

МИНИСТЕРСТВО ОБРАЗОВАНИЯ МОСКОВСКОЙ ОБЛАСТИ
Государственное бюджетное профессиональное образовательное
учреждение Московской области «Люберецкий техникум имени Героя
Советского Союза, летчика-космонавта Ю.А.Гагарина»

ОТЧЕТ

ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИЧЕСКОЙ ПОДГОТОВКЕ

Меринова Дмитрия Александровича

(Фамилия, имя, отчество студента)

по профессиональному модулю

ПМ. 01 Разработка модулей программного обеспечения для
компьютерных систем

ПМ. 02 Осуществление интеграции программных модулей

Специальность

09.02.07 "Информационные системы и программирование"

Код, название

Курс 4 Группа № 195ИС

Период практической подготовки

с «16» февраля 2023 г. по «12» апреля 2023г.

Руководитель практической подготовки

от техникума преподаватель Жирнова Ю. В.

должность, ФИО

Руководитель практической подготовки

от организации Морозов И.С

должность, ФИО

Люберцы 2023

ХАРАКТЕРИСТИКА ПРАКТИКАНТА

Меринова Дмитрия Александровича

(Ф.И.О. практиканта)

Работал в информационном отделе, разработчик WPF приложений

(подразделение, должность, сроки работы)

с «16» февраля 2023 г. по «12» апреля 2023 г.

Количество выходов на работу 39 дней.

Пропущено дней 0, из них по не уважительной причине 0

Прошел производственную (преддипломную) практическую
подготовку по специальности

09.02.07 "Информационные системы и программирование"

Качество выполнения работы

За время прохождения производственной практической подготовки
показал себя с положительной стороны. В течение всего периода Меринов
Дмитрий Александрович внимательно и ответственно относился к
выполняемой работе.

В отношениях с коллегами Меринов Дмитрий Александрович
проявил себя с лучшей стороны: внимательность, умение выслушать и
понять, стремление избежать конфликта, устойчивость к стрессам.

Особенно хочется отметить умение грамотно планировать свою
деятельность в соответствии со стратегией развития коллектива и
выполнение работы с максимальной эффективностью. Руководство
организации оценивает работу Меринова Дмитрия Александровича
положительно.

Руководитель практической подготовки от организации

Руководитель информационного отдела Морозов Иван Сергеевич

(должность, ФИО)

М.П.

Описание организации

1. Название организации: ООО «Авто-сфера»
2. Адрес: г. Дзержинский, ул. Энергетиков, 24
3. Адрес для писем: г. Дзержинский, ул. Энергетиков, 24
4. Телефон: +7 495-225-23-53
5. Факсы +7-495-220-20-50
6. E-mail: kapotnya@avto-start.ru
7. Отрасль: Торговля.
8. Год основания: 2004г
9. Форма собственности: Общество с ограниченной ответственностью

10. История

Компания «Авто-Сфера» была основана в 1991 году, и начала свое развитие с Сервисного центра на Проспекте Мира.

В 1992 году там же открывается первый автосалон, Авто-Сфера становится дилером KIA.

В 1993 году компания получает статус официального дилера MITSUBISHI.

В 2002 году - открытие автосалона и современного технического центра MITSUBISHI на Дмитровском шоссе

В 2004 году Авто - Сфера под брендовым названием Авто-Старт становится официальным дилером ТаГАЗ и получает право продажи автомобилей HYUNDAI.

В 2004 году завершается строительство территории на берегу реки Яуза и начинается продажа и обслуживание автомобилей MITSUBISHI и HYUNDAI

В 2005 году основан Департамент Продажи Поддержанных Автомобилей на Дмитровском шоссе

В 2006 году открывается специализированный дилерский центр KIA на 15-м км МКАД

В 2007 году создается вторая территория по выкупу, обмену и продаже поддержанных автомобилей.

В этом же году открывается Технический Центр на Волоколамском шоссе по обслуживанию автомобилей GEELY.

Уже через 2 года (в 2009 году) компания Авто-Старт принимает в управление территорию в г. Бронницы, специализирующуюся на продаже и ремонту грузовых автомобилей FORD

В декабре 2011 началась реконструкция фасада автосалона KIA на 15-м км МКАД – здесь и открыла своё представительство фирма «Авто-сфера»

11. Дочерние предприятия и/или филиалы

- Компания «Авто-сфера» сама является дочерним предприятием фирмы «Авто-спектр»

12. Предоставляемые услуги: Техническое обслуживание и ремонт автотранспортных средств, продажа автозапчастей.

