МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Программной инженерии

Специальность 1-98 01 03 Программное обеспечение информационной безопасности мобильных систем

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

по дисциплине «Современные технологии программирования мобильных систем»

Тема «Спортивный клуб»

Исполнитель

студент 2 курса группы 8 Кусонский Тимофей Павлович

(Ф.И.О.)

Руководитель работы ассистент Годун А.В

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

Курсовой проект защищен с оценкой

Председатель Пацей Н.В

(подпись)

Минск 2021

Содержание

[ВВЕДЕНИЕ 3](#_Toc72443582)

[1. Аналитический обзор литературы 4](#_Toc72443583)

[2. Анализ требований к программному средству и разработка функциональных требований 9](#_Toc72443584)

[**2.1 Описание функциональности ПС** 9](#_Toc72443585)

[**2.2 Спецификация функциональных требований** 10](#_Toc72443586)

[3. Проектирование программного средства 11](#_Toc72443587)

[**3.1 Архитектура приложения** 11](#_Toc72443588)

[**3.2 Диаграмма классов** 12](#_Toc72443589)

[**3.3 Описание базы данных** 15](#_Toc72443590)

[4. Создание (реализация) программного средства. 17](#_Toc72443591)

[4.1 Реализация классов для работы с данными 17](#_Toc72443592)

[4.2 Шифрование паролей 17](#_Toc72443593)

[4.3 Создание пользователей 18](#_Toc72443594)

[4.4 Стили 18](#_Toc72443595)

[4.5 Контролы 19](#_Toc72443596)

[4.6 Старт формы логина до MainWindow 19](#_Toc72443597)

[5. Тестирование, проверка работоспособности и анализ полученных результатов. 20](#_Toc72443598)

[6. Руководство по установке и использованию 28](#_Toc72443599)

[**6.1 Установка приложения** 28](#_Toc72443600)

[**6.2** **Использование приложения.** 28](#_Toc72443601)

[ЗАКЛЮЧЕНИЕ 30](#_Toc72443602)

[Список использованных источников 31](#_Toc72443603)

[ПРИЛОЖЕНИЕ А 32](#_Toc72443604)

[ПРИЛОЖЕНИЕ Б 39](#_Toc72443605)

# ВВЕДЕНИЕ

Ресторанный бизнес становится все больше и больше и для его управления и упрощением прибегают к автоматизации. Приложения, которые считают выручку, считают зарплаты, которые выводят информацию о блюдах из-за которой официантам проще изучать меню, администраторам добавлять новые блюда, не боясь, что официанты забудут про него.

Везде есть человеческий фактор, однако автоматизация при добавлении блюд, оформлении резервации столиков снижает риск ошибки к минимуму.

Программное средство «Ресторатор» – это программное средство, которое предназначено для упрощения управления рестораном и упрощения поиска информации о блюдах и резервации работникам.

Приложение имеет 2 типа пользователей: администратор и пользователь. Они имеют разные полномочия. Пользователи могут просматривать информацию о блюдах, производить их поиск и просматривать информацию о зарезервированных столиках в определенный день. Администраторы имеют полные права. Они могут удалять и изменять пользователей, резервировать клиентам столики, добавлять, удалять и изменять пункты меню, писать отзывы о работниках своего заведения.

Также будет реализована авторизация всех пользователей. Это значит, что перед тем, как начать пользоваться приложением, нужно будет зарегистрироваться. Хранение всех пользователей будет осуществлено в базе данных, и при авторизации или регистрации будет непосредственное обращение к ней. Также вся информация о блюде и резервации будет хранится в базе данных.

В ходе использования этой программы, пользователи смогут изучить все блюда, их состав и стоимость, что облегчит им работу с клиентами ресторана.

# Аналитический обзор литературы

Выполнения этого курсового проекта не обошлось без использования дополнительной литературы. Таковой является информация на сайте <https://metanit.com/>. На этом сайте имеется много информации, которая помогла в разработке этого курсового проекта. Также использовалась информация с сайта https://ru.stackoverflow.com/.

Самый большое количество информации было получено с официального сайта <https://docs.microsoft.com/en-us/dotnet/csharp/>. Статьи с <https://habr.com/ru/> были также очень полезны.

В качестве аналога я выбрал систему автоматизации ресторана iiko он приближен к моей задумке. Краткий обзор этого сервиса представлен ниже. Я ее нашел на вебсайте: https://restorator-msk.ru/iikorms

Для работников ресторана доступен достаточно обширный функционал. Вот так выглядит главная страница (рисунок 1.1).

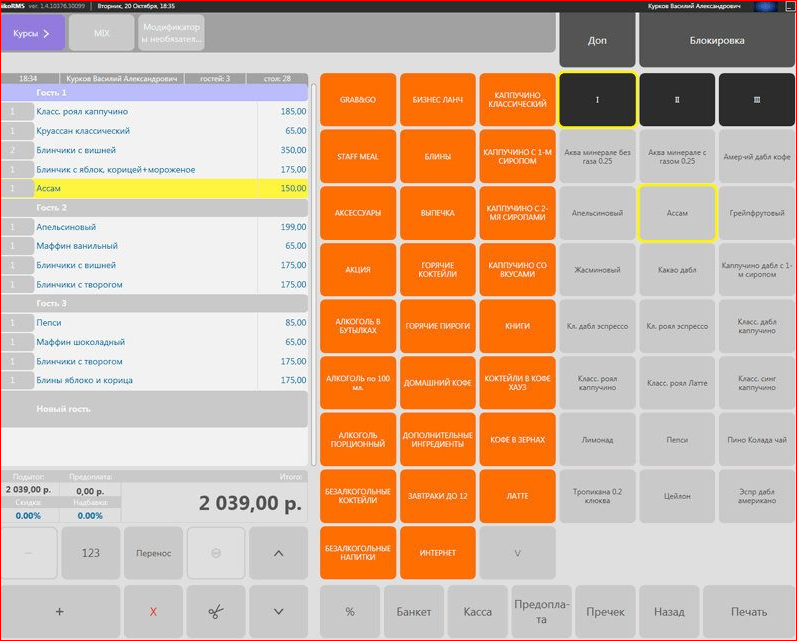


Рисунок 1.1 – Главное окно

Для более быстрого выбора определенной продукции ресторана, некоторые позиции расположены на главном окне (рисунок 1.2).



Рисунок 1.2 – Выбор продукции

Для резервирования столиков используется воссозданная модель ресторана в приложении, при нажатии на какое-либо место, оно загорается как выделенное и его можно зарезервировать (рисунок 1.3).

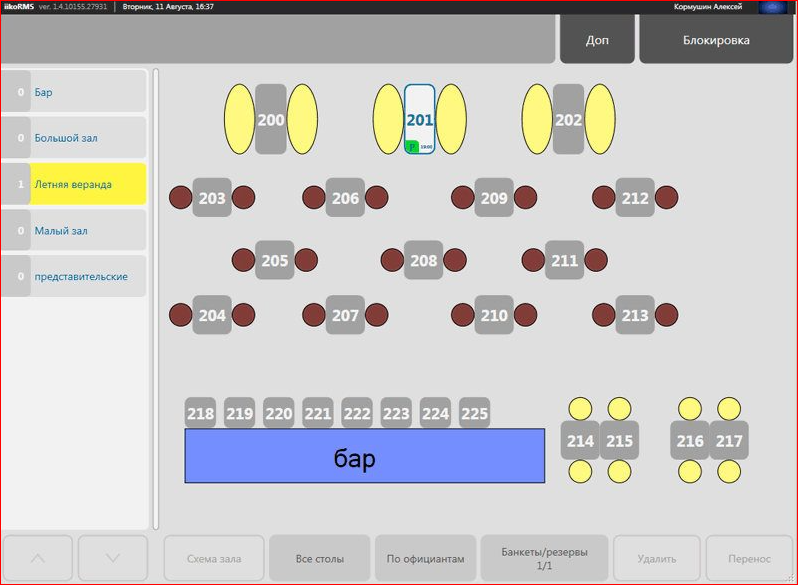


Рисунок 1.3 – модель ресторана

Также в приложении ведется учет товаров и их движения в режиме реального времени (рисунок 1.4).

