СОДЕРЖАНИЕ

Введение 4

1 Технический проект 5

1.1 Анализ предметной области 5

1.2 Постановка задачи 5

1.3 Требования к программному обеспечению и техническим средствам 6

1.4 Проектирование функциональности и структуры программы 7

1.5 Разработка концептуальной модели базы данных 9

1.6 Выбор средств разработки 11

2 Рабочий проект 13

2.1 Проектирование физической модели базы данных 13

2.2 Функциональное взаимодействие модулей программы 15

2.3 Разработка unit-тестов 21

2.4 Входные и выходные данные 22

2.5 Работа с программой 22

Заключение 30

Список использованных источников 31

Приложение А – код приложения 32

Приложение B – код unit-тестов 123

# ВВЕДЕНИЕ

Темой курсового проекта является разработка и создания приложения по теме «АРМ администратора отеля».

Целью данного курсового проекта будет являться написание приложения, позволяющего автоматизировать процесс записи клиентов в отеле.

Написание данного курсового проекта является целесообразным, потому что:

* АРМ позволяет эффективно обрабатывать большие потоки информации;
* АРМ сокращает время работы и облегчает труд работникам.

Для достижения поставленной цели будут рассмотрены и решены следующие задачи:

* разработана структура и функциональность программы;
* разработаны справочные и тестовые материалы;
* произведён выбор средств реализации;
* разработан программный код приложения;

Объектом исследования будут являться бизнес процессы деятельности администратора отеля.

Предметом исследования будет являться оказание услуг отелем.

Основными пользователями данной программы будут администраторы отелей.

Разрабатываемое АРМ администратора отеля будет вести учет специфической информации касающейся оказываемых отелем услуг, а также хранить необходимые данные.

Теоретическими основами и методами решения поставленных задач будут методологии проектирования функциональности приложения, структуры базы данных (БД), особенности реализации программного кода с помощью конкретного языка программирования.

# 1 ТЕХНИЧЕСКИЙ ПРОЕКТ

## 1.1 Анализ предметной области

Ежегодно люди ездят за границу на отдых и им приходиться выбирать среди множества мест для ночлега подходящее им по цене и предоставляемым услугам, будь то гостиница, отель или палатка.

Разрабатываемая база данных будет реализовывать функцию хранения данных небольшой гостиницы. Работа с клиентом будет реализована следующим образом: клиент передает данные администратору для регистрации его в отеле. После регистрации клиент выбирает номер и желаемую дату заезда. Так же клиент может заказать необходимые ему услуги и включить их в стоимость проживания.

## 1.2 Постановка задачи

Необходимо разработать программу по теме «АРМ администратора отеля», а также создать документационное сопровождение программы, включающее в себе технический и рабочие проекты.

Материал по темам должен быть представлен в файлах, поддерживающих текст и изображения.

Данная программа должна будет содержать необходимый функционал для записи в отеле, иметь удобный и понятный пользователю графический интерфейс.

Программа должна обеспечивать выполнение следующих функций:

* отображение главного окна;
* отображение информации о клиентах;
* регистрации клиента и работника в системе;
* отображение информации о номерах;
* оповещение работника о проделанных ошибках;
* проверку корректности данных;
* хранение данных в БД.

Входными данными в приложении будут данные о пользователях, данные об отеле и предоставляемых им услугах, данные о регистрации, которые должны храниться в БД.

Выходными данными в приложении будет информация о регистрации клиента и информация об изменениях в базе данных.

Приложение будет состоять из окон, на которых будет располагаться вся необходимая пользователю информация.

Способ решения поставленных задач зависит от выбора средств разработки.

Тестирование программы будет проводится путем проверки всех модулей программы на корректность их выходных данных и выполняемых ими операций.

Блок тестирования будет проходить путем юнит-тестирования (Модульного тестирования), позволяющего проверить на корректность отдельные модули исходного кода программы.

## 1.3 Требования к программному обеспечению и техническим средствам

Для корректной работы разрабатываемой программы необходимо, чтобы компьютер удовлетворял следующим минимальным требованиям:

* процессор Pentium J3710;
* объем свободной оперативной памяти не менее 512 Мб;
* видеоадаптер Intel HD Graphics 405;
* 200 Мб свободного места на жестком диске;
* стандартная клавиатура с 101/109 клавишами;
* манипулятор типа «мышь».

Для нормального функционирования системы на компьютере должна быть установлена операционная система Windows 7/8/8.1/10, СУБД Microsoft SQL Server.

## 1.4 Проектирование функциональности и структуры программы

Для описания основных функций проектируемой программы воспользуемся use case диаграммой(рисунок 1).

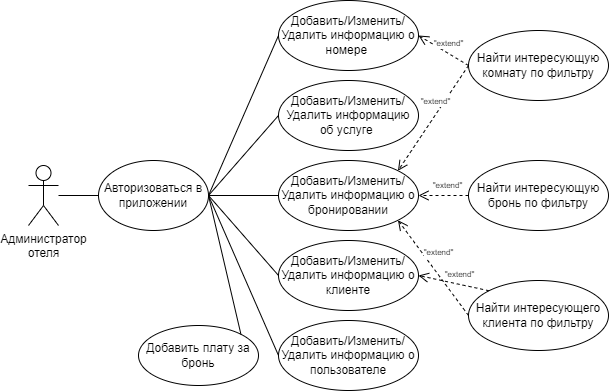


Рисунок 1 – Контекстная use case-диаграмма

С целью удостоверения предъявляемых к программе требований, разрабатываемое приложение будет состоять из следующих окон:

* главное – будет содержать навигационные кнопки, для перехода на другие окна;
* «Номера» – в нем пользователь будет добавлять/изменять/удалять информацию о номерах;
* «Услуги» - в нем пользователь будет добавлять/изменять/удалять информацию об услугах;
* «Бронирование» - в нем пользователь будет добавлять/изменять/удалять информацию о бронировании;
* «Клиенты» - в нем пользователь будет добавлять/изменять/удалять информацию о клиентах;
* «Пользователи» - в нем пользователь будет добавлять информацию о пользователях;
* «Оплата брони» - в нем пользователь сможет добавить плату за бронь;
* «Поиск» - в нем пользователь сможет найти нужную информацию по фильтрам;
* справки – будет выводить справочный материал для пользователя.

Все данные будут хранитсяв БД. Проектируемая структура приложения представлена на рисунке 2.



Рисунок 2 – Схема проектируемой структуры приложения

Тестирование программы планируется проводить путём unit тестов. Это тип тестирования программного обеспечения, при котором тестируются отдельные модули или компоненты программного обеспечения. Цель этого тестирования состоит в том, чтобы убедиться, что каждая единица программного кода работает должным образом.

## 1.5 Разработка концептуальной модели базы данных

Для хранения данных клиентов будет разработана БД. Она будет состоять из следующих таблиц:

Таблица 1 - “User”:

* Id – служебное поле для обеспечения уникальности записи;
* Name –ФИО пользователя;
* Login –логин пользователя;
* Password –пароль от аккаунта пользователя;
* PassportSeries –серия паспорта;
* PassportNumber –номер паспорта;
* Photo –фото пользователя;
* TelephoneNumber –телефонный номер;
* RoleId –ссылка на объект из таблицы “UserRole”. Определяет роль пользователя.

Таблица 2 – “UserRole”:

* Id – служебное поле для обеспечения уникальности записи;
* Value – описание роли.

Таблица 3 – “Room”:

* Id – служебное поле для обеспечения уникальности записи;
* Number –номер комнаты;
* Price –цена номера за день проживания;
* CountOfBeds –количество кроватей;
* Floor – этаж на котором находится комната;
* Description – описание комнаты;
* TypeId – ссылка на объект из таблицы “RoomType”. Определяет тип комнаты.

Таблица 4 – “RoomType”:

* Id – служебное поле для обеспечения уникальности записи;
* Value – значение типа;
* Price –описание типа.

Таблица 5 – “Service”:

* Id – служебное поле для обеспечения уникальности записи;
* NameOf – наименование услуги;
* Description–описание услуги;
* Price – цена за день использования услуги;
* IsUserBuyService – определяет приобрел ли пользователь данную услугу. Содержит булевое значение.

Таблица 6 – “Reservation”:

* Id – служебное поле для обеспечения уникальности записи;
* FromDate – дата заселения;
* ToDate–дата выселения;
* TotalPrice – итоговая цена;
* UserId– ссылка на объект из таблицы “User”. Определяет пользователя, оформившего бронь;
* RoomId– ссылка на объект из таблицы “Room”. Определяет комнату для бронирования.

Таблица 7 – “ReservationHasServices”:

* ReservationId – ссылка на объект из таблицы “Reservation”. Определяет бронь для которой добавляется услуга;
* ServiceId – ссылка на объект из таблицы “Service”. Определяет услугу, добавляемую к брони.

Таблица 8 – “TotalPaid”:

* Id – служебное поле для обеспечения уникальности записи;
* Date– дата оплаты.
* Price – количество денежных средств;
* ReservationId – ссылка на объект из таблицы “Reservation”. Определяет бронь, для которой происходит оплата;

Концептуальная модель БД представлена на рисунке 6.



Рисунок 3 – Концептуальная модель БД

## 1.6 Выбор средств разработки

Из множества различных языков программирования (C++, С#, Java, Delphi,) для реализации курсового проекта был выбран C# и интегрированная среда разработки Microsoft Visual Studio 2022.

C# является объектно-ориентированным и в этом плане много перенял у Java и С++. Он поддерживает полиморфизм, наследование, перегрузку операторов, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений. C# продолжает активно развиваться, и с каждой новой версией появляется все больше интересной функциональности, как, например, лямбда-выражения, динамическое связывание, асинхронные методы и так далее.

Преимущества C# по сравнению с аналогичными программными продуктами:

* поддержка подавляющего большинства продуктов Microsoft;
* типы данных имеют фиксированный размер что повышает мобильность языка и упрощает программирование;
* автоматическая очистка оперативной память;
* большое количество специальных конструкций, разработанных для понимания и написания кода.

Microsoft Visual Studio – полнофункциональная интегрированная среда разработки (IDE) с поддержкой популярных языков программирования, среди которых С, C++, VB.NET, C#, F#, JavaScript, Python. Функциональность Visual Studio охватывает все этапы разработки программного обеспечения, предоставляя современные инструменты для написания кода, проектирования графических интерфейсов, сборки, отладки и тестирования приложений. Возможности Visual Studio могут быть дополнены путем подключения необходимых расширений. Visual Studio поддерживает подсветку синтаксиса, вставку фрагментов кода, отображение структуры и связанных функций. Существенно ускорить работу помогает технология IntelliSense – автозавершение кода по мере ввода.

Для реализации справочной системы был выбран HTML HelpWorkShop. Этот инструмент имеет легкий для использования и интуитивно понятный графический интерфейс. С помощью него можно создавать проекты, гипертекстовые разделы, файлы содержаний, заголовочные файлы, а также прочие элементы, которые могут понадобиться для последующего использования в справочной системе.

Microsoft SQL Server – система управления реляционными БД, разработанная корпорацией Microsoft. Основной используемый язык запросов – Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с небольшими и средними по размеру БД, а также крупными базами данных масштаба предприятия. SQL является общепринятым интерфейсом к БД.

# 2 РАБОЧИЙ ПРОЕКТ

## 2.1 Проектирование физической модели базы данных

Физическая модель данных описывает то, как данные хранятся в компьютере, представляя информацию о структуре записей, их упорядоченности и существующих путях доступа. Также описываются типы, идентификаторы и разрядность полей.

На основе логической модели были создана физическая модель БД.

