МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационные системы и технологии

Специальность 1-98 01 03 «Программное обеспечение информационной

безопасности мобильных систем»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

По дисциплине «Современные технологии программирования мобильных систем»

Тема Программное средство «Бронирование отеля»

Исполнитель

студентка 2 курса группы 7 Русецкая А.А.

(Ф.И.О.)

Руководитель

преподаватель стажер Павловская К.И.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Руководитель Павловская К.И.

(подпись)

Минск 2018

Оглавление

[Введение 3](#_Toc515275461)

[1. Постановка задачи 4](#_Toc515275462)

[1.1 Суть задачи 4](#_Toc515275463)

[1.2 Средства разработки 4](#_Toc515275464)

[2. Архитектура проекта 6](#_Toc515275465)

[2.1 Обобщенная структура 6](#_Toc515275466)

[2.2 UML диаграммы 7](#_Toc515275467)

[2.3 Системная архитектура 8](#_Toc515275468)

[2.4 Функции программы 9](#_Toc515275469)

[2.5 Поведение системы 10](#_Toc515275470)

[2.6 Взаимодействие объектов 11](#_Toc515275471)

[3. Разработка функционала приложения 12](#_Toc515275472)

[3.1 База данных 12](#_Toc515275473)

[3.2 Выполняемые функции 13](#_Toc515275474)

[3.2.1 Авторизация и регистрация 13](#_Toc515275475)

[3.2.2 Панель администатора 14](#_Toc515275476)

[3.2.3 Панель клиента 19](#_Toc515275477)

[4. Руководство по эксплуатации 22](#_Toc515275478)

[5. Тестирование приложения 29](#_Toc515275479)

[Заключение 32](#_Toc515275480)

[Список используемых источников 33](#_Toc515275481)

[ПРИЛОЖЕНИЕ А 34](#_Toc515275482)

[ПРИЛОЖЕНИЕ B 43](#_Toc515275483)

# Введение

На сегодняшний день информационные технологии связанны практически со всеми сферами жизни человека. С их развитием компьютеры и различные программные средства, активно применяются в сфере человеческой деятельности, связанной с обработкой информации и представлением данных.

Целью курсового проекта является разработка программного средства для бронирования номеров и администрирования отеля.

Сегодня гостиничный бизнес - одна из наиболее перспективных и быстроразвивающихся отраслей, приносящей по всему миру многомиллионные прибыли. С каждым годом растет количество гостиниц и отелей, как в нашей стране, так и за рубежом. Поэтому тема разработки система «Бронирование отеля» в настоящее время является актуальной, так как гостиничный бизнес активно развивается.

Программное средство «Бронирование отеля» предназначено для программного управления отелем: бронирование номеров администратором при обращении посетителей посредством разнообразных средств связи, контроль загрузки отеля, заселения и выселения гостей, и бронирования номеров клиентами. В панели администратора оно хранит сведения о номерах, доступных для бронирования, уже зарегистрированных бронях, сотрудниках и клиентах, а также позволяет получить программный отчет о заселениях/выселениях и свободных номерах на определенный день, полученных из базы данных. Производить данные операции с помощью программного средства удобно и быстро. На панели клиента гость может осуществить поиск по критериям и дате бронирования, получить список доступных номеров на выбранную дату, просмотреть свои бронирования.

Базы данных – это некоторый набор данных, организованный по определённым правила и имеющий определённую структуру. В данном курсовом проекте необходимо было освоить проектирование баз данных, а также подключение базы к программному средству для дальнейшей работы с ней.

WPF – технология для построения клиентских приложений Windows, она является частью платформы .NET и представляет собой подсистему для построения эстетически привлекательных графических интерфейсов.

Для работы с WPF использовался язык программирования C# – один из самых востребованных объектно-ориентированных языков IT-отрасли.

Значительно упрощает создание пользовательского интерфейса декларативный язык разметки XAML. Он позволяет создать видимые элементы пользовательского интерфейса, а затем отделить определение пользовательского интерфейса от логики выполнения, используя файлы кода программной части, присоединенные к разметке с помощью определений разделяемых классов.

# Постановка задачи

## 1.1 Суть задачи

Курсовая работа посвящена разработке программного средства «Бронирование отеля», предназначенное для работы с базой данных и пользователями. Приложение должно предоставлять пользователю возможность просмотра ранее зарегистрированных броней, поиск и бронирование номера, а администратору – корректную работу с данными. добавление, удаление, редактирование, поиск данных и просмотр отчетов.

Функции приложения:

* авторизация или регистрация пользователя;
* просмотр, изменение, удаление, добавление броней, сотрудников, клиентов;
* просмотр отчетности;
* поиск номера по критериям;
* просмотр своих броней для клиентов.

Аналоги данного приложения, конечно, существуют. В основном они реализованы не как веб-приложения, а с помощью облачных технологий, что для владельца отеля или его администратора является наиболее удобным средством управления гостиницы.

## 1.2 Средства разработки

При разработке приложения были использованы:

* интегрированная среда разработки Microsoft Visual Studio Community 2017 (версия 15.4.5);
* программная платформа .NET Framework 4.7.03056;
* язык объектно-ориентированного программирования C#;
* расширяемый язык разметки XAML;
* технология WPF;
* Объектно-ориентированная технология доступа к данным Entity Framework (версия 6.2.0);
* Microsoft SQL Server Management Studio (версия 11.0.2);
* Material Design Themes XAML Resources (версия 2.4.0.1044).

Microsoft Visual Studio — линейка продуктов компании [Microsoft](https://ru.wikipedia.org/wiki/Microsoft), включающих [интегрированную среду разработки](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8) программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как [консольные](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F) [приложения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), так и приложения с [графическим интерфейсом](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F), в том числе с поддержкой технологии [Windows Forms](https://ru.wikipedia.org/wiki/Windows_Forms), а также [веб-сайты](https://ru.wikipedia.org/wiki/%D0%A1%D0%B0%D0%B9%D1%82), [веб-приложения](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5), [веб-службы](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%BB%D1%83%D0%B6%D0%B1%D0%B0) как в [родном](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), так и в [управляемом](https://ru.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D1%8F%D0%B5%D0%BC%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) кодах для всех платформ, поддерживаемых [Windows](https://ru.wikipedia.org/wiki/Windows), [Windows Mobile](https://ru.wikipedia.org/wiki/Windows_Mobile), [Windows CE](https://ru.wikipedia.org/wiki/Windows_CE), [.NET Framework](https://ru.wikipedia.org/wiki/.NET_Framework), [Xbox](https://ru.wikipedia.org/wiki/Xbox), [Windows Phone](https://ru.wikipedia.org/wiki/Windows_Phone) [.NET Compact Framework](https://ru.wikipedia.org/wiki/.NET_Compact_Framework) и [Silverlight](https://ru.wikipedia.org/wiki/Silverlight) [1].

C# – язык программирования, сочетающий объектно-ориентированные и аспектно-ориентированные концепции. Разработан в 1998—2001 годах группой инженеров под руководством Андерсa Хейлсбергa в компании Microsoft как основной язык разработки приложений для платформы Microsoft .NET [1].

XAML – это декларативный язык, разработанный Microsoft и используемый для создания графических объектов пользовательского интерфейса. Он не является обязательной частью приложения, мы вообще можем обходиться без него, создавая все элементы в файле связанного с ним кода на языке C#. Однако использование XAML все-таки несет некоторые преимущества. При компиляции приложения в Visual Studio код в xaml-файлах также компилируется в бинарное представление кода xaml, которое называется BAML (Binary Application Markup Language). И затем код baml встраивается в финальную сборку приложения - exe или dll-файл [8].

Entity Framework представляет специальную объектно-ориентированную технологию на базе фреймворка .NET для работы с данными. Entity Framework представляет собой более высокий уровень абстракции чем технология ADO.NET, он позволяет абстрагироваться от самой базы данных и работать с данными независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, то на концептуальном уровне с объектами [3].

Центральной концепцией Entity Framework является понятие сущности или entity. Сущность представляет набор данных, ассоциированных с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их наборами.

Другим ключевым понятием является Entity Data Model. Эта модель сопоставляет классы сущностей с реальными таблицами в БД. Entity Data Model состоит из трех уровней: концептуального, уровень хранилища и уровень сопоставления (маппинга).

Entity Framework предполагает три возможных способа взаимодействия с базой данных:

* Database first: Entity Framework создает набор классов, которые отражают модель конкретной базы данных;
* Model first: сначала разработчик создает модель базы данных, по которой затем Entity Framework создает реальную базу данных на сервере;
* Code first: разработчик создает класс модели данных, которые будут храниться в бд, а затем Entity Framework по этой модели генерирует базу данных и ее таблицы.

В проекте был использован способ взаимодействия с базой под названием Database first.

Material Design ([рус.](https://ru.wikipedia.org/wiki/%D0%A0%D1%83%D1%81%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Материальный дизайн) — пакет NuGet, который предоставляет уже готовые шаблоны дизайна программного обеспечения и приложений операционных систем от компании [Google](https://ru.wikipedia.org/wiki/Google_(%D0%BA%D0%BE%D0%BC%D0%BF%D0%B0%D0%BD%D0%B8%D1%8F)) [4].

# 2. Архитектура проекта

## 2.1 Обобщенная структура

Структура проекта представлено на рисунке 2.1.

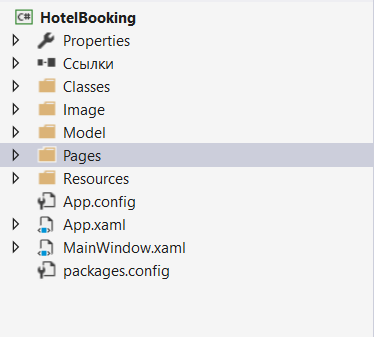


Рисунок 2.1 – Структура проекта HotelBooking

Структура проекта HotelBooking состоит из папки Classes, в которой определены классы: TextBoxWatermarked – для отображения подсказки в текстовом поле, DataGridSearchText – для поиска по таблице, Validation– для валидации данных, ListItem – элементы ListBox.

В папке Image находятся картинки номеров и иконки для приложения. Папка Model содержит модель сопоставления классов сущности с реальными таблицами в БД.

В папке Pages находятся окна приложения и их классы, в папке Resources находится словарь ресурсов стилей. Файл App.config – файл конфигураций, Packages.config – пакеты NuGet, установленные в проекте.

## 2.2 UML диаграммы

Для представления архитектуры, а точнее — различных входящих в нее структур, удобно использовать графические языки. На настоящий момент наиболее проработанным и наиболее широко используемым из них является унифицированный язык моделирования (Unified Modeling Language, UML), хотя достаточно часто архитектуру системы описывают просто набором именованных прямоугольников, соединенных линиями и стрелками, которые представляют возможные связи [2].

Язык UML представляет собой общецелевой язык визуального моделирования, который разработан для спецификации, визуализации, проектирования и документирования компонентов программного обеспечения, бизнес-процессов и других систем. Язык UML одновременно является простым и мощным средством моделирования, который может быть эффективно использован для построения концептуальных, логических и графических моделей сложных систем самого различного целевого назначения. Диаграммы рисуют для визуализации системы с разных точек зрения. Как правило, за исключением наиболее тривиальных случаев, диаграммы дают свернутое представление элементов, из которых составлена система. Один и тот же элемент может присутствовать во всех диаграммах, или только в нескольких (самый распространенный вариант), или не присутствовать ни в одной (очень редко) [7].

UML не заканчивается на моделировании всевозможного ПО. Также данный язык активно сегодня используется для моделирования различных бизнес-процессов, ведения системного проектирования, а также отображения организационных структур.

Диаграмма классов UML представляет собой статическую структурную диаграмму, предназначенную для описания структуры системы, а также демонстрации атрибутов, методов и зависимостей между несколькими различными классами. При моделировании объектно-ориентированных систем этот тип диаграмм используют чаще всего.

Существует четыре типа связей в UML:

* Зависимость – семантически представляет собой связь между двумя элементами модели, в которой изменение одного элемента (независимого) может привести к изменению семантики другого элемента (зависимого);
* Ассоциация – описывает набор связей, существующих между объектами;
* Обобщение – выражает специализацию или наследование, в котором специализированный элемент (потомок) строится по спецификациям обобщенного элемента (родителя);
* Реализация – это семантическая связь между классами, когда один из них (**поставщик**) определяет соглашение, которого второй (**клиент**) обязан придерживаться.

Диаграмма классов проекта представлена на рисунке 2.2.

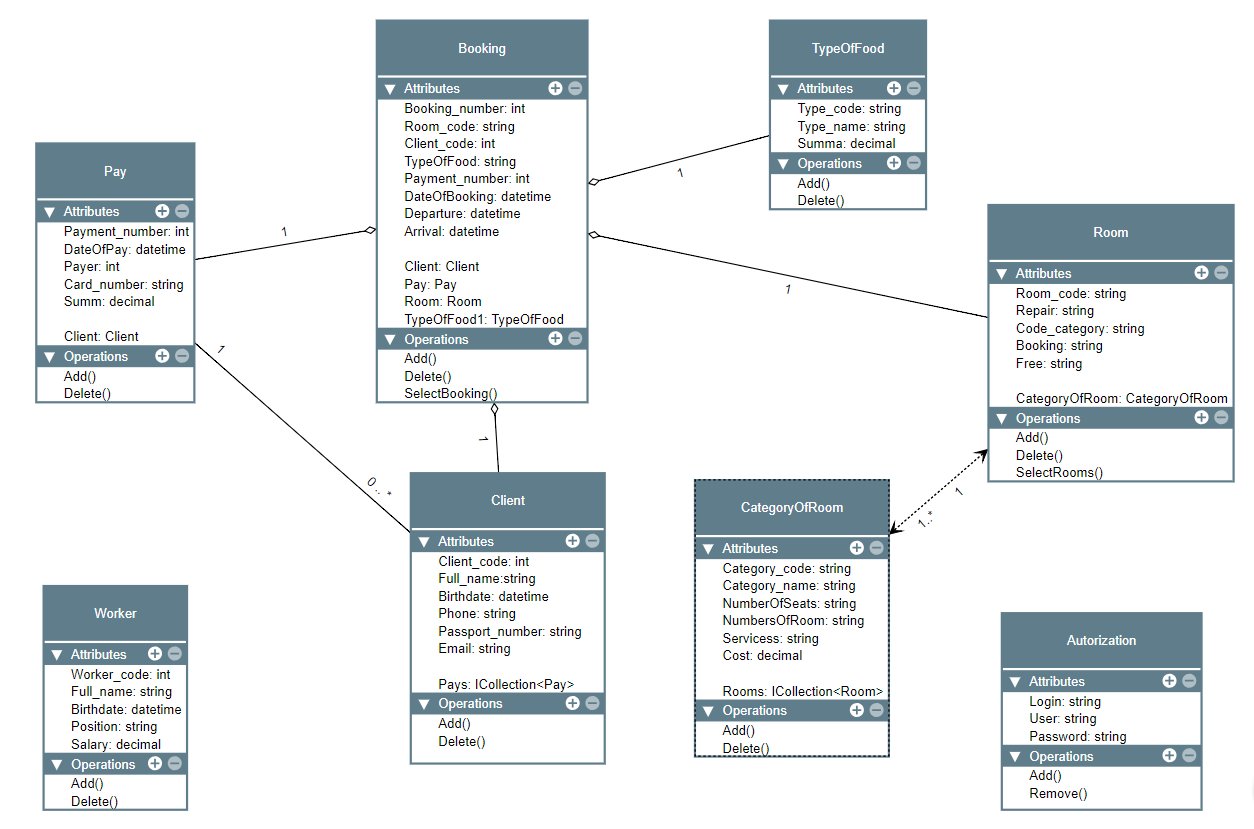


Рисунок 2.2 - Диаграмма классов проекта HotelBooking

## 2.3 Системная архитектура

Вообще говоря, не существует общепринятого термина «архитектура программного обеспечения». Тем не менее, когда дело касается практики, то для большинства разработчиков и так понятно какой код является хорошим, а какой плохим. Хорошая архитектура — это прежде всего выгодная архитектура, делающая процесс разработки и сопровождения программы более простым и эффективным. Программу с хорошей архитектурой легче расширять и изменять, а также тестировать, отлаживать и понимать.