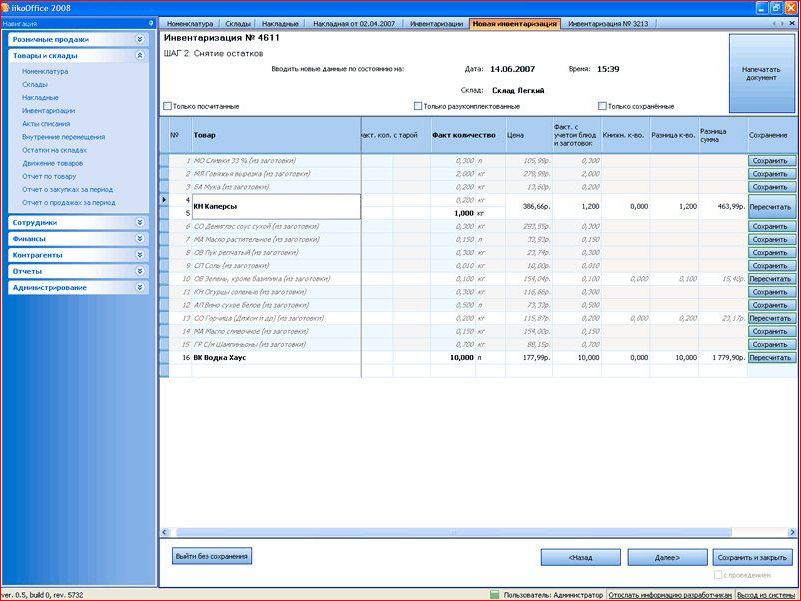


Рисунок 1.4 – Учет товаров

Еще одной из полезных функций данного приложения является автоматизация заработной платы в зависимости от часов работы, собственной выручки и контроль рабочего времени сотрудников. При помощи магнитных карт. За опоздание или неявку на работу штрафы начисляются автоматически (рисунок 1.5).

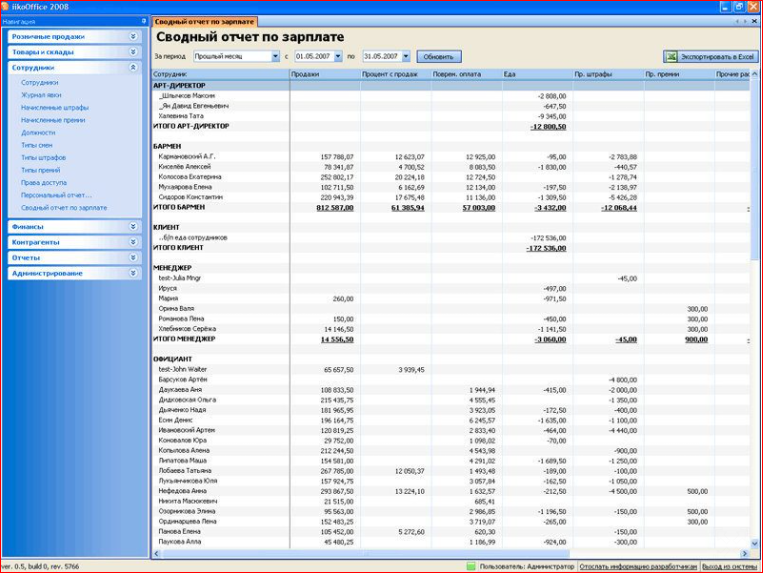


Рисунок 1.5 – Расчет заработной платы

Так же слева экрана присутствует удобное меню перемещения по приложению, можно просматривать информацию о сотрудниках, их должности, изменять права доступа (рисунок 1.6).

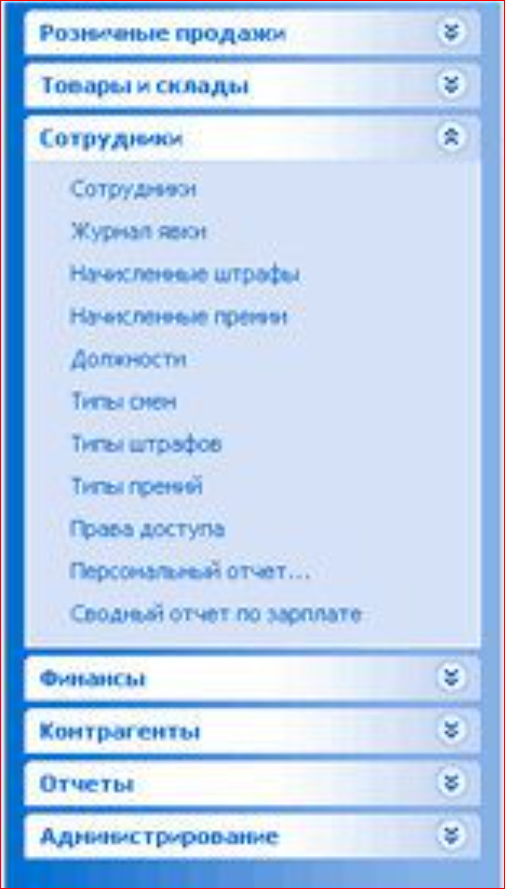


Рисунок 1.6 – Меню перемещения по приложению

# Анализ требований к программному средству и разработка функциональных требований

## **2.1 Описание функциональности ПС**

Основные возможности:

* возможность авторизации и регистрации;
* возможность смены аккаунта, без закрытия программы;
* возможность просмотра, изменения, удаления и добавления пунктов меню;
* возможность изменения мобильного телефона, электронной почты и роли;
* возможность смены аккаунта, без закрытия программы;
* возможность добавлять резервацию определенного столика;
* возможность поиска определенного пункта меню по названию;
* возможность написания отзывов о сотрудниках;
* возможность выборки резерваций по определенной дате.

Основная часть функций будет реализовывать работу с БД;

Ниже, предоставлена диаграмма использования ПС (Use Case)(рисунок 2.1).

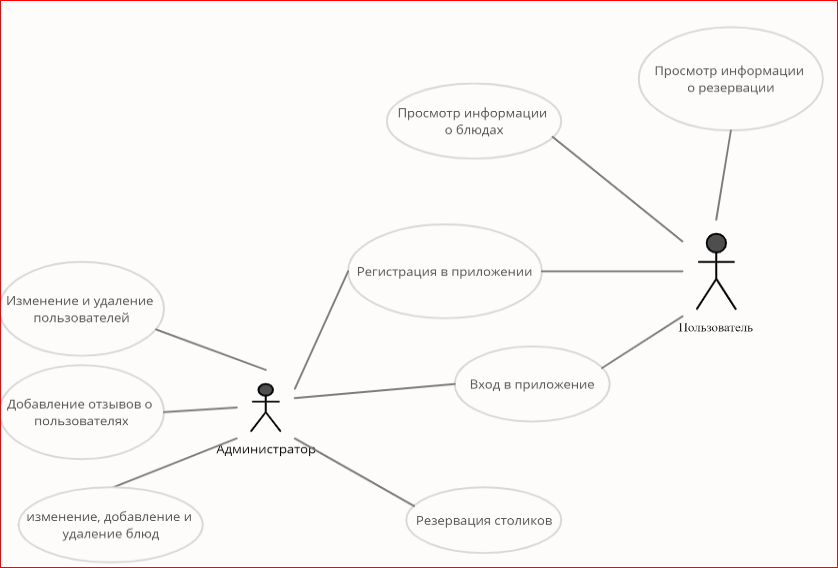


Рисунок 2.1 – «Диаграмма использования»

## **2.2 Спецификация функциональных требований**

1. Авторизация должна быть безопасна для пользователей, т.е. пароли будут храниться в БД в зашифрованном виде (Алгоритм шифрования MD5CryptoServiceProvider);
2. Функции, работающие с БД, не должны блокировать пользовательский интерфейс при отработке. Для этого требуется сделать их асинхронными;
3. При входе пользователя приложение должно пересчитать большое количество данных. Эти вычисления не должны блокировать пользовательский интерфейс при отработке.
4. Пользователь может вводить неверные данные. Из-за этого вычисления будут неверными. Существует необходимость в валидации введенных данных.

