cost = 0;

CreateTable(SelectedPlacesRow, SelectedPlacesCol, CurScene, CurHoll, CurIndex); //Создаем таблицу

ui->labelCost->setText("К оплате: " + QString::number(Cost) + "грн.");

}

FinalyWindow::~*FinalyWindow*()

{

delete ui;

}

void FinalyWindow::CreateTable(std::vector <short> SelectedPlacesRow, std::vector <short> SelectedPlacesCol, Scene \*CurScene, QString CurHoll, int CurIndex){

ui->tableWidget->setRowCount(SelectedPlacesRow.size()); //Устанавливаем необходимое количество строк

for(int i = 0; i < SelectedPlacesRow.size(); i++){ //Выделяю место

for(int j = 0; j < ui->tableWidget->columnCount(); j++){ //для каждой ячейки

ui->tableWidget->setItem(i, j, new QTableWidgetItem);

}

}

for(int i = 0; i < SelectedPlacesRow.size(); i++){ //Заполняю таблицу данными

ui->tableWidget->item(i, 0)->setText(CurScene->name);

ui->tableWidget->item(i, 1)->setText(CurHoll);

ui->tableWidget->item(i, 2)->setText(CurScene->date);

ui->tableWidget->item(i, 3)->setText(CurScene->time);

ui->tableWidget->item(i, 4)->setText(QString::number(SelectedPlacesRow[i] + 1));

ui->tableWidget->item(i, 5)->setText(QString::number(SelectedPlacesCol[i] + 1));

ui->tableWidget->item(i, 6)->setText(QString::number(CurScene->Cost[CurIndex]));

Cost += CurScene->Cost[CurIndex];

}

}

void FinalyWindow::on\_pushButtonOk\_clicked() //При подтверждении покупки

{

\*verific = true; //флаг, указывающий на то, подтверждили ли мы покупку

this->close(); //закрываю форму

}

## Модуль mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <string>

#include <QFileDialog>

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

QString path = qApp->applicationDirPath() + "/VTeatre.sqlite";

mydb = QSqlDatabase::addDatabase("QSQLITE"); //Подключаю БД

mydb.setDatabaseName("./VTeatre.sqlite");

mydb.open();

CurScene = new Scene(); //Создаю объект сцены (пока пустой)

CurScene->SetArrayCountPlaces(); //Заполняю массив количества мест для каждого из типа мест

CurScene->SetDataToTables();

PreviousIndex = 0; //Изначально индекс зала равен 0 - Партеру

CountPurchased = 0; //Изначально количество купленных

CountBooked = 0; //и забронированных равно 0

dt = QDateTime::currentDateTime(); //устанавливаю необходимую дату

ui->dateEdit->setDateTime(dt);

//ui->dateEdit->setMinimumDateTime(dt); // минимальная дата выставляется по текущей дате

ui->tableSeans->setEditTriggers(QAbstractItemView::NoEditTriggers); //запрет редактирования всех ячеек в таблице tableSeans

ui->tableWidget->setEditTriggers(QAbstractItemView::NoEditTriggers); //запрет редактирования всех ячеек в таблице tableWidget

ui->tableSeans->horizontalHeader()->setSectionResizeMode(0, QHeaderView::ResizeToContents); // Ширина столбца с датой по размеру контента

ui->tableSeans->horizontalHeader()->setSectionResizeMode(1, QHeaderView::Stretch); // Ширина столбца с названиями всё остальное пространство

coordinates\_of\_places = new bool\*[CurScene->ArrayCountPlaces[ui->comboBox->currentIndex()][0]]; //выделяю память для массива, содержащего информацию о каждом месте

for(int i = 0; i < CurScene->ArrayCountPlaces[ui->comboBox->currentIndex()][0]; i++)

coordinates\_of\_places[i] = new bool [CurScene->ArrayCountPlaces[ui->comboBox->currentIndex()][1]]; //и заполняю его значением false = "свободно"

for(int i=0; i < CurScene->ArrayCountPlaces[ui->comboBox->currentIndex()][0]; i++)

for(int j = 0; j < CurScene->ArrayCountPlaces[ui->comboBox->currentIndex()][1]; j++)

{

coordinates\_of\_places[i][j]=false;

}

ui->comboBox->setEnabled(false); //Изначально combobox

ui->pushButton->setEnabled(false); //Кнопки оплаты, бронирования

ui->pushButton\_2->setEnabled(false); //поиска, возврата и строки для ввода кода не доступны

ui->pushButton\_3->setEnabled(false);

ui->pushButtonCode->setEnabled(false);

ui->lineEditCode->setEnabled(false);

connect(this, SIGNAL(MyWindowReSize(QResizeEvent\*)), this, SLOT(main\_window\_resize(QResizeEvent\*))); //Соеденения сигнала и слота - используется в дальнейшем для изменения размеров виджетов

}

void MainWindow::pix\_close(int row, int column) //покрасить данную ячейку в красный цвет

{

QPixmap pix\_close1;

pix\_close1.load(":/image/image\_close.png");

pix\_close1 = pix\_close1.scaled(ui->tableWidget->columnWidth(0), ui->tableWidget->rowHeight(0));

ui->tableWidget->item(row, column)->setBackground(QBrush(pix\_close1));

}

void MainWindow::pix\_reserv(int row, int column) //покрасить данную ячейку в желтый цвет

{

QPixmap pix\_reserv;

pix\_reserv.load(":/image/image\_reserv.png");

pix\_reserv = pix\_reserv.scaled(ui->tableWidget->columnWidth(0), ui->tableWidget->rowHeight(0));

ui->tableWidget->item(row, column)->setBackground(QBrush(pix\_reserv));

}

void MainWindow::pix\_checking(int row, int column) //покрасить данную ячейку в голубой цвет

{

QPixmap pix\_checking;

pix\_checking.load(":/image/image\_checking.png");

pix\_checking = pix\_checking.scaled(ui->tableWidget->columnWidth(0), ui->tableWidget->rowHeight(0));

ui->tableWidget->item(row, column)->setBackground(QBrush(pix\_checking));

}

void MainWindow::pix\_standart(int row, int column) //покрасить данную ячейку в зеленый цвет

{

QPixmap pix\_standart;

pix\_standart.load(":/image/image.png");

pix\_standart = pix\_standart.scaled(ui->tableWidget->columnWidth(0), ui->tableWidget->rowHeight(0));

ui->tableWidget->item(row, column)->setBackground(QBrush(pix\_standart));

}

void MainWindow::coordinates\_of\_places\_cleaning(int temp) //Удаление и заполнения массива

{

int current\_temp = ui->comboBox->currentIndex(); //Храним значения для предыдущего зала, это необходимо для очищения

for(int i = 0; i < CurScene->ArrayCountPlaces[temp][0]; i++){

delete [] coordinates\_of\_places[i];