Существует несколько типов системной архитектуры проекта, в частности одноуровневая, двухуровневая, трехуровневая. Для трехуровневой архитектуры характерно наличие User Interface, Business Logic, Data Base. Данный тип архитектуры применятся в программном обеспечении, которое осуществляет работу, с какой-либо базой данных. Одноуровневая системная архитектура не предусматривает деление на уровни. В данном случаи все действия реализации графического интерфейса и решения поставленной задачи выполняются на трех уровнях.

В данном проекте будет рационально использовать трехуровневую архитектуру. В этом случае разрабатываемая система разделяется на три уровня интерфейса пользователя, бизнес-логики, базы данных (рис. 2.3).

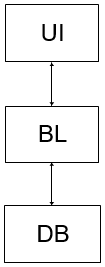


Рисунок 2.3 – Системная структура проекта

## 2.4 Функции программы

Приложение поддерживает разнообразные функции. Варианты использования показаны на диаграмме (рис. 2.4).

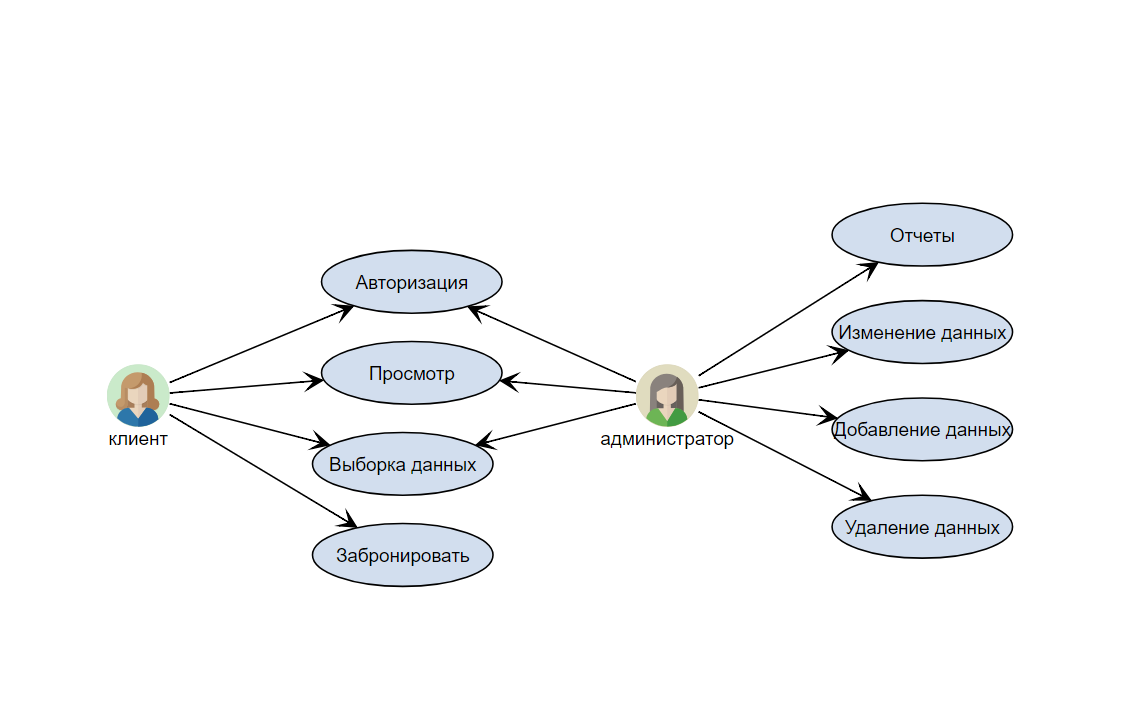


Рисунок 2.4 – Диаграмма вариантов использования

Клиент имеет возможность таких прецедентов как <Авторизироваться>, <Просмотр мои броней>, <Поиск номеров>, <Бронирование номера>.

Администратор имеет возможность таких прецедентов как <Просмотреть отчеты>, <Изменить данные>, <Добавить данные>, <Удалить данные>.

* 1. **Поведение системы**

Для моделирования процесса выполнения операций в языке UML используются диаграммы деятельности. Диаграмма деятельности — это своеобразная блок-схема, которая описывает последовательность выполнения операций во времени. Графически диаграмма деятельности представляется в виде графа, имеющего вершины и ребра.

Диаграмма только устанавливает правила обязательной последовательности действий, которым должен следовать пользователь. Это важно для моделирования бизнес-процессов, поскольку эти процессы часто выполняются параллельно. Такие диаграммы также полезны при разработке параллельных алгоритмов, в которых независимые потоки могут выполнять работу параллельно.

Диаграмма деятельности всех возможных вариантов для администратора приведена на рисунке 2.5.

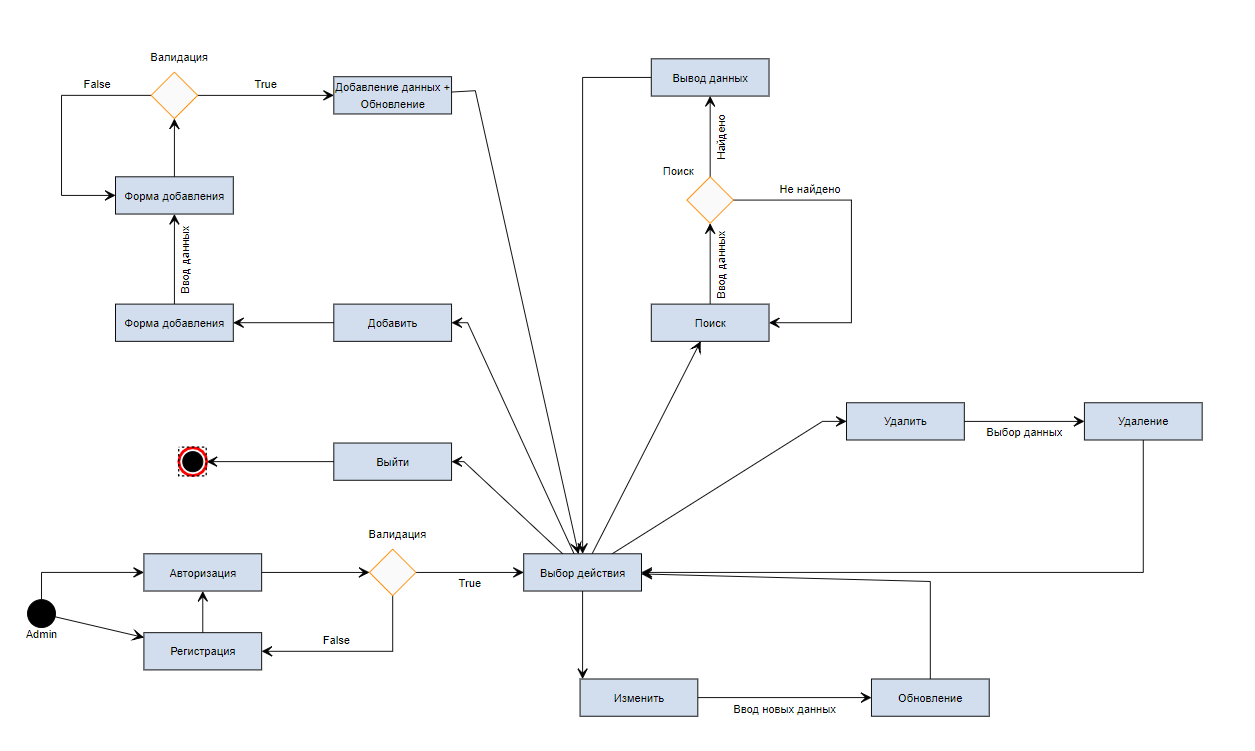


Рис 2.5 – Диаграмма деятельности для проекта

**2.6** **Взаимодействие объектов**

Диаграмма последовательности UML демонстрирует взаимодействия между несколькими объектами, которые упорядочиваются в соответствии с временем их проявления. На такой диаграмме отображается упорядоченное во времени взаимодействие между несколькими объектами. В частности, на ней отображаются все объекты, которые принимают участие во взаимодействии, а также полная последовательность обмениваемых ими сообщений. Главными элементами в данном случае выступают обозначения различных объектов, а также вертикальные линии, отображающие течение времени и прямоугольники, предоставляющие деятельность определенного объекта или же выполнение им какой-либо функции. Диаграмма сотрудничеств.

Представим диаграмму последовательности для функций, выполняемых программой (рис. 2.6).

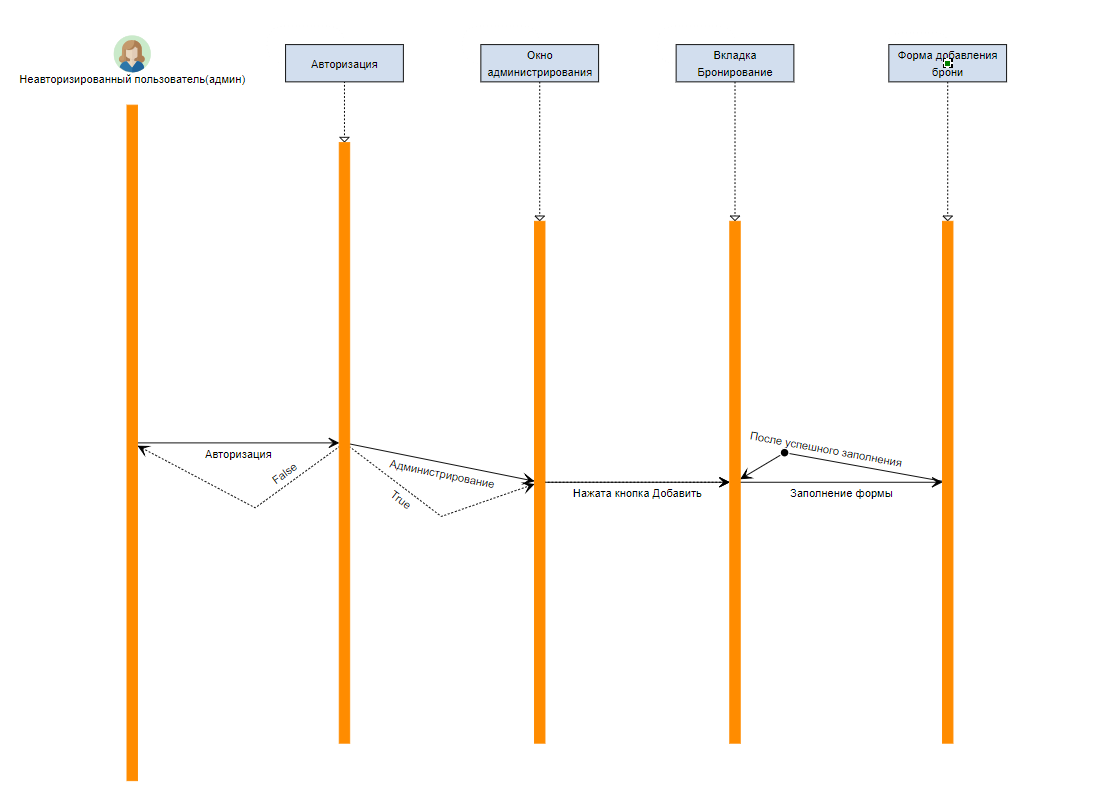


Рисунок 2.6 – Диаграмма последовательности

# 3. Разработка функционала приложения

## 3.1 База данных

База данных (БД) представляет собой совокупность структуриро­ванных данных, хранимых в памяти вычислительной системы и ото­бражающих состояние объектов и их взаимосвязей в рассматриваемой предметной области.

Логическую структуру данных, хранимых в базе, называют мо­делью представления данных. К основным моделям представления данных (моделям данных) относятся иерархическая, сетевая, реля­ционная [5].

Модель данных (в контексте области баз данных) описывает некий набор родовых понятий и признаков, которыми должны обладать все конкретные СУБД и управляемые ими базы данных, если они основываются на этой модели. Наличие модели данных позволяет сравнивать конкретные реализации, используя один общий язык.

Реляционная модель состоит из трех частей, описывающих разные аспекты реляционного подхода: структурной части, манипуляционной части и целостной части.

В структурной части модели фиксируется, что единственной родовой структурой данных, используемой в реляционных баз данных, является нормализованное N-арное отношение.

В манипуляционной части модели определяются два фундаментальных механизма манипулирования реляционными БД – реляционная алгебра и реляционное исчисление. Первый механизм базируется в основном на классической теории множеств (с некоторыми уточнениями и добавлениями), а второй – на классическом логическом аппарате исчисления предикатов первого порядка.

Реляционная модель основана на математическом понятии отношения, представлением которого является таблица. В реляционной модели отношения используются для хранения информации об объектах, представленных в базе данных. Отношение имеет вид двухмерной таблицы, в которой строки соответствуют записям, а столбцы – атрибутам. Каждая запись должна однозначно характеризоваться в таблице. Для этого используют первичные и вторичные ключи. Достоинством реляционной модели является простота и удобство физической реализации.

Реляционная модель базы данных подразумевает нормализацию всех таблиц данных. Нормализация – это формальный метод анализа отношений на основе их первичного ключа и функциональных зависимостей, существующих между их атрибутами.