# Проектирование программного средства

## **3.1 Архитектура приложения**

Паттерн repository-service pattern-архитектурный паттерн, который использует интерфейсы, классы репозиториев и классы сервисов. Классы репозиториев реализуют интерфейсы репозиториев и предназначены для подключения и работы с базой данных, классы сервисов реализуют интерфейс сервисов и предназначены для использования методов классов репозиториев и в последующем вызове своих методов в программе. Данный паттерн мобильный, так как позволяет работать с некоторыми базами данных. И из-за использования интерфейсов и стороннего подключения паттерна к UI менять UI приложения. Позволяет легко менять реализацию интерфейсов, что помогает с легкостью обновлять приложение. Схема паттерна предоставлена на рисунке 3.1.

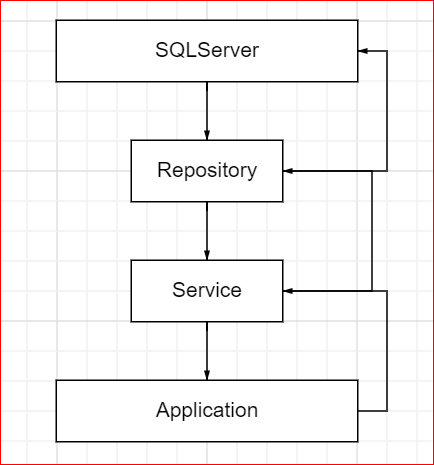


Рисунок 3.1 – «Схема паттерна repository-service pattern»

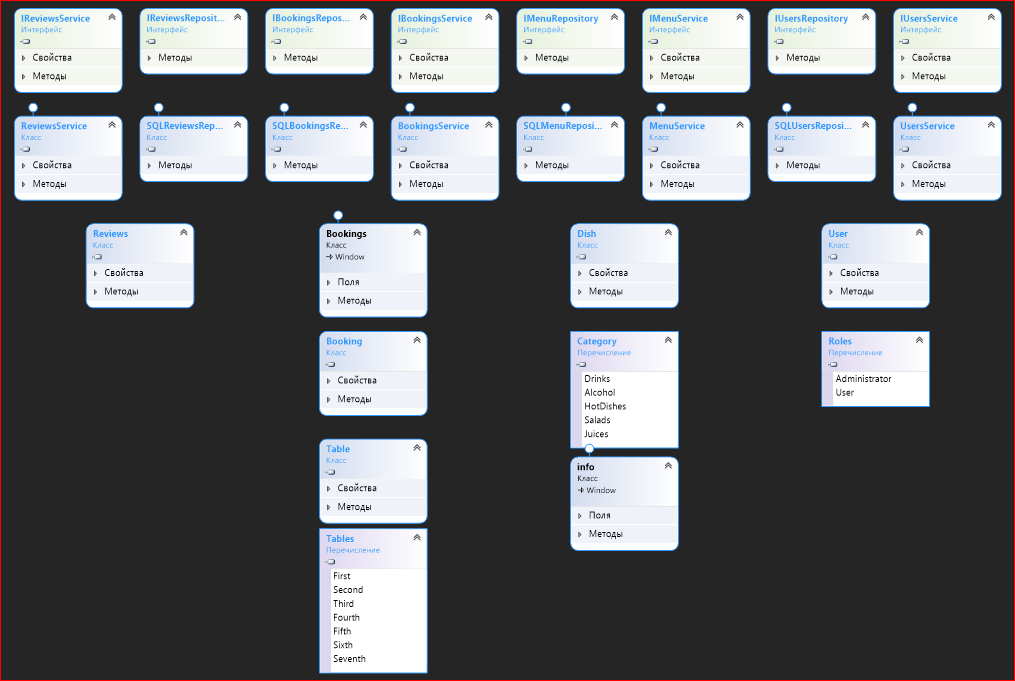


Рисунок 3.2 – «Диаграмма паттерна repository-service pattern»

## **3.2 Диаграмма классов**

Диаграмма классов предоставлена ниже (рисунок 3.3). Более точные диаграммы приведены в приложениях.

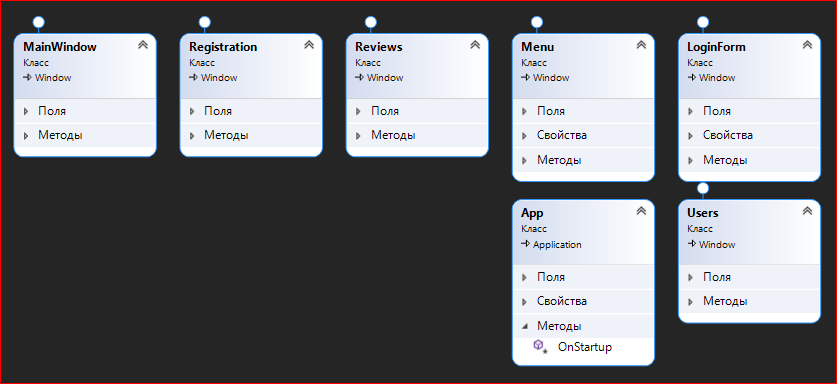


Рисунок 3.3 – «Диаграмма классов форм»

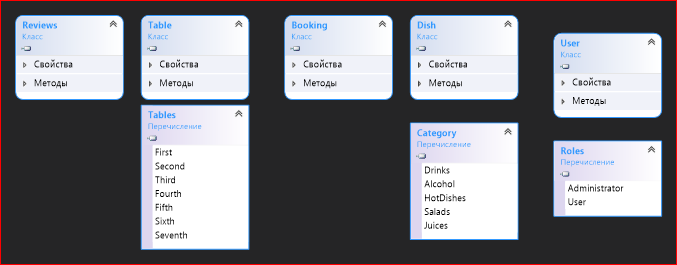


Рисунок 3.4 – «Диаграмма классов»

Основной класс – это класс User(Пользователь).

Свойства класса User:

* id – идентификационный номер пользователя;
* Name – логин пользователя;
* Phone – номер телефона пользователя;
* Email – электронная почта пользователя;
* Password- пароль пользователя;
* RoleId –идентификационный номер роли пользователя.

Перечисление Roles хранит в себе 2 роли:

* Administrator;
* User;

Класс Dish(блюдо).

Свойства класса Dish:

* id – идентификационный номер блюда;
* ProductName – Название блюда;
* Cost – стоимость блюда;
* Description – описание блюда(состав);
* Image - изображение блюда;
* UpdatedBy –содержит id администратора который добавил блюдо;
* UpdatedDate –содержит дату когда администратор добавил блюдо.

Перечисление Category хранит в себе 5 категорий:

* Drinks;
* Alcohol;
* HotDishes;
* Salads;
* Juices.

Класс Booking(бронирование).

Свойства класса Booking:

* id – идентификационный номер брони;
* ClientName – ФИО клиента;
* DataTimeBooking – дата на которую заказали столик;
* Seats– количество мест;
* TableId – id заказанного столика;
* ClientPhoneNumber –содержит номер телефона клиента;
* UpdatedBy – содержит id администратора который добавил бронь;
* Number\_of\_Table – содержит номер столика
* UpdatedDate- содержит дату когда администратор добавил бронь.

Перечисление Tables хранит в себе 7 столиков с названиями:

* First;
* Second;
* Third;
* Fourth;
* Fifth;
* Sixth;
* Seventh.

Класс Table(столик).

Свойства класса Table:

* id – идентификационный номер столика;
* Name – Название столика;
* TotalSeats – число свободных мест за столиком.

Класс Reviews(отзывы).

Свойства класса Reviews:

* id –идентификационный номер отзыва;
* CreatedBy – id администратора, который написал отзыв;
* Description – текст отзыва;
* RecepientId – id пользователя либо администратора, на кого написан отзыв;
* RecepientName – имя пользователя или администратора, на кого написан отзыв;
* CreatedByName- имя администратора, который написал отзыв;

## **3.3 Описание базы данных**

Диаграмма базы данных была разработана в программе Microsoft SQL Server Management Studio 18 и показана на рисунке 3.5.Для реализации функционала и удобной разработки было создано 7 таблиц: Users, которая хранит информацию о всех пользователях; Roles, содержит роли пользователей; Menu, содержит информацию о меню; Categories, содержит категории блюд; Reviews содержит отзывы о пользователях; Bookings, содержит информацию о бронировании; Tables, содержит все столики ресторана.