}

delete []coordinates\_of\_places;

coordinates\_of\_places = new bool\*[CurScene->ArrayCountPlaces[current\_temp][0]]; //выделяю память для массива, содержащего информацию о каждом месте

for(int i = 0; i < CurScene->ArrayCountPlaces[current\_temp][0]; i++)

coordinates\_of\_places[i] = new bool [CurScene->ArrayCountPlaces[current\_temp][1]]; //и заполня его значениями false = "свободно"

for(int i=0; i < CurScene->ArrayCountPlaces[current\_temp][0]; i++)

for(int j = 0; j < CurScene->ArrayCountPlaces[current\_temp][1]; j++)

{

coordinates\_of\_places[i][j]=false;

}

}

void MainWindow::places\_fill()//заполнение мест

{

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

for(int i=0; i < CurScene->ArrayCountPlaces[temp][0]; i++) //исходя из данных, занесенных в объект текущей сцены заполняется таблица мест

{

for(int j=0; j < CurScene->ArrayCountPlaces[temp][1]; j++)

{

if(CurScene->TablesPlaces[temp][i][j] == 0){

pix\_standart(i, j);

continue;

}

if(CurScene->TablesPlaces[temp][i][j] == 1){

pix\_close(i, j);

continue;

}

if(CurScene->TablesPlaces[temp][i][j] == 2){

pix\_reserv(i, j);

continue;

}

}

}

}

MainWindow::~*MainWindow*() //Деструктор

{

delete ui;

}

void MainWindow::on\_informationAbout\_triggered() //Окно About

{

About \*window = new About(this);

window->show();

}

void MainWindow::on\_tableWidget\_cellClicked(int row, int column) // по нажатию на ячейку

{

if(coordinates\_of\_places[row][column])//проверяет активно ли нажатое место

{

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

if(CurScene->TablesPlaces[temp][row][column] == 0) //Если да, тоокрашиваю его обратно в правильный цвет

pix\_standart(row,column);

if(CurScene->TablesPlaces[temp][row][column] == 1)

pix\_close(row,column);

if(CurScene->TablesPlaces[temp][row][column] == 2)

pix\_reserv(row,column);

coordinates\_of\_places[row][column] = false; //удаляю из массива с выделенными местами

for(int i = 0; i < SelectedPlacesRow.size(); i++){ //очищаю векторы, хранящие номера мест

if(SelectedPlacesRow[i] == row && SelectedPlacesCol[i] == column){

SelectedPlacesRow.erase(SelectedPlacesRow.begin() + i);

SelectedPlacesCol.erase(SelectedPlacesCol.begin() + i);

}

}

}

else

{ //Если же до этого ячейка не была выбрана

coordinates\_of\_places[row][column] = true; //то в массиве отмечаю как выделенную

pix\_checking(row,column); //меняю ее цвет

SelectedPlacesRow.push\_back(row); //И добавляю в вектор колонку и ряд этого места

SelectedPlacesCol.push\_back(column);

}

CustomizePrice();

}

void MainWindow::on\_comboBox\_currentIndexChanged(int index) // по изменению пункта в combobox

{

coordinates\_of\_places\_cleaning(PreviousIndex); //пересоздаю массив

create\_a\_MainTable(); //создаю таблицу мест

places\_fill(); //заполняю ее

customizeTableInf(); //изменяю таблицу информации о местах

SelectedPlacesRow.clear(); //очищаю векторы, хранящие номера мест

SelectedPlacesCol.clear();

PreviousIndex = index; //Предыдущий индекс обновляю на текущий

CustomizePrice();

}

void MainWindow::on\_tableSeans\_cellClicked(int row, int column) // по нажатию на название постановки

{

if(column == 1) //проверяю на нажатие именно на название

{

ui->comboBox->setEnabled(true); //Делаю доступными все элементы которые раньше были недоступны

ui->pushButton->setEnabled(true);

ui->pushButton\_2->setEnabled(true);

ui->pushButton\_3->setEnabled(true);

ui->pushButtonCode->setEnabled(true);

ui->lineEditCode->setEnabled(true);

delete CurScene; //удаляю устаревшую сцену

CurScene = new Scene(); //и создаю новую

CurScene->set\_name(ui->tableSeans->item(row, 1)->text()); //Устанавливаю для нее имя

CurScene->set\_time(ui->tableSeans->item(row, 0)->text()); //время

CurScene->set\_date(ui->dateEdit->text()); //дату

CurScene->set\_cost(); //и цены

CurScene->SetArrayCountPlaces(); //Количество мест для каждого зала

CurScene->SetDataToTables(); //и информацию о каждом месте

coordinates\_of\_places\_cleaning(ui->comboBox->currentIndex()); //очищаю массив выбранных мест

create\_a\_MainTable(); //создаю заново таблицу мест

places\_fill(); //заполняю ее

customizeTableInf(); //обновляю таблицу с информацией

SelectedPlacesRow.clear(); //очищаю векторы, хранящие номера мест

SelectedPlacesCol.clear();

ui->comboBox->setCurrentIndex(0);

QString name = ui->tableSeans->item(row,column)->text(); //Изменяю подпись текущей постаановки

ui->label\_4->setText(name);

}

CustomizePrice();

}

void MainWindow::on\_dateEdit\_dateChanged(const QDate &date) //После изменения текущей даты в календаре

{

QString date\_seansa = ui->dateEdit->text();

QSqlQuery qry1("select \* from Postanovka where date\_seansa='" + date\_seansa + "'");

std::vector<QString>Name;

std::vector<QString>Time;

while(qry1.next()) //считываю из базы данных информацию о времени и названии спектаклей на этот день

{

Time.push\_back(qry1.value(1).toString());

Name.push\_back(qry1.value(0).toString());//записывает данные спектакля в вектор

}

ui->tableSeans->setRowCount(Name.size()); // указываем количество строк

for(int i = 0; i < Name.size(); i++) //выделяю память

{

ui->tableSeans->setItem(i, 0, new QTableWidgetItem); // вставляем ячейку

ui->tableSeans->setItem(i, 1, new QTableWidgetItem); // вставляем ячейку

ui->tableSeans->item(i, 0)->setText(Time[i]); //и заполняю данными каждую ячейку

ui->tableSeans->item(i, 1)->setText(Name[i]);

}

}

void MainWindow::on\_pushButton\_clicked() // купить

{

bool \*verification = new bool [1]; //указатель, указывающий на то, подтверждена ли покупка

\*verification = false; //изначально равен false

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

std::vector <short> PurRow;

std::vector <short> PurCol;

for(int i = 0; i < SelectedPlacesRow.size(); i++){ //Если покупаю уже забронированные

if(CurScene->TablesPlaces[temp][SelectedPlacesRow[i]][SelectedPlacesCol[i]] == 2){