На основании реляционной модели данных составим физическую структуру базы данных. Физическая структура базы данных представлена схемой данных на рисунке 3.1

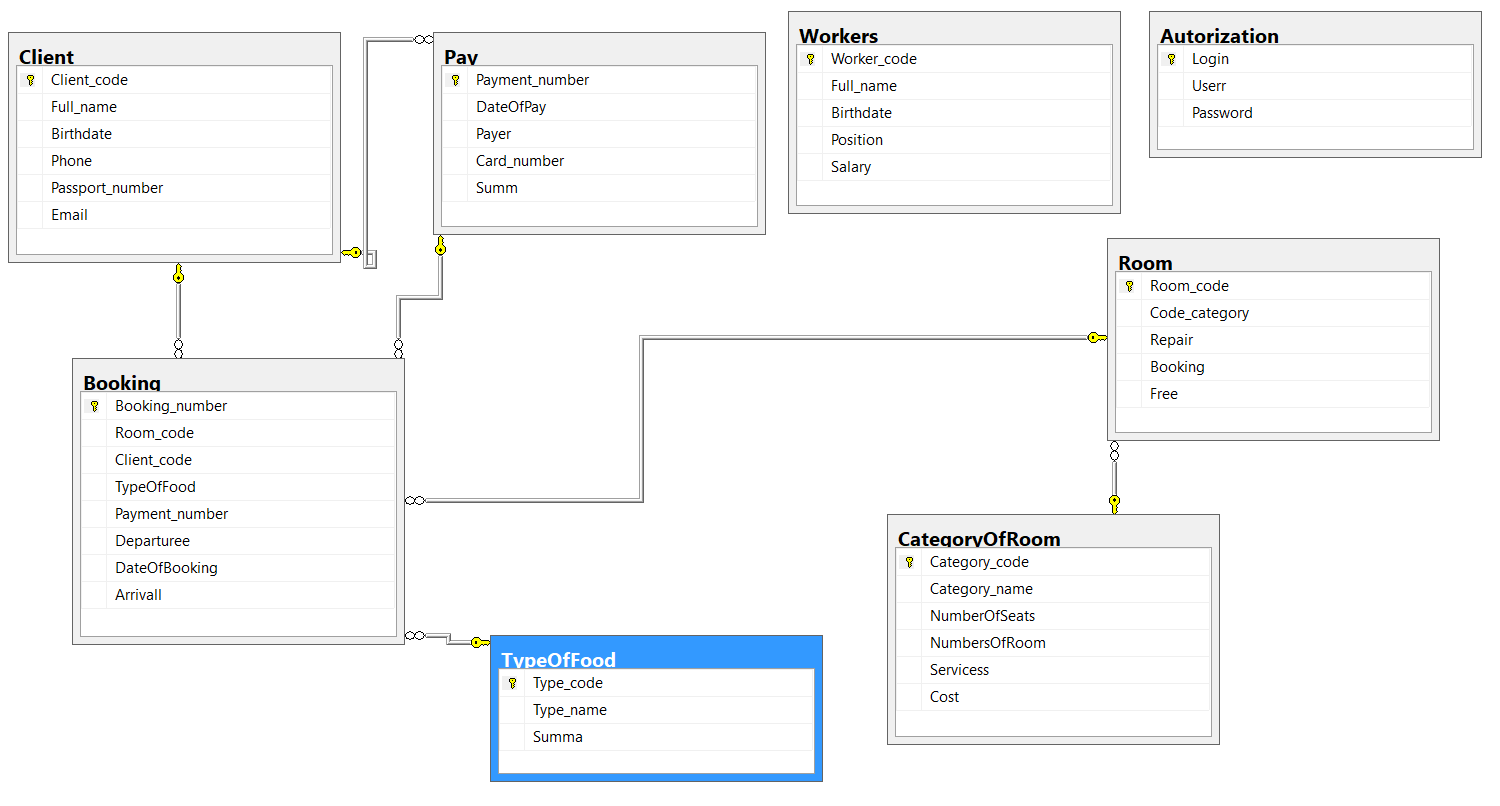


Рисунок 3.1 - Структура базы данных HotelBooking

## 3.2 Выполняемые функции

### **3.2.1 Авторизация и регистрация**

Авторизация пользователя происходит в начальном окне MainWindow.xaml. Функционал окна прописана в MainWindow.xaml.cs и имеет следующий вид, представленный на рисунке 3.2.

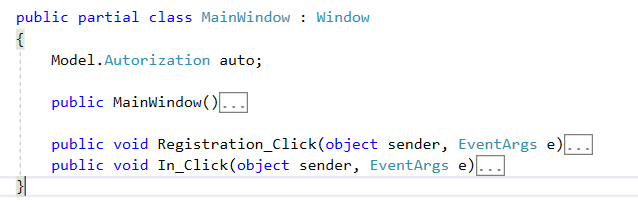


Рисунок 3.2 – Структура класса MainWindow.

Класс MainWindow содержит метод In\_Click, который вызывает процедуру Enter, а та, в свою очередь, делает проверку на действительность логина и пароля, возвращая нам сущность Autorization. Данный метод представлен на рисунке 3.3.

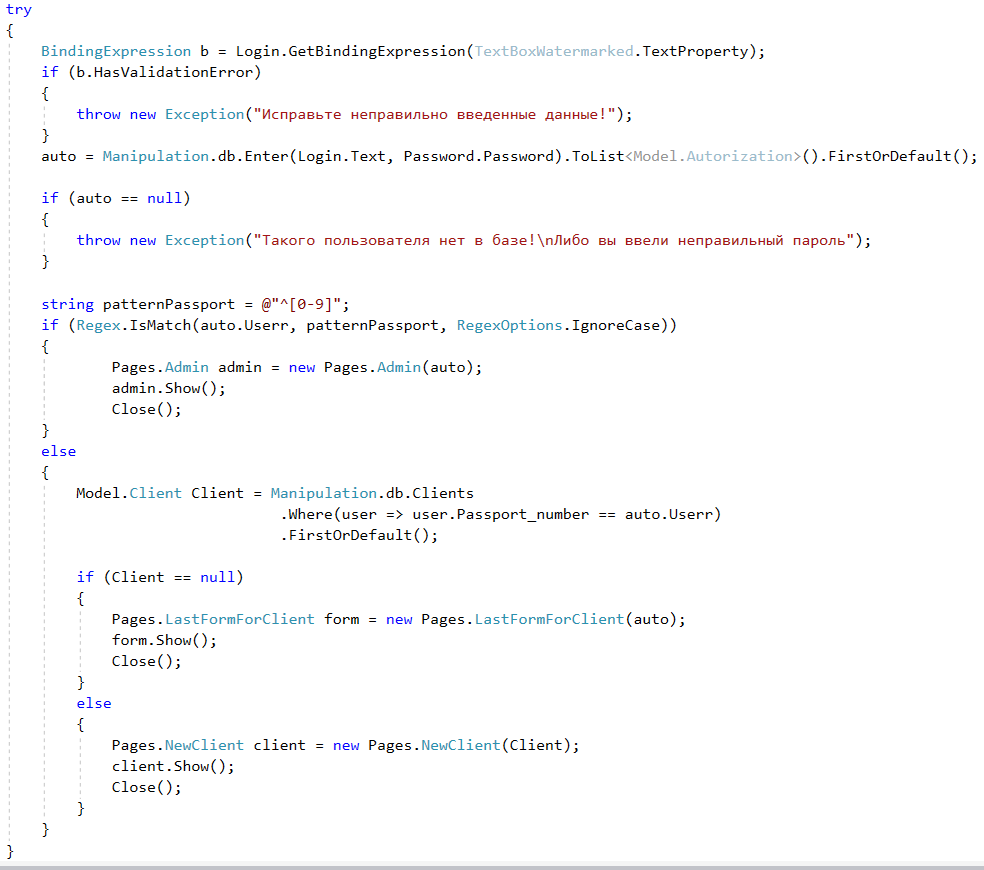


Рисунок 3.3 – Метод, реализующий вход пользователя

**3.2.2 Панель администатора**

Операции добавления, удаления и редактирования и поиск осуществляются на панели Администратора. Чтобы добавить данные о клиенте, брони или сотруднике в базу, необходимо нажать на кнопку «Добавить» на нужной вкладке, после чего откроется модальное окно для добавления, в котором нужно заполнить всю форму.

Cтруктура класса окна Admin представлена в приложении А.

Класс Admin содержит обработчики события кнопки добавления для каждой сущности (рисунок 3.4), которые впоследствии вызывают окно для заполнения данных. После заполнения текстовых полей окна Администратор нажимает кнопку «Добавить», которая обрабатывает данные и добавляет их в базу (рисунок 3.5).

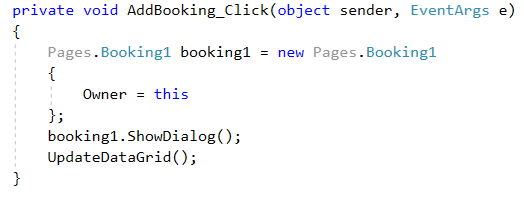


Рисунок 3.5 – Обработчик события кнопки, реализующий открытие окна добавления

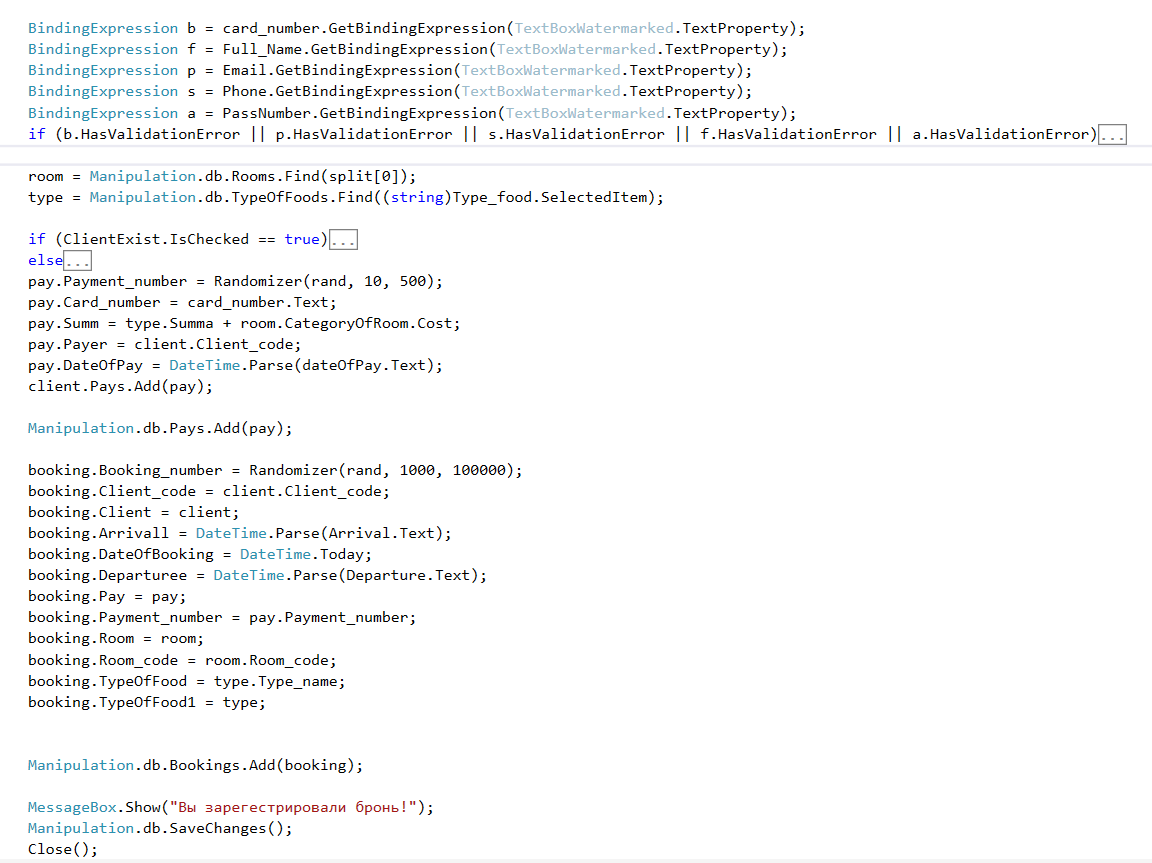


Рисунок 3.6 – Метод, реализующий добавление брони в базу

Для удаления данных в таблице DataGrid требуется выделить нужную строку и во вкладке в меню операций нажать кнопку «Удалить». Если пользователь не выберет строку, то выскочит предупреждение «Выберите строку, чтобы удалить ее». Пример обработчика события клика по кнопке приведен на рисунке 3.6.

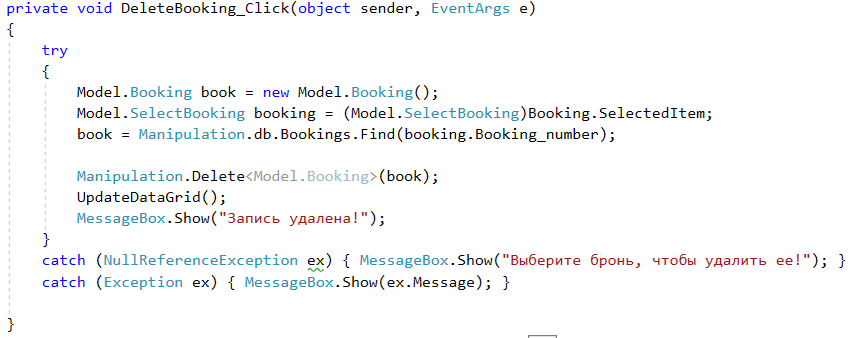


Рисунок 3.6 – Обработчик события кнопки, реализующий удаление брони

Редактирование происходит после двойного клика в ячейке таблицы DataGrid, после изменения данных вызывается событие CellEditEnding. Оно обрабатывается соответствующим обработчиком, который обновляет измененные данные в базе. Пример обработчика события представлен на рисунке 3.7.

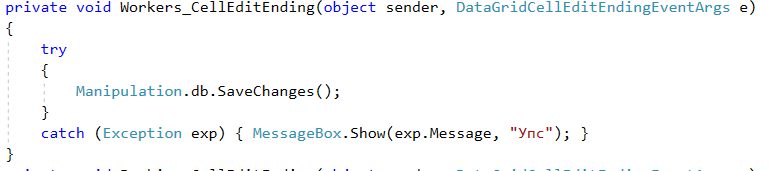


Рисунок 3.7 – Обработчик события ячейки, реализующий сохранение изменений

Чтобы воспользоваться поиском, необходимо в меню в поле ввода ввести данные для осуществления операции. Поиск производиться по колонке, в таблице DataGrid, – Главный гость. Операция происходи при вводе любого символа благодаря созданному свойству зависимости в классе DataGridSearchText (рисунок 3.8). Однако создание свойства зависимости не является конечным этапом, далее мы должны связать поле ввода с таблицей. Устанавливаем в атрибутах DataGrid «data:DataGridTextSearch.SearchValue="{Binding ElementName=Search, Path=Text, UpdateSourceTrigger=PropertyChanged}"».

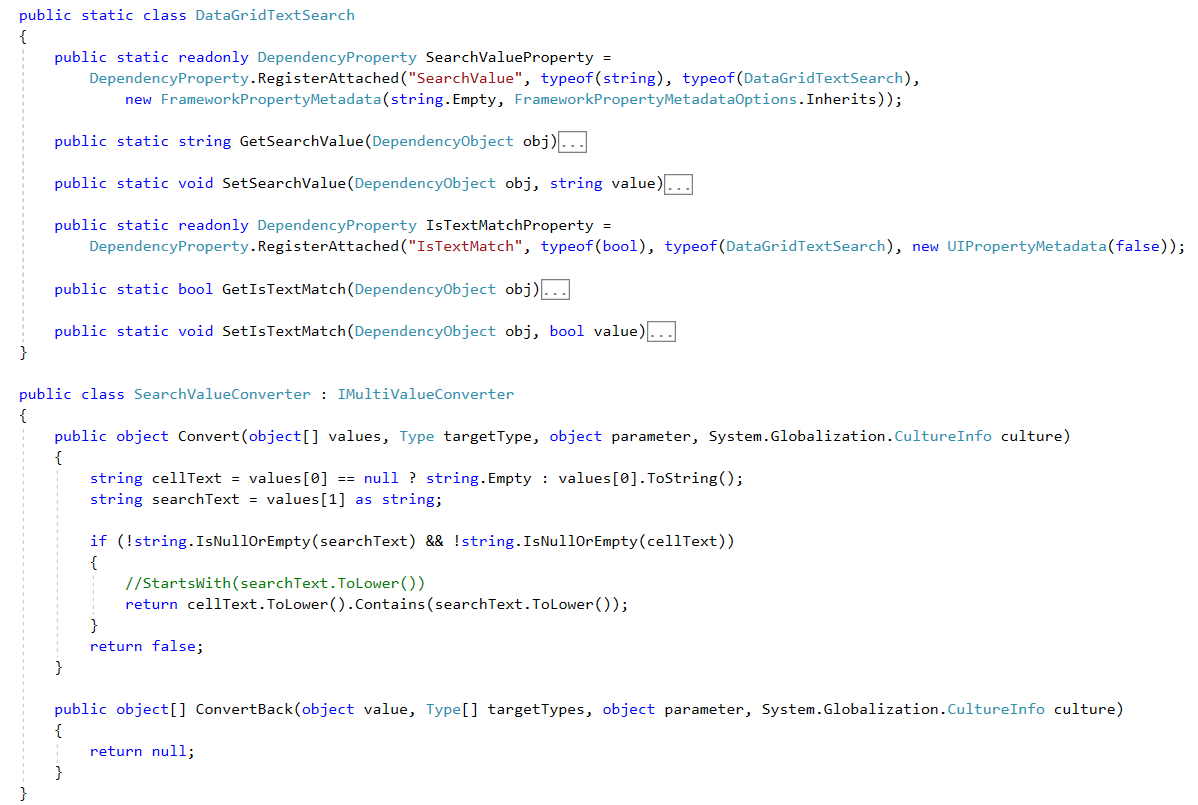


Рисунок 3.8 – Класс DataGridSearchText

Так же Администратор может просматривать отчеты о заселении/выселении и свободных номерах на любую дату. Для этого нужно перейти во вкладку Отчетность, выбрать нужный отчет и нажать кнопку «+», ввести дату и нажать кнопку «Показать». Далее на экране будет изображена таблица с данными, либо окошко с сообщением «На текущую дату выездов не запланировано».