Таблица 1 – Физическая модель таблицы «User»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Подпись поля | Тип данных | Длина | Ключ |
| Id | Идентификатор | int | 4 | PK |
| Name | ФИО | nvarchar | 250 | - |
| Login | Логин | nvarchar | 250 | - |
| Password | Пароль | nvarchar | 250 | - |
| PassportSeries | Серия паспорта | int | 4 | - |
| PassportNumber | Номер паспорта | int | 4 | - |
| Photo | Фото | image | 16 | - |
| TelephoneNumber | Телефонный номер | nvarchar | 250 |  |
| RoleId | Номер роли | int | 4 | FK |

Таблица 2 – Физическая модель таблицы «UserRole»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Подпись поля | Тип данных | Длина | Ключ |
| Id | Идентификатор | int | 4 | PK |
| Value | Значение | nvarchar | 250 | - |

Таблица 3 – Физическая модель таблицы «Room»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Подпись поля | Тип данных | Длина | Ключ |
| Id | Идентификатор | int | 4 | PK |
| Number | Номер комнаты | nvarchar | 50 | - |
| Price | Цена | decimal | 9 | - |
| CountOfBeds | Количество кроватей | int | 255 | - |
| Floor | Этаж | int | 255 | - |
| Description | Описание | nvarchar | 4000 | - |
| TypeId | Номер тип | int | 4 | FK |

Таблица 4 – Физическая модель таблицы «RoomType»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Подпись поля | Тип данных | Длина | Ключ |
| Id | Идентификатор | int | 4 | PK |
| Value | Вопрос | nvarchar | 255 | - |
| Description | Описание | nvarchar | 3000 |  |

Таблица 5 – Физическая модель таблицы «Service»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Подпись поля | Тип данных | Длина | Ключ |
| Id | Идентификатор | int | 4 | PK |
| NameOf | Наименование | nvarchar | 250 | - |
| Description | Описание | nvarchar | 3500 | - |
| Price | Цена | decimal | 9 | - |
| IsUserBuyService | Услуга куплена? | bit | 1 | - |

Таблица 6 – Физическая модель таблицы «ReservationHasServices»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Подпись поля | Тип данных | Длина | Ключ |
| ReservationId | Идентификатор брони | int | 4 | PK/FK |
| ServiceId | Идентификатор услуги | int | 4 | PK/FK |

Таблица 7 – Физическая модель таблицы «Reservation»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Подпись поля | Тип данных | Длина | Ключ |
| Id | Идентификатор | int | 4 | PK |
| FromDate | Дата заселения | datetime | 8 | - |
| ToDate | Дата выселения | datetime | 8 | - |
| TotalPrice | Итоговая цена | decimal | 9 | - |
| UserId | Идентификатор пользователя | int | 4 | FK |
| RoomId | Идентификатор комнаты | int | 4 | FK |

Таблица 8 – Физическая модель таблицы «TotalPaid»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Подпись поля | Тип данных | Длина | Ключ |
| Id | Идентификатор | int | 4 | PK |
| Date | Дата начисления | datetime | 8 | - |
| Price | Начисленные средства | decimal | 9 | - |
| ReservationId | Идентификатор брони | int | 4 | FK |

## 2.2 Функциональное взаимодействие модулей программы

На основе разработанного технического проекта было написано приложение по теме «АРМ администратора отеля». Для создания программы был использован язык программирования C# и ИСР Visual studio 2019. В программе используются следующие языковые модули:

* System – содержит фундаментальные и базовые классы;
* System.Windows.Forms – содержит классы для создания работы с формами;
* System.Media – содержит классы для воспроизведения звуковых файлов и работы с системными звуками;
* System.Security.Cryptography - Предоставляет криптографические службы, включая безопасное кодирование и декодирование данных, а также множество других операций, таких как хэширование, генерация случайных чисел и проверка подлинности сообщений.;
* System.Windows– Предоставляет несколько важных классов базовых элементов Windows Presentation Foundation (WPF), различные классы, которые поддерживают систему свойств и логику событий WPF, а также другие типы, более широко применяемые в ядре и инфраструктуре WPF;
* System.Linq– Предоставляет классы и интерфейсы, поддерживающие запросы с использованием LINQ;
* System.IO - Содержит типы, позволяющие осуществлять чтение и запись в файлы и потоки данных, а также типы для базовой поддержки файлов и папок;
* System.Globalization - Содержит классы, определяющие сведения, относящиеся к культуре, такие как язык, название страны, используемые календари, шаблоны форматирования дат, денежных сумм и чисел, а также порядок сортировки строк. Эти классы применяются при создании многоязыковых приложений;
* System.Drawing - Предоставляет доступ к основным графическим функциям GDI+;
* System.Data - предоставляет доступ к классам, представляющим архитектуру ADO.NET. ADO.NET позволяет создавать компоненты, эффективно управляющие данными из нескольких источников данных;
* System.Collections - Содержит интерфейсы и классы, определяющие различные коллекции объектов, такие как списки, очереди, битовые массивы, хэш-таблицы и словари;
* System.ComponentModel - Предоставляет классы, используемые для реализации поведения компонентов и элементов управления во время разработки и во время выполнения. Это пространство имен содержит базовые классы и интерфейсы для реализации атрибутов и преобразователей типов, привязки к источникам данных и лицензирования компонентов;
* System.Runtime.CompilerServices - Предоставляет средства, позволяющие разработчикам компиляторов, использующим управляемый код, задавать в метаданных атрибуты, влияющие на поведение среды CLR во время выполнения;

Также были разработаны следующие пользовательские модули:

* MainViewModel.cs (главное окно программы) – осуществляет переход между другими модулями;
* RoomViewModel.cs (окно «Номера») – отвечает за добавление, изменение и удаление данных об номерах отеля;
* ServicesViewModel.cs (окно «Услуги») – отвечает за добавление, изменение и удаление данных об услугах отеля;
* ReservationViewModel.cs (окно «Бронирование») - отвечает за добавление, изменение и удаление брони отеля;
* UserViewModel.cs (окно «Клиенты») - отвечает за добавление, изменение и удаление данных об клиентах отеля;
* PaidViewModel.cs (окно «Оплата») - отвечает за добавление оплаты бронирования;
* SearchViewModel.cs (окна: «Поиск пользователя», «Поиск комнаты», «Поиск брони»)- отвечает за поиск необходимой информации по фильтрам;
* Hotel Manual.chm (справка) – обеспечивает отображения справочной информации о программе и работе с ней.

В модуле MainViewModel.cs описаны и используются следующие процедуры:

* private void OpenNewWindow(object obj) – открывает выбранное пользователем окно;
* private Brush GetColorOfReservation(int id) – возвращает экземпляр Brush в зависимости от того оплачена бронь или нет. Id – идентификатор брони;
* public void UpdateReservation(object obj) – обновляет календарь брони.

В модуле RoomViewModel.cs описаны и используются следующие процедуры:

* private void UpdateCollections() – обновляет коллекции содержащиеся в модуле;
* private void UpdateRoomInfo(int id) – обновляет информацию выбранной комнаты по ее id;
* private void AddRoom(object obj) – добавляет данные комнаты в БД;
* private void UpdateRoom(object obj) – обновляет данные комнаты в БД;
* private void DeleteRoom(object obj) – удаляет данные комнаты из БД.

В модуле RoomViewModel.cs описаны и используются следующие процедуры:

* private void UpdateCollections() – обновляет коллекции содержащиеся в модуле;
* private void UpdateSelectedItemInfo(int id) – обновляет информацию выбранной услуге по ее id;
* private void AddService(object obj) – добавляет данные услуги в БД;
* private void UpdateService (object obj) – обновляет данные услуги в БД;
* private void DeleteService (object obj) – удаляет данные услуги из БД.

В модуле RoomViewModel.cs описаны и используются следующие процедуры:

* private void UpdateCollections() – обновляет коллекции содержащиеся в модуле;
* private void UpdateSelectedItemInfo(int id) – обновляет информацию выбранной брони по ее id;
* private bool CheckInfo() – проверяет корректность полученных данных;
* private bool CheckDate(Reservation notIncludeReservation) – проверяет не пересекаются ли даты новой брони с уже существующими. notIncludeReservation – бронь которая не будет учитываться при проверке;
* private void AddReservation(object obj) – добавляет данные брони в БД;
* private void UpdateReservation (object obj) – обновляет данные брони в БД;
* private void DeleteReservation (object obj) – удаляет данные брони из БД;
* private void Checkout(object obj) – рассчитывает итоговую цену и цену за день для добавляемой брони;
* ReservationSearch(object obj) – открывает окно поиска брони;
* UserSearch(object obj) – открывает окно поиска пользователя;
* RoomSearch(object obj) – открывает окно поиска комнаты;
* Checkout(Reservation reservation) – рассчитывает итоговую цену и цену за день для определенной брони.

В модуле UserViewModel.cs описаны и используются следующие процедуры:

* private void UpdateCollections() – обновляет коллекции содержащиеся в модуле;
* private void UpdateSelectedItemInfo(int id) – обновляет информацию выбранном клиенте по его id;
* private bool CheckInfo() – проверяет корректность полученных данных;
* public string TelephoneNumberFilter(string str, list<char> charsToRemove) – меняет формат полученного телефонного номера перед добавлением его в БД. str – строка с номером, charsToRemove – символы для удаления;
* private void AddUser(object obj) – добавляет данные пользователя в БД;
* private void UpdateUser (object obj) – обновляет данные пользователя в БД;
* private void DeleteUser (object obj) – удаляет данные пользователя из БД;

В модуле PaidViewModel.cs описаны и используются следующие процедуры:

* private void UpdateData(int id) – обновляет данные о стоимости брони;
* private void AddPaid(object obj) – добавляет данные об оплате брони в БД;

В модуле SearchViewModel.cs описаны и используются следующие процедуры:

* private void ReservationFilter(int id) – фильтрует коллекцию бронирований;
* private void UserFilter(object obj) – фильтрует коллекцию пользователей;
* private void RoomFilter(object obj) – фильтрует коллекцию комнат;

## 2.3 Разработка unit-тестов

Для осуществления корректной работы разработанного приложения, в части правильности вычисления значений функции, был разработан тестовый проект, включающий в себя набор unit-тестов. Данные тесты охватывают все возможные тестовые случаи, которые необходимо проверить:

* проверка на корректность положительного дробного числа с корректными входными данными;
* проверка на корректность положительного дробного числа с отрицательным числом на входе;
* проверка на корректность положительного дробного числа с некорректно заданной строкой на входе;
* проверка на корректность положительного дробного числа с излишком запятых на входе;
* проверка на корректность положительного дробного числа с булевым значением на входе;
* проверка получения массива байтов по картинке хранящийся в папке проекта;
* проверка получения хешкода, где в качестве аргумента используется строка со значением«1»;
* проверка получения хешкода, где в качестве аргумента используется пустая строка;
* проверка получения хешкода, где в качестве аргумента используется пустота - null.

Код данных тестов представлен в приложении В.

## 2.4 Входные и выходные данные

Входными данными для модуля MainViewModel.cs являются данные об брони и номерах из БД, которые загружаются в коллекцию Nodes. К коллекции Nodes привязан элемент Canvas находящийся на главной форме, он отображает данные в виде календаря.

Входными данными для модуля PaidViewModel.cs является внесенная плата за бронь.

Выходными данными для модуля ReservationViewModel.cs являются данные клиента, номер, выбранные услуги, дата заселения и дата выселения.

Выходными данными для модуля RoomViewModel.cs являются номер комнаты, количество кроватей, цена за день, этаж и тип номера.

Выходными данными для модуля ServicesViewModel.cs являются наименование услуги, описание услуги и цена.

Выходными данными для модуля UserViewModel.cs являются ФИО клиента, серия паспорта, номер паспорта, номер телефона.

Выходными данными для модуля SearchViewModel.cs фильтры, которые могут меняться в зависимости от выбранного окна.

## 2.5 Работа с программой

При запуске программы отображается окно загрузки(рисунок 4). После загрузки отобразится окно авторизации(рисунок 5).

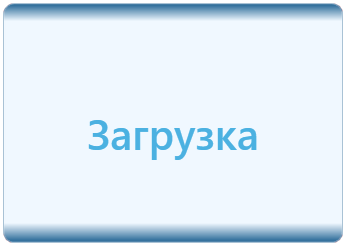


Рисунок 4 – Окно загрузки

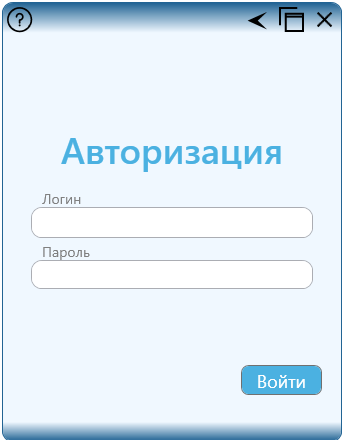


Рисунок 5 – Окно авторизации

Пользователь вводит свои данные и после успешной авторизации попадает в главное окно(рисунок 6). В главном окно осуществляются переходы к окнам: «Номера» (рисунок 7), «Услуги»(рисунок 8), «Бронирование» (рисунок 9), «Клиенты» (рисунок 10), «Пользователи», «Оплата брони» (рисунок 12) и «Профиль». Окно «Профиль» и «Пользователи» идентичны по интерфейсу(рисунок 11).