PurRow.push\_back(SelectedPlacesRow[i]); //То добавляю в массив для удаления

PurCol.push\_back(SelectedPlacesCol[i]);

}

if(CurScene->TablesPlaces[temp][SelectedPlacesRow[i]][SelectedPlacesCol[i]] == 1){//Если поукупаю уже купленные

QMessageBox::warning(this, "Предупреждение", "Место: " + QString::number(SelectedPlacesCol[i] + 1) + " Ряд: "+ QString::number(SelectedPlacesRow[i] + 1) +" уже куплено.");

SelectedPlacesRow.erase(SelectedPlacesRow.begin() + i); //То удаляю из вектора и ничего не делаю с ними

SelectedPlacesCol.erase(SelectedPlacesCol.begin() + i);

i--;

}

}

if(SelectedPlacesRow.size()){//если выбрано хотябы 1 не купленные место

FinalyWindow \*wnd = new FinalyWindow(SelectedPlacesRow, SelectedPlacesCol, CurScene, ui->comboBox->currentText(), ui->comboBox->currentIndex(), verification); //вызаваю окно подтверждения покупки

wnd->*exec*();

if(\*verification){ //если покупка действительно прошла успешно

DeleteBooked(PurRow, PurCol); //то удаляю из базы данных бронированные, которые мы уже купили

for(int i = 0; i < SelectedPlacesRow.size(); i++){ //И обновляю объект текущей сцены

CurScene->TablesPlaces[temp][SelectedPlacesRow[i]][SelectedPlacesCol[i]] = 1;

}

QSqlQuery qry0("select count\_place from DataforStatistic where date='" + ui->dateEdit->text()+ "' and type\_place='" +ui->comboBox->currentText()+ "'"); //Обновляю БД

qry0.first();

QSqlQuery qry("update DataforStatistic set count\_place="+ QString::number(SelectedPlacesRow.size() + qry0.value("count\_place").toInt()) +" where date='" +ui->dateEdit->text()+ "' and type\_place='" +ui->comboBox->currentText()+ "'");

CurScene->InsertTablesToDataBase(SelectedPlacesRow, SelectedPlacesCol, ui->comboBox->currentIndex(), 1);

customizeTableInf();

}

places\_fill(); //Обновляю таблицу мест

coordinates\_of\_places\_cleaning(ui->comboBox->currentIndex()); //очищаю массив

SelectedPlacesRow.clear(); //очищаю векторы, хранящие номера мест

SelectedPlacesCol.clear();

CustomizePrice(); //изменяю данные о количестве выбранных мест

}

}

void MainWindow::on\_pushButton\_2\_clicked() // забронировать

{

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

QString Code = "";

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

if(CurScene->TablesPlaces[temp][SelectedPlacesRow[i]][SelectedPlacesCol[i]] == 2 || CurScene->TablesPlaces[temp][SelectedPlacesRow[i]][SelectedPlacesCol[i]] == 1){

SelectedPlacesRow.erase(SelectedPlacesRow.begin() + i); //если бронирую уже забронированный

SelectedPlacesCol.erase(SelectedPlacesCol.begin() + i); //или купленный

i--; //то ничего с ними не делаю

}

else

CurScene->TablesPlaces[temp][SelectedPlacesRow[i]][SelectedPlacesCol[i]] = 2; //в противоположном случае - отмечаю как бронированное

}

if(SelectedPlacesCol.size()){ //если остались места, которые необходимо бронировать

srand(time(NULL));

for(int i = 0; i < 10; i++){ //генерирую случайный код

Code += QString::number(std::rand() % 10);

}

//QMessageBox::information(QMessageBox::NoIcon,QObject::tr("Бронирование"),tr("Ваш код: ") + Code);//и вывожу его

QMessageBox::information(this, QObject::tr("Бронирование"),tr("Ваш код: ") + Code );

}

CurScene->InsertTablesToDataBase(SelectedPlacesRow, SelectedPlacesCol, ui->comboBox->currentIndex(), 2, Code); //обновляю БД

places\_fill(); //Обновляю таблицу мест

coordinates\_of\_places\_cleaning(ui->comboBox->currentIndex()); //очищаю массив мест

customizeTableInf(); //изменяю данные таблицу информации

SelectedPlacesRow.clear(); //очищаю векторы, хранящие номера мест

SelectedPlacesCol.clear();

CustomizePrice(); //изменяю информацию о выделенных местах

}

void MainWindow::on\_action\_exit\_triggered() // пункт Выход

{

this->close();

}

void MainWindow::on\_action\_statistic\_sale\_triggered() // окно статистики

{

Statistic \*wnd = new Statistic;

wnd->*exec*();

}

void MainWindow::on\_pushButton\_3\_clicked()//вернуть

{

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

QSqlQuery qry0("select count\_place from DataforStatistic where date='" + ui->dateEdit->text()+ "' and type\_place='" +ui->comboBox->currentText()+ "'"); //Обновляю БД

qry0.first();

QSqlQuery qry("update DataforStatistic set count\_place="+ QString::number(qry0.value("count\_place").toInt() - SelectedPlacesRow.size()) +" where date='" +ui->dateEdit->text()+ "' and type\_place='" +ui->comboBox->currentText()+ "'");

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

if(CurScene->TablesPlaces[temp][SelectedPlacesRow[i]][SelectedPlacesCol[i]] == 0){ //проверяю на то, занято ли место

SelectedPlacesRow.erase(SelectedPlacesRow.begin() + i); //если да, то удаляю его из вектора

SelectedPlacesCol.erase(SelectedPlacesCol.begin() + i);

i--;

}

else

CurScene->TablesPlaces[temp][SelectedPlacesRow[i]][SelectedPlacesCol[i]] = 0; //если нет то заношу ищзменения в объект

}

CurScene->InsertTablesToDataBase(SelectedPlacesRow, SelectedPlacesCol, ui->comboBox->currentIndex(), 0); //и заношу данные изменения в БД

places\_fill(); //перерисовываю таблицу

coordinates\_of\_places\_cleaning(ui->comboBox->currentIndex()); //очищаю массив мест

customizeTableInf(); //обновляю таблицу информации

SelectedPlacesRow.clear(); //очищаю векторы, хранящие номера мест

SelectedPlacesCol.clear();

CustomizePrice(); //обновляю данные о выделенных местах

}

void MainWindow::create\_a\_MainTable() //создаю таблицу с местами

{

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

ui->tableWidget->setRowCount(CurScene->ArrayCountPlaces[temp][0]); //Устанавливаю необходимое число строк

ui->tableWidget->setColumnCount(CurScene->ArrayCountPlaces[temp][1]); //Устанавливаю необходичое число столбцов

for(int i=0; i < ui->tableWidget->rowCount(); i++){ //выделяю для каждой ячейки память

for(int j=0; j < ui->tableWidget->columnCount(); j++){

ui->tableWidget->setItem(i, j, new QTableWidgetItem);