Обработчик события кнопки «Показать» вызывает процедуру из базы данных SelectBooking, с передачей в нее 3 параметров поиска – (Дата заезда, Дата выезда, количество человек). Процедура находит все записи в таблице, которые удовлетворяют условию и возвращает их в наборе. Результирующим набором является сложный тип SelectBooking, который создается с помощью средств нашей модели сущностей. Для этого в списке процедур мы должны вызвать контекстное меню правой клавишей и выбрать пункт «Правка». Далее исправляем, если требуется, тип возвращаемого значения.

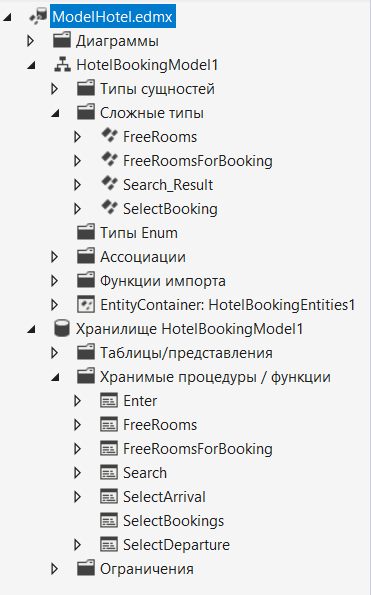


Рисунок 3.9 – Процедуры и сложные типы проекта

Сложные типы и процедуры, используемые в проекте можно посмотреть на рисунке 3.9.

Пример обработчика события для предоставления отчета показан на рисунке 3.10.



Рисунок 3.10 – Метод, реализующий получение отчета о свободных номерах

**3.2.3 Панель клиента**

Cтруктура класса окна NewClient (панель клиента) представлена в приложении B.

Класс NewClient содержит обработчик события кнопки Found\_Click(), который обращается к процедуре Search в базе данных для поиска номеров соответствующих заданным критериям (рисунок 3.11). В результате мы получаем коллекцию сложных типов Search\_Result. Далее разбираем каждый элемент и приводим к требуемому типу, чтобы поместить все в ListBox, в котором определен шаблон каждого элемента списка.

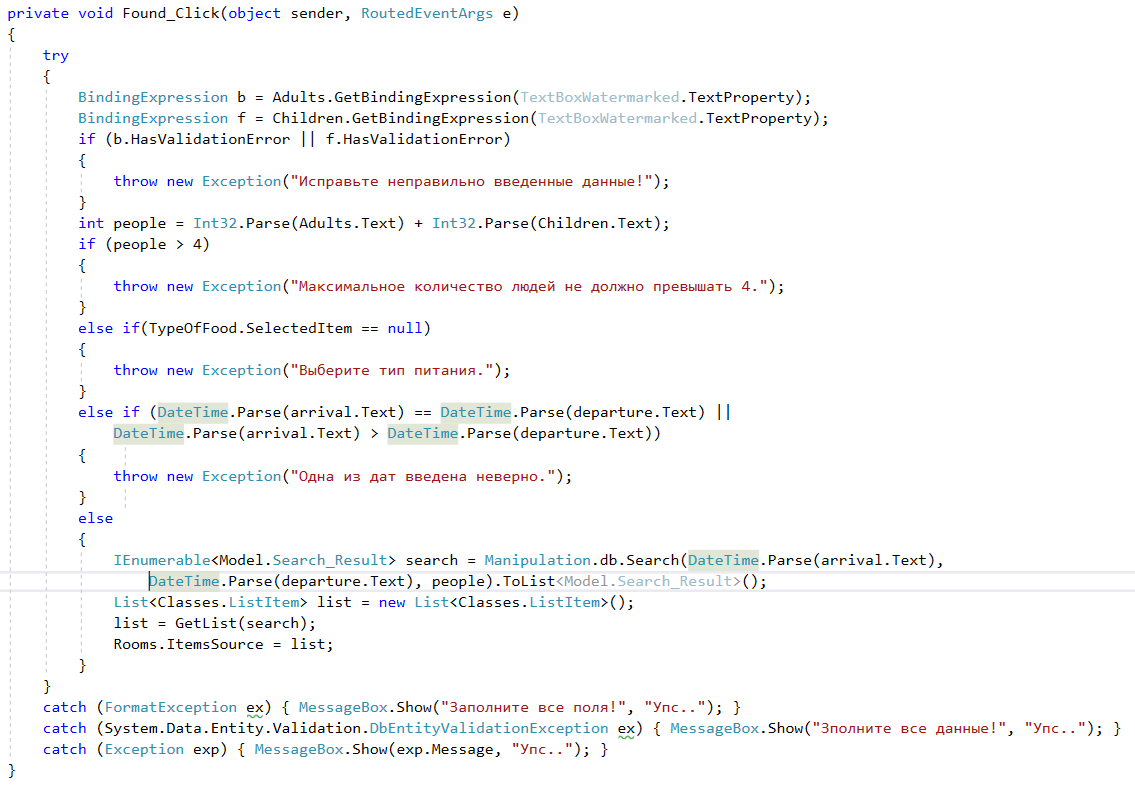


Рисунок 3.11 – Обработчик события кнопки, реализующий поиск номеров по заданным критериям

Когда список результатов заполнен элементами, пользователь может просмотреть все доступные номера и забронировать номер, нажав кнопку «Понравилось? Забронируйте». Обработчик события клика по этой кнопке представлен на рисунке 3.12. Для клиента так же высчитывается скидка на основе уже существующих броней.

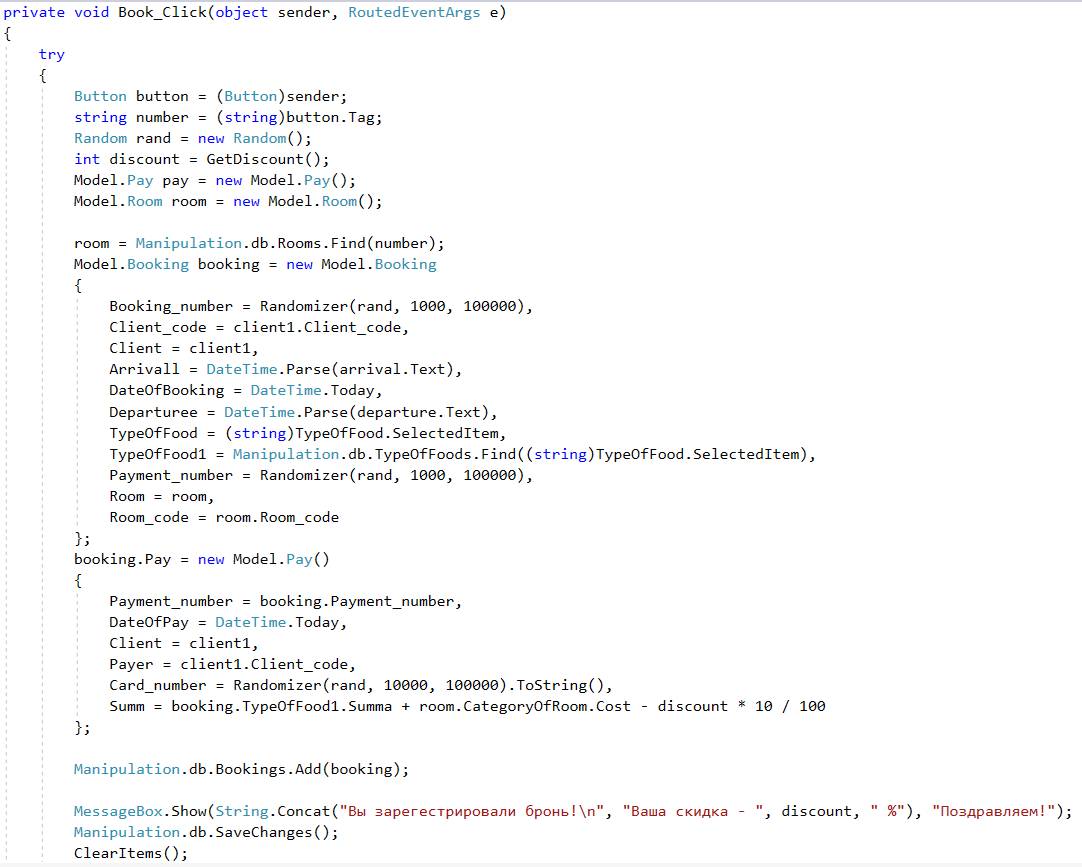


Рисунок 3.12 – Обработчик события кнопки, реализующий бронирование номера с предоставлением скидки

Дополнением к панели клиента является кнопка «Мои бронирования». При ее нажатии скрывается область поиска и результата, и появляется страница MyBooking (элемент Page).

Появление страницы реализовано через элемент Frame, у которого меняется свойство Visibility. Далее в поле ввода нужно ввести свою фамилию и нажать кнопку «Показать». Чуть ниже этих элементов появится таблица с бронями клиента, либо окошко с надписью «На текущий момент брони отсутствуют».

Обработчик события клика по кнопке представлен на рисунке 3.13.

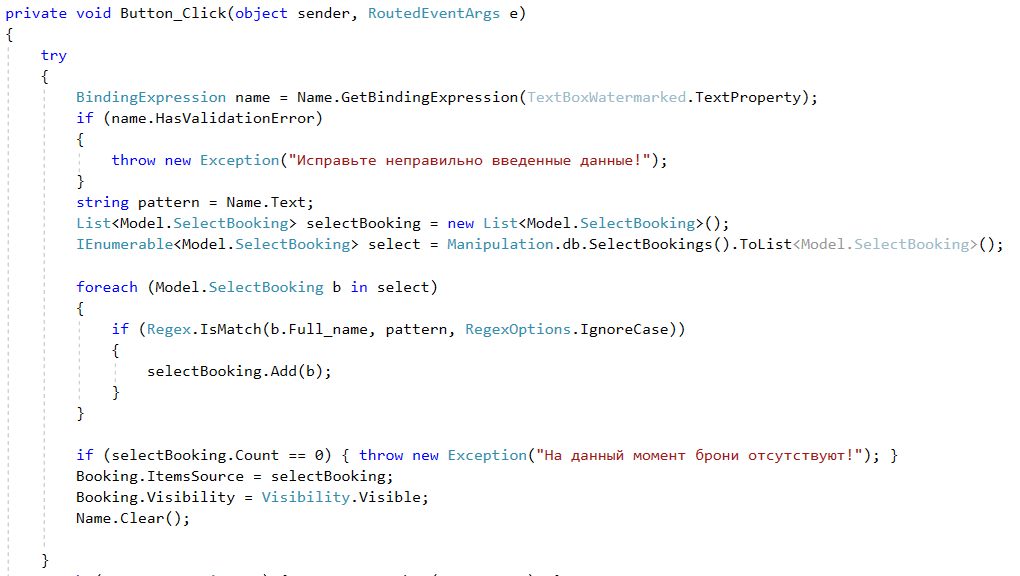


Рисунок 3.13 – Обработчик события кнопки, реализующий вывод броней клиента

На странице как панели Администратора, так и панели Клиента, есть кнопка «Выйти», которая дает возможность выхода из панели и возвращения на начальный экран авторизации. Ее обработчик приведен на рисунке 3.14.

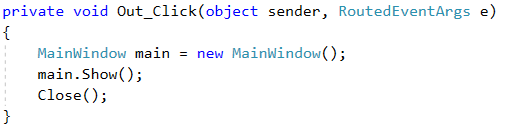


Рисунок 3.14 – Обработчик события кнопки, реализующий выход из аккаунта

# 4. Руководство по эксплуатации

При запуске приложения открывается окно авторизации плользователя (рисунок 4.1).

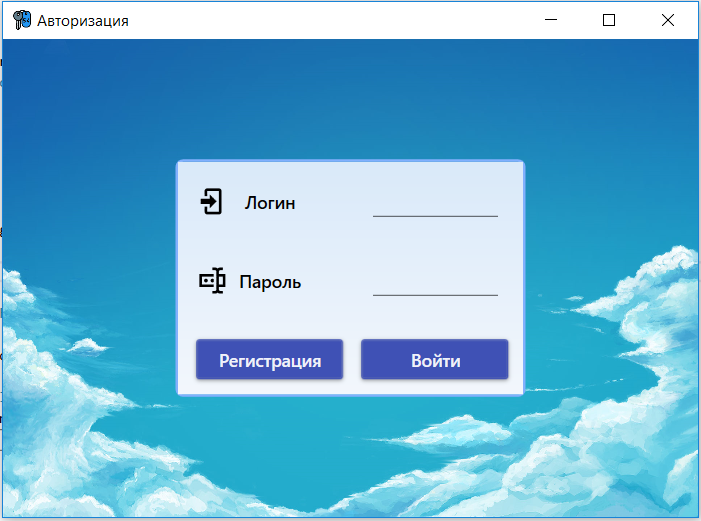


Рисунок 4.1 – Окно авторизации

При нажатии на кнопку «Регистрация» открывается окно регистрации, где пользователь должен зарегистрироваться (рисунок 4.2).

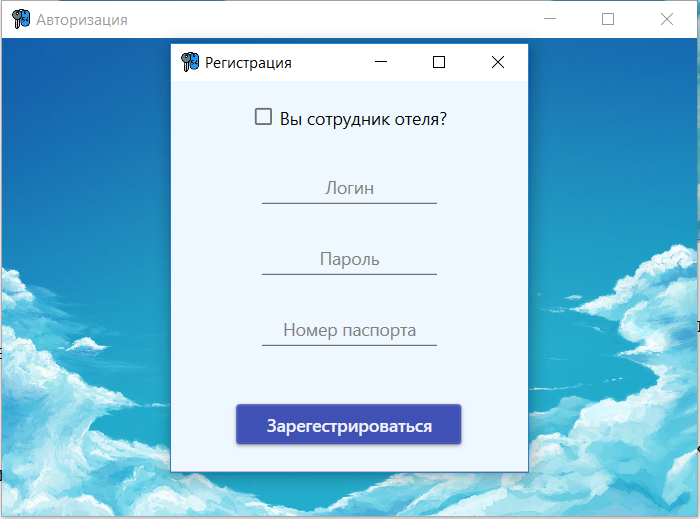


Рисунок 4.2 – Окно регистрации

После ввода логина и пароля Администратор нажимает на кнопку «Войти», и, если все правильно введено, открывается окно панели Администратора (рисунок 4.3).