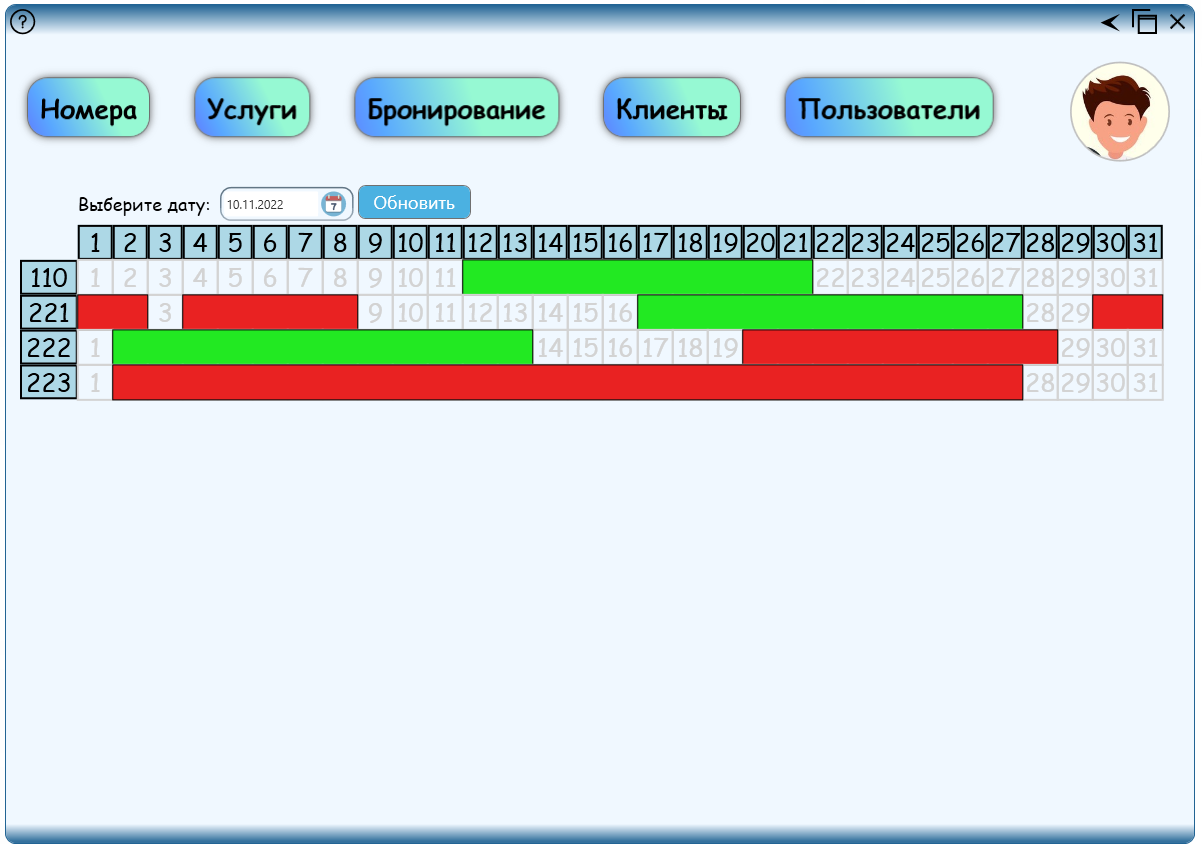


Рисунок 6 – Главное окно

Для добавления данных о номере используется кнопка добавить, для переключения между вкладками используются соответствующие вкладки(рисунок 7). «Тип номера» содержит весь список типов номера хранящихся в бд.

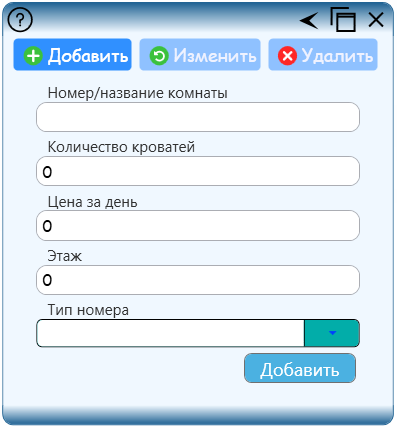


Рисунок 7 – Окно «Номера»

Для добавления данных об услуге используется кнопка добавить, для переключения между вкладками используются соответствующие вкладки(рисунок 8).

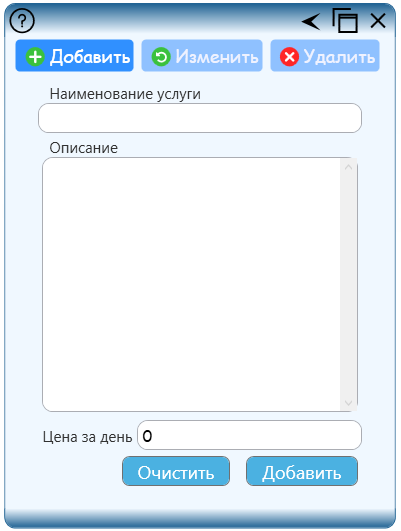


Рисунок 8 – Окно «Услуги»

Для добавления данных о брони используется кнопка добавить, для переключения между вкладками используются соответствующие вкладки(рисунок 9). «Клиент» содержит весь пользователей являющихся клиентами. «Номер» содержит весь список Номеров хранящихся в бд.

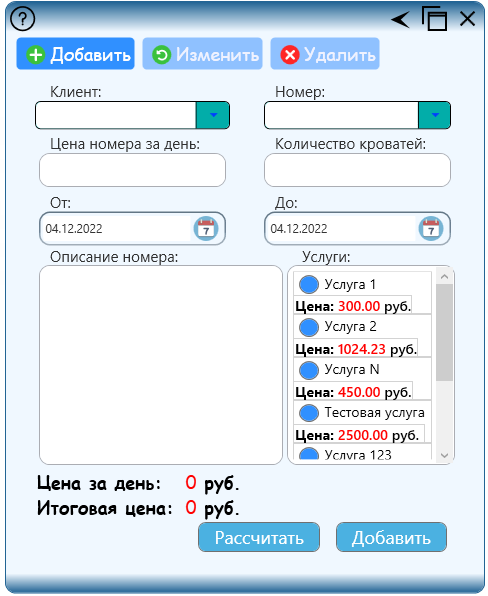


Рисунок 9 – Окно «Бронирование»

Для добавления данных о клиентах используется кнопка добавить, для переключения между вкладками используются соответствующие вкладки(рисунок 10).

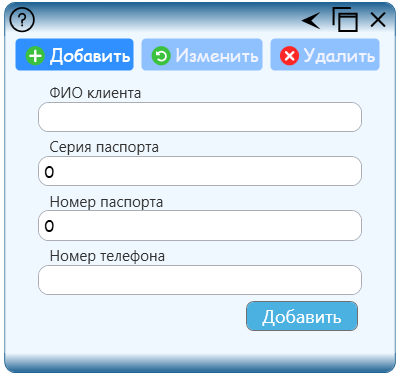


Рисунок 10 – Окно «Клиенты»

Для добавления данных о пользователях системы используется кнопка Зарегистрировать, для загрузки изображений из локального носителя используется кнопка «Загрузить картикну»(рисунок 11).

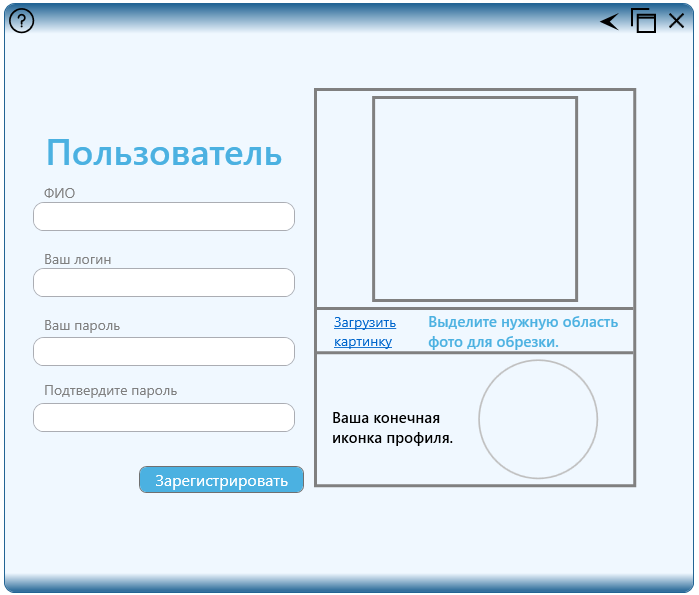


Рисунок 11 – Окно «Пользователи»

Для добавления платы за бронь используется кнопка Добавить, для(рисунок 12).

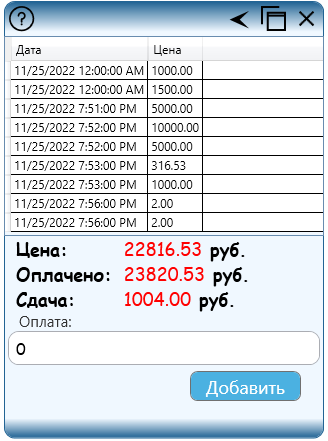


Рисунок 12 – Окно «Оплата брони»

Для фильтрации используются различные параметры. Поиск бронирования по клиенту и номеру(рисунок 13).

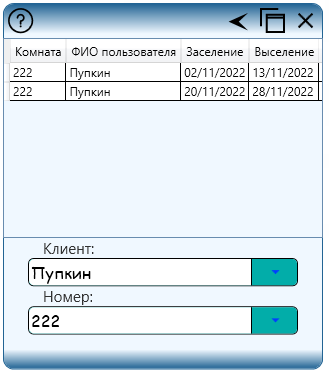


Рисунок 13 – Окно «Поиск бронирования»

Для получения справочной информации требуется нажатие на кнопку «Помощь» в виде вопроса, которая находится в каждом окне программы.

Для создания справочной системы использовалась программа HTML HelpWorkshop. Окно помощи(рисунок 14) включает в себя следующие страницы:

* Окно Авторизации;
* Главное окно;
* Окно Бронирования;
* Окно Клиентов;
* Окно Номеров;
* Окно Пользователей;
* Окно Услуг.

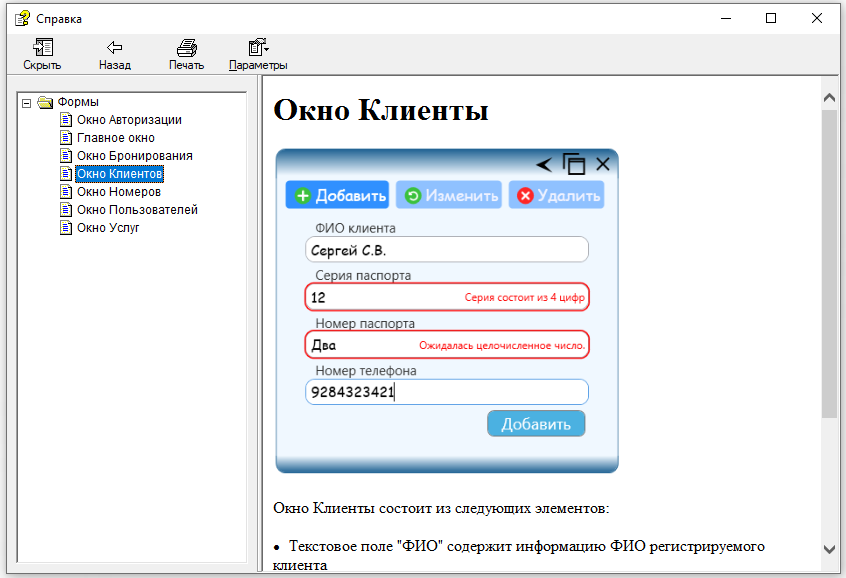


Рисунок 14 – Окно справки

# ЗАКЛЮЧЕНИЕ

В результате выполнения данного курсового проекта было написано приложение по теме «АРМ администратора отеля».

Основными достоинствами программы являются:

* программа проста эксплуатацию;
* наличие удобного графического интерфейса;
* данный программный продукт будет предназначен для широкого круга пользователей;
* наличие справочной системы.