}

}

row\_height = (ui->tableWidget->height() -10) / ui->tableWidget->rowCount(); //подсчитываю высоту и ширину ячеек

column\_width = (ui->tableWidget->width() -35) / ui->tableWidget->columnCount();

for(int i = 0; i < ui->tableWidget->rowCount(); i++)

{

ui->tableWidget->setRowHeight(i, row\_height); // высота строк

for(int j = 0; j < ui->tableWidget->columnCount(); j++)

{

if(i == 0)

ui->tableWidget->setColumnWidth(j, column\_width); // ширина столбцов

QTableWidgetItem \*item = new QTableWidgetItem;

item->setText(QString::number(j + 1));

item->setTextAlignment(Qt::AlignCenter);

item->setFlags(item->flags() & (~Qt::ItemIsSelectable)); // устанавливаю флаг ItemIsSelectable в false

item->setToolTip("Ряд " + QString::number(i+1));

ui->tableWidget->setItem( i, j, item );

}

}

ui->tableWidget->verticalHeader()->setStyleSheet("border-style: solid;" // стиль для хидера

"border-width: 1px;"

"border-color: black;"

"border-radius: 0px;"

"color: black;");

ui->tableWidget->setStyleSheet("QTableWidget{" // стиль для таблицы и tooltip-a

"border-style: solid;"

"border-width: 5px;"

"border-radius: 1px;"

"border-color: #d3bb9c;"

"}"

"QTableWidget QToolTip{"

"color:red;"

"border-style: solid;"

"border-width: 1px;"

"border-color: green;"

"border-radius: 0px;"

"}"

"QHeaderView{"

"background-color: transparent;"

"}");

ui->tableWidget->verticalHeader()->*setVisible*(true);

ui->tableWidget->verticalHeader()->setMaximumSectionSize(ui->tableWidget->verticalHeader()->defaultSectionSize());//Устанавливаю размеры вертикального хедера

ui->tableWidget->verticalHeader()->setMinimumSectionSize(ui->tableWidget->verticalHeader()->defaultSectionSize());

}

void MainWindow::on\_options\_room\_triggered() //Окно настройки зала

{

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

for(int i = 0; i < CurScene->ArrayCountPlaces[temp][0]; i++){ //очищаю массив мест

delete [] coordinates\_of\_places[i];

}

delete []coordinates\_of\_places;

OptionsForHall \*wind = new OptionsForHall(this); //открываю окно

wind->*exec*();

CurScene->DeleteTables(); //очищаю таблицы

CurScene->SetArrayCountPlaces(); //устанавливаю обновленные данные о количестве мест

CurScene->SetDataToTables(); //Заполняю таблицы данными

SelectedPlacesRow.clear(); //создаю таблицу с местами

SelectedPlacesCol.clear();

create\_a\_MainTable(); //создаю таблицу мест

places\_fill(); //заполняю таблицу мест

customizeTableInf(); //обновляю таблицу информации

coordinates\_of\_places = new bool\*[CurScene->ArrayCountPlaces[temp][0]]; //Выделяю память для массива мест

for(int i = 0; i < CurScene->ArrayCountPlaces[temp][0]; i++)

coordinates\_of\_places[i] = new bool [CurScene->ArrayCountPlaces[temp][1]];

for(int i=0; i < CurScene->ArrayCountPlaces[temp][0]; i++)

for(int j = 0; j < CurScene->ArrayCountPlaces[temp][1]; j++)

{

coordinates\_of\_places[i][j]=false;

}

}

void MainWindow::on\_action\_addScene\_triggered() // нажатие на "Добавить постановку"

{

AddScene \*window = new AddScene(this);

window->*exec*();

on\_dateEdit\_dateChanged(QDate::fromString(ui->dateEdit->text(), "dd.MM.yyyy")); //Обновляю таблицу сеансов

}

void MainWindow::customizeTableInf() //обновляю таблицу информации

{

CountPurchased = 0; //Количество купленных и бронированных

CountBooked = 0;

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

for(int i=0; i < CurScene->ArrayCountPlaces[temp][0] ; i++) //Подсчитываю количество купленных и бронированных

{

for(int j=0; j < CurScene->ArrayCountPlaces[temp][1]; j++)

{

if(CurScene->TablesPlaces[temp][i][j] == 1)

CountPurchased++;

if(CurScene->TablesPlaces[temp][i][j] == 2)

CountBooked++;

}

}

//И заполняю таблицу информации данными

int CountOfPlaces = CurScene->ArrayCountPlaces[temp][0] \* CurScene->ArrayCountPlaces[temp][1];

ui->tableInfo->item(0,0)->setText("Всего мест: " + QString::number(CountOfPlaces));

ui->tableInfo->item(0,1)->setText("Продано: " + QString::number(CountPurchased));

ui->tableInfo->item(1,1)->setText("Бронь: " + QString::number(CountBooked));

ui->tableInfo->item(1,0)->setText("Свободно: " + QString::number(CountOfPlaces - CountPurchased - CountBooked));

}

void MainWindow::on\_action\_hovered()

{

}

void MainWindow::on\_tableSeans\_customContextMenuRequested(const QPoint &pos)

{

/\* Создаем объект контекстного меню \*/

QMenu \* menu = new QMenu(this);

/\* Создаём действия для контекстного меню \*/

QAction \* editDevice = new QAction(trUtf8("Редактировать"), this);

QAction \* delDevice = new QAction(trUtf8("Удалить"), this);

/\* Подключаем СЛОТы обработчики для действий контекстного меню \*/

connect(editDevice, SIGNAL(triggered()), this, SLOT(slotEditRecord())); // Обработчик вызова диалога редактирования

connect(delDevice, SIGNAL(triggered()), this, SLOT(slotDelRecord())); // Обработчик вызова диалога редактирования

/\* Устанавливаем действия в меню \*/

menu->addAction(editDevice);

menu->addAction(delDevice);

/\* Вызываем контекстное меню \*/

menu->popup(ui->tableSeans->viewport()->mapToGlobal(pos));

}

void MainWindow::slotEditRecord()

{

int row = ui->tableSeans->currentRow();

if (QMessageBox::warning(this, trUtf8("Редактирование записи"), trUtf8("Вы уверены, что хотите редактировать эту запись?"), QMessageBox::Yes | QMessageBox::No) == QMessageBox::No)

return;

else {

Scene \*tempScene = new Scene();

QSqlQuery qry("select \* from Postanovka where name='" + ui->tableSeans->item(row, 1)->text() + "' and time\_seansa='" + ui->tableSeans->item(row, 0)->text() + "' and date\_seansa='" + ui->dateEdit->text() + "'");

qry.first();

tempScene->set\_time(ui->tableSeans->item(row, 0)->text());

tempScene->set\_name(ui->tableSeans->item(row, 1)->text());

tempScene->set\_date(ui->dateEdit->text());

tempScene->set\_cost(qry.value("cost\_parter").toInt(), qry.value("cost\_benuar").toInt(), qry.value("cost\_beletaj").toInt());