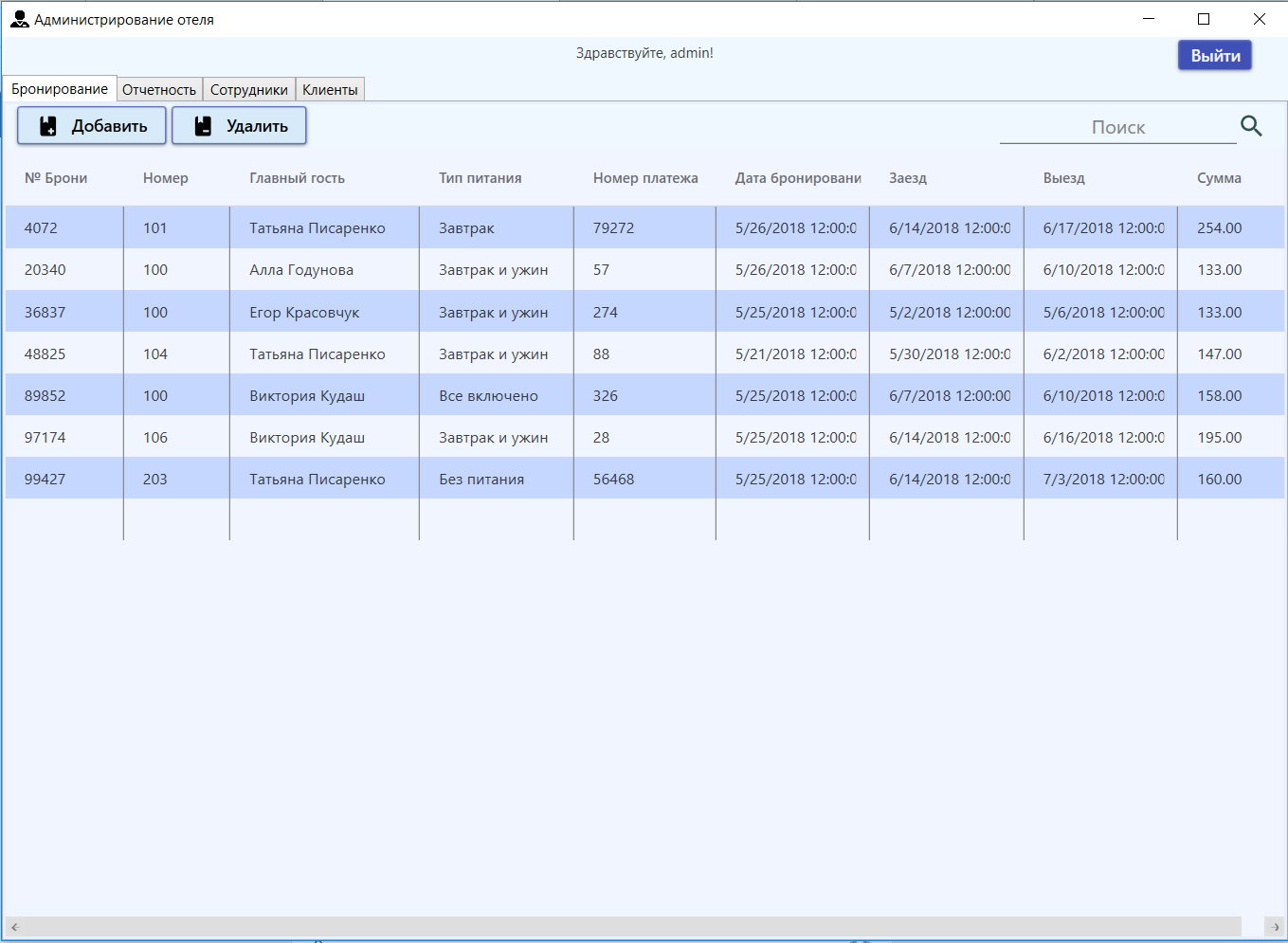


Рисунок 4.3 – Окно панели Администратора

В окне он может нажать на кнопку «Добавить». После появится окно для добавления брони (рисунок 4.4). Обязательно следует указать есть ли клиент в базу. Далее требуется заполнить поля данными и нажать кнопку «Добавить», после появится сообщение «Вы зарегистрировали бронь» и окно будет закрыто.

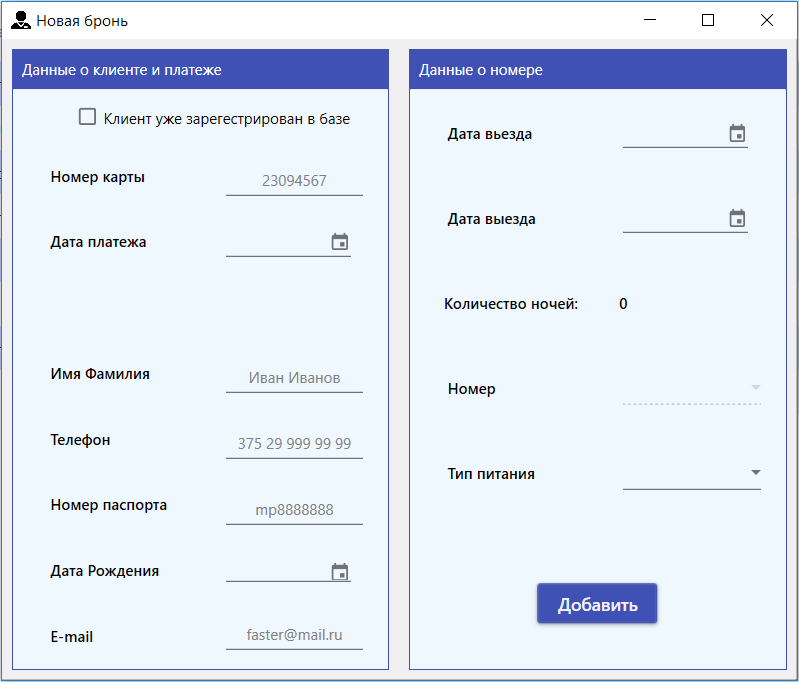


Рисунок 4.4 – Окно добавления брони

Далее администратор перейдет на вкладку «Отчетность» (рисунок 4.5), чтобы посмотреть свободные номера, к примеру, на завтрашний день. Он нажимает на кнопку «+» и вводит дату, далее жмет кнопку «Показать» и отображается отчет в виде таблицы (рисунок 4.6).

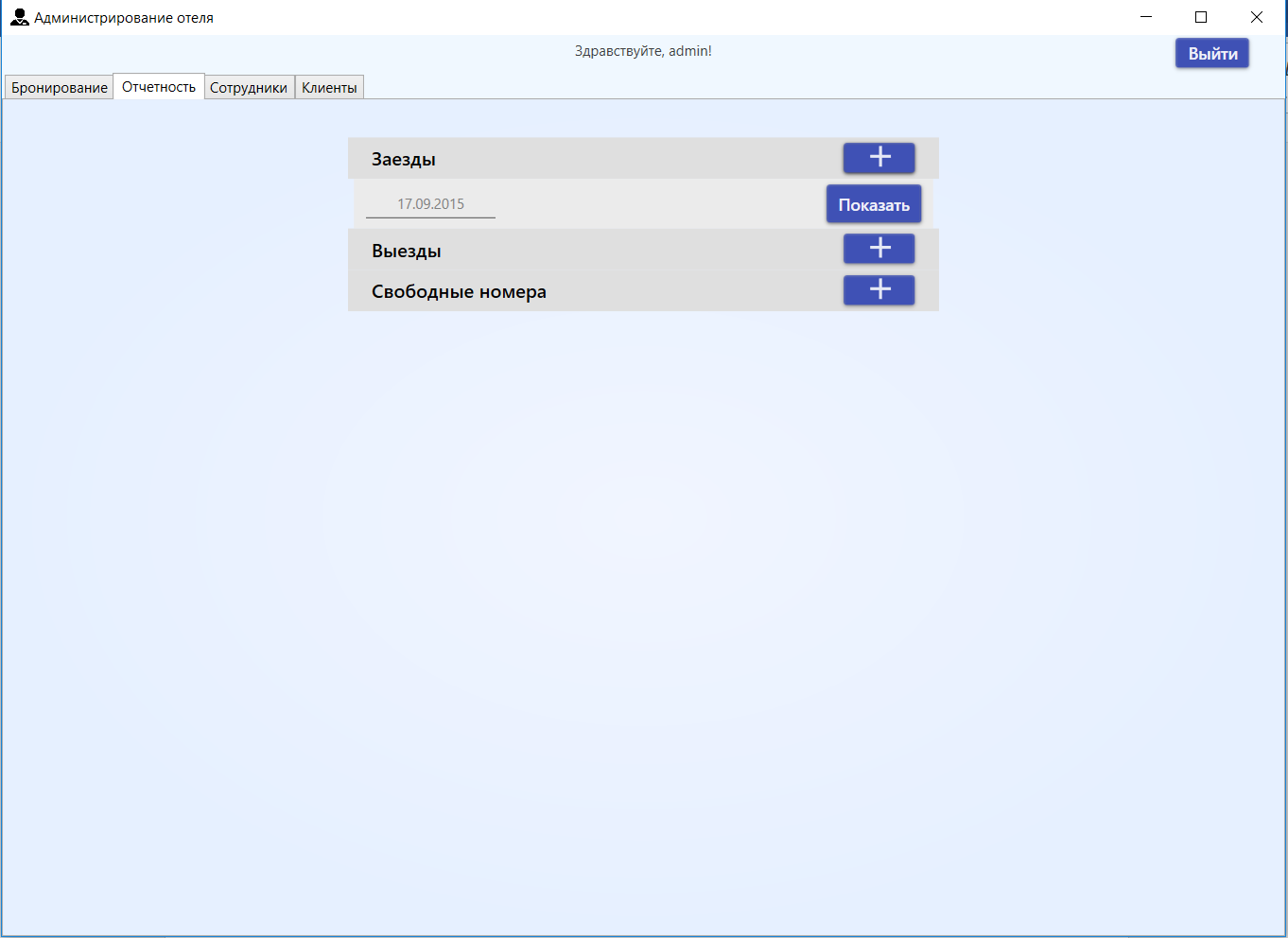


Рисунок 4.5 – Вкладка «Отчетность»

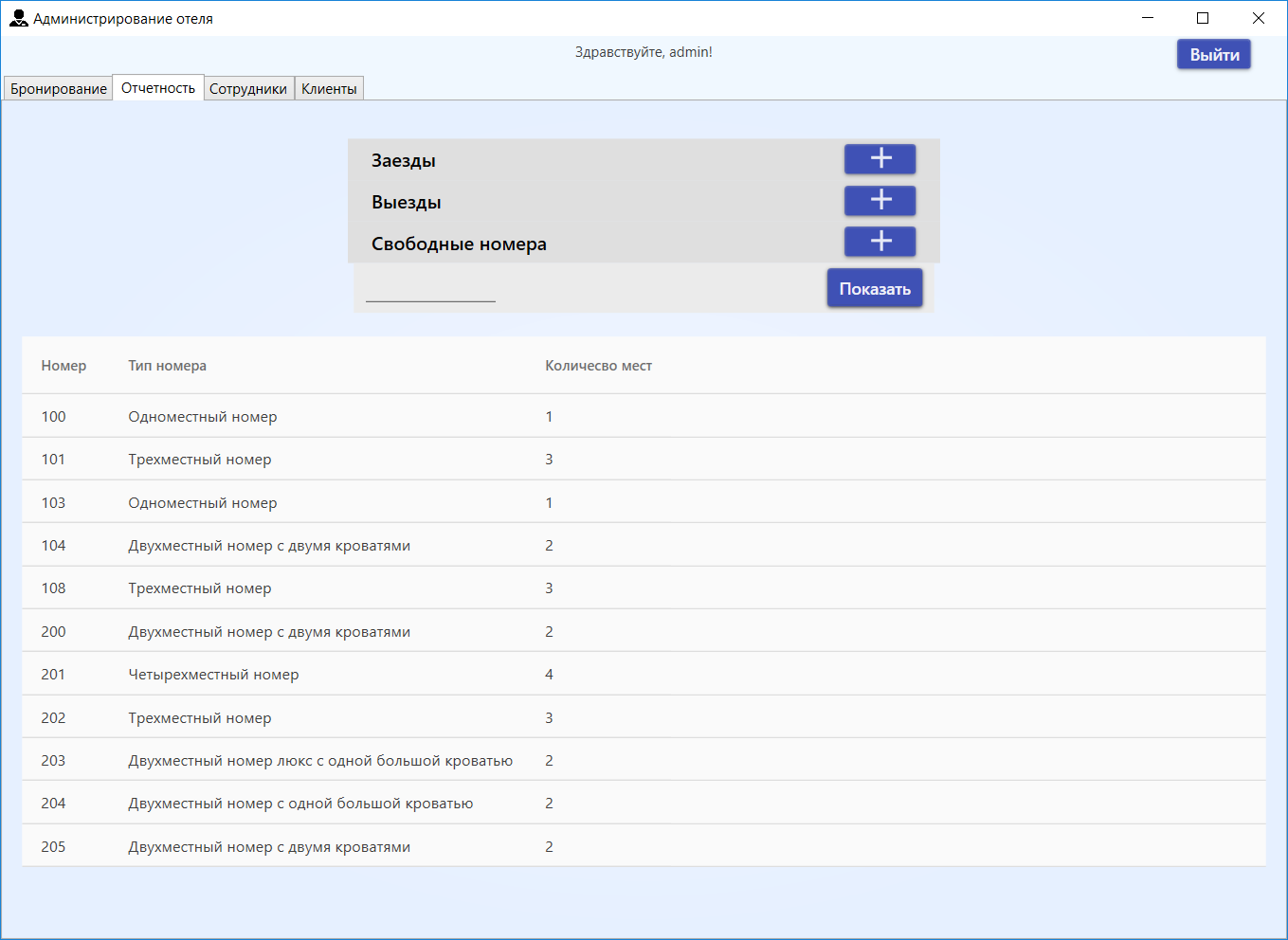


Рисунок 4.6 – Вкладка «Отчетность», результат поиска

На вкладках «Сотрудники» и «Клиенты» так же есть кнопка «Добавить», которые вызывают окна добавления (рисунок 4.7).

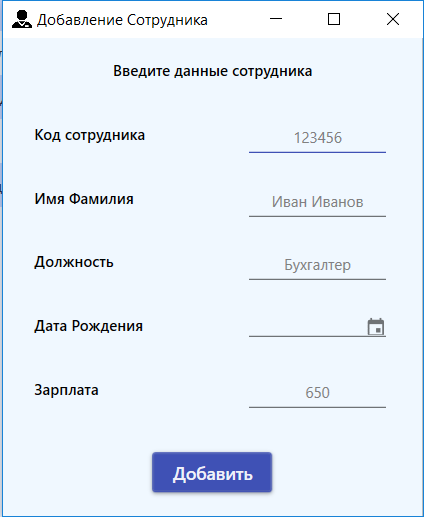
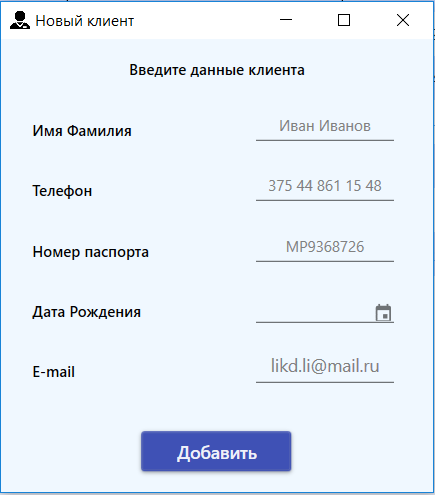
 

Рисунок 4.7 – Окна добавления сотрудника и клиента

Далее Администратор может воспользоваться поиском и найти нужную бронь по фамилии или имени клиента (рисунок 4.8). Для этого нужно в поле ввода начать вводить буквы.

