Курсовая работа

По дисциплине «Разработка программных приложений»

на тему «Система учета заказов и их выполнение в строительной фирме (ремонт квартир)»

Выполнил студент группы ИД-401кис

Набиев А.И.

Руководитель : Суздалев А.С.

Челябинск

2022

# Введение

В настоящее время большое распространение в сферах деятельности современных предприятий получили различные реализации баз данных. Это обосновано тем, что данный раздел информационных технологий имеет значительную степень применимости в практических ситуациях и довольно просто внедряется в каждом конкретном случае. К тому же, в последние годы существует множество всевозможных реализаций баз данных (БД) и систем управления базами данных (СУБД), что даёт возможность для стратегического варьирования и подбора наилучшего варианта, как для узкоспециализированного, так и для широко специализированного предприятия либо подразделения.

Проанализировав данную ситуацию, можно сказать, что на рынке нет строительной компании, подрядчиков или бригад по ремонту квартир, которые не нуждаются в учете данных. Заказы, клиенты, услуги, договоры – всё должно храниться в структурированной базе данных. А еще лучше, чтобы сотрудники предприятия имели возможность вести этот учет в простом и удобном приложении.

# Основная часть

Цель работы: спроектировать базу данных и разработать клиентское приложение по ее управлению.

Задачи работы:

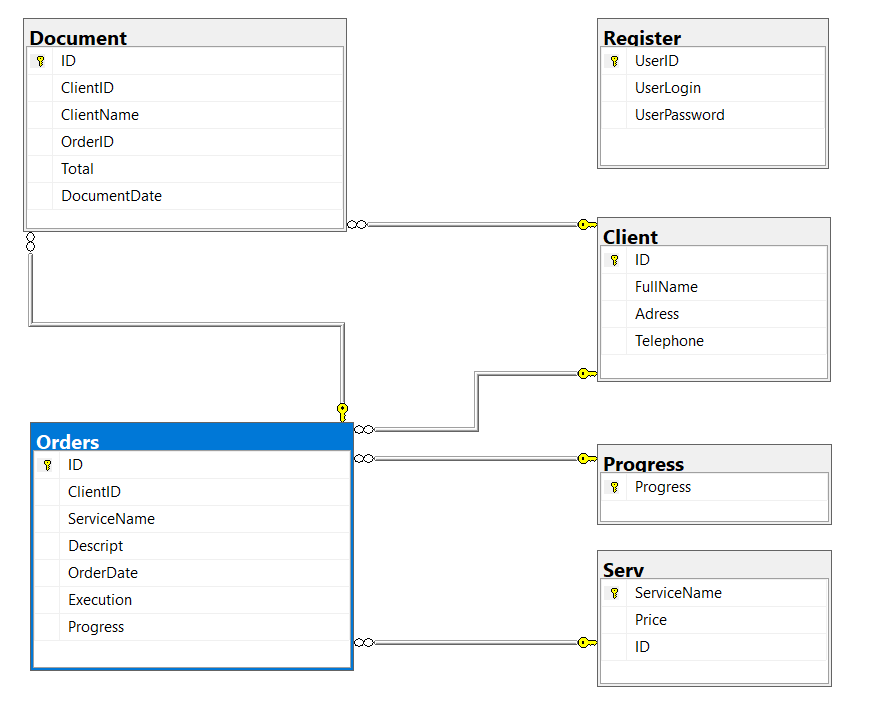
* Продумать алгоритм создания базы данных;
* Создание базы данных для приложения;
* Создание приложения на C#;
* Протестировать созданную программу;

Ресурс работы: Microsoft SQL Server Management Studio, интегрированная среда разработки программного обеспечения Visual Studio.

Практическая значимость работы:

Созданное приложение позволит вести учет заказов в строительной компании или же компании по ремонтам внутри помещения.

# База данных

****

## Таблица «Register»

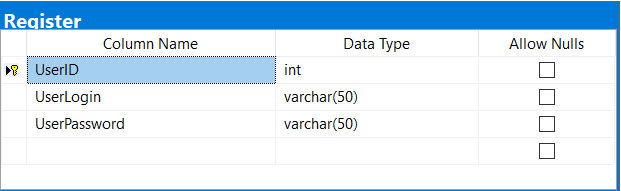
****

Таблица «Register» служит для обеспечения доступа к клиентскому приложению, а также регистрации новых пользователей. Содержит поля:

* «UserID»: первичный ключ таблицы, с автоинкрементном от 1 с шагом 1, является обязательным, тип данных: целое число, 4 байта;
* «UserLogin»: логин пользователя, является обязательным, тип данных: строка переменной длины до 50 символов;
* «UserPassword»: пароль пользователя, является обязательным, тип данных: строка переменной длины до 50 символов;

## Таблица «Progress»

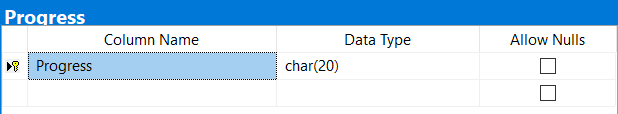
****

Таблица «Progress» служит как переменная о процессе выполнения для таблицы заказов, и содержит одно поле:

* «Progress»: первичный ключ таблицы, является обязательным, тип данных: строка фиксированной длинны до 20 символов;

## Таблица «Client»

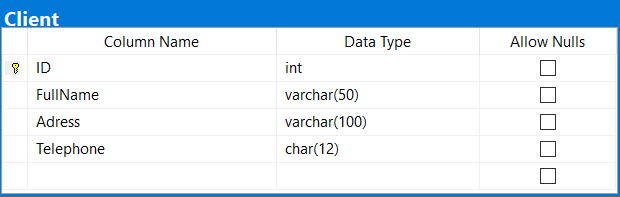
****

Таблица «Client» хранит информацию о клиентах и содержит следующие поля:

* «ID»: первичный ключ таблицы, с автоинкрементном от 1 с шагом 1, является обязательным, тип данных: целое число, 4 байта;
* «FullName»: ФИО клиента, является обязательным, тип данных: строка переменной длины до 50 символов;
* «Adress»: адрес клиента (объекта, на котором будут производиться работы), тип данных: строка переменной длины до 100 символов;
* «Telephone»: номер телефона клиента, тип данных: строка фиксированной длинны 12 символов;