AddScene \*temp = new AddScene(true, tempScene, this);

temp->*exec*();

on\_dateEdit\_dateChanged(QDate::fromString(ui->dateEdit->text(), "dd.MM.yyyy"));

}

}

void MainWindow::slotDelRecord(){ //Удалить постановку

int row = ui->tableSeans->currentRow();

QSqlQuery qry1("delete from Postanovka where name='" + ui->tableSeans->item(row, 1)->text() + "' and time\_seansa='" + ui->tableSeans->item(row, 0)->text() + "' and date\_seansa='" + ui->dateEdit->text() + "'");

QSqlQuery qry2("delete from Employed\_place where name\_seansa='" + ui->tableSeans->item(row, 1)->text() + "' and time\_seansa='" + ui->tableSeans->item(row, 0)->text() + "' and date\_seansa='" + ui->dateEdit->text() + "'");

if(ui->tableSeans->item(row, 1)->text() == CurScene->name && ui->tableSeans->item(row, 0)->text() == CurScene->time && ui->dateEdit->text() == CurScene->date){

delete CurScene; //если удалил ту постановку, тоторая сейчас открыта

CurScene = new Scene();

ui->comboBox->setEnabled(false); //то необходимо ее закрыть

ui->pushButton->setEnabled(false); //и очистить все массивы и векторы которые с ней связаны

ui->pushButton\_2->setEnabled(false);

ui->pushButton\_3->setEnabled(false);

ui->pushButtonCode->setEnabled(false);

ui->lineEditCode->setEnabled(false);

ui->tableWidget->setVerticalHeader(false);

ui->tableWidget->verticalHeader()->setStyleSheet("border-width: 0px;");

ui->label\_4->setText("");

ui->tableWidget->setRowCount(0);

ui->tableWidget->setColumnCount(0);

customizeTableInf();

CustomizePrice();

SelectedPlacesRow.clear();

SelectedPlacesCol.clear();

}

on\_dateEdit\_dateChanged(QDate::fromString(ui->dateEdit->text(), "dd.MM.yyyy"));

}

void MainWindow::DeleteBooked(std::vector <short> PurRow, std::vector<short>PurCol) //удаляю забронированные места

{

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

QSqlQuery qry("delete from Employed\_place where type\_place = '" + QString::number(ui->comboBox->currentIndex()) +"' and date\_seansa = '" + CurScene->date + "' and time\_seansa='" + CurScene->time + "' and name\_seansa='" + CurScene->name + "' and row=" + QString::number(PurRow[i]) + " and column=" + QString::number(PurCol[i]));

}

}

void MainWindow::CustomizePrice(){ //изменение данных о выбранных местах

int CountChecked = 0;

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

for(int i=0; i < CurScene->ArrayCountPlaces[temp][0]; i++){ //подсчитываю количество выбранных

for(int j = 0; j < CurScene->ArrayCountPlaces[temp][1]; j++){

if(coordinates\_of\_places[i][j])

CountChecked++;

}

}

ui->LabelCountSel->setText("Выбрано: " + QString::number(CountChecked));//и изменяю данные таблицы

}

void MainWindow::on\_pushButtonCode\_clicked() //Поиск мест по введенному коду

{

places\_fill(); //перерисовываю таблицу мест

coordinates\_of\_places\_cleaning(ui->comboBox->currentIndex()); //очищаю массив мест

SelectedPlacesRow.clear(); //очищаю вектор с выбранными местами

SelectedPlacesCol.clear();

QSqlQuery qry("select \* from Employed\_place where code='" + ui->lineEditCode->text() + "' and name\_seansa='" + CurScene->name + "' and date\_seansa='" + CurScene->date + "' and time\_seansa='" + CurScene->time + "' and type\_place='" + ui->comboBox->currentText() + "'");

bool Enabled = true;

while(qry.next()){ //поиск в БД необходимых мест

coordinates\_of\_places[qry.value(5).toInt()][qry.value(6).toInt()] = true;

SelectedPlacesRow.push\_back(qry.value(5).toInt()); //если нашел, то дабвляю в вектор выбранных мест

SelectedPlacesCol.push\_back(qry.value(6).toInt());

pix\_checking(qry.value(5).toInt(), qry.value(6).toInt()); //и изменяю цвет ячейки

Enabled = false;

}

if(Enabled){//если не найдено, то вывести сообщение пользователю

QMessageBox::critical(this,QObject::tr("Ошибка"),tr("Для данной постановки код не найден."));

}

}

void MainWindow::*resizeEvent*(QResizeEvent\* e) //изменение размера. Из класса QObject

{

emit MyWindowReSize(e); //Посылаю сигнал для перерисовки некоторых виджетов

QWidget::*resizeEvent*(e); //После чего слот продолжает свою стандартную работу

}

void MainWindow::main\_window\_resize(QResizeEvent \*event)

{

}

## Модуль optionsforhall.cpp

#include "optionsforhall.h"

#include "ui\_optionsforhall.h"

#include <QtXml>

#include <QFileDialog>

#include <QDir>

#include <QMessageBox>

OptionsForHall::OptionsForHall(QWidget \*parent) :

QDialog(parent),

ui(new Ui::OptionsForHall)

{

ui->setupUi(this);

ui->tableWidget->setEditTriggers(QAbstractItemView::NoEditTriggers); //запрет редактирования всех ячеек в таблице tableWidget

QString path = qApp->applicationDirPath() + "/Data.json";

QString jsonFileName = "./Data.json";

QString jsonText;

QFile jsonFile(jsonFileName);

try{

if(jsonFile.*open*(QIODevice::ReadOnly | QIODevice::Text))

{

jsonText = jsonFile.readAll();

QJsonDocument jsonDoc = QJsonDocument::fromJson(jsonText.toUtf8());

parseJSON(jsonDoc);

jsonFile.*close*();

}

else{

throw "Error opening JSON file.";

}

}

catch(const char\*msg){

QMessageBox::critical(this, "Ошибка", msg);

}

}

OptionsForHall::~*OptionsForHall*()

{

delete ui;

}

void OptionsForHall::on\_spinBox\_Row\_valueChanged(int arg1)//при изменении количества рядов

{

int temp = ui->ComboBoxType->currentIndex();

dataJsonFile[temp][0]=arg1;

DrawTable();

}

void OptionsForHall::on\_spinBox\_Column\_valueChanged(int arg1)//при изменении количества мест в ряду

{

int temp = ui->ComboBoxType->currentIndex();

dataJsonFile[temp][1]=arg1;

DrawTable();

}

void OptionsForHall::DrawTable()