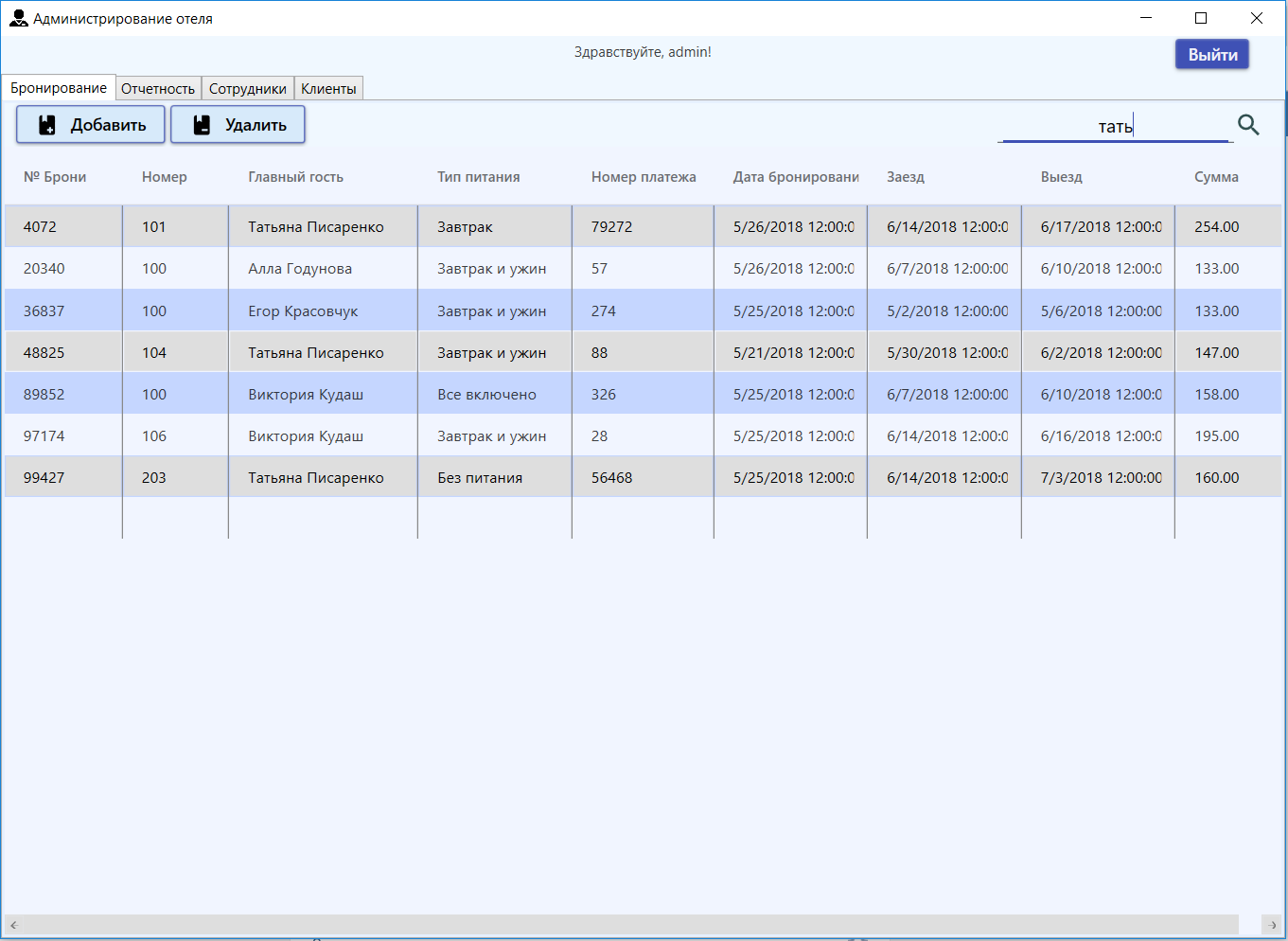


Рисунок 4.8 – Поиск

Для клиента после авторизации открывается панель Клиента (рисунок 4.9). В окне он может нажать на кнопку «Мои бронирования». После появится страница, в которой потребуется ввести свою фамилию (рисунок 4.10).

Чтобы вернуться к поиску номеров нужно нажать кнопку «Назад к бронированию». При этом закроется страница и появится изначальное окно как на рисунке 4.9.

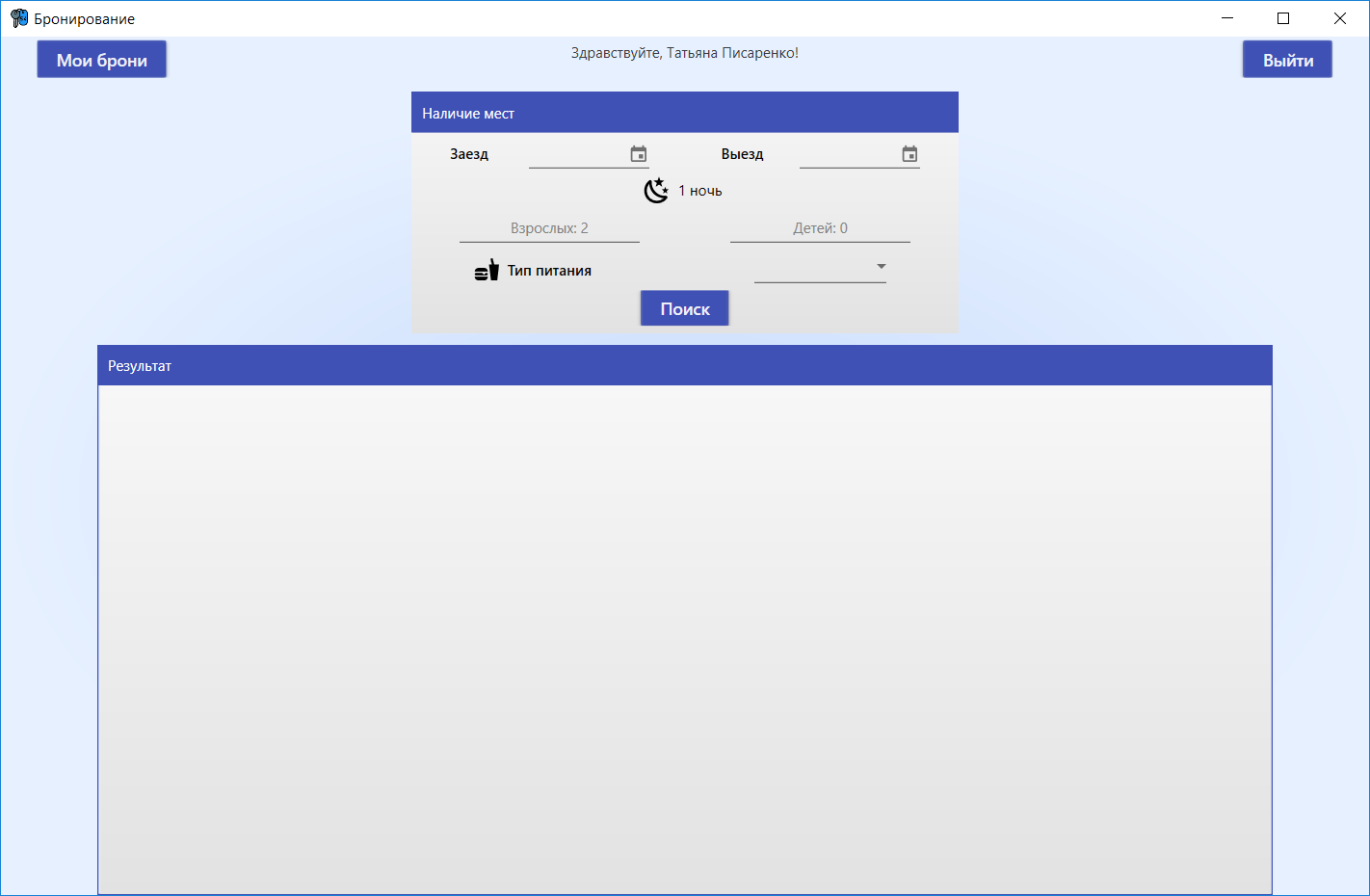


Рисунок 4.9 – Панель клиента

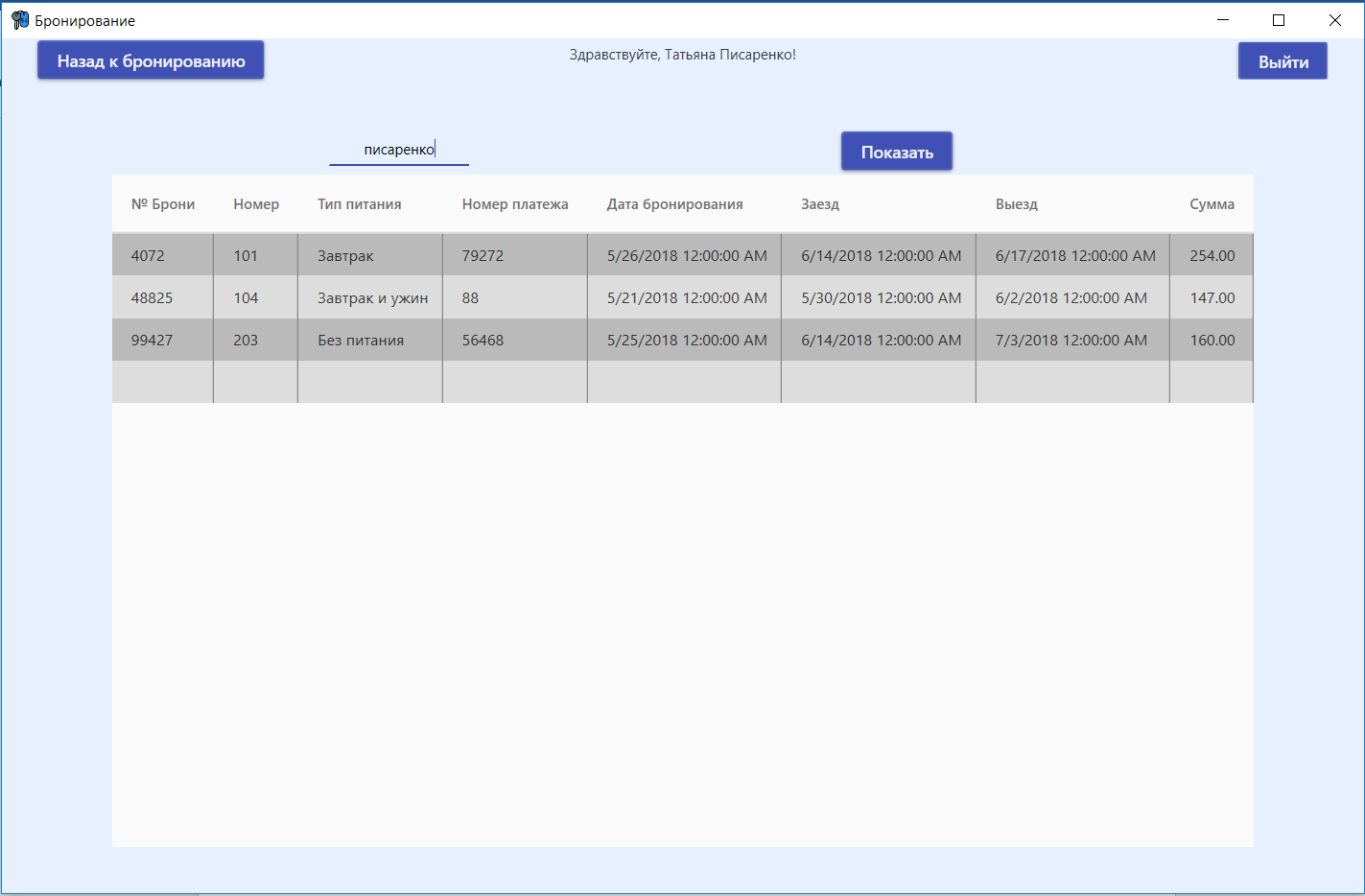


Рисунок 4.10 – Окно чека

Далее клиент должен ввести критерии поиска и нажать кнопку «Поиск» (рисунок 4.11), далее ему будет представлен список возможных номеров для бронирования на эту дату.

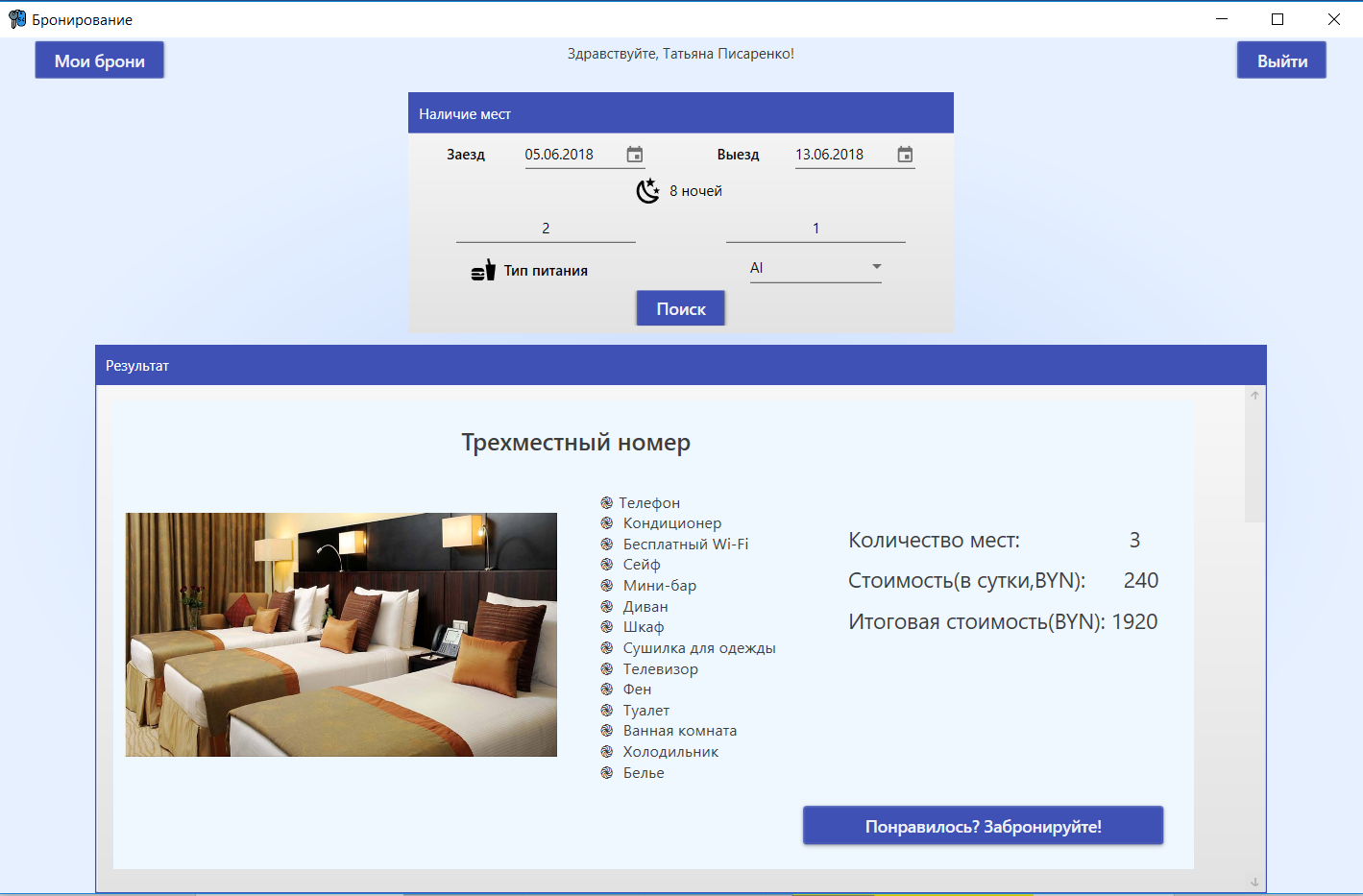


Рисунок 4.11 – Сформированная таблица с заказами

Для того, чтобы забронировать клиент должен нажать на кнопку «Понравилось? Забронируйте!». Далее покажется модальное окно с сообщением об успешной регистрации брони и текущей скидкой (рисунок 4.12).