## Таблица «Serv»

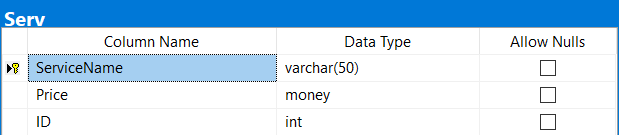
****

Таблица «Serv» является перечнем услуг компании и содержит следующие поля:

* «ServiceName»: название услуги, первичный ключ таблицы, является обязательным, тип данных: строка переменной длины до 50 символов;
* «Price»: цена услуги, является обязательным, тип данных денежный.
* «ID»: идентификатор услуги, потребовался для работы методов в приложении, автоинкремент от 1 с шагом 1, является обязательным, тип данных: целое число, 4 байта;

## Таблица «Orders»

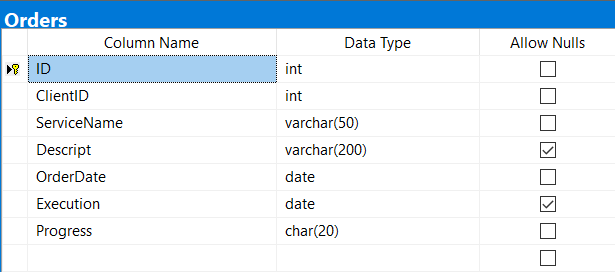
****

Таблица «Orders» представляет собой информацию о заказах и содержит следующие поля:

* «ID»: первичный ключ таблицы, с автоинкрементном от 1 с шагом 1, является обязательным, тип данных: целое число, 4 байта;
* «ClientID»: внешний ключ, ссылается на идентификатор клиента, является обязательным, тип данных: целое число, 4 байта;
* «ServiceName»: внешний ключ название услуги, является обязательным, тип данных: строка переменной длины до 50 символов;
* «OrderDate»: дата заказа, является обязательным, имеет автозаполнение по текущей дате, тип данных: дата;
* «Execution»: контрольный срок заказа (срок выполнения), не является обязательным, тип данных: дата;
* «Progress»: внешний ключ таблицы Progress, тип данных: строка фиксированной длинны 20 символов.

## Таблица «Document»

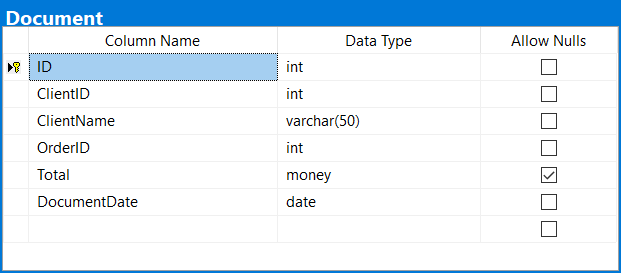
****

Таблица «Document» является договором (чеком) между компанией и клиентом и содержит следующие поля:

* «ID»: первичный ключ таблицы, с автоинкрементном от 1 с шагом 1, является обязательным, тип данных: целое число, 4 байта;
* «ClientID»: внешний ключ, ссылается на идентификатор клиента, тип данных: целое число, 4 байта
* «ClientName»: ФИО клиента, является обязательным, тип данных: строка переменной длины до 50 символов;
* «OrderID»: внешний ключ, ссылается на идентификатор заказа, тип данных: целое число, 4 байта;
* «Total»: сумма итого, не является обязательным, тип данных: денежный;
* «DocumentDate»: дата заключения договора, записывается автоматически по текущей дате, тип данных: дата.

# API

## Описание работы API

При написании API для своего приложения я использовал средства интегрированной среды разработки программного обеспечения Visual Studio и набор утилит для Open API Specification(Swagger).

**API** — описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.

**API (интерфейс прикладного программирования)** упрощает процесс программирования при создании приложений, абстрагируя базовую реализацию и предоставляя только объекты или действия, необходимые разработчику. Если графический интерфейс для почтового клиента может предоставить пользователю кнопку, которая выполнит все шаги для выборки и выделения новых писем, то API для ввода/вывода файлов может дать разработчику функцию, которая копирует файл из одного места в другое, не требуя от разработчика понимания операций файловой системы, происходящих за кулисами.

## Добавление моделей

В проекте API я добавил классы моделей для каждой сущности из базы данных. **Модель** — это набор классов, представляющих данные, которыми управляет приложение. Модель моего приложения содержит классы:

**Client** – модель сущности Client из базы данных. Содержит те же поля, что и в базе данных.

public class Client

{

public int ID { get; set; }

public string FullName { get; set; }

public string Adress { get; set; }

public string Telephone { get; set; }

}

**Serv** – модель сущности Serv из базы данных. Содержит те же поля, что и в базе данных.

public class Serv

{

public string ServiceName { get; set; }

public decimal Price { get; set; }

public int ID { get; set; }

}

**Orders** – модель сущности Orders из базы данных. Содержит те же поля, что и в базе данных.

public class Orders

{

public int ID { get; set; }

public int ClientID { get; set; }

public string ServiceName { get; set; }

public string Descript { get; set; }

public DateTime OrderDate { get; set; }

public DateTime Execution { get; set; }

public string Progress { get; set; }

}

**Document** – модель сущности Document из базы данных. Содержит те же поля, что и в базе данных.

public class Document

{

public int ID { get; set; }

public int ClientID { get; set; }

public string ClientName { get; set; }

public int OrderID { get; set; }

public decimal Total { get; set; }

public DateTime DocumentDate { get; set; }

}

**Register** – модель сущности Register из базы данных. Содержит те же поля, что и в базе данных.

public class Register

{

[Key]

public int UserID { get; set; }

public string UserLogin { get; set; }

public string UserPassword { get; set; }

}

## Добавление контекста базы данных

**Контекст базы данных** —это основной класс, который координирует функциональные возможности Entity Framework для модели данных. Этот класс является производным от класса Microsoft.EntityFrameworkCore.DbContext.

public class RepairDbContext : DbContext

{

public DbSet<Client> Clients { get; set; }

public DbSet<Serv> Servs { get; set; }

public DbSet<Orders> Orders { get; set; }

public DbSet<Document> Documents { get; set; }

public DbSet<Progress> Progress { get; set; }

public DbSet<Register> Register { get; set; }

public RepairDbContext(DbContextOptions<RepairDbContext> options) : base(options)