{

QTableWidgetItem \*item;

int temp = ui->ComboBoxType->currentIndex();

ui->tableWidget->setRowCount(dataJsonFile[temp][0]); //задаю таблице нужное количечство строк

ui->tableWidget->setColumnCount(dataJsonFile[temp][1]); //и столбцов

for(int i=0; i < dataJsonFile[temp][0]; i++){ //освобождаю для них память

for(int j=0; j < dataJsonFile[temp][1]; j++){

ui->tableWidget->setItem(i, j, new QTableWidgetItem);

}

}

int row\_height = (ui->tableWidget->height() - 10) / ui->tableWidget->rowCount(); //подсчитываю высоту строк

int column\_width = (ui->tableWidget->width() - 35) / ui->tableWidget->columnCount(); //и ширину столбцов

if(row\_height > 100) //задаю максимальные значения для высоты и ширины ячеек

row\_height = 100;

if(column\_width > 200)

column\_width = 200;

QPixmap pix1; //храню пиксмап с изображением зеленого овала

pix1.load(":/image/image.png");

pix1 = pix1.scaled(column\_width, row\_height); //устанавливаю для него нужный размер

for(int i = 0; i < ui->tableWidget->rowCount();i++)

{

ui->tableWidget->setRowHeight(i, row\_height); // высота строк

for(int j = 0; j < ui->tableWidget->columnCount(); j++)

{

if(i == 0)

ui->tableWidget->setColumnWidth(j, column\_width); // ширина столбцов

item = new QTableWidgetItem;

item->setBackground(QBrush(pix1));

item->setText(QString::number(j + 1));

item->setTextAlignment(Qt::AlignCenter);

item->setFlags(item->flags() & (~Qt::ItemIsSelectable)); // устанавливаю флаг ItemIsSelectable в false

item->setToolTip("Ряд " + QString::number(i+1));

ui->tableWidget->setItem( i, j, item ); //вставляю в ячейку созданный Item

}

}

ui->tableWidget->verticalHeader()->setFixedWidth(25); // размер хидера

ui->tableWidget->verticalHeader()->setStyleSheet("border-style: solid;" // стиль для хидера

"border-width: 1px;"

"border-color: black;"

"border-radius: 0px;"

"color: black;");

ui->tableWidget->setStyleSheet("QTableWidget{" // стиль для таблицы и tooltip-a

"border-style: solid;"

"border-width: 5px;"

"border-radius: 1px;"

"border-color: #d3bb9c;"

"}"

"QTableWidget QToolTip{"

"color:red;"

"border-style: solid;"

"border-width: 1px;"

"border-color: green;"

"border-radius: 0px;"

"}");

}

void OptionsForHall::parseJSON(QJsonDocument& jsonDoc)

{

QJsonObject jsonObject = jsonDoc.object();

QJsonArray optionsArray = jsonObject["options"].toArray();

for(int i=0; i<3; i++)

{

dataJsonFile[i][0] = optionsArray[i].toObject()["row"].toDouble();

dataJsonFile[i][1] = optionsArray[i].toObject()["column"].toDouble();

type\_placeJsonFile[i] = optionsArray[i].toObject()["type\_place"].toString();

}

int temp = ui->ComboBoxType->currentIndex();

ui->spinBox\_Row->setValue(dataJsonFile[temp][0]);

ui->spinBox\_Column->setValue(dataJsonFile[temp][1]);

}

void OptionsForHall::on\_ComboBoxType\_currentIndexChanged(int index)

{

ui->spinBox\_Row->setValue(dataJsonFile[index][0]);

ui->spinBox\_Column->setValue(dataJsonFile[index][1]);

}

void OptionsForHall::on\_pushButtonOK\_clicked() //при нажатии на "ОК" изменяю БД

{

QString path = qApp->applicationDirPath() + "./Data.json";

QString jsonFileName = "./Data.json";

if(jsonFileName != NULL)

{

QFile jsonFile(jsonFileName);

if(jsonFile.*open*(QIODevice::WriteOnly | QIODevice::Truncate))

{

QJsonObject json, json1;

QJsonArray data;

for(int i = 0; i < 3; i++)

{

json1["row"]=dataJsonFile[i][0];

json1["column"]=dataJsonFile[i][1];

json1["type\_place"]=type\_placeJsonFile[i];

data.append(json1);

}

json["options"]=data;

QJsonDocument saveDoc(json);

jsonFile.write(saveDoc.toJson());

jsonFile.*close*();

}

else

{

QMessageBox msgBox;

msgBox.setWindowTitle("Error");

msgBox.setText("Error writing JSON file.");

msgBox.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);

msgBox.*exec*();

}

}

}

## Модуль scene.cpp

#include "scene.h"

Scene::Scene()

{

name = date = time = "";

cost\_parter = cost\_benuar = cost\_beletaj = 0;

CountOfTypesPlaces = 3;

nameofplace[0] = "Партер";

nameofplace[1] = "Бенуар";

nameofplace[2] = "Бельэтаж";

}

void Scene::set\_name(QString arg){

name = arg;

}

void Scene::set\_time(QString arg){

time = arg;

}

void Scene::set\_cost(){

QSqlQuery qry("select cost\_parter, cost\_benuar, cost\_beletaj from Postanovka where name = '" + name + "' and time\_seansa='" + time + "' and date\_seansa='" + date + "'");

qry.next();

Cost.push\_back(qry.value(0).toDouble());

Cost.push\_back(qry.value(1).toDouble());

Cost.push\_back(qry.value(2).toDouble());

}

void Scene::set\_cost(int par, int ben, int bel){

Cost.push\_back(par);

Cost.push\_back(ben);

Cost.push\_back(bel);

}

void Scene::set\_date(QString arg){

date = arg;

}

void Scene::SetArrayCountPlaces(){

OptionsForHall options;

for(int i = 0; i < 3; i++){

ArrayCountPlaces[i][0] = options.dataJsonFile[i][0];

ArrayCountPlaces[i][1] = options.dataJsonFile[i][1];

}

}

void Scene::SetDataToTables(){

TablesPlaces = new int \*\* [CountOfTypesPlaces];

for(int i = 0; i < CountOfTypesPlaces; i++){

TablesPlaces[i] = new int \* [ArrayCountPlaces[i][0]];

for(int j = 0; j < ArrayCountPlaces[i][0]; j++)

TablesPlaces[i][j] = new int [ArrayCountPlaces[i][1]];

}

for(int k = 0; k < CountOfTypesPlaces; k++){

QSqlQuery qry("select \* from Employed\_place where type\_place='" + nameofplace[k] + "' and time\_seansa='" + time +"' and date\_seansa='" + date + "' and name\_seansa='" + name +"'" );

for(int i = 0; i < ArrayCountPlaces[k][0]; i++){

for(int j = 0; j < ArrayCountPlaces[k][1]; j++){

TablesPlaces[k][i][j] = 0;