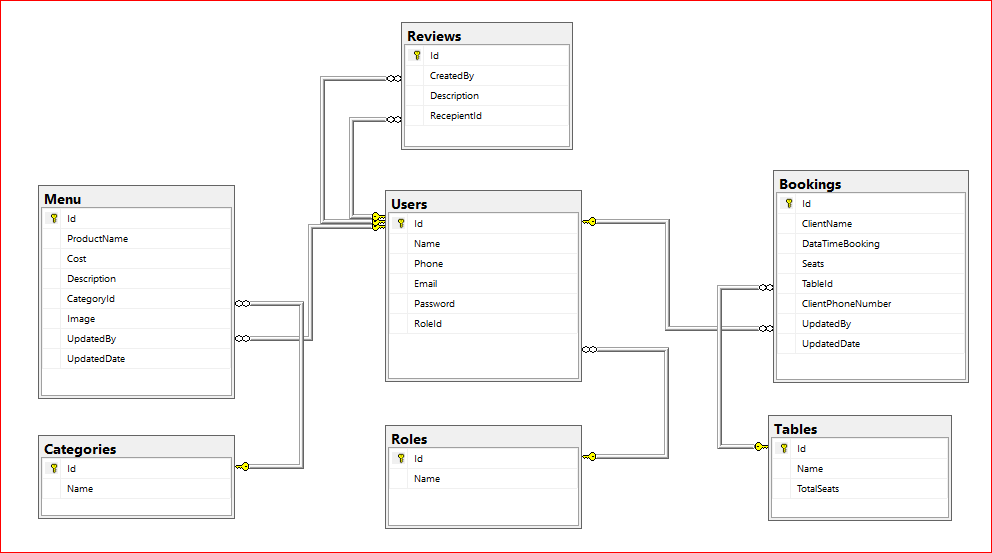


Рисунок 3.5 – «Диаграмма базы данных»

Таблица Users состоит из 6 столбцов:

* Идентификатор id (первичный ключ);
* Логин пользователя Name;
* Мобильный телефон пользователя Phone;
* Электронная почта пользователя Email;
* Хешированный пароль Password;
* Id роли RoleId(вторичный ключ);

Таблица Roles состоит из 2 столбцов:

* Идентификатор id (первичный ключ);
* Имя роли Name;

Таблица Menu состоит из 8 столбцов:

* Идентификатор id (первичный ключ);
* Название блюда ProductName;
* Стоимость блюда Cost;
* Описание блюда(состав) Description;
* Id категорий блюд CategoryId(вторичный ключ);
* Изображение блюда Image;
* Id пользователя, который добавил блюдо UpdatedBy(вторичный ключ);
* Дата создания блюда UpdatedDate;

Таблица Categories состоит из 2 столбцов:

* Идентификатор id (первичный ключ);
* Название категории Name;

Таблица Reviews состоит из 4 столбцов:

* Идентификатор id (первичный ключ);
* Id администратора, который написал отзыв CreatedBy(вторичный ключ);
* Текст отзыва Description;
* Id пользователя, на которого писался отзыв RecepientI(вторичный ключ);

Таблица Bookings состоит из 8 столбцов:

* Идентификатор id (первичный ключ);
* Имя клиента ClientName;
* Дата, на которую забронирован столик DataTimeBooking;
* Количество мест Seats;
* Id столика TableId(вторичный ключ);
* Номер телефона клиента ClientPhoneNumber;
* Id администратора, который добавил бронь UpdatedBy(вторичный ключ);
* Дата создания брони UpdatedDate;

Таблица Tables состоит из 3 столбцов:

* Идентификатор id (первичный ключ);
* Название столика Name;
* Общее число мест за столиком TotalSeats;

Идентификаторы во всех таблицах генерируются автоматически (identity).

# Создание (реализация) программного средства.

4.1 Реализация классов для работы с данными

Связь программного средства с базой данных реализована при помощи технологии ADO.NET. Классы осуществляющие работу с базой данных находятся в папке Repositories и содержат строку подключения к базе данных, работа осуществляется с помощью sql запросов.



Рисунок 4.1 – «Строка подключения к базе данных»

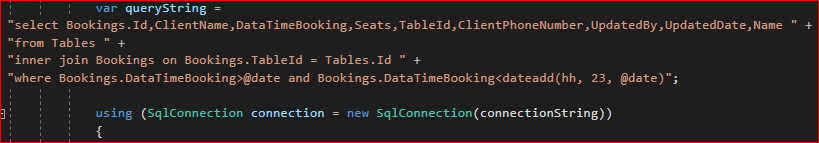


Рисунок 4.2 – «Строка sql запроса»

Вся информация для работы с базой данных из сервисов передается в репозитории, после чего в репозиториях осуществляется поиск по этой информации в базе данных и новые данные переходят обратно в сервисы.

4.2 Шифрование паролей

Для безопасности данных пользователя в базе данных пароль имеет зашифрованный вид. При сравнении паролей введенный пользователем пароль кэшируется и в зашифрованном виде сравнивается с паролем, который храниться в базе данных.

Для хеширования пароля я разработал метод GetPasswordHash (рисунок 4.3), который принимает значение пароля и хэширует, на выходи получается значение, показанное на рисунке 4.4.

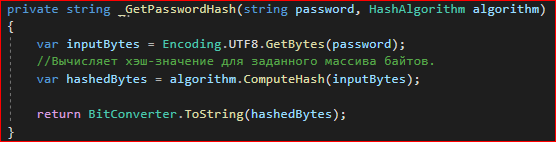


Рисунок 4.3 – «Получение хэша пароля»



Рисунок 4.4 – «Полученный хэш»



Рисунок 4.5 – «Вызов метода хеширования»



Рисунок 4.6 – «Выбор алгоритма хэширования»

4.3 Создание пользователей

Регистрация осуществляется вводом определенных данных на форме, которая передает их в класс UserService, который вызывает метод класса SQLUsersRepository, этот метод принимает данные записывает их в переменные и передает в sql запрос, который производит создание пользователей в базе данных.



Рисунок 4.7 – «Код создания нового пользователя в базе данных»

4.4 Стили

Существуют библиотеки стилей, например MaterialDesign , но мне его дизайн не понравился и я в интернете нашел готовые файлы со стилями(<https://professorweb.ru/my/WPF/themes/DarkBlue_UI/theme_db.php>), которые я подключил в свой проект в папку Resourses.



Рисунок 4.8 – «Стили для проекта»

4.5 Контролы

Для реализации раздельного выбора даты времени я подключил контрол, который называется Extended WPF Toolkit, на данный момент его популярность упала, однако раньше он был достаточно популярен. ( [GitHub - xceedsoftware/wpftoolkit: All the controls missing in WPF. Over 1 million downloads.](https://github.com/xceedsoftware/wpftoolkit)).

На рисунке 4.9 изображено использование таймпикера из пакета контролов ectk.



Рисунок 4.9 – «TimePicker»



Рисунок 4.10 – «TimePicker в UI»

4.6 Старт формы логина до MainWindow

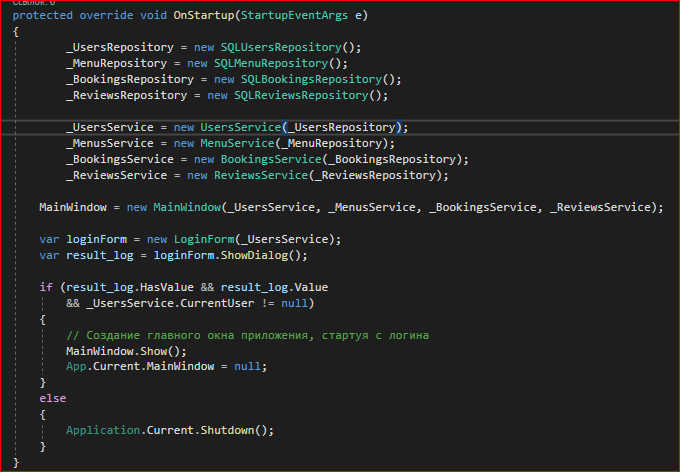
Запуск логин формы осуществляется в файле App.xaml (рисунок 4.11). Для этого во время запуска приложения мы открываем форму логина и при верном входе в аккаунт вызываем главное окно приложения, после чего форма авторизации закрывается. Однако при закрытии формы авторизации приложение полностью завершит работу.