В ходе выполнения курсового проекта были достигнуты поставленные цели и решены следующие задачи:

* исследованы основные объекты отеля;
* спроектированы структура приложения и БД;
* разработаны справочные и тестовые материалы;
* определены минимальные требования к техническим и программным средствам;
* описаны входные и выходные данные;
* выбраны средства реализации приложения;
* разработана БД;
* разработан программный код приложения;
* создано руководство пользователя.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Хейлсберг А. Язык программирования C#. М.:Вильямс,2018. 784 с.
2. ШарпДж. MicrosoftVisualC#. Подробное руководство.М.:Вильямс, 2017.848 с.
3. Крис Ф. Основы Windows Presentation Foundation / Крис Андерсон ; [пер. с англ. А. Слинкина]. - Москва : ДМК Пресс ; Санкт-Петербург : БХВ-Петербург, 2008. - 427 с. : ил.; 24 см.; ISBN 978-5-94074-363-7 (ДМК Пресс)
4. СтанекУ.Р. MS SQL Server 2017.Справочникадминистратора.М.:Вильямс, 2017. 720 с.
5. ОппельЭ. Дж.,ВайнбергП.Н., ГроффДж. Р. SQL 2019. М.:Вильямс, 2019. 960 с.
6. Документация Microsoft. URL: https://docs.microsoft.com/ru-ru/documentation (дата обращения: 01.12.2021).
7. Оппель Э. Дж. SQL: Полное руководство. М.:Вильямс, 2015. 960 с.
8. Библиотека классов .NET Framework. URL: https:// msdn.microsoft.com/ru-ru/library/mt472912(v=vs.110).aspx (дата обращения: 01.12.2021).

ПРИЛОЖЕНИЕ А - код приложения

Функции MainViewModel.cs

public class MainViewModel : BaseViewModel

{

public abstract class Node

{

public double XPos { get; set; }

public double YPos { get; set; }

public Node()

{

OnClickCommand = new RelayCommand(OnClick);

}

public RelayCommand OnClickCommand

{

get;

private set;

}

public abstract void OnClick(object obj);

}

private DateTime \_date;

private ObservableCollection<Node> \_nodes;

private ObservableCollection<Room> \_rooms;

public DateTime Date

{

get => \_date;

set

{

SetPropertyChanged(ref \_date, value, nameof(Date));

UpdateReservation(null);

}

}

public ObservableCollection<Node> Nodes

{

get => \_nodes;

set

{

SetPropertyChanged(ref \_nodes, value, nameof(Nodes));

}

}

public ObservableCollection<Room> Rooms

{

get => \_rooms;

set

{

SetPropertyChanged(ref \_rooms, value, nameof(Rooms));

}

}

public MainViewModel()

{

OpenNewWindowCommand = new RelayCommand(OpenNewWindow);

UpdateContentCommand = new RelayCommand(UpdateReservation);

Nodes = new ObservableCollection<Node>();

Rooms = new ObservableCollection<Room>();

using(HotellDbEntities context = new HotellDbEntities())

{

List<Room> roomList = context.Room.ToList();

foreach(var item in roomList)

{

Rooms.Add(item);

}

}

Date = DateTime.Now;

}

public RelayCommand OpenNewWindowCommand

{

get;

private set;

}

public RelayCommand UpdateContentCommand

{

get;

private set;

}

private void OpenNewWindow(object obj)

{

var parametr = obj.ToString();

if(parametr == "Room"){

(new RoomWindow()).ShowDialog();

}

else if (parametr == "Service") {

(new ServicesWindow()).ShowDialog();

}

else if (parametr == "Reservation")

{

(new ReservationWindow()).ShowDialog();

}

else if (parametr == "Users")

{

(new UserWindow()).ShowDialog();

}

else if (parametr == "Employees")

{

(new RegistrationWindow()).ShowDialog();

}

else if (parametr == "UserProfile")

{

(new RegistrationWindow(CurrentUser.Id)).ShowDialog();

}

UpdateReservation(null);

}

private int scale = 35;

private Brush GetColorOfReservation(int id)

{

using(HotellDbEntities context = new HotellDbEntities())

{

Reservation reservation = context.Reservation.Where(res => res.Id == id).FirstOrDefault();

decimal price = reservation.TotalPrice;

List<TotalPaid> totalPaid = reservation.TotalPaid.ToList();

decimal paid = 0;

foreach(var item in totalPaid)

{

paid += item.Price;

}

if(paid >= price)

{

return new SolidColorBrush(Color.FromArgb(255, 34, 233, 34));

}

else

{

return new SolidColorBrush(Color.FromArgb(255, 233, 34, 34));

}

}

}

public void UpdateReservation(object obj)

{

Nodes = new ObservableCollection<Node>();

Rooms = new ObservableCollection<Room>();

using (HotellDbEntities context = new HotellDbEntities())

{

int row = 0;

List<Room> roomList = context.Room.ToList();

foreach (var item in roomList)

{

Rooms.Add(item);

}

foreach (var room in Rooms)

{

List<Reservation> reservations = context.Reservation.Where(res =>

((res.FromDate.Month == Date.Month && res.FromDate.Year == Date.Year) ||

(res.ToDate.Month == Date.Month && res.ToDate.Year == Date.Year)) &&

(res.RoomId == room.Id)).ToList();

for(int i = 0; i < 31; i++)

{

Nodes.Add(new ShapeNode

{

ReservationId = -1,

XPos = i \* 35,

YPos = row \* scale,

Stroke = Brushes.LightGray,

StrokeThickness = 2,

Fill = new SolidColorBrush(Color.FromArgb(0, 0, 0, 0)),

Geometry = new RectangleGeometry { Rect = new Rect(0, 0, scale, scale) }

});

Nodes.Add(new TextNode

{

XPos = i \* 35,

YPos = row \* scale,

Text = $"{i + 1}"

});

}

/\*List<Reservation> reservations = room.Reservation.Where(res =>

(res.FromDate.Month == Date.Month && res.FromDate.Year == Date.Year) ||

(res.ToDate.Month == Date.Month && res.ToDate.Year == Date.Year)).ToList();\*/

if (reservations.Count < 1)

{

row += 1;

continue;

}

foreach (var reservation in reservations)

{

if (reservation.FromDate.Month != Date.Month && reservation.ToDate.Month == Date.Month)

{

int width = reservation.ToDate.Day \* scale;

Nodes.Add(new ShapeNode

{

ReservationId = reservation.Id,

XPos = 0,

YPos = row \* scale,

Stroke = Brushes.Black,

StrokeThickness = 1,

Fill = GetColorOfReservation(reservation.Id),

Geometry = new RectangleGeometry { Rect = new Rect(0, 0, width, scale) }

});

}

else if (reservation.FromDate.Month == Date.Month && reservation.ToDate.Month != Date.Month)

{

int width = (scale \* 31) - (reservation.FromDate.Day \* scale) + 35;

Nodes.Add(new ShapeNode

{

ReservationId = reservation.Id,

XPos = reservation.FromDate.Day \* scale - 35,

YPos = row \* scale,

Stroke = Brushes.Black,

StrokeThickness = 1,

Fill = GetColorOfReservation(reservation.Id),

Geometry = new RectangleGeometry { Rect = new Rect(0, 0, width, scale) }

});

}

else if (reservation.FromDate.Month != Date.Month && reservation.ToDate.Month != Date.Month &&

Date.Month > reservation.FromDate.Month && Date.Month < reservation.ToDate.Month)

{

int width = (scale \* 31);

Nodes.Add(new ShapeNode

{

ReservationId = reservation.Id,

XPos = 0,

YPos = row \* scale,

Stroke = Brushes.Black,

StrokeThickness = 1,

Fill = GetColorOfReservation(reservation.Id),

Geometry = new RectangleGeometry { Rect = new Rect(0, 0, width, scale) }

});

}

else if (reservation.FromDate.Month == Date.Month && reservation.ToDate.Month == Date.Month)

{

int width = (reservation.ToDate.Day \* scale) - (reservation.FromDate.Day \* scale) + scale;

Nodes.Add(new ShapeNode

{

ReservationId = reservation.Id,

XPos = reservation.FromDate.Day \* scale - scale,

YPos = row \* scale,

Stroke = Brushes.Black,

StrokeThickness = 1,

Fill = GetColorOfReservation(reservation.Id),

Geometry = new RectangleGeometry { Rect = new Rect(0, 0, width, scale) }

});

}

}

row += 1;

}

}

}

}

public class ShapeNode : Node

{

public Geometry Geometry { get; set; }

public Brush Stroke { get; set; }

public Brush Fill { get; set; }

public double StrokeThickness { get; set; }

public int ReservationId { get; set; }

public override void OnClick(object obj)

{

//TODO: Сделать

ShapeNode node = obj as ShapeNode;

CurrentUser.ReservationId = node.ReservationId;

(new PaidWindow()).ShowDialog();

}

}

public class TextNode : Node

{

public string Text { get; set; }

public override void OnClick(object obj)

{

}

}

Функции PaidViewModel.cs

public class PaidViewModel : BaseViewModel

{

private int \_id;

private decimal \_price;

private decimal \_paid;

private decimal \_change;

private decimal \_addedPrice;

private ObservableCollection<TotalPaid> \_totalPaid;

public int Id

{

get => \_id;

set => SetPropertyChanged(ref \_id, value, nameof(Id));

}

public decimal Price

{

get => \_price;

set => SetPropertyChanged(ref \_price, value, nameof(Price));

}

public decimal Paid

{

get => \_paid;

set => SetPropertyChanged(ref \_paid, value, nameof(Paid));

}

public decimal Change

{

get => \_change;

set => SetPropertyChanged(ref \_change, value, nameof(Change));

}

public decimal AddedPrice

{

get => \_addedPrice;

set => SetPropertyChanged(ref \_addedPrice, value, nameof(AddedPrice));

}

public ObservableCollection<TotalPaid> TotalPaid

{

get => \_totalPaid;

set => SetPropertyChanged(ref \_totalPaid, value, nameof(TotalPaid));

}

public PaidViewModel()

{

int reservationId = CurrentUser.ReservationId;

AddPaidCommand = new RelayCommand(AddPaid);

UpdateData(reservationId);

}

private void UpdateData(int id)

{

TotalPaid = new ObservableCollection<TotalPaid>();

Price = 0;

Paid = 0;

Change = 0;

try

{

using (HotellDbEntities context = new HotellDbEntities())

{

Reservation res = context.Reservation.Where(reservation => reservation.Id == id).FirstOrDefault();

if (res != null)

{

Price = res.TotalPrice;

List<TotalPaid> totalPaid = res.TotalPaid.ToList();

foreach (var paid in totalPaid)

{

TotalPaid.Add(paid);

Paid += paid.Price;

}

}

if(Paid > Price)

{

Change = Paid - Price;

}

}

}

catch

{

}

}

public RelayCommand AddPaidCommand

{

get;

private set;

}

private void AddPaid(object obj)

{

if (AddedPrice <= 0)

{

MessageBox.Show("Введите корректные данные.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Exclamation);

return;

}

//TODO: Сделать нормальную проверку (существует ли такая комната уже или нет)

try

{

using (HotellDbEntities context = new HotellDbEntities())

{

Reservation res = context.Reservation.Where(reservation => reservation.Id == CurrentUser.ReservationId).FirstOrDefault();

var src = DateTime.Now;

TotalPaid paid = new TotalPaid

{

Date = new DateTime(src.Year, src.Month, src.Day, src.Hour, src.Minute, 0),

Price = AddedPrice,

ReservationId = CurrentUser.ReservationId

};

context.TotalPaid.Add(paid);

context.SaveChanges();

MessageBox.Show("Сумма успешно внесена.", "Добавление", MessageBoxButton.OK, MessageBoxImage.Information);

AddedPrice = 0;

UpdateData(CurrentUser.ReservationId);

}

}

catch

{

}

}

}

Функции ReservationViewModel.cs

public class ReservationViewModel : BaseViewModel

{

private int \_id;

private DateTime \_fromDate = DateTime.Now;

private DateTime \_toDate = DateTime.Now;

private int \_userId;

private int \_roomId;

private double \_priceForOneDay;

private double \_allPrice;

private ObservableCollection<Service> \_services;

private ObservableCollection<User> \_users;

private ObservableCollection<Room> \_rooms;

private ObservableCollection<Reservation> \_reservations;

private ObservableCollection<Reservation> \_filteredReservations;

private Service \_selectedService;

private User \_selectedUser;

private Room \_selectedRoom;

private Reservation \_selectedReservation;

private string \_selectedRoomType;

public int Id

{

get => \_id;

set => SetPropertyChanged(ref \_id, value, nameof(Id));

}

public DateTime FromDate

{

get => \_fromDate;

set

{

if (ToDate != null)

{

if(value > ToDate)

{

MessageBox.Show("Дата заселения должна быть меньше даты выселения.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Exclamation);

return;

}

}

SetPropertyChanged(ref \_fromDate, value, nameof(FromDate));