{

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Client>().ToTable("Client");

modelBuilder.Entity<Serv>().ToTable("Serv");

modelBuilder.Entity<Orders>().ToTable("Orders");

modelBuilder.Entity<Document>().ToTable("Document");

modelBuilder.Entity<Progress>().ToTable("Progress");

modelBuilder.Entity<Register>().ToTable("Register");

}

}

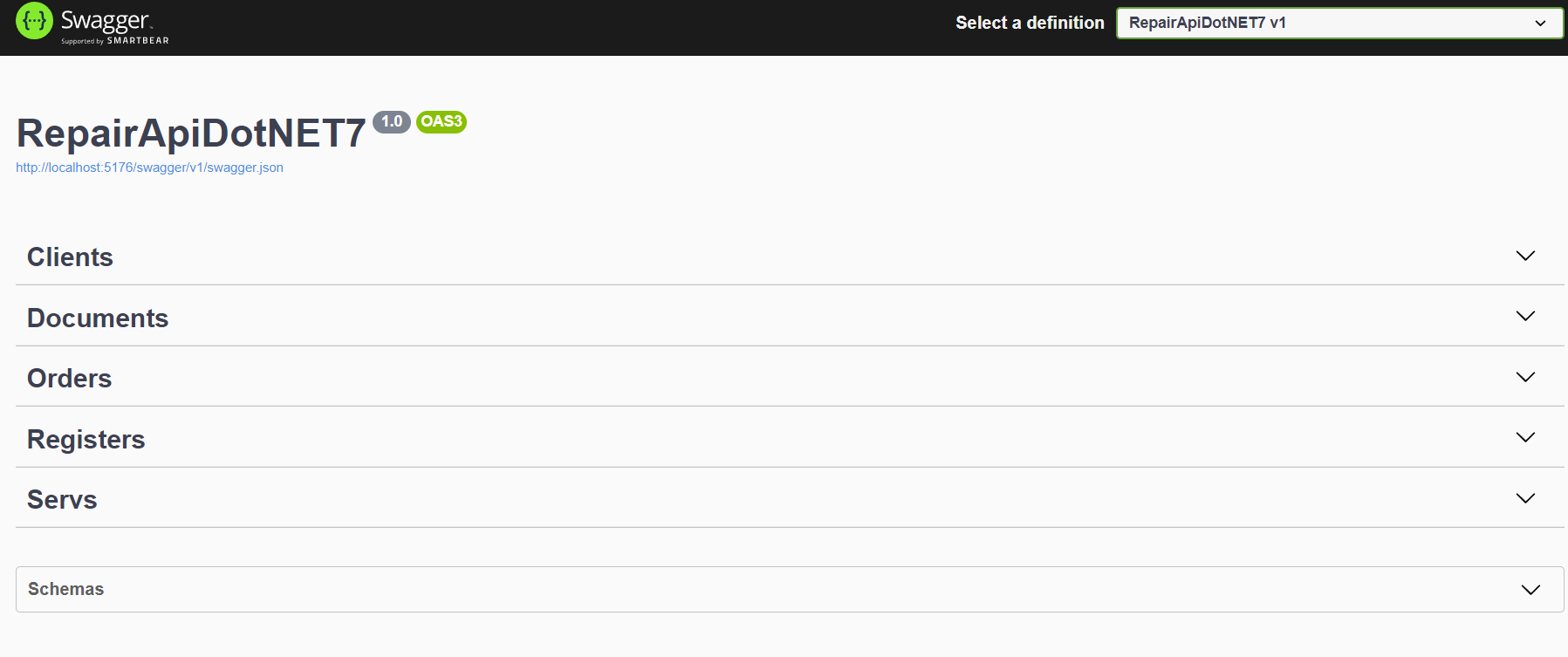
builder.Services.AddDbContext<RepairDbContext>(options => options.UseSqlServer(builder.Configuration.GetConnectionString("RepairDbConnectionString")));

## Пользовательский интерфейс Swagger

**Swagger (OpenAPI**) — это не зависящая от языка спецификация для описания REST API. Она позволяет компьютерам и пользователям лучше понять возможности REST API без прямого доступа к исходному коду. Ее основные цели: свести к минимуму объем работ, необходимых для соединения отдельных служб; сократить время, необходимое для точного документирования службы. Две основные реализации OpenAPI для .NET — это Swashbuckle и NSwag.

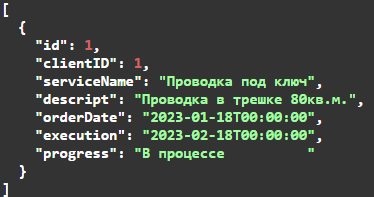
Иными словами: OpenAPI — это спецификация. Swagger — это инструментарий, использующий спецификацию OpenAPI. Например, OpenAPIGenerator и SwaggerUI[[1]](#footnote-1).

Пользовательский интерфейс Swagger обеспечивает пользовательский веб-интерфейс, предоставляющий сведения о службе с использованием созданной спецификации OpenAPI. Swashbuckle и NSwag включают встроенную версию пользовательского интерфейса Swagger, чтобы его можно было разместить в приложении ASP.NET Core, используя вызов регистрации ПО промежуточного слоя. Пользовательский веб-интерфейс выглядит следующим образом:

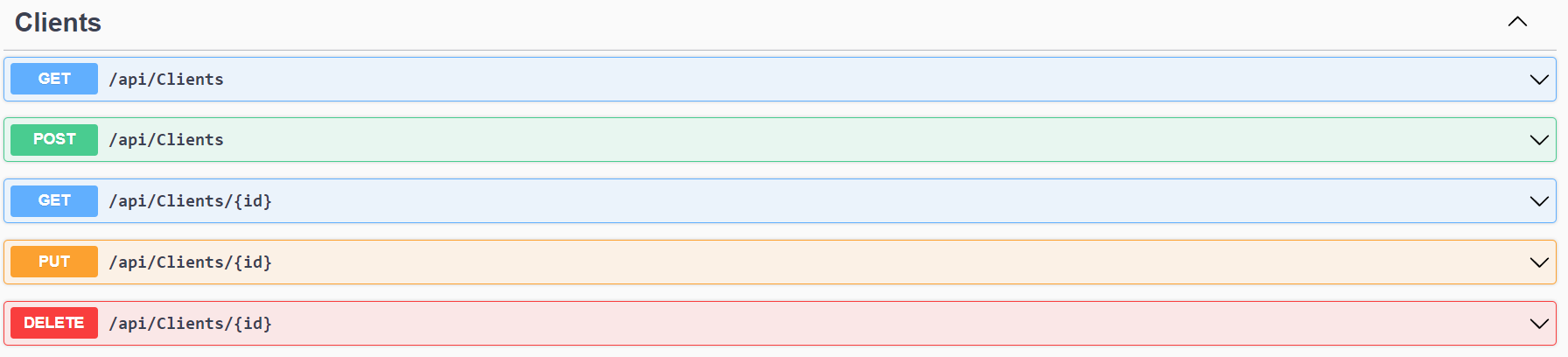


Каждый метод открытого действия в ваших контроллерах можно протестировать в пользовательском интерфейсе. Выберите имя метода, чтобы развернуть соответствующий раздел. Добавьте все необходимые параметры и запустите.