}

}

while(qry.next()){

if(qry.value(5).toInt() < ArrayCountPlaces[k][0] && qry.value(6).toInt() < ArrayCountPlaces[k][1]){

if(qry.value(7).toString() == "Куплено"){

TablesPlaces[k][qry.value(5).toInt()][qry.value(6).toInt()] = 1;

}

if(qry.value(7).toString() == "Забронировано"){

TablesPlaces[k][qry.value(5).toInt()][qry.value(6).toInt()] = 2;

}

}

}

}

}

void Scene::DeleteTables(){

for(int k = 0; k < CountOfTypesPlaces; k++){

for(int i = 0; i < ArrayCountPlaces[k][0]; i++){

delete []TablesPlaces[k][i];

}

delete [] TablesPlaces[k];

}

delete [] TablesPlaces;

}

void Scene::InsertTablesToDataBase(std::vector<short> SelectedPlacesRow, std::vector<short> SelectedPlacesCol, int Index, int operation, QString Code){

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

int row = SelectedPlacesRow[i];

int column = SelectedPlacesCol[i];

if(operation == 0){//Вернуть

QSqlQuery qry("delete from Employed\_place where type\_place = '" + nameofplace[Index] +"' and date\_seansa = '" + date + "' and time\_seansa='" + time + "' and name\_seansa='" + name + "' and row=" + QString::number(row) + " and column=" + QString::number(column));

}

if(operation == 1 ){//Купить

QSqlQuery qry("insert into Employed\_place (type\_place, date\_seansa, time\_seansa, name\_seansa, row, column, state) values ('" + nameofplace[Index] + "', '" + date + "', '" + time + "', '" + name + "', " + QString::number(row) + ", " + QString::number(column) + ", 'Куплено')");

}

if(operation == 2){//Забронировать

QSqlQuery qry("insert into Employed\_place (code, type\_place, date\_seansa, time\_seansa, name\_seansa, row, column, state) values ('" + Code + "', '" + nameofplace[Index] + "', '" + date + "', '" + time + "', '" + name + "', " + QString::number(row) + ", " + QString::number(column) + ", 'Забронировано')");

}

}

}

## Модуль statistic.cpp

#include "statistic.h"

#include "ui\_statistic.h"

Statistic::Statistic(QWidget \*parent) :

QDialog(parent),

ui(new Ui::Statistic)

{

ui->setupUi(this);

//Changing ticks on y axis

//Making setup for physical graph

ui->widget->addGraph();

ui->widget->graph(0)->setPen(QPen(QColor("#0026A4"), 3, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));

//ui->widget->graph(0)->setBrush(QBrush(QPixmap("./balboa.jpg"))); // fill with texture of specified image

ui->widget->graph(0)->setLineStyle(QCPGraph::lsLine);

//ui->widget->graph(0)->setScatterStyle(QCPScatterStyle(QCPScatterStyle::ssDisc, 5));

ui->widget->graph(0)->setName("Партер");

//Making setup for emotional graph

ui->widget->addGraph();

ui->widget->graph(1)->setPen(QPen(QColor("#0BB3A8"), 3, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));

//ui->widget->graph(1)->setBrush(QBrush(QPixmap("./balboa.jpg"))); // fill with texture of specified image

ui->widget->graph(1)->setLineStyle(QCPGraph::lsLine);

//ui->widget->graph(1)->setScatterStyle(QCPScatterStyle(QCPScatterStyle::ssDisc, 5));

ui->widget->graph(1)->setName("Бенуар");

//Making setup for intelectual graph

ui->widget->addGraph();

ui->widget->graph(2)->setPen(QPen(QColor("#973A12"), 3, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));

//ui->widget->graph(2)->setBrush(QBrush(QPixmap("./balboa.jpg"))); // fill with texture of specified image

ui->widget->graph(2)->setLineStyle(QCPGraph::lsLine);

//ui->widget->graph(2)->setScatterStyle(QCPScatterStyle(QCPScatterStyle::ssDisc, 5));

ui->widget->graph(2)->setName("Бельэтаж");

//Подписываем оси Ox и Oy

ui->widget->xAxis->setLabel("Дата");

ui->widget->yAxis->setLabel("Количество проданных мест");

// configure bottom axis to show date instead of number:

ui->widget->xAxis->setTickLabelType(QCPAxis::ltDateTime);

ui->widget->xAxis->setDateTimeFormat("dd.MM.yyyy");

ui->widget->xAxis->setTickLabelRotation(45);

ui->widget->xAxis->setAutoTickStep(true);

ui->widget->xAxis->setTickLabels(false);

ui->widget->yAxis->setAutoTickStep(true);

ui->widget->yAxis->setTickLabels(false);

//ui->widget->xAxis->setTickStep(86400000);

//Making legend

ui->widget->legend->setVisible(true);

QFont legendFont = font(); // start out with MainWindow's font..

legendFont.setPointSize(9); // and make a bit smaller for legend

ui->widget->legend->setFont(legendFont);

ui->widget->legend->setBrush(QBrush(QColor(255,255,255,150)));

// by default, the legend is in the inset layout of the main axis rect. So this is how we access it to change legend placement:

ui->widget->axisRect()->insetLayout()->setInsetAlignment(0, Qt::AlignTop|Qt::AlignRight);

}

Statistic::~*Statistic*()

{

delete ui;

}

void Statistic::on\_pushButton\_clicked()

{

DataforStatistic \*data = new DataforStatistic(7);

data->InsertData();

CreateGraph(data, 7);

}

void Statistic::CreateGraph(DataforStatistic \*data, int CountDays){

QVector <double> date;

QVector <double> par;

QVector <double> ben;

QVector <double> bel;

int MaxValueY = 0;

for(int i = 0; i < CountDays; i++){

QDateTime t = QDateTime::fromString(data->DataOfPlaces[i][0][1], "dd.MM.yyyy");

date.push\_back(t.toTime\_t());

par.push\_back(data->DataOfPlaces[i][0][0].toDouble());

ben.push\_back(data->DataOfPlaces[i][1][0].toDouble());

bel.push\_back(data->DataOfPlaces[i][2][0].toDouble());

if(par[i] > MaxValueY)

MaxValueY = par[i];

if(ben[i] > MaxValueY)

MaxValueY = ben[i];

if(bel[i] > MaxValueY)

MaxValueY = bel[i];

}

for(int i = 0; i < CountDays; i++){

qDebug()<<par[i]<<" "<<date[i];

}

ui->widget->graph(0)->setData(date, par);

ui->widget->graph(1)->setData(date, ben);

ui->widget->graph(2)->setData(date, bel);

QDateTime s = QDateTime::fromString(data->DataOfPlaces[CountDays - 1][0][1], "dd.MM.yyyy");