Рисунок 4.11 – «App.xaml»

# Тестирование, проверка работоспособности и анализ полученных результатов.

При тестировании приложения, были применены сценарии, которые могли бы привести к ошибке. В этой главе мы рассмотрим эти сценарии и посмотрим на их обработку.

В момент регистрации, возможна такая ситуация, в которой пользователь ничего не заполнил все поля. Обработка данного сценария представлена на рисунке 5.1.

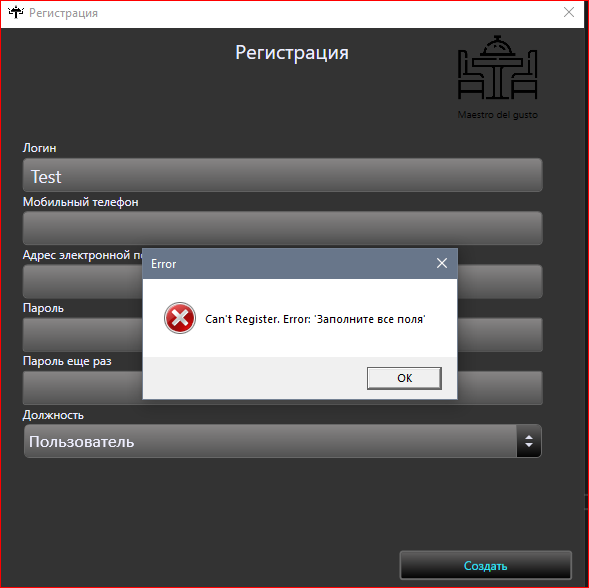


Рисунок 5.1 – «Ошибка отсутствия в некоторых полях информации»

Так же можно ошибиться с вводом мобильного телефона и электронной почты, например, не дописав нужное количество символов. Обработка этого сценария предоставлена на рисунке 5.2 и 5.3.

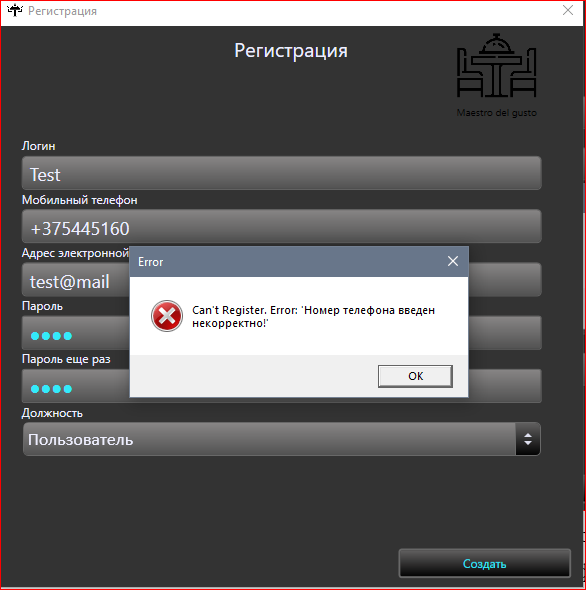


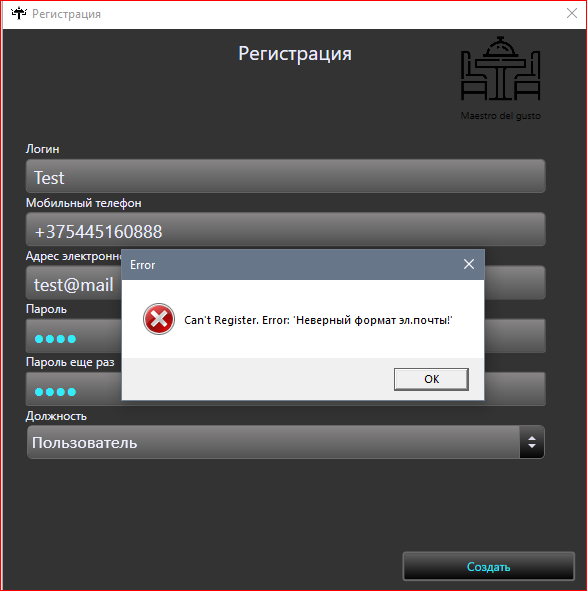
Рисунок 5.2 – «Ошибка некорректного ввода мобильного телефона»

Рисунок 5.3 – «Ошибка некорректного ввода электронной почты»

Так же стоит блокировка на регистрацию больше одно администратора, при попытке создания нового пользователя с ролью “Администратор” выдается исключение. Обработка этого сценария предоставлена на рисунке 5.4.

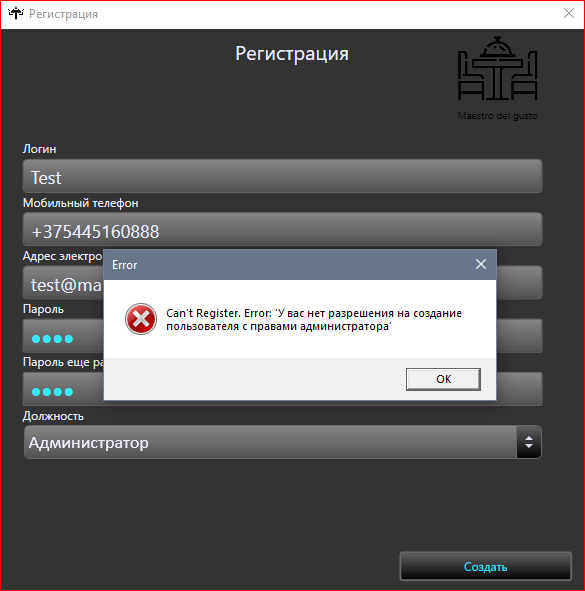


Рисунок 5.4 – «Ошибка некорректного ввода мобильного телефона»

При попытке создания аккаунта с логином, который уже используется другим пользователем, выдается исключение. Обработка этого сценария предоставлена на рисунке 5.5.

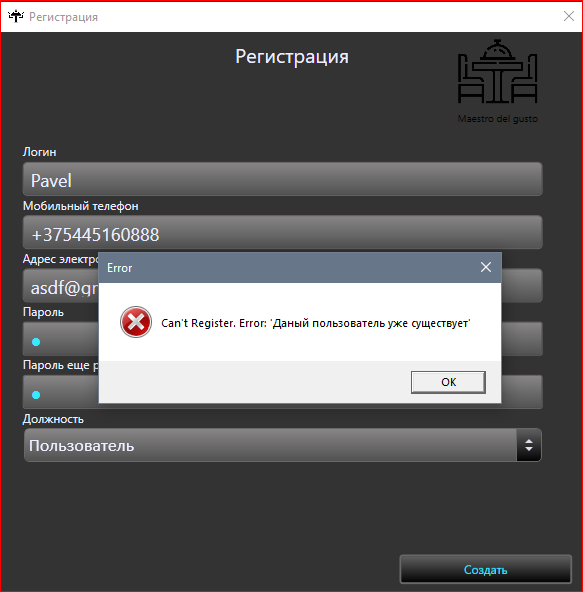


Рисунок 5.5 – «Ошибка создания пользователя с логином, который уже занят»

При попытке войти в несуществующий аккаунт, выдается исключение это видно на рисунке 5.6.

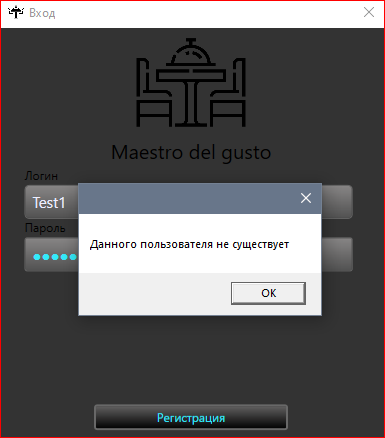


Рисунок 5.6 – «Ошибка входа в несуществующий аккаунт»

При вводе неверного пароля при входе, выдается исключение, это видно на рисунке 5.7

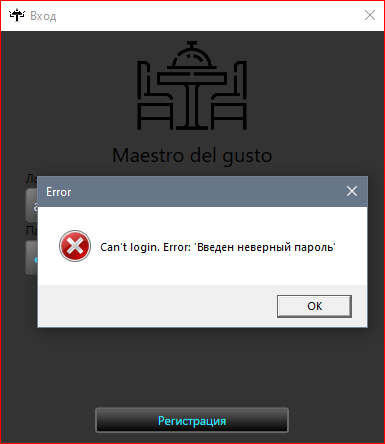


Рисунок 5.7 – «Ошибка неверный пароль»

В момент добавления определенного пункта меню администратором, он может случайно попытаться добавить уже существующий пункт меню, программа не даст ему это сделать и выдаст ошибку, это показано на рисунке 5.8.