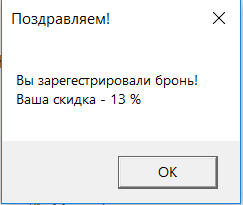


Рисунок 4.12 – Итоговое сообщение

# 5. Тестирование приложения

Программы пишут для того, чтобы они работали быстро и, самое главное, правильно. Самая большая неприятность при возникновении ошибочной ситуации состоит в том, что обычно это приводит к экстренному завершению программы. Во многих языках программирования, в том числе и в С#, предусмотрены механизмы, позволяющие «сохранить лицо» даже в достаточно сложных ситуациях. Поэтому в курсовом проекте были предусмотрены меры для обработки различных исключительных ситуаций:

Предусмотрена валидация поля для ввода логина и пароля. Результат представлен на рисунках 5.1-5.2

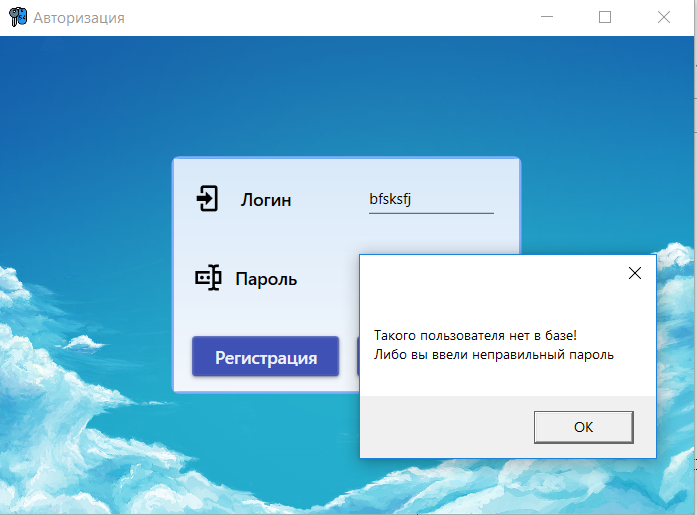
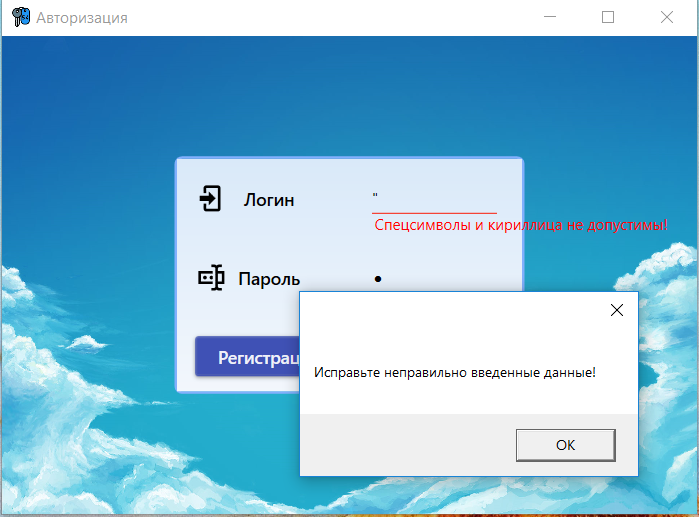


Рисунок 5.1-5.2 – Окно авторизации при вводе неверных данных

Предусмотрена валидация для добавления (рисунок 5.3).

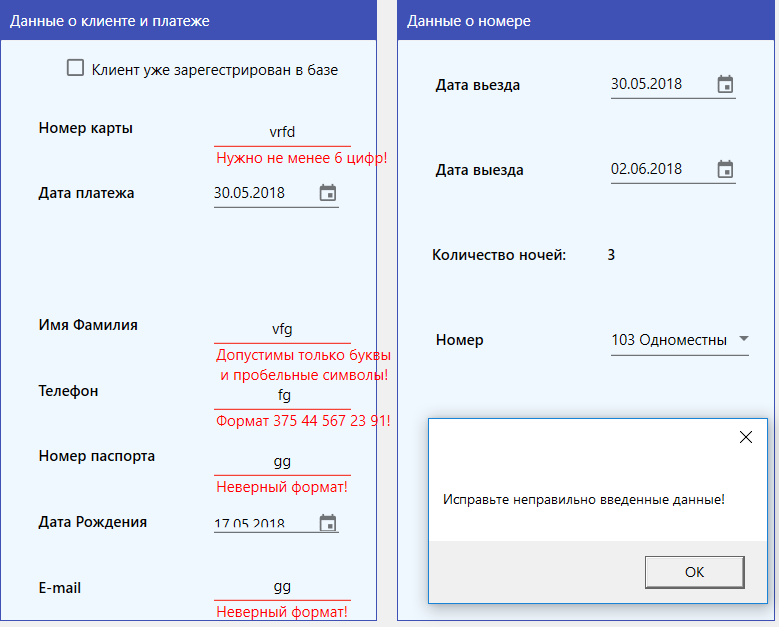


Рисунок 5.3 – Обработка исключений, связанных с вводом некорректной информации при добавлении

При введении неправильной даты для просмотра отчета будет также выдана ошибка (рисунке 5.4).

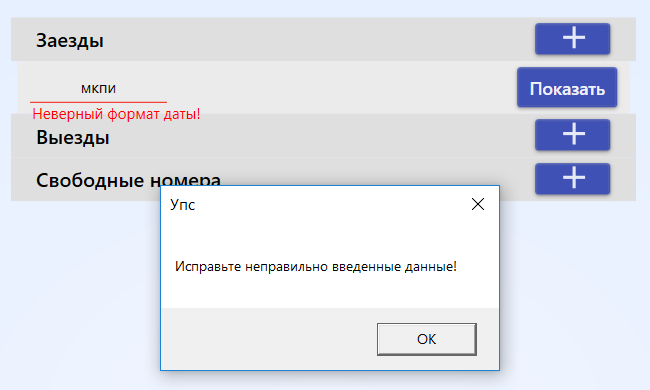


Рисунок 5.4 – Обработка исключений, связанных с вводом некорректной даты при просмотре отчета

При неверном введении даты при поиске клиенту выдаст ошибку (рисунок 5.5), если не был указан тип питания – выдаст ошибку (рисунок 5.6).

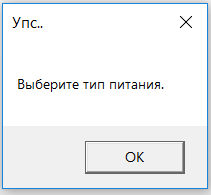
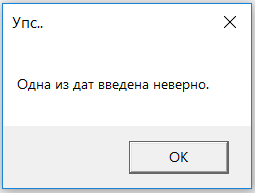


Рисунок 5.5-5.6 – Обработка исключений, связанных с вводом некорректной даты или типа питания

При неверном введении фамилии приложение выдаст ошибку на рисунке 5.7.

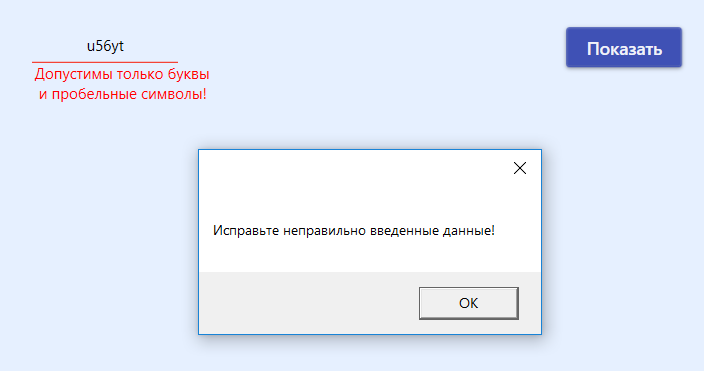


Рисунок 5.7 – Обработка исключений, связанных с вводом некорректной даты или типа питания

# Заключение

Итогом выполнения курсового проекта стало программное средство, представляющее собой приложение «Бронирование отеля», написанное на языке C#, с использованием технологии WPF, Entity Framework 6, Material Design Theme. Данное приложение выполняет все функции, заявленные в задании. Программное средство имеет простой и понятный интерфейс, а также содержит все валидацию, для предотвращения ошибок ввода со строны пользователя.

В результате выполнения проекта были закреплены теоретические и практические базовые навыки по дисциплине «Современные технологии программирования мобильных систем».

В процессе выполнения первого раздела курсовой работы уделила особое внимание теоретической части и постановке задач, которые должен выполнять курсовой проект, была обоснована актуальность выбранной темы.

Во втором разделе курсового проекта были тщательно проанализированы современные технологии, позволяющие создавать качественные десктопные приложения, была выбрана архитектура проекта, созданы 4 вида UML-диаграмм. В третьем разделе описаны выбранная база данных и функции, предоставляемые пользователю приложением.

Раздел номер четыре представляет пример практического использования приложения и выполнение всех требуемых функций. В пятом разделе приведены наглядные примеры сделанной валидации данных.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает корректно, а требования технического задания выполнены в полном объеме.

# Список используемых источников

1. Рихтер Дж. Р55 CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. — СПб.: Питер, 2013. — 896 с.: ил. — (Серия «Мастер-класс»).
2. Пацей, Н.В. Курс лекций по языку программирования C# / Н.В. Пацей. – Минск: БГТУ, 2016. – 175 с.
3. Форум программистов .NET [Электронный ресурс] / Работа с Entity Framework 6 – Дата доступа по: 25.05.2018 – https://professorweb.ru/my/entity-framework/6/level1.
4. Material Design in XAML [Электронный ресурс] / Material Design in XAML Toolkit – Дата доступа по: 23.05.2018 – http://materialdesigninxaml.net/
5. Файловый архив студентов [Электронный ресурс] / Базы данных и системы управления базами данных – Дата доступа по: 24.05.2018 – https://studfiles.net/preview/4586879/.
6. Руководство по WPF [Электронный ресурс] / Руководство по WPF – Дата доступа по 15.05.2018 – <https://metanit.com/sharp/wpf/>.
7. Syl [Электронный ресурс] / UML-диаграмма. Виды диаграмм UML – Дата доступа по: 25.05.2018 – https://www.syl.ru/article/206012/new\_uml-diagramma-vidyi-diagramm-uml.
8. Общие сведения о языке XAML (WPF) [Электронный ресурс] / Обзор XAML (WPF) – Дата доступа по: 25.05.2018 – <https://msdn.microsoft.com/ru-ru/library/ms752059(v=vs.110).aspx>.

# ПРИЛОЖЕНИЕ А

Листинг окна Window\_add.

public partial class Admin : Window

{

Model.Autorization autorization;

public Admin(Model.Autorization autor)

{

try

{

autorization = new Model.Autorization();

autorization = autor;

InitializeComponent();

DataContext = new Model.SelectBooking();

}

catch(Exception ex) { MessageBox.Show(ex.Message); }

}

private void Window\_Loaded(object sender, RoutedEventArgs e)

{

LoginAdmin.Content = String.Concat("Здравствуйте, ", autorization.Login, "!");

}

private void Window\_Activated(object sender, EventArgs e)

{

try

{

UpdateDataGrid();

}

catch (Exception exp) { MessageBox.Show(exp.Message, "Упс"); }

}

private void AddClient\_Click(object sender, EventArgs e)

{

try

{

Client1 client1 = new Client1

{

Owner = this

};

client1.ShowDialog();

UpdateDataGrid();

}

catch (Exception ex) { MessageBox.Show(ex.Message); }

}

private void AddWorker\_Click(object sender, EventArgs e)

{

try

{

Workerr workerr = new Workerr

{

Owner = this

};

workerr.ShowDialog();

UpdateDataGrid();

}

catch (Exception ex) { MessageBox.Show(ex.Message); }

}

private void AddBooking\_Click(object sender, EventArgs e)

{

Pages.Booking1 booking1 = new Pages.Booking1

{

Owner = this

};

booking1.ShowDialog();

UpdateDataGrid();

}

private void DeleteBooking\_Click(object sender, EventArgs e)

{

try

{

Model.Booking book = new Model.Booking();

Model.SelectBooking booking = (Model.SelectBooking)Booking.SelectedItem;

book = Manipulation.db.Bookings.Find(booking.Booking\_number);

Manipulation.Delete<Model.Booking>(book);

UpdateDataGrid();

MessageBox.Show("Запись удалена!");

}

catch (NullReferenceException ex) { MessageBox.Show("Выберите бронь, чтобы удалить ее!"); }

catch (Exception ex) { MessageBox.Show(ex.Message); }

}

private void DeleteWorker\_Click(object sender, EventArgs e)

{

try

{

Model.Worker worker = new Model.Worker();

worker = (Model.Worker)Workers.SelectedItem;

Manipulation.Delete<Model.Worker>(worker);

UpdateDataGrid();

MessageBox.Show("Запись удалена!");

}

catch (ArgumentNullException ex) { MessageBox.Show("Выберите работника, чтобы удалить его!"); }

catch (Exception ex) { MessageBox.Show(ex.Message); }

}

private void DeleteClient\_Click(object sender, EventArgs e)

{

try

{

Model.Client client = new Model.Client();

client = (Model.Client)Clients.SelectedItem;

Manipulation.Delete<Model.Client>(client);

UpdateDataGrid();

MessageBox.Show("Запись удалена!");

}

catch (ArgumentNullException ex) { MessageBox.Show("Выберите клиента, чтобы удалить его!"); }

catch (Exception ex) { MessageBox.Show(ex.Message); }

}

private void PlusButton\_Click(object sender, EventArgs e)

{

try

{

Collapsed();

ArrivalP.Visibility = Visibility.Visible;

}

catch (Exception ex) { MessageBox.Show(ex.Message); }

}

private void PlusButtonD\_Click(object sender, EventArgs e)

{

try

{

Collapsed();

Departure.Visibility = Visibility.Visible;

}

catch (Exception ex) { MessageBox.Show(ex.Message); }

}

private void PlusButtonF\_Click(object sender, EventArgs e)

{

try

{

Collapsed();

FreeRooms.Visibility = Visibility.Visible;

}

catch (Exception ex) { MessageBox.Show(ex.Message); }

}

private void Workers\_CellEditEnding(object sender, DataGridCellEditEndingEventArgs e)

{

try

{

Manipulation.db.SaveChanges();

}

catch (Exception exp) { MessageBox.Show(exp.Message, "Упс"); }

}

private void Booking\_CellEditEnding(object sender, DataGridCellEditEndingEventArgs e)

{

try

{

Manipulation.db.SaveChanges();

}

catch (Exception exp) { MessageBox.Show(exp.Message, "Упс"); }

}

private void Clients\_CellEditEnding(object sender, DataGridCellEditEndingEventArgs e)

{

try

{

Manipulation.db.SaveChanges();

}

catch (Exception exp) { MessageBox.Show(exp.Message, "Упс"); }

}

private void Arrivals\_Click(object sender, RoutedEventArgs e)

{

try

{

BindingExpression p = ArrivalBox.GetBindingExpression(TextBoxWatermarked.TextProperty);

if (p.HasValidationError) { throw new Exception("Исправьте неправильно введенные данные!"); }

List<Model.SelectBooking> selectBooking = new List<Model.SelectBooking>();

IEnumerable<Model.SelectBooking> select = Manipulation.db.SelectArrival(DateTime.Parse(ArrivalBox.Text)).ToList<Model.SelectBooking>();

foreach (Model.SelectBooking b in select)

{

selectBooking.Add(b);

}

See.Columns.Clear();

See.Items.Clear();

ArrivalBox.Clear();

if (selectBooking.Count == 0) { throw new Exception("На текущую дату заездов нет."); }

See.Visibility = Visibility.Visible;