Пример результата запроса GET. Получение информации по заказу:



**Методы справочника Clients:**

****

**GET** – получение списка клиентов.

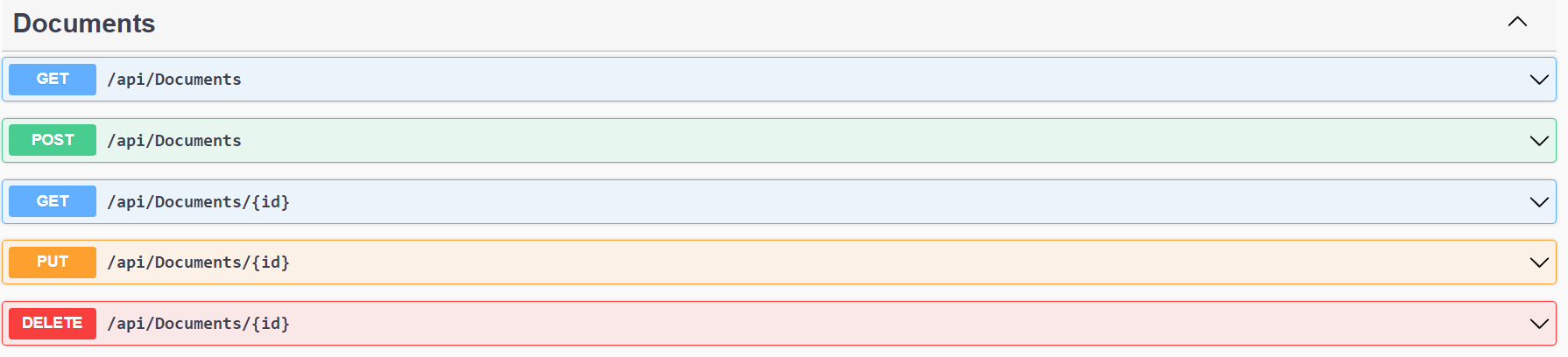
**POST** – добавление нового клиента.

**GET(id)** – получение клиента по идентификатору.

**PUT** – изменение информации о клиенте.

**DELETE** – удаление записи о клиенте.

**Методы справочника Documents:**

****

**GET** – получение списка документов.

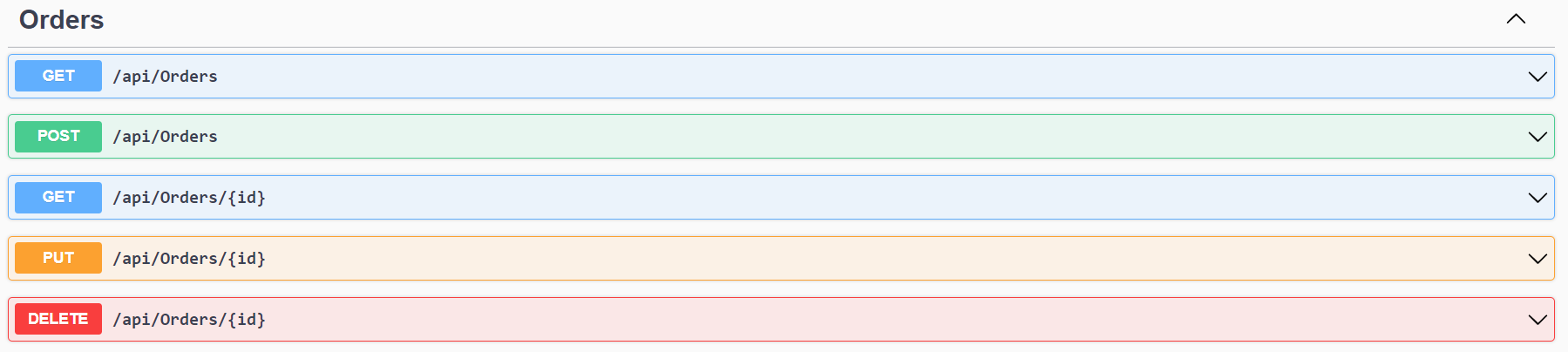
**POST** – добавление нового документа.

**GET(id)** – получение документа по идентификатору.

**PUT** – изменение информации в документе.

**DELETE** – удаление документа.

**Методы справочника Orders:**

****

**GET** – получение списка заказов.

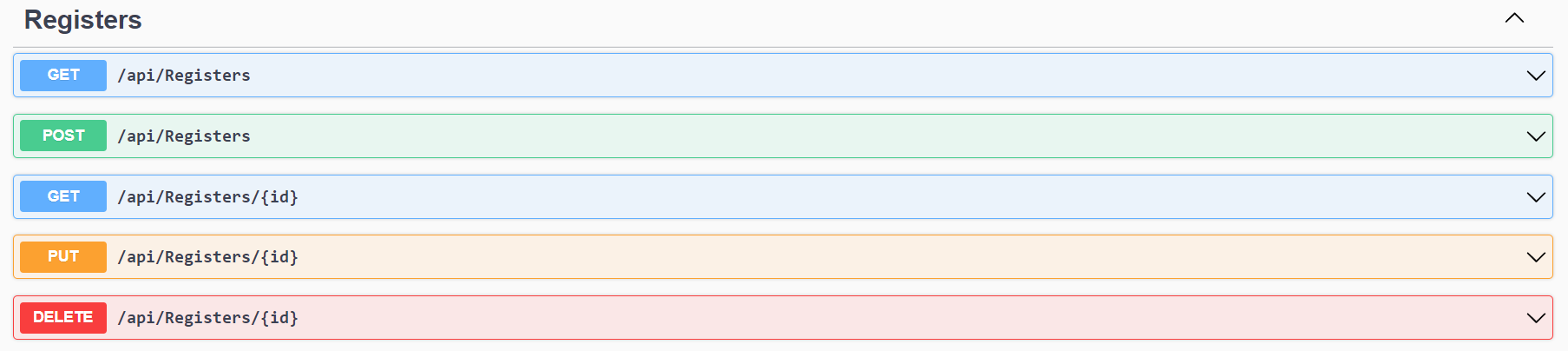
**POST** – добавление нового заказа.