}

}

public DateTime ToDate

{

get => \_toDate;

set

{

if (FromDate != null)

{

if (value < FromDate)

{

MessageBox.Show("Дата выселения должна быть больше даты заселения.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Exclamation);

return;

}

}

SetPropertyChanged(ref \_toDate, value, nameof(ToDate));

}

}

public int UserId

{

get => \_userId;

set => SetPropertyChanged(ref \_userId, value, nameof(UserId));

}

public int RoomId

{

get => \_roomId;

set => SetPropertyChanged(ref \_roomId, value, nameof(RoomId));

}

public double PriceForOneDay

{

get => \_priceForOneDay;

set => SetPropertyChanged(ref \_priceForOneDay, value, nameof(PriceForOneDay));

}

public double AllPrice

{

get => \_allPrice;

set => SetPropertyChanged(ref \_allPrice, value, nameof(AllPrice));

}

public ObservableCollection<Service> Services

{

get => \_services;

set

{

SetPropertyChanged(ref \_services, value, nameof(Services));

}

}

public ObservableCollection<User> Users

{

get => \_users;

set => SetPropertyChanged(ref \_users, value, nameof(Users));

}

public ObservableCollection<Room> Rooms

{

get => \_rooms;

set => SetPropertyChanged(ref \_rooms, value, nameof(Rooms));

}

public ObservableCollection<Reservation> Reservations

{

get => \_reservations;

set => SetPropertyChanged(ref \_reservations, value, nameof(Reservations));

}

public ObservableCollection<Reservation> FilteredReservations

{

get => \_filteredReservations;

set => SetPropertyChanged(ref \_filteredReservations, value, nameof(FilteredReservations));

}

public Service SelectedService

{

get => \_selectedService;

set

{

SetPropertyChanged(ref \_selectedService, value, nameof(SelectedService));

}

}

public User SelectedUser

{

get => \_selectedUser;

set

{

SetPropertyChanged(ref \_selectedUser, value, nameof(SelectedUser));

}

}

public Room SelectedRoom

{

get => \_selectedRoom;

set

{

SetPropertyChanged(ref \_selectedRoom, value, nameof(SelectedRoom));

if(value != null)

{

using (HotellDbEntities context = new HotellDbEntities())

{

SelectedRoomType = context.RoomType.Where(type => type.Id == value.TypeId).FirstOrDefault().Value + "\n\n";

SelectedRoomType += context.RoomType.Where(type => type.Id == value.TypeId).FirstOrDefault().Description;

}

}

}

}

public Reservation SelectedReservation

{

get => \_selectedReservation;

set

{

SetPropertyChanged(ref \_selectedReservation, value, nameof(SelectedReservation));

if (value != null)

{

UpdateSelectedItemInfo(value.Id);

}

}

}

public string SelectedRoomType

{

get => \_selectedRoomType;

set

{

SetPropertyChanged(ref \_selectedRoomType, value, nameof(SelectedRoomType));

}

}

public ReservationViewModel()

{

AddReservationCommand = new RelayCommand(AddReservation);

UpdateReservationCommand = new RelayCommand(UpdateReservation);

DeleteReservationCommand = new RelayCommand(DeleteReservation);

CheckoutCommand = new RelayCommand(Checkout);

ReservationSearchCommand = new RelayCommand(ReservationSearch);

UserSearchCommand = new RelayCommand(UserSearch);

RoomSearchCommand = new RelayCommand(RoomSearch);

UpdateCollections();

}

private void UpdateCollections()

{

Services = new ObservableCollection<Service>();

Rooms = new ObservableCollection<Room>();

Users = new ObservableCollection<User>();

Reservations = new ObservableCollection<Reservation>();

FilteredReservations = new ObservableCollection<Reservation>();

try

{

using (HotellDbEntities context = new HotellDbEntities())

{

List<Service> serviceList = context.Service.ToList();

foreach (var item in serviceList)

{

Services.Add(item);

}

List<Room> roomList = context.Room.ToList();

foreach (var item in roomList)

{

Rooms.Add(item);

}

List<User> userList = context.User.Where(user => user.RoleId == 2).ToList();

foreach (var item in userList)

{

Users.Add(item);

}

List<Reservation> reservationList = context.Reservation.ToList();

foreach (var item in reservationList)

{

Reservations.Add(item);

}

FilteredReservations = Reservations;

}

}

catch

{

}

}

private void UpdateSelectedItemInfo(int id)

{

try

{

using (HotellDbEntities context = new HotellDbEntities())

{

Reservation reservation = context.Reservation.Where(entity => entity.Id == id).FirstOrDefault();

if (reservation != null)

{

\_fromDate = reservation.FromDate;

\_toDate = reservation.ToDate;

FromDate = reservation.FromDate;

ToDate = reservation.ToDate;

SelectedUser = Users.Where(user => user.Id == reservation.UserId).FirstOrDefault();

SelectedRoom = Rooms.Where(room => room.Id == reservation.RoomId).FirstOrDefault();

Services = new ObservableCollection<Service>();

List<Service> serviceList = context.Service.ToList();

foreach (var item in serviceList)

{

Service service = reservation.Service.Where(ser => ser.Id == item.Id).FirstOrDefault();

if(service != null)

{

item.IsUserBuyService = true;

}

Services.Add(item);

}

/\*foreach (var service in reservation.Service)

{

Services.Where(res => res.Id == service.Id).FirstOrDefault().IsUserBuyService = true;

}\*/

Checkout(reservation);

}

else

{

//TODO: tada

//MessageBox.Show("Комната не найдена в базе данных.");

}

}

}

catch

{

}

}

private bool CheckInfo()

{

if (SelectedUser == null || SelectedRoom == null || ToDate == null || FromDate == null)

{

MessageBox.Show($"Заполните все поля.", "Внимание", MessageBoxButton.OK, MessageBoxImage.Information);

return false;

}

return true;

}

private bool CheckDate(Reservation notIncludeReservation)

{

if(notIncludeReservation == null)

{

notIncludeReservation = new Reservation();

notIncludeReservation.Id = -1;

}

using (HotellDbEntities contesxt = new HotellDbEntities())

{

List<Reservation> reservations = contesxt.Reservation.Where(

entity => (ToDate >= entity.FromDate && FromDate <= entity.FromDate || FromDate <= entity.ToDate && ToDate >= entity.ToDate

|| FromDate <= entity.FromDate && ToDate >= entity.ToDate || FromDate >= entity.FromDate && ToDate <= entity.ToDate)

&& entity.RoomId == SelectedRoom.Id && entity.Id != notIncludeReservation.Id).ToList();

if (reservations.Count > 0)

{

string reservationsInfo = $"\n" +

$"Пользователь: \t\t{reservations[0].User.Name}\n" +

$"Дата заселения: \t\t{reservations[0].FromDate}\n" +

$"Дата выселения: \t\t{reservations[0].ToDate}\n";

MessageBox.Show($"Номер уже занят на эту дату.{reservationsInfo}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Exclamation);

return true;

}

else { return false; }

}

}

public RelayCommand AddReservationCommand

{

get;

private set;

}

public RelayCommand UpdateReservationCommand

{

get;

private set;

}

public RelayCommand DeleteReservationCommand

{

get;

private set;

}

public RelayCommand ReservationSearchCommand

{

get;

private set;

}

public RelayCommand UserSearchCommand

{

get;

private set;

}

public RelayCommand RoomSearchCommand

{

get;

private set;

}

public RelayCommand CheckoutCommand

{

get;

private set;

}

private void AddReservation(object obj)

{

if (CheckInfo())

{

try

{

if (CheckDate(null))

{

return;

}

List<Service> selectedServices = Services.Where(service => service.IsUserBuyService == true).ToList();

int id = 0;

string services = null;

foreach (var service in selectedServices)

{

services += "[" + service.NameOf + "] ";

}

Checkout(new object());

if (MessageBox.Show(

$"Проверьте корректность введенных данных.\n\n" +

$"Пользователь:\t\t{SelectedUser.Name}\n" +

$"Комната:\t\t{SelectedRoom.Number}\n" +

$"Дата заселения:\t\t{FromDate.ToString("dd/MM/yyyy")}\n" +

$"Дата выселения:\t\t{ToDate.ToString("dd/MM/yyyy")}\n" +

$"Услуги: \t\t{services}\n" +

$"К оплате: \t\t{AllPrice}\n" +

$"Добавить бронь?", "Проверка корректности", MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)

{

using (HotellDbEntities context = new HotellDbEntities())

{

Reservation reservation = new Reservation

{

FromDate = FromDate,

ToDate = ToDate,

UserId = SelectedUser.Id,

RoomId = SelectedRoom.Id,

TotalPrice = (decimal)AllPrice

};

context.Reservation.Add(reservation);

context.SaveChanges();

id = reservation.Id;

}

using (HotellDbEntities context = new HotellDbEntities())

{

Reservation reservation = context.Reservation.Where(entity => entity.Id == id).FirstOrDefault();

foreach (var service in selectedServices)

{

Service dbService = context.Service.Where(entity => entity.Id == service.Id).FirstOrDefault();

reservation.Service.Add(dbService);

}

context.SaveChanges();

}

using(HotellDbEntities context = new HotellDbEntities())

{

Reservation reservation = context.Reservation.Where(entity => entity.Id == id).FirstOrDefault();

MessageBox.Show("Данные о брони успешно добавлены.", "Оповещение", MessageBoxButton.OK, MessageBoxImage.Information);

}

}

}

catch

{

}

finally

{

//UpdateCollections();

Reservations = new ObservableCollection<Reservation>();

using (HotellDbEntities context = new HotellDbEntities())

{

List<Reservation> reservationList = context.Reservation.ToList();

foreach (var item in reservationList)

{

Reservations.Add(item);

}

}

}

}

}

private void UpdateReservation(object obj)

{

if (CheckInfo())

{

int id = 0;

try

{

if (CheckDate(SelectedReservation))

{

return;

}

List<Service> selectedServices = Services.Where(service => service.IsUserBuyService == true).ToList();

string services = null;

foreach (var service in selectedServices)

{

services += "[" + service.NameOf + "] ";

}

Checkout(new object());

if (MessageBox.Show(

$"Проверьте корректность введенных данных.\n\n" +

$"Пользователь:\t\t{SelectedUser.Name}\n" +

$"Комната:\t\t{SelectedRoom.Number}\n" +

$"Дата заселения:\t\t{FromDate.ToString("dd/MM/yyyy")}\n" +

$"Дата выселения:\t\t{ToDate.ToString("dd/MM/yyyy")}\n" +

$"Услуги: \t\t{services}\n" +

$"К оплате: \t\t{AllPrice}\n" +

$"Добавить бронь?", "Проверка корректности", MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)

{

using (HotellDbEntities context = new HotellDbEntities())

{

Reservation reservation = new Reservation();

reservation = context.Reservation.Where(entity => entity.Id == SelectedReservation.Id).FirstOrDefault();

if (reservation == null)

{

MessageBox.Show("Бронь не существует.", "Внимание", MessageBoxButton.OK, MessageBoxImage.Exclamation);

return;

}

reservation.FromDate = FromDate;

reservation.ToDate = ToDate;

reservation.UserId = SelectedUser.Id;

reservation.RoomId = SelectedRoom.Id;

reservation.TotalPrice = (decimal)AllPrice;

context.SaveChanges();

id = reservation.Id;

}

using (HotellDbEntities context = new HotellDbEntities())

{

Reservation reservation = context.Reservation.Where(entity => entity.Id == id).FirstOrDefault();

List<Service> ser = reservation.Service.ToList();

foreach(var service in ser)

{

reservation.Service.Remove(service);

}

context.SaveChanges();

}

using (HotellDbEntities context = new HotellDbEntities())

{

Reservation reservation = context.Reservation.Where(entity => entity.Id == id).FirstOrDefault();

foreach (var service in selectedServices)

{

Service dbService = context.Service.Where(entity => entity.Id == service.Id).FirstOrDefault();

reservation.Service.Add(dbService);

}

context.SaveChanges();

MessageBox.Show("Данные о брони успешно обновлены.", "Оповещение", MessageBoxButton.OK, MessageBoxImage.Information);

}

UpdateCollections();

SelectedReservation = Reservations.Where(reservation => reservation.Id == id).FirstOrDefault();

SelectedRoom = Rooms.Where(room => room.Id == SelectedReservation.RoomId).FirstOrDefault();

SelectedUser = Users.Where(user => user.Id == SelectedReservation.UserId).FirstOrDefault();

}

}

catch

{

}

finally

{

//TODO: Переместить обновление формы из всех finally выше в блок try. Иначе при сбое будет еще одна ошибка(нахуя этот блок нужен вообще?)