GenerateTable();

foreach (Model.SelectBooking b in select)

{

See.Items.Add(b);

}

}

catch(FormatException ex) { MessageBox.Show("Введите правильную дату!", "Упс"); }

catch (NullReferenceException ex) { MessageBox.Show(ex.Message, "Упс"); }

catch (Exception exp) { MessageBox.Show(exp.Message, "Упс"); }

}

private void Departure\_Click(object sender, RoutedEventArgs e)

{

try

{

BindingExpression p = DepartureBox.GetBindingExpression(TextBoxWatermarked.TextProperty);

if (p.HasValidationError) { throw new Exception("Исправьте неправильно введенные данные!"); }

List<Model.SelectBooking> selectBooking = new List<Model.SelectBooking>();

IEnumerable<Model.SelectBooking> select = Manipulation.db.SelectDeparture(DateTime.Parse(DepartureBox.Text)).ToList<Model.SelectBooking>();

foreach (Model.SelectBooking b in select)

{

selectBooking.Add(b);

}

See.Columns.Clear();

See.Items.Clear();

DepartureBox.Clear();

if (selectBooking.Count == 0) { throw new Exception("На текущую дату заездов нет."); }

See.Visibility = Visibility.Visible;

GenerateTable();

foreach (Model.SelectBooking b in select)

{

See.Items.Add(b);

}

}

catch (NullReferenceException ex) { MessageBox.Show(ex.Message, "Упс"); }

catch (FormatException ex) { MessageBox.Show("Введите правильную дату!", "Упс"); }

catch (Exception exp) { MessageBox.Show(exp.Message, "Упс"); }

}

private void GetFreeRooms\_Click(object sender, RoutedEventArgs e)

{

try

{

BindingExpression p = FreeC.GetBindingExpression(TextBoxWatermarked.TextProperty);

if (p.HasValidationError) { throw new Exception("Исправьте неправильно введенные данные!"); }

List<Model.FreeRooms> selectRooms = new List<Model.FreeRooms>();

IEnumerable<Model.FreeRooms> select = Manipulation.db.FreeRooms(DateTime.Parse(FreeC.Text)).ToList<Model.FreeRooms>();

foreach (Model.FreeRooms b in select)

{

selectRooms.Add(b);

}

See.Columns.Clear();

See.Items.Clear();

FreeC.Clear();

if (selectRooms.Count == 0) { throw new Exception("На текущую дату свободных комнат нет."); }

See.Visibility = Visibility.Visible;

#region a

DataGridTextColumn t1 = new DataGridTextColumn();

t1.Header = "Номер ";

t1.Binding = new Binding("Room\_code");

See.Columns.Add(t1);

DataGridTextColumn t2 = new DataGridTextColumn();

t2.Header = "Тип номера";

t2.Binding = new Binding("Category\_name");

See.Columns.Add(t2);

DataGridTextColumn t3 = new DataGridTextColumn();

t3.Header = "Количесво мест";

t3.Binding = new Binding("NumberOfSeats");

See.Columns.Add(t3);

#endregion

foreach (Model.FreeRooms b in select)

{

See.Items.Add(b);

}

}

catch (FormatException ex) { MessageBox.Show("Введите правильную дату!", "Упс"); }

catch (NullReferenceException ex) { MessageBox.Show(ex.Message, "Упс"); }

catch (Exception exp) { MessageBox.Show(exp.Message, "Упс"); }

}

private void Collapsed()

{

ArrivalP.Visibility = Visibility.Collapsed;

Departure.Visibility = Visibility.Collapsed;

FreeRooms.Visibility = Visibility.Collapsed;

}

public void UpdateDataGrid()

{

try

{

List<Model.SelectBooking> selectBooking = new List<Model.SelectBooking>();

IEnumerable<Model.SelectBooking> select = Manipulation.db.SelectBookings().ToList<Model.SelectBooking>();

foreach (Model.SelectBooking b in select)

{

selectBooking.Add(b);

}

Manipulation.db.SaveChanges();

Booking.ItemsSource = selectBooking;

Workers.ItemsSource = Manipulation.db.Workers.ToList<Model.Worker>();

Clients.ItemsSource = Manipulation.db.Clients.ToList<Model.Client>();

}

catch (Exception exp) { MessageBox.Show(exp.Message, "Упс"); }

}

public void GenerateTable()

{

try

{

DataGridTextColumn t1 = new DataGridTextColumn

{

Header = "№ Брони",

Binding = new Binding("Booking\_number")

};

See.Columns.Add(t1);

DataGridTextColumn t2 = new DataGridTextColumn

{

Header = "Номер",

Binding = new Binding("Room\_code")

};

See.Columns.Add(t2);

DataGridTextColumn t3 = new DataGridTextColumn

{

Header = "Гость ",

Binding = new Binding("Full\_name")

};

See.Columns.Add(t3);

DataGridTextColumn t4 = new DataGridTextColumn

{

Header = "Тип питания",

Binding = new Binding("Type\_name")

};

See.Columns.Add(t4);

DataGridTextColumn t5 = new DataGridTextColumn

{

Header = "№ Платежа",

Binding = new Binding("Payment\_number")

};

See.Columns.Add(t5);

DataGridTextColumn t6 = new DataGridTextColumn

{

Header = "Дата бронирования",

Binding = new Binding("DateOfBooking")

};

See.Columns.Add(t6);

DataGridTextColumn t7 = new DataGridTextColumn

{

Header = "Заезд ",

Binding = new Binding("Arrivall")

};

See.Columns.Add(t7);

DataGridTextColumn t8 = new DataGridTextColumn

{

Header = "Выезд ",

Binding = new Binding("Departuree")

};

See.Columns.Add(t8);

DataGridTextColumn t9 = new DataGridTextColumn

{

Header = "Сумма",

Binding = new Binding("Summ")

};

See.Columns.Add(t9);

}

catch (Exception exp) { MessageBox.Show(exp.Message + "/" +exp.InnerException); }

}

private void Out\_Click(object sender, RoutedEventArgs e)

{

MainWindow main = new MainWindow();

main.Show();

Close();

}

}

# ПРИЛОЖЕНИЕ B

Листинг окна NewClient.

public partial class NewClient : Window

{

public Model.Client client1;

public NewClient(Model.Client cl)

{

try

{

client1 = cl;

InitializeComponent();

DataContext = new Model.Client();

}

catch (Exception exp) { MessageBox.Show(exp.Message + "/" + exp.InnerException); }

}

private void Window\_Loaded(object sender, RoutedEventArgs e)

{

try

{

List<string> typesOfFood = new List<string>();

foreach (Model.TypeOfFood t in Manipulation.db.TypeOfFoods)

{

typesOfFood.Add(t.Type\_code);

}

TypeOfFood.ItemsSource = typesOfFood;

LoginCient.Content = String.Concat("Здравствуйте, ", client1.Full\_name, "!");

arrival.DisplayDateStart = DateTime.Today;

arrival.DisplayDateEnd = DateTime.Parse("31.12.2019");

departure.DisplayDateStart = DateTime.Today;

departure.DisplayDateEnd = DateTime.Parse("31.12.2019");

}

catch (Exception exp) { MessageBox.Show(exp.Message + "/" + exp.InnerException); }

}

private void Out\_Click(object sender, RoutedEventArgs e)

{

MainWindow main = new MainWindow();

main.Show();

Close();

}

private void Departure\_SelectedDateChanged(object sender, SelectionChangedEventArgs e)

{

try

{

if (arrival.SelectedDate != null && departure.SelectedDate != null)

{

int nig = Booking1.CountsOfNights(DateTime.Parse(arrival.Text), DateTime.Parse(departure.Text));

if (nig > 1)

{

CountOfNights.Content = String.Concat(nig, " ночей");

}

}

}

catch (Exception exp) { MessageBox.Show(exp.Message); }

}

private void Found\_Click(object sender, RoutedEventArgs e)

{

try

{

BindingExpression b = Adults.GetBindingExpression(TextBoxWatermarked.TextProperty);

BindingExpression f = Children.GetBindingExpression(TextBoxWatermarked.TextProperty);

if (b.HasValidationError || f.HasValidationError)

{

throw new Exception("Исправьте неправильно введенные данные!");

}

int people = Int32.Parse(Adults.Text) + Int32.Parse(Children.Text);

if (people > 4)

{

throw new Exception("Максимальное количество людей не должно превышать 4.");

}

else if(TypeOfFood.SelectedItem == null)

{

throw new Exception("Выберите тип питания.");

}

else if (DateTime.Parse(arrival.Text) == DateTime.Parse(departure.Text) || DateTime.Parse(arrival.Text) > DateTime.Parse(departure.Text))

{

throw new Exception("Одна из дат введена неверно.");

}

else

{

IEnumerable<Model.Search\_Result> search = Manipulation.db.Search(DateTime.Parse(arrival.Text), DateTime.Parse(departure.Text), people).ToList<Model.Search\_Result>();

List<Classes.ListItem> list = new List<Classes.ListItem>();

list = GetList(search);

Rooms.SelectionMode = SelectionMode.Single;

Rooms.ItemsSource = list;

}

}

catch (System.FormatException ex) { MessageBox.Show("Зaполните все поля!", "Упс.."); }

catch (System.Data.Entity.Validation.DbEntityValidationException ex) { MessageBox.Show("Зполните все данные!", "Упс.."); }

catch (Exception exp) { MessageBox.Show(exp.Message, "Упс.."); }

}

private void Book\_Click(object sender, RoutedEventArgs e)

{

try

{

Button button = (Button)sender;

string number = (string)button.Tag;

Random rand = new Random();

int discount = GetDiscount();

Model.Pay pay = new Model.Pay();

Model.Room room = new Model.Room();

room = Manipulation.db.Rooms.Find(number);

Model.Booking booking = new Model.Booking

{

Booking\_number = Randomizer(rand, 1000, 100000),

Client\_code = client1.Client\_code,

Client = client1,

Arrivall = DateTime.Parse(arrival.Text),

DateOfBooking = DateTime.Today,

Departuree = DateTime.Parse(departure.Text),

TypeOfFood = (string)TypeOfFood.SelectedItem,

TypeOfFood1 = Manipulation.db.TypeOfFoods.Find((string)TypeOfFood.SelectedItem),

Payment\_number = Randomizer(rand, 1000, 100000),

Room = room,

Room\_code = room.Room\_code

};

booking.Pay = new Model.Pay()

{

Payment\_number = booking.Payment\_number,

DateOfPay = DateTime.Today,

Client = client1,

Payer = client1.Client\_code,

Card\_number = Randomizer(rand, 10000, 100000).ToString(),

Summ = booking.TypeOfFood1.Summa + room.CategoryOfRoom.Cost - discount \* 10 / 100

};

Manipulation.db.Bookings.Add(booking);

MessageBox.Show(String.Concat("Вы зарегестрировали бронь!\n", "Ваша скидка - ", discount, " %"), "Поздравляем!");

Manipulation.db.SaveChanges();

ClearItems();

}

catch (System.Data.Entity.Validation.DbEntityValidationException ex) { MessageBox.Show("Зполните все данные!"); }

catch (Exception exp) { MessageBox.Show(exp.Message, "Упс.."); }

}

private void MyBooking\_Click(object sender, RoutedEventArgs e)

{

High2.Visibility = Visibility.Collapsed;

MyBooking.Visibility = Visibility.Visible;

ShowBooking.Visibility = Visibility.Collapsed;

Out.Visibility = Visibility.Visible;

}

private void Outer\_Click(object sender, RoutedEventArgs e)

{

High2.Visibility = Visibility.Visible;

MyBooking.Visibility = Visibility.Collapsed;

ShowBooking.Visibility = Visibility.Visible;

Out.Visibility = Visibility.Collapsed;

}

private List<Classes.ListItem> GetList(IEnumerable<Model.Search\_Result> search)

{

string str = (string)CountOfNights.Content;

string[] split = str.Split(' ');

List<Classes.ListItem> list = new List<Classes.ListItem>();

foreach (Model.Search\_Result s in search)

{

Classes.ListItem listItem = new Classes.ListItem();

if (s.Category\_name.Contains("Одноместный номер"))

{

listItem.Path = "../Image/1чел1кров.jpg";

listItem.Room\_code = s.Room\_code;

listItem.Category\_name = s.Category\_name;

listItem.Cost = s.Cost;

listItem.NumberOfSeats = s.NumberOfSeats;

listItem.Servicess = GetServices(s.Servicess);

listItem.AndCost = s.Cost \* Int32.Parse(split[0]);

}

else if (s.Category\_name.Contains("Двухместный номер с двумя кроватями"))

{

listItem.Path = "../Image/2чел2кров.jpg";

listItem.Category\_name = s.Category\_name;

listItem.Room\_code = s.Room\_code;

listItem.Cost = s.Cost;

listItem.NumberOfSeats = s.NumberOfSeats;

listItem.Servicess = GetServices(s.Servicess);

listItem.AndCost = s.Cost \* Int32.Parse(split[0]);

}

else if (s.Category\_name.Contains("Двухместный номер с одной большой кроватью"))

{

listItem.Path = "../Image/21кровать.jpg";

listItem.Category\_name = s.Category\_name;

listItem.Room\_code = s.Room\_code;

listItem.Cost = s.Cost;

listItem.NumberOfSeats = s.NumberOfSeats;

listItem.Servicess = GetServices(s.Servicess);

listItem.AndCost = s.Cost \* Int32.Parse(split[0]);

}

else if (s.Category\_name.Contains("Двухместный номер люкс с одной большой кроватью"))

{

listItem.Path = "../Image/люкс2чел1кр.jpg";

listItem.Category\_name = s.Category\_name;

listItem.Room\_code = s.Room\_code;

listItem.Cost = s.Cost;

listItem.NumberOfSeats = s.NumberOfSeats;

listItem.Servicess = GetServices(s.Servicess);

listItem.AndCost = s.Cost \* Int32.Parse(split[0]);

}

else if (s.Category\_name.Contains("Трехместный номер"))

{

listItem.Path = "../Image/3чел3кро.jpg";

listItem.Category\_name = s.Category\_name;

listItem.Room\_code = s.Room\_code;

listItem.Cost = s.Cost;

listItem.NumberOfSeats = s.NumberOfSeats;

listItem.Servicess = GetServices(s.Servicess);

listItem.AndCost = s.Cost \* Int32.Parse(split[0]);

}

else if (s.Category\_name.Contains("Четырехместный номер"))

{

listItem.Path = "../Image/4чел3кроват.jpg";

listItem.Category\_name = s.Category\_name;

listItem.Room\_code = s.Room\_code;

listItem.Cost = s.Cost;

listItem.NumberOfSeats = s.NumberOfSeats;

listItem.Servicess = GetServices(s.Servicess);

listItem.AndCost = s.Cost \* Int32.Parse(split[0]);

}

list.Add(listItem);

}

return list;

}

public int Randomizer(Random rand, int min, int max)

{

int number = rand.Next(min, max);

return number;

}

public string GetServices(string str)

{

string And = "";

string[] split = str.Split(',');

foreach (string s in split)

{

And += String.Concat("\u058E " + s + "\n");

}

return And;

}

public void ClearItems()

{

Rooms.ItemsSource = null;

}

private int GetDiscount()

{

int Discount;

string pattern = client1.Full\_name;

List<Model.SelectBooking> selectBooking = new List<Model.SelectBooking>();

IEnumerable<Model.SelectBooking> select = Manipulation.db.SelectBookings().ToList<Model.SelectBooking>();

if (select.Count() > 0)

{

int caf = select.Count();

if (caf > 2 && caf < 5)

{

caf += 3;

}

else if(caf > 5 && caf < 10)

{

caf += 5;

}

return Discount = caf \* 110 / 100;

}

else

{

return 0;

}

}

}