**GET(id)** – получение заказа по идентификатору.

**PUT** – изменение информации в заказе.

**DELETE** – удаление заказа.

**Методы справочника Registers:**

****

**GET** – получение списка пользователей.

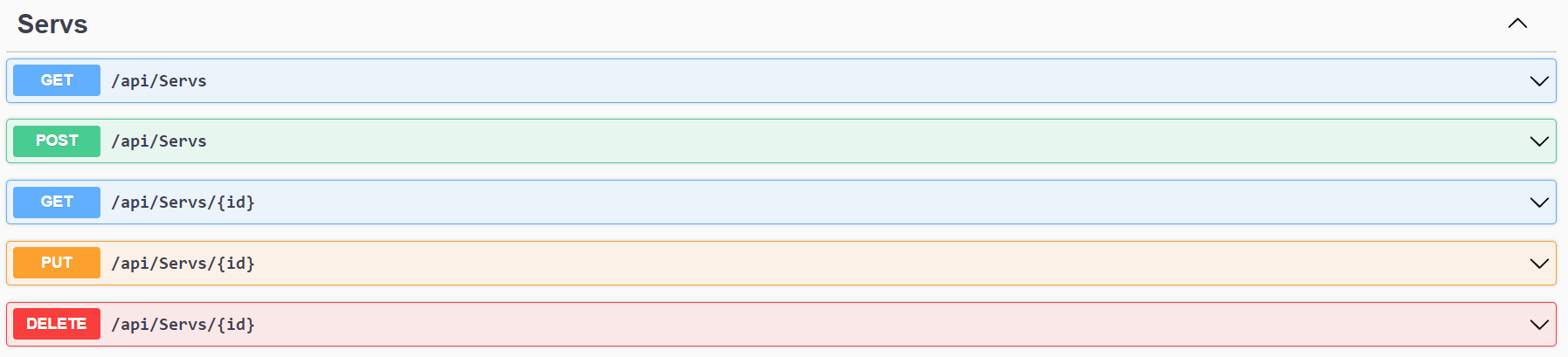
**POST** – добавление нового пользователя.

**GET(id)** – получение пользователя по идентификатору.

**PUT** – изменение информации о пользователе.

**DELETE** – удаление пользователя.

**Методы справочника Servs:**

****

**GET** – получение списка услуг.

**POST** – добавление новой услуги.

**GET(id)** – получение услуги по идентификатору.

**PUT** – изменение информации об услуги.

**DELETE** – удаление услуги.

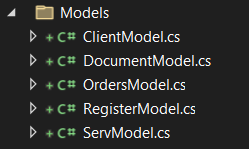
# Приложение

## Описание работы клиентского приложения

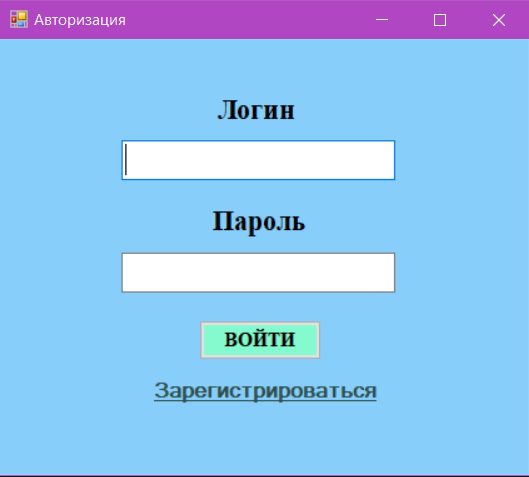
Клиентской называется часть приложения, с которой напрямую взаимодействует конечный пользователь. Это может быть либо приобретенное компанией серийное коммерческое программное обеспечение, либо прикладная программа, разработанная внутри компании с помощью инструментальных средств третьих фирм.

Наличие клиентских приложений способствует упрощению работы конечного пользователя базы данных. От него скрыты объекты базы данных, содержащие реальную информацию, программный код, а также происходящие внутри нее события. Современные технологии делают работу с прикладными программами для конечного пользователя более интуитивной: они дают ему возможность сосредоточится на выполнении своих прямых обязанностей и, тем самым, способствуют повышению производительности его труда.[[2]](#footnote-2)

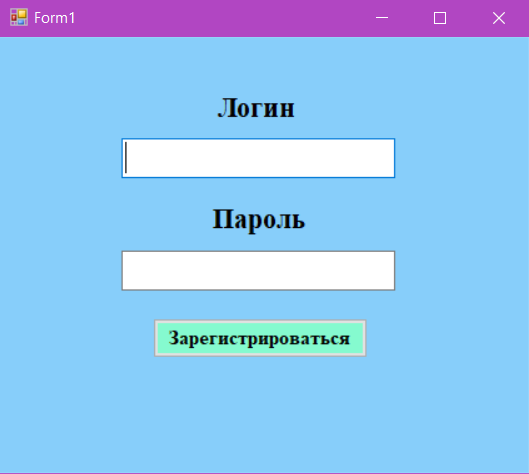
В проекте клиентского приложения я также добавил классы моделей для каждой сущности из базы данных.



## Формы авторизации и регистрации:

****

Чтобы войти в приложение, пользователю нужно ввести логин и пароль. Пароль сравнивается с данными в базе и, если все данные введены верно, то пользователь переходит на главную форму. Также имеется возможность зарегистрироваться, перейдя по ссылке на форму регистрации.