}

}

}

private void DeleteReservation(object obj)

{

if (SelectedReservation == null)

{

MessageBox.Show("Выберите бронь.", "Внимание", MessageBoxButton.OK, MessageBoxImage.Exclamation);

return;

}

try

{

using (HotellDbEntities context = new HotellDbEntities())

{

Reservation reservation = new Reservation();

reservation = context.Reservation.Where(entity => entity.Id == SelectedReservation.Id).FirstOrDefault();

if (reservation == null)

{

MessageBox.Show("Брони не существует.", "Внимание", MessageBoxButton.OK, MessageBoxImage.Exclamation);

return;

}

List<Service> ser = reservation.Service.ToList();

foreach (var service in ser)

{

reservation.Service.Remove(service);

}

List<TotalPaid> totalPaid = reservation.TotalPaid.ToList();

foreach (var paid in totalPaid)

{

reservation.TotalPaid.Remove(paid);

}

context.SaveChanges();

}

using (HotellDbEntities context = new HotellDbEntities())

{

Reservation reservation = new Reservation();

reservation = context.Reservation.Where(entity => entity.Id == SelectedReservation.Id).FirstOrDefault();

context.Reservation.Remove(reservation);

context.SaveChanges();

MessageBox.Show("Данные о брони успешно удалены.", "Оповещение", MessageBoxButton.OK, MessageBoxImage.Information);

UpdateCollections();

SelectedReservation = null;

SelectedRoom = null;

SelectedUser = null;

}

}

catch

{

}

}

private void Checkout(object obj)

{

if(CheckInfo())

{

int countOfDays = (ToDate - FromDate).Days + 1;

double priceForOneDay = 0;

double allPrice = 0;

List<Service> selectedServices = Services.Where(service => service.IsUserBuyService == true).ToList();

foreach(var service in selectedServices)

{

priceForOneDay += Convert.ToDouble(service.Price);

allPrice += Convert.ToDouble(service.Price \* countOfDays);

}

priceForOneDay += Convert.ToDouble(SelectedRoom.Price);

allPrice += Convert.ToDouble(SelectedRoom.Price \* countOfDays);

PriceForOneDay = Math.Round(priceForOneDay, 2);

AllPrice = Math.Round(allPrice, 2);

}

}

private void ReservationSearch(object obj)

{

new ReservationSearchWindow().ShowDialog();

Reservation reservation = Reservations.Where(res => res.Id == CurrentUser.ReservationId).FirstOrDefault();

if (reservation != null)

{

SelectedReservation = reservation;

}

}

private void UserSearch(object obj)

{

new UserSearchWindow().ShowDialog();

User selectedUser = Users.Where(user => user.Id == CurrentUser.UserId).FirstOrDefault();

if (selectedUser != null)

{

SelectedUser = selectedUser;

}

}

private void RoomSearch(object obj)

{

RoomSearchWindow win = new RoomSearchWindow();

win.ShowDialog();

if (win.RoomsDataGrid.SelectedItem != null)

{

Room selectedRoom = Rooms.Where(room => room.Id == (win.RoomsDataGrid.SelectedItem as Room).Id).FirstOrDefault();

if (selectedRoom != null)

{

SelectedRoom = selectedRoom;

}

}

}

private void Checkout(Reservation reservation)

{

if (CheckInfo())

{

int countOfDays = (ToDate - FromDate).Days + 1;

double priceForOneDay = 0;

double allPrice = 0;

List<Service> selectedServices = reservation.Service.ToList();

foreach (var service in selectedServices)

{

priceForOneDay += Convert.ToDouble(service.Price);

allPrice += Convert.ToDouble(service.Price \* countOfDays);

}

priceForOneDay += Convert.ToDouble(SelectedRoom.Price);

allPrice += Convert.ToDouble(SelectedRoom.Price \* countOfDays);

PriceForOneDay = Math.Round(priceForOneDay, 2);

AllPrice = Math.Round(allPrice, 2);

}

}

}

Функции RoomViewModel.cs

public class RoomViewModel : BaseViewModel

{

private int \_id;

private string \_number;

private decimal \_price;

private int \_countOfBeds;

private int \_floor;

private ObservableCollection<RoomType> \_roomTypes;

private ObservableCollection<Room> \_rooms;

private RoomType \_selectedRoomType;

private Room \_selectedRoom;

public int Id

{

get => \_id;

set => SetPropertyChanged(ref \_id, value, nameof(Id));

}

public string Number

{

get => \_number;

set => SetPropertyChanged(ref \_number, value, nameof(Number));

}

public decimal Price

{

get => \_price;

set => SetPropertyChanged(ref \_price, value, nameof(Price));

}

public int CountOfBeds

{

get => \_countOfBeds;

set => SetPropertyChanged(ref \_countOfBeds, value, nameof(CountOfBeds));

}

public int Floor

{

get => \_floor;

set => SetPropertyChanged(ref \_floor, value, nameof(Floor));

}

public ObservableCollection<RoomType> RoomTypes

{

get => \_roomTypes;

set => SetPropertyChanged(ref \_roomTypes, value, nameof(RoomTypes));

}

public ObservableCollection<Room> Rooms

{

get => \_rooms;

set => SetPropertyChanged(ref \_rooms, value, nameof(Rooms));

}

public RoomType SelectedRoomType

{

get => \_selectedRoomType;

set => SetPropertyChanged(ref \_selectedRoomType, value, nameof(SelectedRoomType));

}

public Room SelectedRoom

{

get => \_selectedRoom;

set

{

SetPropertyChanged(ref \_selectedRoom, value, nameof(SelectedRoom));

if(value != null)

{

UpdateRoomInfo(value.Id);

}

}

}

public RoomViewModel()

{

AddRoomCommand = new RelayCommand(AddRoom);

UpdateRoomCommand = new RelayCommand(UpdateRoom);

DeleteRoomCommand = new RelayCommand(DeleteRoom);

RoomSearchCommand = new RelayCommand(RoomSearch);

try

{

using (HotellDbEntities context = new HotellDbEntities())

{

List<RoomType> roomTypes = context.RoomType.ToList();

RoomTypes = new ObservableCollection<RoomType>();

foreach (var item in roomTypes)

{

RoomTypes.Add(item);

}

}

using (HotellDbEntities context = new HotellDbEntities())

{

List<Room> rooms = context.Room.ToList();

Rooms = new ObservableCollection<Room>();

foreach (var item in rooms)

{

Rooms.Add(item);

}

}

}

catch

{

}

}

private void UpdateCollections()

{

Rooms = new ObservableCollection<Room>();

try

{

using (HotellDbEntities context = new HotellDbEntities())

{

List<Room> rooms = context.Room.ToList();

Rooms = new ObservableCollection<Room>();

foreach (var item in rooms)

{

Rooms.Add(item);

}

}

}

catch

{

}

}

private void UpdateRoomInfo(int id)

{

try

{

using (HotellDbEntities context = new HotellDbEntities())

{

Room room = context.Room.Where(entity => entity.Id == id).FirstOrDefault();

if (room != null)

{

Number = room.Number;

CountOfBeds = room.CountOfBeds;

Price = room.Price;

Floor = room.Floor;

SelectedRoomType = RoomTypes.Where(type => type.Id == room.TypeId).FirstOrDefault();

}

else

{

//TODO: tada

//MessageBox.Show("Комната не найдена в базе данных.");

}

}

}

catch

{

}

}

public RelayCommand AddRoomCommand

{

get;

private set;

}

public RelayCommand UpdateRoomCommand

{

get;

private set;

}

public RelayCommand DeleteRoomCommand

{

get;

private set;

}

public RelayCommand RoomSearchCommand

{

get;

private set;

}

private void AddRoom(object obj)

{

if(Number == null || Price == null || CountOfBeds == null)

{

return;

}

//TODO: Сделать нормальную проверку (существует ли такая комната уже или нет)

using (HotellDbEntities context = new HotellDbEntities())

{

Room newRoom = new Room()

{

Number = Number,

Price = Price,

CountOfBeds = CountOfBeds,

Floor = Floor,

TypeId = SelectedRoomType.Id

};

context.Room.Add(newRoom);

context.SaveChanges();

//TODO: Не забудь обновить обсерные коллекции после добавления и удаления комнат

MessageBox.Show("Комната успешно добавлена.", "Добавление", MessageBoxButton.OK, MessageBoxImage.Information);

}

}

private void UpdateRoom(object obj)

{

try

{

using (HotellDbEntities context = new HotellDbEntities())

{

Room room = context.Room.Where(entity => entity.Id == SelectedRoom.Id).FirstOrDefault();

if (room != null)

{

room.Number = Number;

room.Price = Price;

room.CountOfBeds= CountOfBeds;

room.Floor = Floor;

room.TypeId= SelectedRoomType.Id;

context.SaveChanges();

MessageBox.Show("Комната успешно изменена.", "Изменения", MessageBoxButton.OK, MessageBoxImage.Information);

UpdateCollections();

}

else

{

//TODO: tada

//MessageBox.Show("Комната не найдена в базе данных.");

}

}

}

catch

{

}

}

private void DeleteRoom(object obj)

{

if(SelectedRoom != null)

{

if(MessageBox.Show(

$"Вы точно хотите удалить комнату - [{SelectedRoom.Id}] {SelectedRoom.Number}?",

"Удаление",

MessageBoxButton.YesNo,

MessageBoxImage.Question) == MessageBoxResult.Yes)

{

try

{

using (HotellDbEntities context = new HotellDbEntities())

{

//Сделать проверку на наличие связей у комнаты и у прочих сущностей при удалении

Room room = context.Room.FirstOrDefault(entity => entity.Id == SelectedRoom.Id);

context.Room.Remove(room);

context.SaveChanges();

MessageBox.Show("Комната успешно удалена.", "Удаление");

UpdateCollections();

}

}

catch

{ }

}

}

}

private void RoomSearch(object obj)

{

RoomSearchWindow win = new RoomSearchWindow();

win.ShowDialog();

//TODO: ОШИБКА! SelectedItem нужно ниже

if (win.RoomsDataGrid != null)

{

Room selectedRoom = Rooms.Where(room => room.Id == (win.RoomsDataGrid.SelectedItem as Room).Id).FirstOrDefault();

if (selectedRoom != null)

{

SelectedRoom = selectedRoom;

}

}

}

}

Функции SearchViewModel.cs

public class SearchViewModel : BaseViewModel

{

private int \_passportSeries;

private int \_passportNumber;

private double \_roomPrice;

private int \_countOfBeds;

private ObservableCollection<User> \_users;

private ObservableCollection<Room> \_rooms;

private ObservableCollection<Reservation> \_reservations;

private ObservableCollection<RoomType> \_roomTypes;

private ObservableCollection<Reservation> \_filteredReservations;

private ObservableCollection<User> \_filteredUsers;

private ObservableCollection<Room> \_filteredRooms;

private User \_selectedUser;

private User \_filteredUser;

private Room \_selectedRoom;

private RoomType \_selectedRoomType;

private Reservation \_selectedReservation;

[Required(ErrorMessage = "Это поле обязательно")]

[Range(1000, 9999, ErrorMessage = "Серия состоит из 4 цифр")]

public int PassportSeries

{

get => \_passportSeries;

set

{

ValidateProperty(value, "PassportSeries");

SetPropertyChanged(ref \_passportSeries, value, nameof(PassportSeries));

UserFilter();

}

}

[Required(ErrorMessage = "Это поле обязательно")]

[Range(100000, 999999, ErrorMessage = "Номер состоит из 6 цифр")]

public int PassportNumber

{

get => \_passportNumber;

set

{

ValidateProperty(value, "PassportNumber");

SetPropertyChanged(ref \_passportNumber, value, nameof(PassportNumber));

UserFilter();

}

}

public double RoomPrice

{

get => \_roomPrice;

set

{

SetPropertyChanged(ref \_roomPrice, value, nameof(RoomPrice));

RoomFilter();

}

}

public int CountOfBeds

{

get => \_countOfBeds;

set

{

SetPropertyChanged(ref \_countOfBeds, value, nameof(CountOfBeds));

RoomFilter();

}

}

public ObservableCollection<User> Users

{

get => \_users;

set

{

SetPropertyChanged(ref \_users, value, nameof(Users));

}

}

public ObservableCollection<Room> Rooms

{

get => \_rooms;

set

{

SetPropertyChanged(ref \_rooms, value, nameof(Rooms));