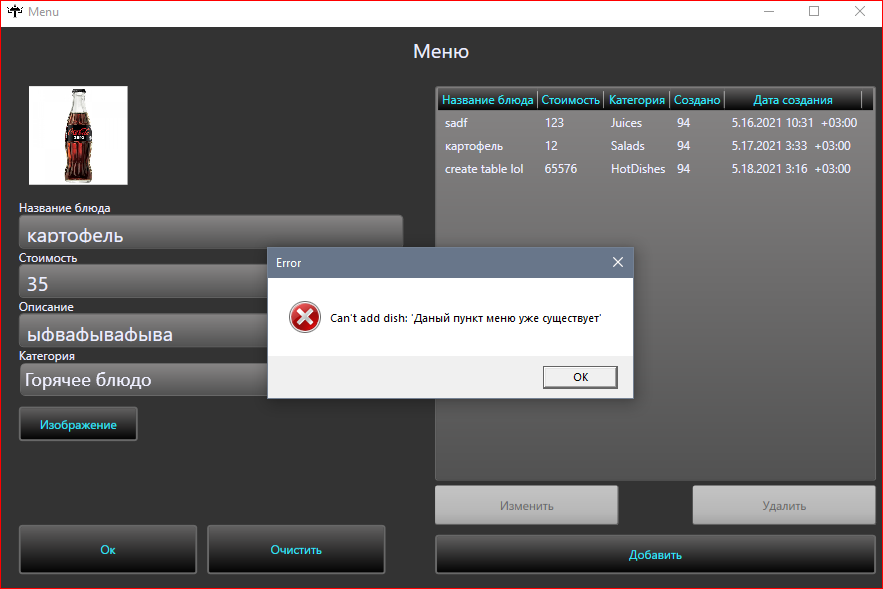


Рисунок 5.8 – «Ошибка добавления пункта меню, который уже существует»

При попытке зарезервировать больше мест чем есть свободных мест за столиком будет выдано исключение, это показано на рисунке 5.9.

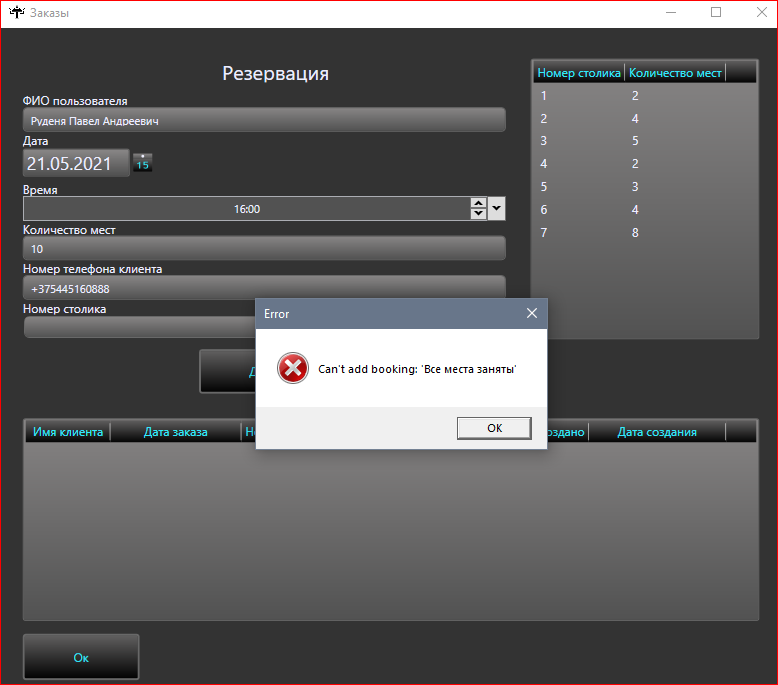


Рисунок 5.9 – «Ошибка попытки зарезервировать количество мест больше чем есть свободных за столиком»

# Руководство по установке и использованию

## **6.1 Установка приложения**

Установите MicrosoftSQLServer по ссылке:. [SQL Server Downloads | Microsoft](https://www.microsoft.com/en-us/sql-server/sql-server-downloads)(рисунок 6.1)

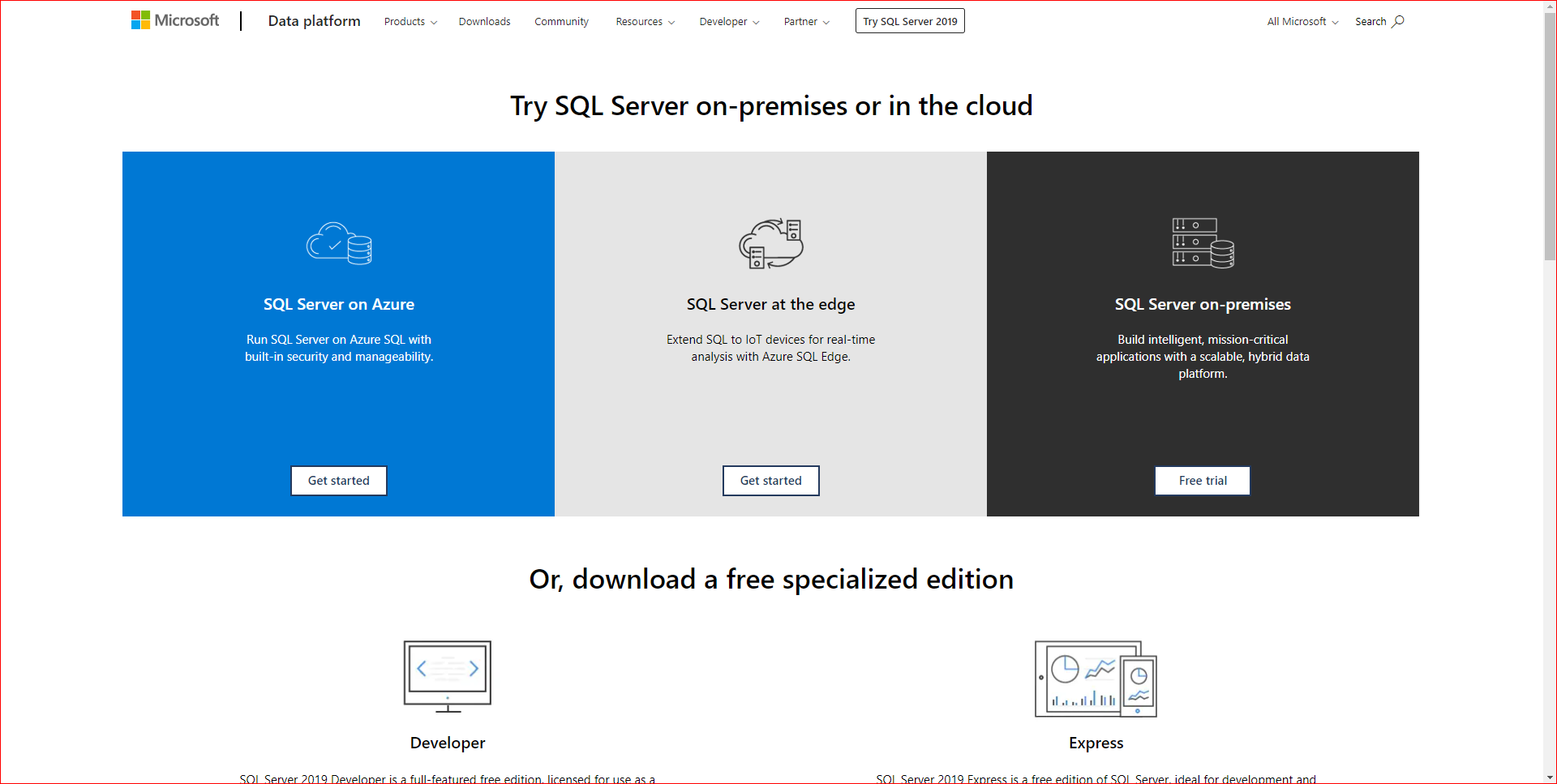


Рисунок 6.1 – «Установка MicrosoftSQLServer»

Для установки программного средства скачайте архив с GitHub <https://github.com/SelDanilEv/EconoMiC_installer>.