****

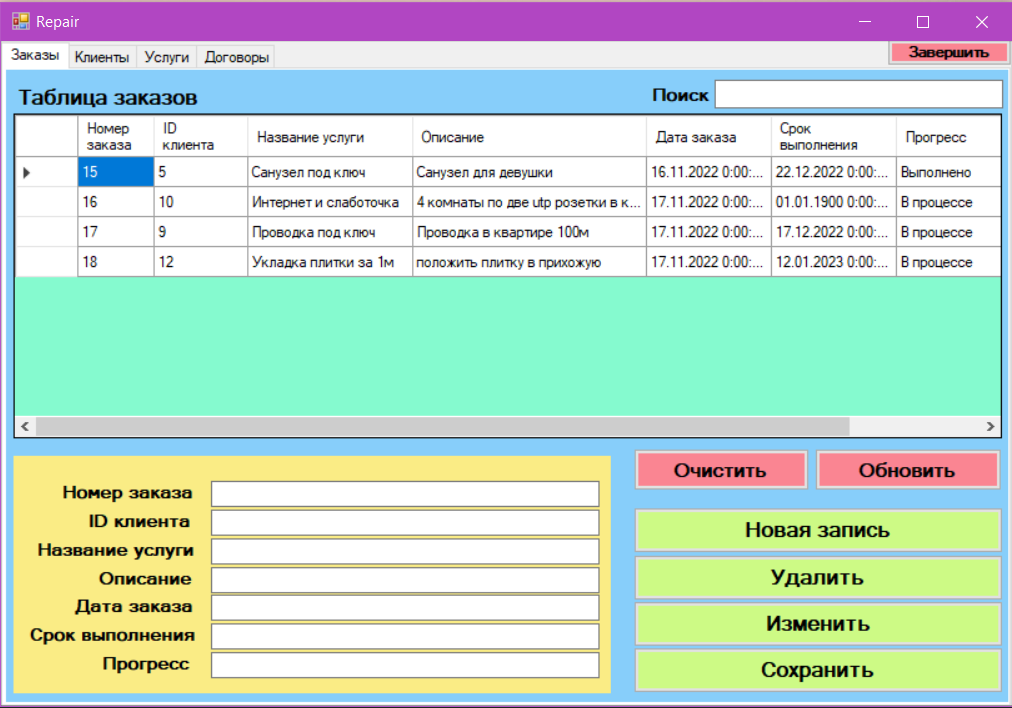
## Главная форма:

В этом проекте я решил реализовать весь главный функционал программы на одной форме. Поэтому при входе в приложение пользователь сразу встречает страницу заказов.

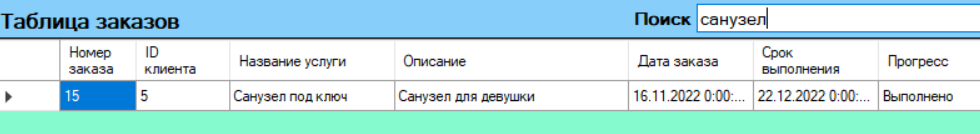
Выбор такого решения могу объяснить удобством в использовании, помимо добавления новых записей (что как раз удобнее делать в отдельном окне), никаких лишних переходов на другие формы здесь нет. А навигация между таблицами реализована через “tabPage”, проще говоря, вкладки.

Вывод данных из таблиц представлен в dataGridView – элемент управления, предоставляющий настраиваемую таблицу для отображения данных.

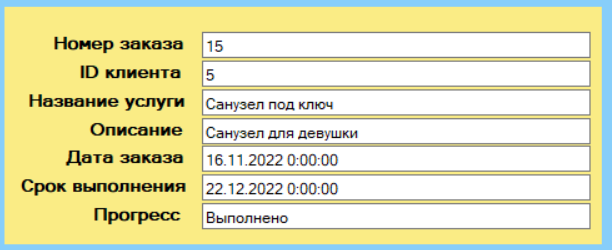
## Вкладка заказы:

****

Выше таблицы располагается поле поиска, где пользователь может выполнить поиск по ключевому слову.

****

В левом нижнем углу формы находятся поля детального представления, в которые выводятся значения строки, в ячейку которой нажалпользователь.Также с помощью этой панели можно изменить значения записей в dataGridView, а оттуда уже в базе данных.

****

В правом нижнем углу располагаются кнопки взаимодействия с записями.

****

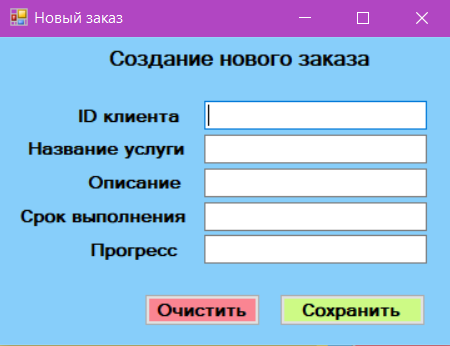
Кнопка «Обновить» заново выводит данные в dataGridView из запроса.

Кнопка «Очистить» очищает текст из полей в левом углу.

«Изменить» вносит изменения из панели с textbox’ами в dataGridView по выбранной строке.

«Удалить» соответственно удаляет выбранную строку.  
А кнопка «Сохранить» отправляет соответствующие запросы на удаление и изменение.

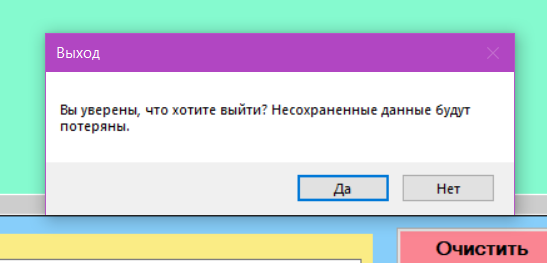
«Новая запись» открывает форму добавления записи. Поля в форме добавления соответствуют столбцам в базе данных, за исключением данных автозаполнения, таких как ID и текущей даты.