}

}

public ObservableCollection<Reservation> Reservations

{

get => \_reservations;

set

{

SetPropertyChanged(ref \_reservations, value, nameof(Reservations));

}

}

public ObservableCollection<RoomType> RoomTypes

{

get => \_roomTypes;

set

{

SetPropertyChanged(ref \_roomTypes, value, nameof(RoomTypes));

}

}

public ObservableCollection<Reservation> FilteredReservations

{

get => \_filteredReservations;

set

{

SetPropertyChanged(ref \_filteredReservations, value, nameof(FilteredReservations));

}

}

public ObservableCollection<User> FilteredUsers

{

get => \_filteredUsers;

set

{

SetPropertyChanged(ref \_filteredUsers, value, nameof(FilteredUsers));

}

}

public ObservableCollection<Room> FilteredRooms

{

get => \_filteredRooms;

set

{

SetPropertyChanged(ref \_filteredRooms, value, nameof(FilteredRooms));

}

}

public User SelectedUser

{

get => \_selectedUser;

set

{

SetPropertyChanged(ref \_selectedUser, value, nameof(SelectedUser));

ReservationFilter();

}

}

public User FilteredUser

{

get => \_filteredUser;

set

{

SetPropertyChanged(ref \_filteredUser, value, nameof(FilteredUser));

if (value != null)

{

CurrentUser.UserId = value.Id;

}

}

}

public Room SelectedRoom

{

get => \_selectedRoom;

set

{

SetPropertyChanged(ref \_selectedRoom, value, nameof(SelectedRoom));

ReservationFilter();

}

}

public RoomType SelectedRoomType

{

get => \_selectedRoomType;

set

{

SetPropertyChanged(ref \_selectedRoomType, value, nameof(SelectedRoomType));

RoomFilter();

}

}

public Reservation SelectedReservation

{

get => \_selectedReservation;

set

{

SetPropertyChanged(ref \_selectedReservation, value, nameof(SelectedReservation));

if (value != null)

{

CurrentUser.ReservationId = value.Id;

}

}

}

public SearchViewModel()

{

Users = new ObservableCollection<User>();

Rooms = new ObservableCollection<Room>();

Reservations = new ObservableCollection<Reservation>();

FilteredReservations = new ObservableCollection<Reservation>();

FilteredUsers = new ObservableCollection<User>();

FilteredRooms = new ObservableCollection<Room>();

RoomTypes = new ObservableCollection<RoomType>();

\_countOfBeds = 1;

using(HotellDbEntities context = new HotellDbEntities())

{

List<User> userList = context.User.Where(user => user.RoleId == 2).ToList();

foreach (var item in userList)

{

Users.Add(item);

}

List<Room> roomList = context.Room.ToList();

foreach (var item in roomList)

{

if ((decimal)\_roomPrice < item.Price){

\_roomPrice = (double)item.Price;

}

Rooms.Add(item);

}

List<Reservation> reservationList = context.Reservation.ToList();

foreach (var item in reservationList)

{

Reservations.Add(item);

}

List<RoomType> roomTypesList = context.RoomType.ToList();

foreach (var item in roomTypesList)

{

RoomTypes.Add(item);

}

FilteredReservations = Reservations;

FilteredUsers = Users;

FilteredRooms = Rooms;

RoomPrice = \_roomPrice;

}

}

private void ReservationFilter()

{

List<Reservation> reservations = Reservations.ToList();

FilteredReservations = new ObservableCollection<Reservation>();

if (SelectedUser != null)

{

reservations = reservations.Where(res => res.UserId == SelectedUser.Id).ToList();

}

if (SelectedRoom != null)

{

reservations = reservations.Where(res => res.RoomId == SelectedRoom.Id).ToList();

}

foreach (var res in reservations)

{

FilteredReservations.Add(res);

}

}

private void UserFilter()

{

List<User> users = Users.ToList();

FilteredUsers = new ObservableCollection<User>();

if (PassportSeries >= 1000)

{

users = users.Where(user => user.PassportSeries == PassportSeries).ToList();

}

if (PassportNumber >= 100000)

{

users = users.Where(user => user.PassportNumber == PassportNumber).ToList();

}

foreach (var user in users)

{

FilteredUsers.Add(user);

}

}

private void RoomFilter()

{

List<Room> rooms = Rooms.ToList();

FilteredRooms = new ObservableCollection<Room>();

if (RoomPrice > 0)

{

rooms = rooms.Where(room => room.Price <= (decimal)RoomPrice).ToList();

}

if (CountOfBeds > 0)

{

rooms = rooms.Where(room => room.CountOfBeds >= CountOfBeds).ToList();

}

if (SelectedRoomType != null)

{

rooms = rooms.Where(room => room.RoomType == SelectedRoomType).ToList();

}

foreach (var room in rooms)

{

FilteredRooms.Add(room);

}

}

}

Функции ServicesViewModel.cs

public class ServiceViewModel : BaseViewModel

{

private int \_id;

private string \_nameOf;

private string \_description;

private decimal \_price;

private ObservableCollection<Service> \_services;

private Service \_selectedService;

public int Id

{

get => \_id;

set => SetPropertyChanged(ref \_id, value, nameof(Id));

}

public string NameOf

{

get => \_nameOf;

set => SetPropertyChanged(ref \_nameOf, value, nameof(NameOf));

}

public string Description

{

get => \_description;

set => SetPropertyChanged(ref \_description, value, nameof(Description));

}

public decimal Price

{

get => \_price;

set => SetPropertyChanged(ref \_price, value, nameof(Price));

}

public ObservableCollection<Service> Services

{

get => \_services;

set => SetPropertyChanged(ref \_services, value, nameof(Services));

}

public Service SelectedService

{

get => \_selectedService;

set

{

SetPropertyChanged(ref \_selectedService, value, nameof(SelectedService));

if(value != null)

{

UpdateSelecetdItemInfo(value.Id);

}

}

}

public ServiceViewModel()

{

AddServiceCommand = new RelayCommand(AddService);

UpdateServiceCommand = new RelayCommand(UpdateService);

DeleteServiceCommand = new RelayCommand(DeleteService);

UpdateCollections();

}

private void UpdateCollections()

{

Services = new ObservableCollection<Service>();

try

{

using (HotellDbEntities context = new HotellDbEntities())

{

List<Data.Service> serviceList = context.Service.ToList();

foreach (var item in serviceList)

{

Services.Add(item);

}

}

}

catch

{

}

}

private void UpdateSelecetdItemInfo(int id)

{

try

{

using (HotellDbEntities context = new HotellDbEntities())

{

Service service = context.Service.Where(entity => entity.Id == id).FirstOrDefault();

if (service != null)

{

NameOf = service.NameOf;

Description = service.Description;

Price = (decimal)service.Price;

}

else

{

//TODO: tada

//MessageBox.Show("Комната не найдена в базе данных.");

}

}

}

catch

{

}

}

public RelayCommand AddServiceCommand

{

get;

private set;

}

public RelayCommand UpdateServiceCommand

{

get;

private set;

}

public RelayCommand DeleteServiceCommand

{

get;

private set;

}

private void AddService(object obj)

{

if (NameOf == null || Description == null || Price < 0)

{

MessageBox.Show("Введите корректные данные.", "Ошибка");

return;

}

//TODO: Сделать нормальную проверку (существует ли такая комната уже или нет)

try

{

using (HotellDbEntities context = new HotellDbEntities())

{

Data.Service service = context.Service.Where(entity => entity.NameOf == NameOf).FirstOrDefault();

if (service != null)

{

if (MessageBox.Show(

$"Услуга с названием \"[{service.NameOf}]\" уже существует? Добавить снова?",

"Внимание",

MessageBoxButton.YesNo,

MessageBoxImage.Warning) != MessageBoxResult.Yes)

{

return;

}

}

Data.Service newService = new Data.Service()

{

NameOf = NameOf,

Description = Description,

Price = Price,

IsUserBuyService = false,

};

context.Service.Add(newService);

context.SaveChanges();

//TODO: Не забудь обновить обсерные коллекции после добавления и удаления комнат

MessageBox.Show("Услуга успешно добавлена.", "Добавление", MessageBoxButton.OK, MessageBoxImage.Information);

}

}

catch

{

}

finally

{

UpdateCollections();

}

}

private void UpdateService(object obj)

{

try

{

using (HotellDbEntities context = new HotellDbEntities())

{

Data.Service service = context.Service.Where(entity => entity.Id == SelectedService.Id).FirstOrDefault();

if (service != null)

{

service.NameOf = NameOf;

service.Description = Description;

service.Price = Price;

context.SaveChanges();

MessageBox.Show("Услуга успешно изменена.", "Изменения", MessageBoxButton.OK, MessageBoxImage.Information);

}

else

{

//TODO: tada

//MessageBox.Show("Комната не найдена в базе данных.");

}

}

}

catch

{

}

finally

{

UpdateCollections();

}

}

private void DeleteService(object obj)

{

if (SelectedService != null)

{

if (MessageBox.Show(

$"Вы точно хотите удалить услугу - [{SelectedService.Id}] {SelectedService.NameOf}?",

"Удаление",

MessageBoxButton.YesNo,

MessageBoxImage.Question) == MessageBoxResult.Yes)

{

try

{

using (HotellDbEntities context = new HotellDbEntities())

{

//Сделать проверку на наличие связей у комнаты и у прочих сущностей при удалении

Data.Service service = context.Service.FirstOrDefault(entity => entity.Id == SelectedService.Id);

context.Service.Remove(service);

context.SaveChanges();

MessageBox.Show("Услуга успешно удалена.", "Удаление", MessageBoxButton.OK, MessageBoxImage.Information);

}

}

catch

{

}

finally

{

UpdateCollections();

}

}

}

}

}

Функции UserViewModel.cs

public class UserViewModel : BaseViewModel

{

private int \_id;

private string \_name = "";

private string \_login;

private string \_password;

private string \_confirmPassword;

private int \_passportSeries;

private int \_passportNumber;

private BitmapImage \_photo;

private string \_telephoneNumber;

private int \_roleId;

private ObservableCollection<User> \_users;

private User \_selectedUser;

public int Id

{

get => \_id;

set => SetPropertyChanged(ref \_id, value, nameof(Id));

}

[Required(ErrorMessage = "Имя обязательно")]

[StringLength(100, MinimumLength = 5, ErrorMessage ="Длина должна быть в рамках [{1}-{2}]")]

public string Name

{

get => \_name;

set

{

ValidateProperty(value, "Name");

SetPropertyChanged(ref \_name, value, nameof(Name));

}

}

[Required(ErrorMessage = "Логин обязателен")]

[StringLength(25, MinimumLength = 3, ErrorMessage = "Длина должна быть в рамках [3-25]")]

public string Login

{

get => \_login;

set

{

ValidateProperty(value, "Login");

SetPropertyChanged(ref \_login, value, nameof(Login));

}

}

[Required(ErrorMessage = "Пароль обязателен")]

[StringLength(100, MinimumLength = 3, ErrorMessage = "Длина должна быть в рамках [3-100]")]

public string Password

{

get => \_password;

set

{

ValidateProperty(value, "Password");

SetPropertyChanged(ref \_password, value, nameof(Password));

}

}

[Required(ErrorMessage = "Пароль обязателен.")]

[StringLength(100, MinimumLength = 3, ErrorMessage = "Длина должна быть в рамках [3-100]")]

public string ConfirmPassword

{

get => \_confirmPassword;

set

{

ValidateProperty(value, "ConfirmPassword");

SetPropertyChanged(ref \_confirmPassword, value, nameof(ConfirmPassword));

}

}

[Required(ErrorMessage = "Это поле обязательно")]

[Range(1000, 9999, ErrorMessage = "Серия состоит из 4 цифр")]

public int PassportSeries

{

get => \_passportSeries;

set

{

ValidateProperty(value, "PassportSeries");

SetPropertyChanged(ref \_passportSeries, value, nameof(PassportSeries));

}

}

[Required(ErrorMessage = "Это поле обязательно")]

[Range(100000, 999999, ErrorMessage = "Номер состоит из 6 цифр")]

public int PassportNumber

{

get => \_passportNumber;

set

{

ValidateProperty(value, "PassportNumber");

SetPropertyChanged(ref \_passportNumber, value, nameof(PassportNumber));

}

}

public BitmapImage Photo

{

get => \_photo;

set => SetPropertyChanged(ref \_photo, value, nameof(Photo));

}

[Required(ErrorMessage = "Телефон обязателен.")]