Распакуйте его в выбранную вами папку. После распаковки запустите файл server.sql(рисунок 6.2)

В MicrofotSQLServer нажмите Execute для создания базы данных приложения.

Далее запустите Kusinskiy\_Timophey.exe.

## **Использование приложения.**

При запуске приложения вы попадете на окно входа. Если у вас еще нет аккаунта, то перейдите во вкладку регистрации. Зарегистрируйтесь. После регистрации откроется окно входа, войдите в свой аккаунт. Если введенные данные были правильными, то после нажатия кнопки входа откроется главное окно.

Основные элементы главного окна показаны на рисунке 6.3.

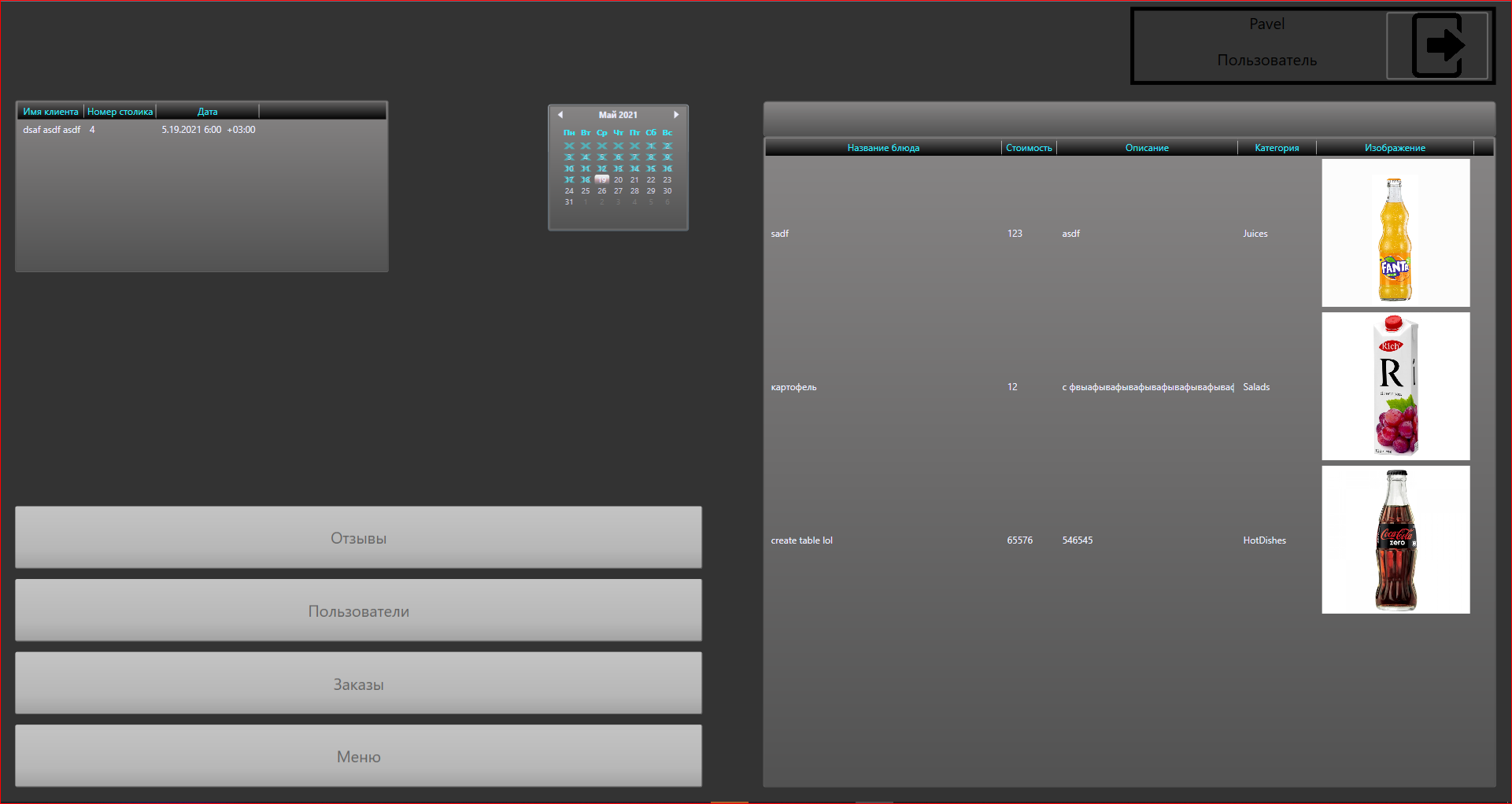


Рис. 6.3 – «Основные элементы»

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта были рассмотрены основные аналоги программного средства, их преимущества и недостатки, были изучены их возможности.

В ходе написания курсового проекта были использованы паттерны:

* repository;
* unit of work;

Так же была изучена технология ADO.NET. Использован и доработан стиль DarkBlue UI и применен алгоритм хеширования MD5.

При разработке проекта был изучен и использован паттерн repository-service pattern.

В программном средстве был реализован следующий функционал:

* + Поддерживать работу c базой данных;
  + Добавление блюда в меню;
  + Выполнять поисковые запросы;
  + Выполнять регистрацию и авторизацию;
  + Просматривать информацию о наличии мест;
  + Просматривать информацию о составе блюда;
  + Оставлять отзывы.

В рамках курсового проекта было разработано программное средство «Ресторатор)», которое соответствует поставленным задачам.

Список использованных источников

1. Сайт о программировании Metanit: Руководство по WPF;
2. Сайт о программировании Microsoft.com: c# documentation;
3. Сайт о программировании Habr: Загрузка и обработка изображений в .NET Core;
4. Сайт о автоматизации ресторанов restorator-msk: iiko-система автоматизации ресторана;
5. Сайт о программировании professorweb: Стиль DarkBlue UI;
6. Сайт github;

ПРИЛОЖЕНИЕ А

Листинг 1. Код класса UsersService

public class UsersService : IUsersService

{

private readonly IUsersRepository \_UsersRepository;

private User \_CurrenrtUser;

private HashAlgorithm \_HashAlgorithm = new MD5CryptoServiceProvider();

public User CurrentUser

{

get

{

return \_CurrenrtUser;

}

}

public UsersService(IUsersRepository usersRepository)

{

\_UsersRepository = usersRepository;

}

/// <summary>

/// Регистрация

/// </summary>

/// <param name="name"> Имя</param>

/// <param name="phone"> Номер телефона</param>

/// <param name="email"> Емэйл </param>

/// <param name="password"> Пароль </param>

/// <param name="roleId"> Роль </param>

/// <returns></returns>

public int? Create(

string name,

string phone,

string email,

string password,

int roleId)

{

//Проверка роли

if (\_UsersRepository.IsUserExists(name))

{

throw new Exception("Даный пользователь уже существует");

}

//Проверка phone\_number

if (!Regex.Match(phone, @"^(\+[3][7][5][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9])\*$").Success)

{

throw new Exception("Номер телефона введен некорректно!");

}

//Проверка email

var emailPattern = @"^\w+([-+.']\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*$";

if (!Regex.IsMatch(email, emailPattern))

{

throw new Exception("Неверный формат эл.почты!");

}

//Проверка роли

if (roleId == 1

&& \_UsersRepository.IsAdminExists())

{

throw new Exception("У вас нет разрешения на создание пользователя с правами администратора");

}

password = \_GetPasswordHash(password, \_HashAlgorithm);

return \_UsersRepository.Create(name, phone, email, password, roleId);

}

private string \_GetPasswordHash(string password, HashAlgorithm algorithm)

{

var inputBytes = Encoding.UTF8.GetBytes(password);

//Вычисляет хэш-значение для заданного массива байтов.