****

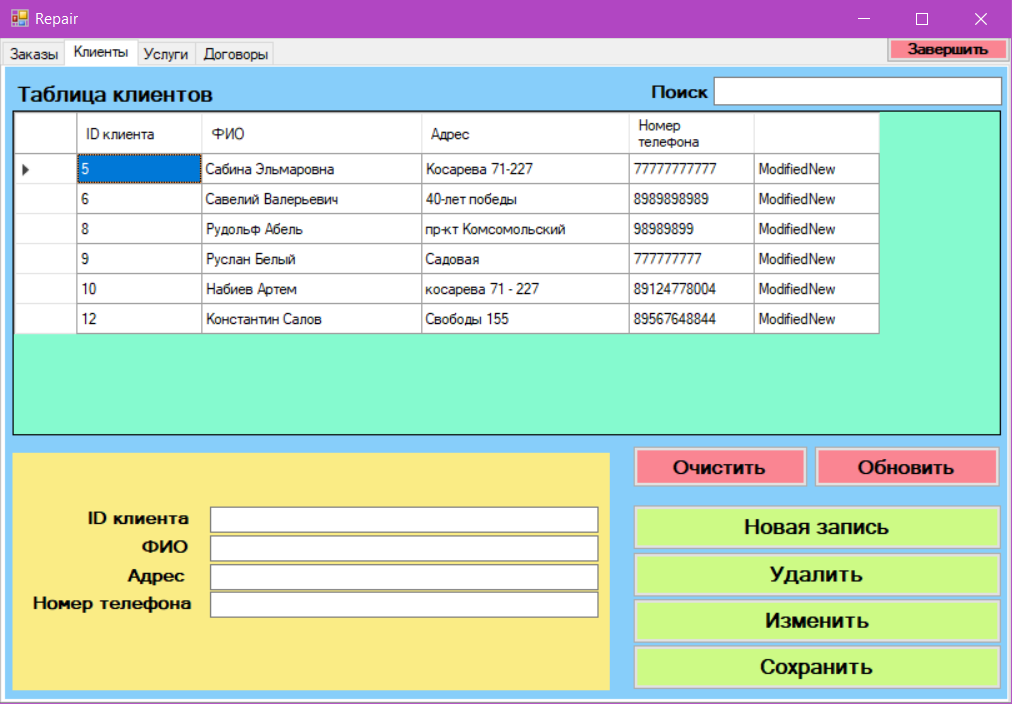
Очистить – очищает текст в полях, а сохранить отправляет запрос на добавление записи.

Аналогичный функционал представлен на трех других вкладках формы.

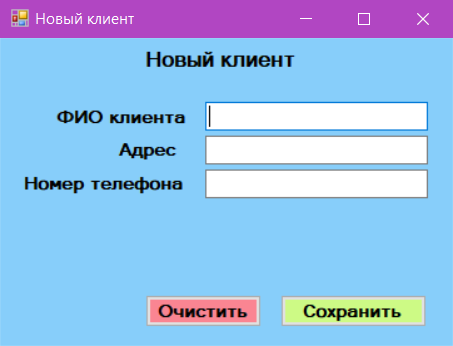
Kнопка «Завершить» закрывает приложение с сопутствующим сообщением пользователю о возможной потери данных.

****

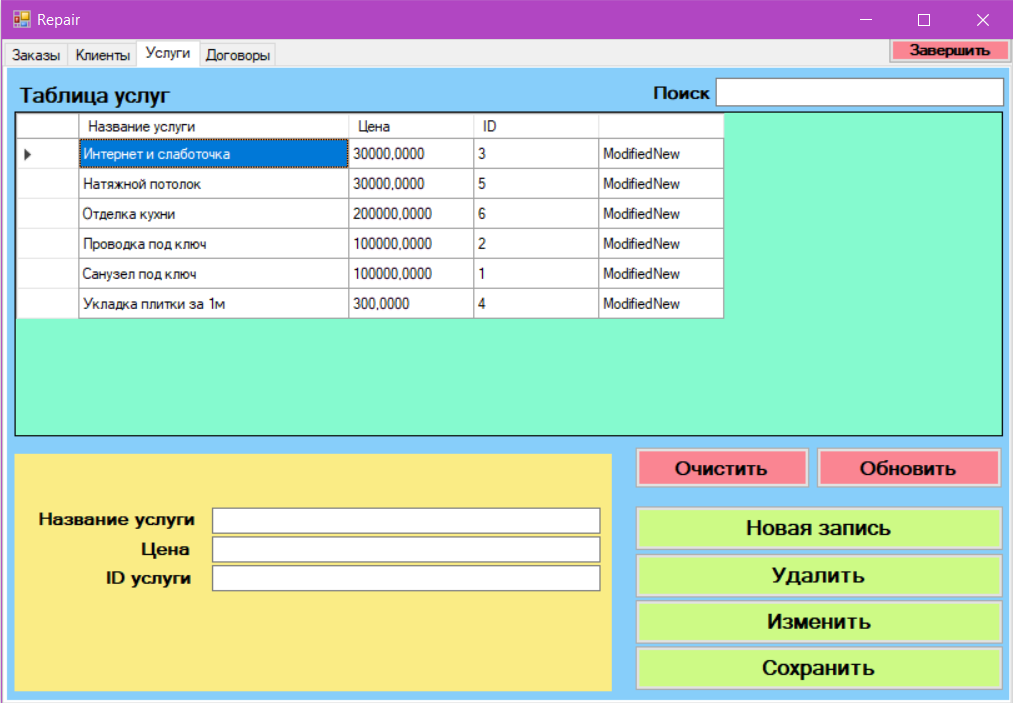
## Вкладка Клиенты:

****

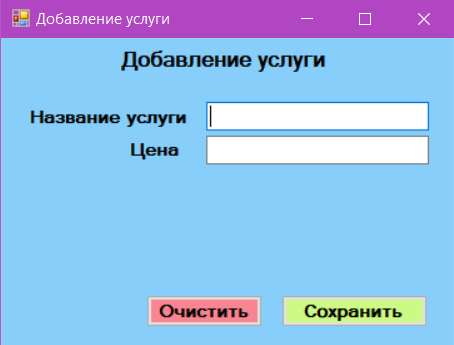
Форма добавления клиента:

****

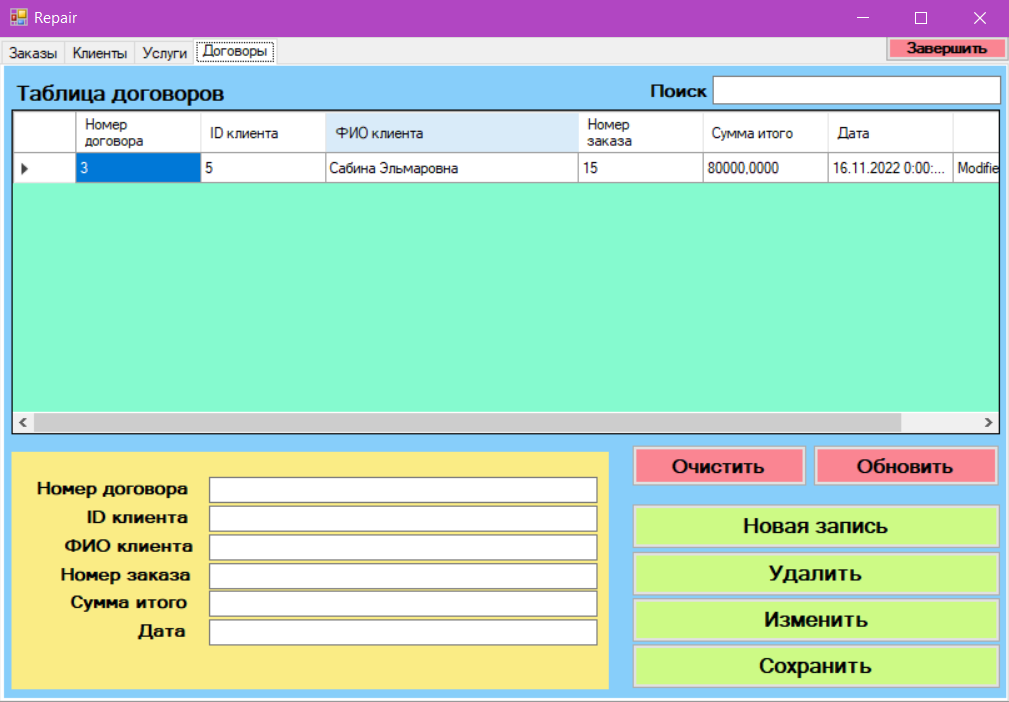
## Вкладка Услуги:



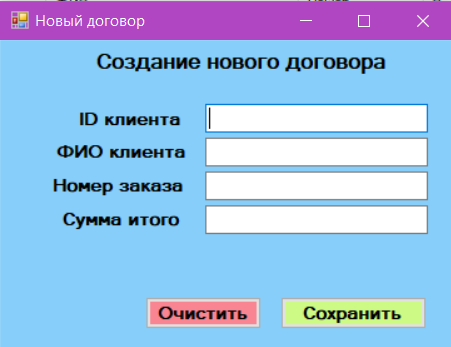
Форма добавления услуги:

****

## Вкладка Договоры:

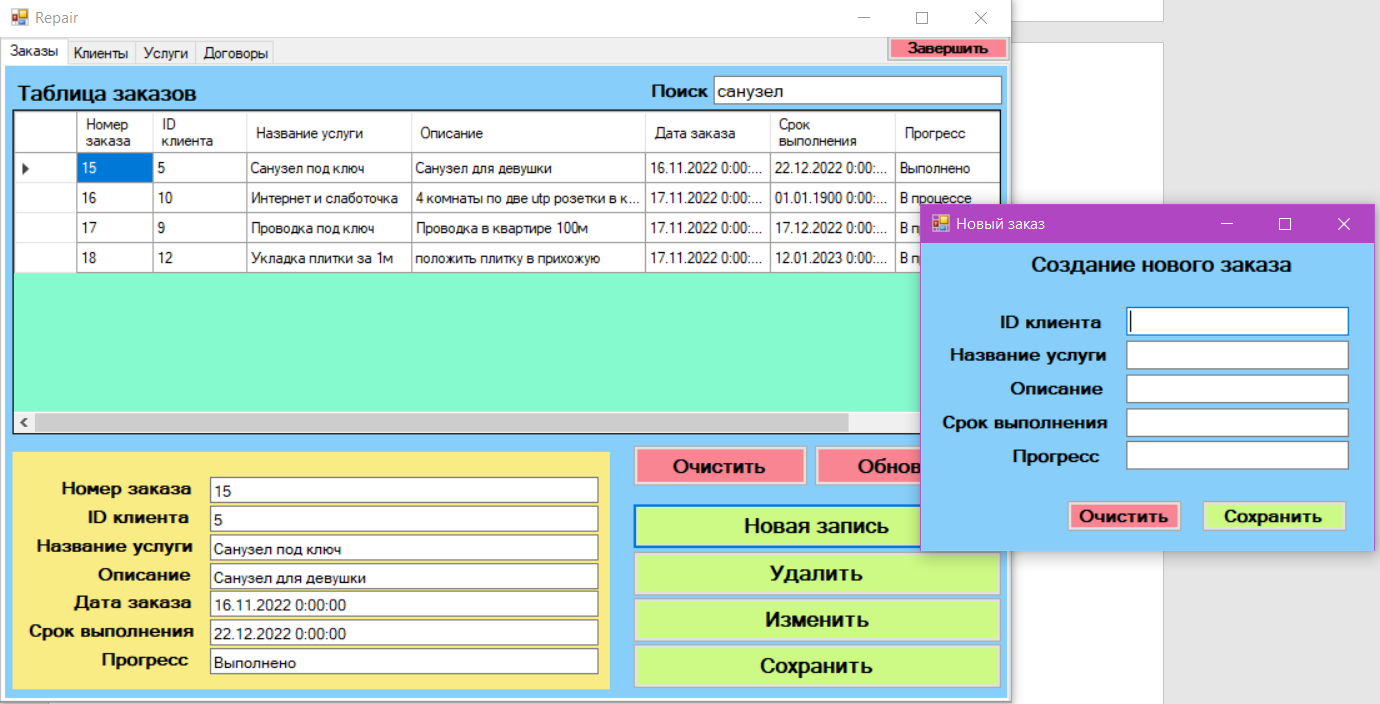
****

Форма создания нового договора:

****

# Заключение

Таким образом, в век развитых информационных технологий людям часто требуется помощь в учете данных. Информационные системы становятся всё более популярными, так как облегчают множество операций на предприятии. Поэтому данная тема является актуальной. Я создал АРМ менеджера строительной фирмы, использующее собственную базу данных, а так же её изменение. В работе описан алгоритм программы, этапы работы над ним, приведены скриншоты результатов работы. Цель работы достигнута.



# Литература 1. «Введение в системы баз данных» К. Дж. Дейт. 2. SQL. Полное руководство. Джеймс Р. Грофф, Пол Н. Вайнберг, Эндрю Дж. Оппель. 3. В. Ю. Пирогов. Информационные системы и базы данных: организация и проектирование: учебное пособие. 4. Владимир Петров. Информационные системы: учебное пособие.

1. [↑](#footnote-ref-1)
2. [↑](#footnote-ref-2)