[RegularExpression(@"\(?\d{3}\)?-? \*\d{3}-? \*-?\d{2} \*-?\d{2}",

ErrorMessage = "Неверный формат номера.")]

public string TelephoneNumber

{

get => \_telephoneNumber;

set

{

ValidateProperty(value, "TelephoneNumber");

SetPropertyChanged(ref \_telephoneNumber, value, nameof(TelephoneNumber));

}

}

public int RoleId

{

get => \_roleId;

set => SetPropertyChanged(ref \_roleId, value, nameof(RoleId));

}

public ObservableCollection<User> Users

{

get => \_users;

set => SetPropertyChanged(ref \_users, value, nameof(Users));

}

public User SelectedUser

{

get => \_selectedUser;

set

{

SetPropertyChanged(ref \_selectedUser, value, nameof(SelectedUser));

if(value != null)

{

UpdateSelectedItemInfo(value.Id);

}

}

}

/\*private void ValidateProperty<T>(T value, string name)

{

Validator.ValidateProperty(value, new ValidationContext(this, null, null)

{

MemberName = name,

});

}\*/

public UserViewModel()

{

AddUserCommand = new RelayCommand(AddUser);

UpdateUserCommand = new RelayCommand(UpdateUser);

DeleteUserCommand = new RelayCommand(DeleteUser);

UserSearchCommand = new RelayCommand(UserSearch);

UpdateCollections();

}

private void UpdateCollections()

{

Users = new ObservableCollection<User>();

try

{

using (HotellDbEntities context = new HotellDbEntities())

{

List<User> userList = context.User.Where(user => user.RoleId == 2).ToList();

foreach (var item in userList)

{

Users.Add(item);

}

}

}

catch

{

}

}

private bool CheckInfo()

{

if (Name.Length < 5 || PassportNumber < 100000 || PassportNumber > 999999

|| PassportSeries < 1000 || PassportSeries > 9999 || TelephoneNumber.Length < 10)

{

MessageBox.Show($"Введите данные.\nПоля не должны содержать сообщение об ошибке.", "Внимание", MessageBoxButton.OK, MessageBoxImage.Information);

return false;

}

return true;

}

private void UpdateSelectedItemInfo(int id)

{

try

{

using (HotellDbEntities context = new HotellDbEntities())

{

User user = context.User.Where(entity => entity.Id == id).FirstOrDefault();

if (user != null)

{

Name = user.Name;

PassportSeries = (int)user.PassportSeries;

PassportNumber = (int)user.PassportNumber;

TelephoneNumber = user.TelephoneNumber;

}

else

{

//TODO: tada

//MessageBox.Show("Комната не найдена в базе данных.");

}

}

}

catch

{

}

}

public RelayCommand AddUserCommand

{

get;

private set;

}

public RelayCommand UpdateUserCommand

{

get;

private set;

}

public RelayCommand DeleteUserCommand

{

get;

private set;

}

public RelayCommand UserSearchCommand

{

get;

private set;

}

private List<char> charsToRemove = new List<char>() { ' ', '-', '(', ')' };

public string TelephoneNumberFilter(string str, List<char> charsToRemove)

{

foreach (char c in charsToRemove)

{

str = str.Replace(c.ToString(), String.Empty);

}

str = str.Insert(8, " ");

str = str.Insert(6, " ");

str = str.Insert(3, " ");

return str;

}

private void AddUser(object obj)

{

if (CheckInfo())

{

if (MessageBox.Show(

$"Последние корректные данные.\n\n" +

$"Имя:\t\t{Name}\n" +

$"Серия паспорта:\t{PassportSeries}\n" +

$"Номер паспорта:\t{PassportNumber}\n" +

$"Номер телефона:\t{TelephoneNumber}\n" +

$"Добавить пользователя?", "Проверка корректности", MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)

{

using(HotellDbEntities context = new HotellDbEntities())

{

User user = new User();

user = context.User.Where(entity => entity.PassportSeries == PassportSeries && entity.PassportNumber == PassportNumber).FirstOrDefault();

if(user != null)

{

MessageBox.Show(

"Пользователь с такими паспортными данными уже существует.\nЕсли вы хотите изменить данные пользователя, то перейдите в раздел \"Изменить\"",

"Внимание",

MessageBoxButton.OK,

MessageBoxImage.Exclamation);

return;

}

string telephonerNumber = TelephoneNumberFilter(TelephoneNumber, charsToRemove);

user = context.User.Where(entity => entity.TelephoneNumber == telephonerNumber).FirstOrDefault();

if (user != null)

{

if (

!(MessageBox.Show("Пользователь с таким телефонным номером уже существует.\nПродолжить?", "Оповещение", MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)){

return;

}

}

}

try

{

using (HotellDbEntities context = new HotellDbEntities())

{

User user = new User()

{

Name = Name,

PassportNumber = PassportNumber,

PassportSeries = PassportSeries,

TelephoneNumber = TelephoneNumberFilter(TelephoneNumber, charsToRemove),

RoleId = context.UserRole.Where(entity => entity.Value == "Клиент").FirstOrDefault().Id

};

context.User.Add(user);

context.SaveChanges();

MessageBox.Show("Клиент успешно добавлен.", "Оповещение", MessageBoxButton.OK, MessageBoxImage.Information);

}

}

catch

{

}

finally

{

UpdateCollections();

}

}

}

}

private void UpdateUser(object obj)

{

if (CheckInfo())

{

if (MessageBox.Show(

$"Последние корректные данные.\n\n" +

$"Имя:\t\t{Name}\n" +

$"Серия паспорта:\t{PassportSeries}\n" +

$"Номер паспорта:\t{PassportNumber}\n" +

$"Номер телефона:\t{TelephoneNumber}\n" +

$"Обновить данные пользователя?", "Проверка корректности", MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)

{

try

{

using (HotellDbEntities context = new HotellDbEntities())

{

User user = new User();

user = context.User.Where(entity => entity.Id == SelectedUser.Id).FirstOrDefault();

if(user == null)

{

MessageBox.Show("Пользователя не существует.", "Внимание", MessageBoxButton.OK, MessageBoxImage.Exclamation);

return;

}

user.Name = Name;

user.PassportSeries = PassportSeries;

user.PassportNumber = PassportNumber;

user.TelephoneNumber = TelephoneNumberFilter(TelephoneNumber, charsToRemove);

context.SaveChanges();

Id = user.Id;

MessageBox.Show("Данные о клиенте успешно обновлены.", "Оповещение", MessageBoxButton.OK, MessageBoxImage.Information);

}

}

catch

{

}

finally

{

UpdateCollections();

SelectedUser = Users.Where(user => user.Id == Id).FirstOrDefault();

}

}

}

}

private void DeleteUser(object obj)

{

if(SelectedUser == null)

{

MessageBox.Show("Выберите пользователя.", "Внимание", MessageBoxButton.OK, MessageBoxImage.Exclamation);

return;

}

try

{

using (HotellDbEntities context = new HotellDbEntities())

{

User user = new User();

user = context.User.Where(entity => entity.Id == SelectedUser.Id).FirstOrDefault();

if (user == null)

{

MessageBox.Show("Пользователя не существует.", "Внимание", MessageBoxButton.OK, MessageBoxImage.Exclamation);

return;

}

context.User.Remove(user);

context.SaveChanges();

MessageBox.Show("Данные о клиенте успешно удалены.", "Оповещение", MessageBoxButton.OK, MessageBoxImage.Information);

}

}

catch

{

}

finally

{

UpdateCollections();

SelectedUser = null;

}

}

private void UserSearch(object obj)

{

UserSearchWindow win = new UserSearchWindow();

win.ShowDialog();

//TODO: Переделать везде поиск так, как снизу (везде нету Selecteditem)

if (win.UsersDataGrid.SelectedItem != null)

{

User selectedUser = Users.Where(user => user.Id == (win.UsersDataGrid.SelectedItem as User).Id).FirstOrDefault();

if (selectedUser != null)

{

SelectedUser = selectedUser;

}

}

}

}

ПРИЛОЖЕНИЕ B - код unit-тестов

Функции MainViewModel.cs

[TestClass]

public class HotelTests

{

[TestMethod]

public void Test\_HashCode\_GetHashCode1\_xMpCOKC5I4INzFCab3WEmwreturned()

{

// arrange

string testString = "1";

string expected = "xMpCOKC5I4INzFCab3WEmw==";

// act

string actual = HashCode.GetHashCode(testString);

// assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void Test\_HashCode\_GetHashCodeEmpty\_Nullreturned()

{

// arrange

string testString = "";

string expected = null;

// act

string actual = HashCode.GetHashCode(testString);

// assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void Test\_HashCode\_GetHashCodeNull\_Nullreturned()

{

// arrange

string testString = null;

string expected = null;

// act

string actual = HashCode.GetHashCode(testString);

// assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void Test\_FileService\_BitmapImage\_To\_Byte\_Array\_ByteArrayreturned()

{

// arrange

string imagePath = Directory.GetCurrentDirectory() + "\\Images\\testImage.png";

byte[] expected = new byte[]

{

…

};

// act

byte[] actual = FileService.BitmapImageToByteArray(new BitmapImage(new Uri(imagePath)));

// assert

CollectionAssert.AreEqual(expected, actual);

}

[TestMethod]

public void Test\_PostitiveDecimalNumberVerificationRule\_Wrong\_Parametr\_Type\_Falsereturned()

{

// arrange

PostitiveDecimalNumberVerificationRule validation = new PostitiveDecimalNumberVerificationRule();

bool testParametr = true;

bool expectedValid = false;

object expectedErrorContent = "Неудалось преобразовать в строку.";

// act

bool actualValid = validation.Validate(testParametr, null).IsValid;

object actualErrorContent = validation.Validate(testParametr, null).ErrorContent;

// assert

Assert.AreEqual(expectedValid, actualValid);

Assert.AreEqual(expectedErrorContent, actualErrorContent);

}

[TestMethod]

public void Test\_PostitiveDecimalNumberVerificationRule\_Too\_Many\_Commas\_Falsereturned()

{

// arrange

PostitiveDecimalNumberVerificationRule validation = new PostitiveDecimalNumberVerificationRule();

string testParametr = "123,2354,24";

bool expectedValid = false;

object expectedErrorContent = "Неудалось преобразовать число. Удалите лишние запятые.";

// act

bool actualValid = validation.Validate(testParametr, null).IsValid;

object actualErrorContent = validation.Validate(testParametr, null).ErrorContent;

// assert

Assert.AreEqual(expectedValid, actualValid);

Assert.AreEqual(expectedErrorContent, actualErrorContent);

}

[TestMethod]

public void Test\_PostitiveDecimalNumberVerificationRule\_Not\_Decimal\_Parametr\_Falsereturned()

{

// arrange

PostitiveDecimalNumberVerificationRule validation = new PostitiveDecimalNumberVerificationRule();

string testParametr = "Один и три четверти";

bool expectedValid = false;

object expectedErrorContent = "Ожидалась дробное число.";

// act

bool actualValid = validation.Validate(testParametr, null).IsValid;

object actualErrorContent = validation.Validate(testParametr, null).ErrorContent;

// assert

Assert.AreEqual(expectedValid, actualValid);

Assert.AreEqual(expectedErrorContent, actualErrorContent);

}

[TestMethod]

public void Test\_PostitiveDecimalNumberVerificationRule\_Negative\_Decimal\_Parametr\_Falsereturned()

{

// arrange

PostitiveDecimalNumberVerificationRule validation = new PostitiveDecimalNumberVerificationRule();

string testParametr = "-124.42";

bool expectedValid = false;

object expectedErrorContent = "Число должно быть положительным.";

// act

bool actualValid = validation.Validate(testParametr, null).IsValid;

object actualErrorContent = validation.Validate(testParametr, null).ErrorContent;

// assert

Assert.AreEqual(expectedValid, actualValid);

Assert.AreEqual(expectedErrorContent, actualErrorContent);

}

[TestMethod]

public void Test\_PostitiveDecimalNumberVerificationRule\_Positive\_Decimal\_Parametr\_Truereturned()

{

// arrange

PostitiveDecimalNumberVerificationRule validation = new PostitiveDecimalNumberVerificationRule();

string testParametr = "24.12";

bool expectedValid = true;

object expectedErrorContent = null;

// act

bool actualValid = validation.Validate(testParametr, null).IsValid;

object actualErrorContent = validation.Validate(testParametr, null).ErrorContent;

// assert

Assert.AreEqual(expectedValid, actualValid);

Assert.AreEqual(expectedErrorContent, actualErrorContent);

}

}