var hashedBytes = algorithm.ComputeHash(inputBytes);

return BitConverter.ToString(hashedBytes);

}

/// <summary>

/// Удаление пользователей

/// </summary>

/// <param name="userId"> ID пользователя</param>

/// <returns></returns>

public bool Delete(int userId)

{

if (userId == CurrentUser.Id)

{

throw new Exception("Вы не можете удалить самого себя");

}

return \_UsersRepository.Delete(userId);

}

/// <summary>

/// Возврат пользователя по userID

/// </summary>

/// <param name="userId">ID пользователя</param>

/// <returns></returns>

public User Get(int userId)

{

return new User();

}

/// <summary>

/// Список пользователей

/// </summary>

/// <returns></returns>

public ObservableCollection<User> Get()

{

ObservableCollection<User> list = new ObservableCollection<User>(\_UsersRepository.Get());

return list;

}

/// <summary>

/// Авторизация

/// </summary>

/// <param name="name"> Имя пользователя </param>

/// <param name="password"> Пароль </param>

/// <returns></returns>

public bool Login(string name, string password)

{

var user = \_UsersRepository.Get(name);

if (user !=null)

{

password = \_GetPasswordHash(password, \_HashAlgorithm);

if (user.Password == password)

{

\_CurrenrtUser = user;

return true;

}

else

{

throw new Exception("Введен неверный пароль");

}

}

return false;

}

/// <summary>

/// Изменение пользователей

/// </summary>

/// <param name="user"> Обьект класса User </param>

/// <returns></returns>

public bool Update(User user)

{

//Проверка phone\_number

if (!Regex.Match(user.Phone, @"^(\+[3][7][5][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9])\*$").Success)

{

throw new Exception("Номер телефона введен некорректно!");

}

//Проверка email

var emailPattern = @"^\w+([-+.']\w+)\*@\w+([-.]\w+)\*\.\w+([-.]\w+)\*$";

if (!Regex.IsMatch(user.Email, emailPattern))

{

throw new Exception("Неверный формат эл.почты!");

}

if (user.Id==CurrentUser.Id)

{

throw new Exception("Вы не можете изменять самого себя");

}

else

{

return \_UsersRepository.Update(user);

}

}

}

Листинг 2. Код класса SQLUsersRepository

public class SQLUsersRepository : IUsersRepository

{

string connectionString =

"Data Source=(local);Initial Catalog=Restaurateur;"

+ "Integrated Security=true";//Ссылка на подключение базы данных

//Регистрация

public int? Create(

string name,

string phone,

string email,

string password,

int roleId)

{

int? userId = null;

var queryString =

"INSERT INTO [Users] ([Name] ,[Phone] ,[Email] ,[Password] ,[RoleId]) OUTPUT Inserted.ID VALUES (@Name ,@Phone,@Email,@Password,@RoleId)";//sql запрос

// Создание и открытие соединения в блоке using.

using (SqlConnection connection =

new SqlConnection(connectionString))

{

//Создание объеков команд и параметров.

using (var command = new SqlCommand(queryString, connection))

{

command.Parameters.AddWithValue("@Name", name);

command.Parameters.AddWithValue("@Phone", phone);

command.Parameters.AddWithValue("@Email", email);

command.Parameters.AddWithValue("@Password", password);

command.Parameters.AddWithValue("@RoleId", roleId);

//Открытие подключения

connection.Open();

userId = (int?)command.ExecuteScalar();

}

}

return userId;

}

public bool Delete(int userId)//Удаление пользователей

{

var queryString = "Delete from Users where id=@id";//sql запрос

using (SqlConnection connection = new SqlConnection(connectionString))

{

// Create the Command and Parameter objects.

var command = new SqlCommand(queryString, connection);

command.Parameters.AddWithValue("@id", userId);

connection.Open();

var reader = command.ExecuteReader();

reader.Read();

}

return true;

}

public User Get(int userId)

{

return new User();

}

public ObservableCollection<User> Get()//Создание списка пользователей

{

User user = null;

var list = new ObservableCollection<User>();

var queryString = "select id,name,Password,Phone,email,RoleId from Users";

using (SqlConnection connection = new SqlConnection(connectionString))

{

// Create the Command and Parameter objects.

var command = new SqlCommand(queryString, connection);

// Создание объеков команд и параметров.

connection.Open();

using (var reader = command.ExecuteReader())

{

while (reader.Read())

{

object dbVal = null;

user = new User();

user.Id = (int)reader.GetValue(0);

dbVal = reader.GetValue(1);

if (!(dbVal is DBNull))

{

user.Name = (dbVal as string).Trim();

}

dbVal = (string)reader.GetValue(2);

if (!(dbVal is DBNull))

{

user.Password = (dbVal as string).Trim();

}

dbVal = (string)reader.GetValue(3);

if (!(dbVal is DBNull))

{

user.Phone = (dbVal as string).Trim();

}

dbVal = (string)reader.GetValue(4);

if (!(dbVal is DBNull))

{

user.Email = (dbVal as string).Trim();

}

user.RoleId = (Roles)reader.GetValue(5);

list.Add(user);

}

}

}

return list;

}

public User Get(string name)//Вход

{

User user = null;

var queryString = "select top 1 id,Name,Password,Phone,Email,RoleId from Users where name=@name";//sql запрос

// Создание и открытие соединения в блоке using.

using (SqlConnection connection = new SqlConnection(connectionString))

{

var command = new SqlCommand(queryString, connection);

command.Parameters.AddWithValue("@name", name);

// Создание объеков команд и параметров.

connection.Open();

using(var reader = command.ExecuteReader())

{

if(reader.Read())

{

object dbVal = null;

user = new User();

user.Id = (int)reader.GetValue(0);

dbVal = reader.GetValue(1);

if(!(dbVal is DBNull))

{

user.Name = (dbVal as string).Trim();

}

dbVal = (string)reader.GetValue(2);

if (!(dbVal is DBNull))

{

user.Password = (dbVal as string).Trim();

}

dbVal = (string)reader.GetValue(3);

if (!(dbVal is DBNull))

{

user.Phone = (dbVal as string).Trim();

}

dbVal = (string)reader.GetValue(4);

if (!(dbVal is DBNull))

{

user.Email = (dbVal as string).Trim();

}

user.RoleId = (Roles)reader.GetValue(5);

}

}

}

return user;

}

public bool IsAdminExists()

{

var queryString = "select top (1) Users.Id from Users inner join Roles on Roles.Id = Users.RoleId where Roles.Name = 'Administrator'";//sql запрос

// Создание и открытие соединения в блоке using.

using (SqlConnection connection = new SqlConnection(connectionString))

{

// Create the Command and Parameter objects.

var command = new SqlCommand(queryString, connection);

// Создание объеков команд и параметров.

connection.Open();

using (var reader = command.ExecuteReader())

{

if (reader.Read())

{

var idVal = reader.GetValue(0);

return !((idVal is DBNull)

|| (int)idVal < 0);

}

}

}

return false;

}

public bool IsUserExists(string userName)

{

var queryString = "select top (1) Users.Name from Users where Users.Name = @Name";//sql запрос

// Создание и открытие соединения в блоке using.

using (SqlConnection connection = new SqlConnection(connectionString))

{

// Create the Command and Parameter objects.

var command = new SqlCommand(queryString, connection);

command.Parameters.AddWithValue("@Name", userName);

connection.Open();

using (var reader = command.ExecuteReader())

{

if (reader.Read())

{

string idVal = (string)reader.GetValue(0);

return userName == idVal.Trim();

}

}

}

return false;

}

public bool Update(User user)//Обновление

{

var queryString = "Update Users Set Phone = @Phone, Email = @Email, RoleId = @RoleId where id = @id";//sql запрос

using (SqlConnection connection = new SqlConnection(connectionString))

{

// Create the Command and Parameter objects.

var command = new SqlCommand(queryString, connection);

command.Parameters.AddWithValue("@id", user.Id);

command.Parameters.AddWithValue("@Phone", user.Phone);

command.Parameters.AddWithValue("@Email", user.Email);

command.Parameters.AddWithValue("@RoleId", user.RoleId);

connection.Open();

var reader = command.ExecuteReader();

reader.Read();

}

return true;

}

}

ПРИЛОЖЕНИЕ Б

Диаграмма размещения