QDateTime f = QDateTime::fromString(data->DataOfPlaces[0][0][1], "dd.MM.yyyy");

ui->widget->yAxis->setRange(0, MaxValueY \* 1.1);//Для оси Oy

ui->widget->xAxis->setRange(s.toTime\_t(), f.toTime\_t());//Для оси Ox

ui->widget->xAxis->setTickLabels(true);

ui->widget->yAxis->setTickLabels(true);

QVector<double> ticks;

for(int i = 0; i < MaxValueY + 10; i += MaxValueY / 10){

ticks.push\_back(i);

}

ui->widget->yAxis->setAutoTicks(false);

ui->widget->yAxis->setTickVector(ticks);

//И перерисуем график на нашем widget

ui->widget->replot();

qDebug()<<s.toTime\_t();

qDebug()<<f.toTime\_t();

}

void Statistic::on\_pushButton\_2\_clicked()

{

DataforStatistic \*data = new DataforStatistic(30);

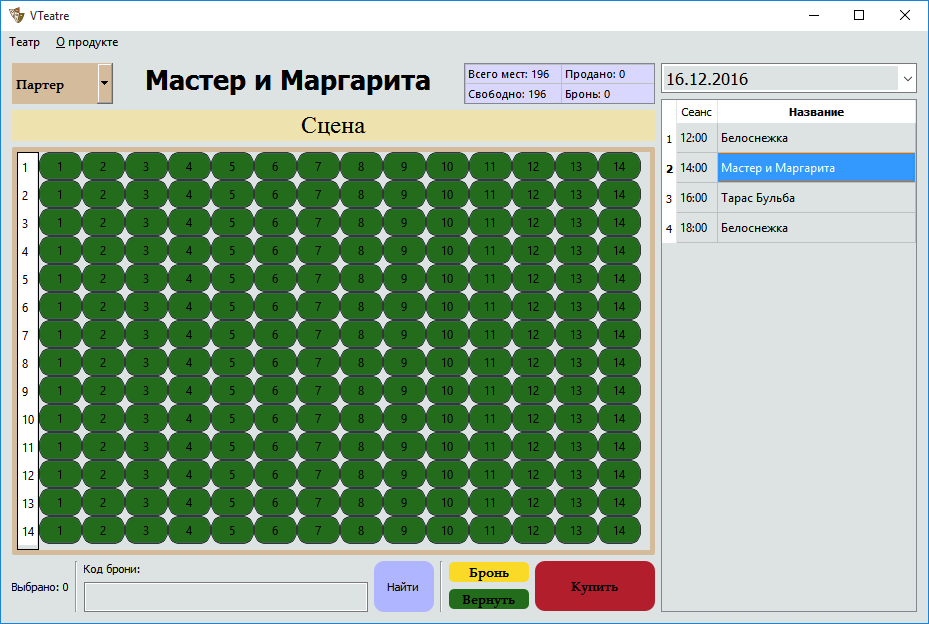
data->InsertData();

CreateGraph(data, 30);

}

# додаток б

# Інтерфейс програми

 Рисунок Б.1 – Головне вікно программи

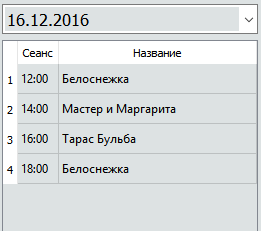


Рисунок Б.2 – Панель з виставами

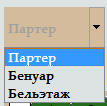


Рисунок Б.3 – Список з вибіром місця



Рисунок Б.4 – Панель з кнопками керування сценою



Рисунок Б.5 – Таблиця з інформацією

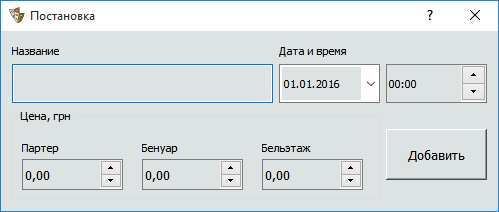


Рисунок Б.6 – Вікно додавання вистави

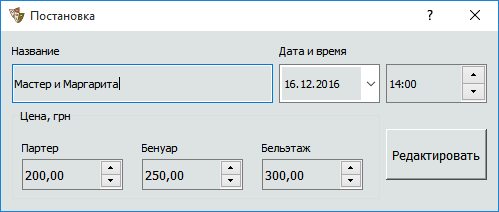


Рисунок Б.7 – Вікно редагування вистави

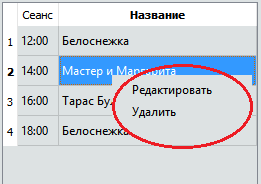


Рисунок Б.8 – Контекстне меню

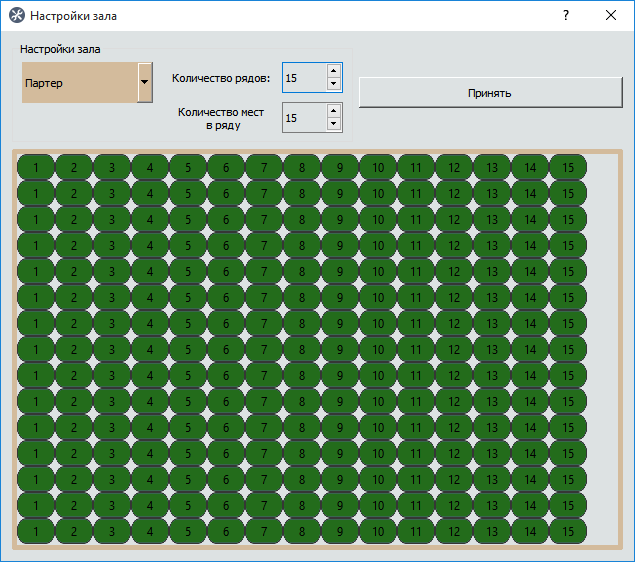


Рисунок Б.9 – Вікно налаштування залу

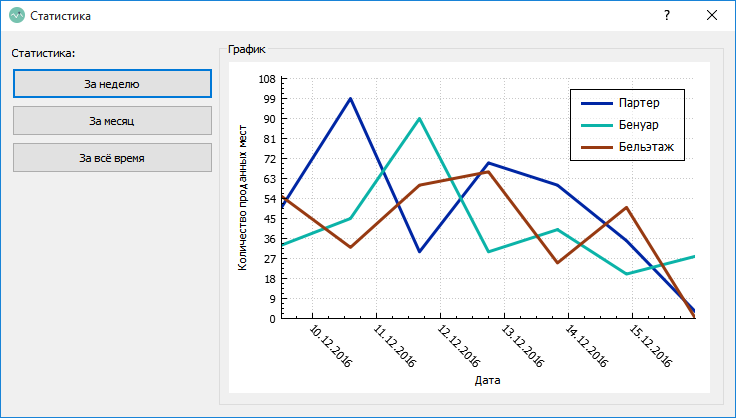


Рисунок Б.9 – Вікно статистики продаж