Описание подразделения, в котором была пройдена практика

1. Название подразделения: Отдел безопасности
2. Руководитель подразделения: Морозов Иван Сергеевич
3. Куратор практики: Морозов Иван Сергеевич
4. Структура и функции подразделения (в краткой форме должностные обязанности):

Цели:

Отдел безопасности автосалона участвует в организации современной системы защиты информационной системы по продаже автомобилей. Работает для того, чтобы после приобретения у нас автомобиля, покупатель с удовольствием пользовался услугами нашего автосервиса, и не переживал за утечку данных, указанных при покупке

Руководитель отдела безопасности:

4.1. Приходит на работу за 30 минут до начала рабочего дня, аккуратно одетым.

4.2. Встречает менеджеров в салоне продаж, помогает сориентироваться в предлагаемом ассортименте автомобилей.

4.3. Провожает покупателей на стоянку автомобилей, знакомит их с предлагаемыми к продаже автомобилями и предлагает возможные варианты оплаты. Если на стоянке нет того автомобиля, который желает приобрести покупатель, оформляет заявку на желаемый автомобиль и сообщает о сроке и условиях выполнения заявки.

4.4. Обеспечивает безопасность сделки. Объясняет преимущества приобретения автомобиля в авто центре, предоставляя покупателю информацию обо всех предлагаемых скидках, условиях и возможностях гарантийного обслуживания.

4.5. Сообщает каждому потенциальному покупателю о возможностях комплекса авто услуг автоцентра и приглашает воспользоваться услугами автосервиса.

4.6. Сопровождает беседу с покупателем демонстрацией каталогов, прайс-листов и всеми имеющимися в распоряжении менеджера по продажам рекламными материалами.

4.7. Во всех ситуациях действует технологично, применяя современные техники продаж и привлечения покупателей.

4.8. Передает покупателей, принявших решение о покупке продавцам-консультантам, продавцам-оформителям и контролирует оформление сделки.

5. Резюме руководителя подразделения

- Высшее образование по специальности «Программист 1С»
- стаж работы в данной сфере 5 лет
- достижения в профессиональной сфере: Является сооснователем ООО «Авто-сфера». До «Авто-сферы» работал в салоне «KIA Motors» в отделе информационной безопасности

Введение

Описание подразделения	4
Описание предметной области	7
Постановка задачи	8
Установка и настройка инструментального ПО	10
Microsoft SQL Server Management Studio	10
SQL EXPRESS	14
Visual Studio 2022	16
Создание ERD-диаграммы в Visio	20
User Story	22
Use case	23
Разработка базы данных автосалона	24
Разработка WPF приложения	28
Код приложения	36
Страница авторизации	36
Вывод данных в DataGrid	44
ListView	51
Корзина	61
Заключение	64
Список литературы	66

Описание предметной области

Начнём наше изучение предметной области с программного обеспечения:

Программное обеспéчение — программа или множество программ, используемых для управления компьютером.

Visual Studio 2022 - это интегрированная среда разработки (IDE), которую разработала компания Microsoft для создания приложений для операционных систем Windows, а также для веб-разработки и разработки мобильных приложений. Среда поддерживает множество языков программирования, включая C#, C++, Visual Basic, Python, JavaScript и многие другие.

Visual Studio обеспечивает разработчикам широкий набор инструментов для создания, отладки, тестирования и развертывания приложений. Это позволяет ускорить процесс разработки и упростить взаимодействие между разработчиками в команде. В Visual Studio также есть множество плагинов и расширений, которые позволяют дополнительно расширить функциональность IDE для удобства разработки и облегчения задач.

Microsoft SQL Server Management Studio (SSMS) – это интегрированная среда для управления базами данных Microsoft SQL Server. Она предоставляет графический интерфейс для управления и настройки экземпляров SQL Server, а также позволяет создавать, изменять и удалять базы данных, таблицы, представления, процедуры и другие объекты баз данных.

Постановка задачи

Автосалон - это как магазин, в котором представлены новые и поддержанные автомобили, а также их технические характеристики. Основными элементами предметной области являются поставляемые автомобили с различными признаками, а также информация о покупателях, продавцах и заказах.

База данных будет использоваться для учета продаж и поставок автомобилей и будет относиться к классу баз данных управления предприятием. Созданная база данных поможет упростить процедуру поиска информации о товарах и ценах на них. Созданная база данных также включает информацию о покупателях, сотрудниках машинах, заказах.

Приложение для работы с информационной системой автосалона упрощает процедуру поиска необходимой информации о машинах и ценах на них. С ее помощью сотрудник может легко узнать информацию о поставленном или проданном автомобиле, клиент может просматривать существующий в наличии модельный ряд авто, руководитель отслеживать деятельность сотрудников и динамику продаж.

Как у администратора, так и у клиента есть возможность найти интересующий его автомобиль, выставить нужные фильтры, отсортировать список автомобилей по наличию, по возрастанию цены и по определённой марке автомобиля.

На рисунке 1 определим задачи которые должны быть решены на производственной практической подготовке:

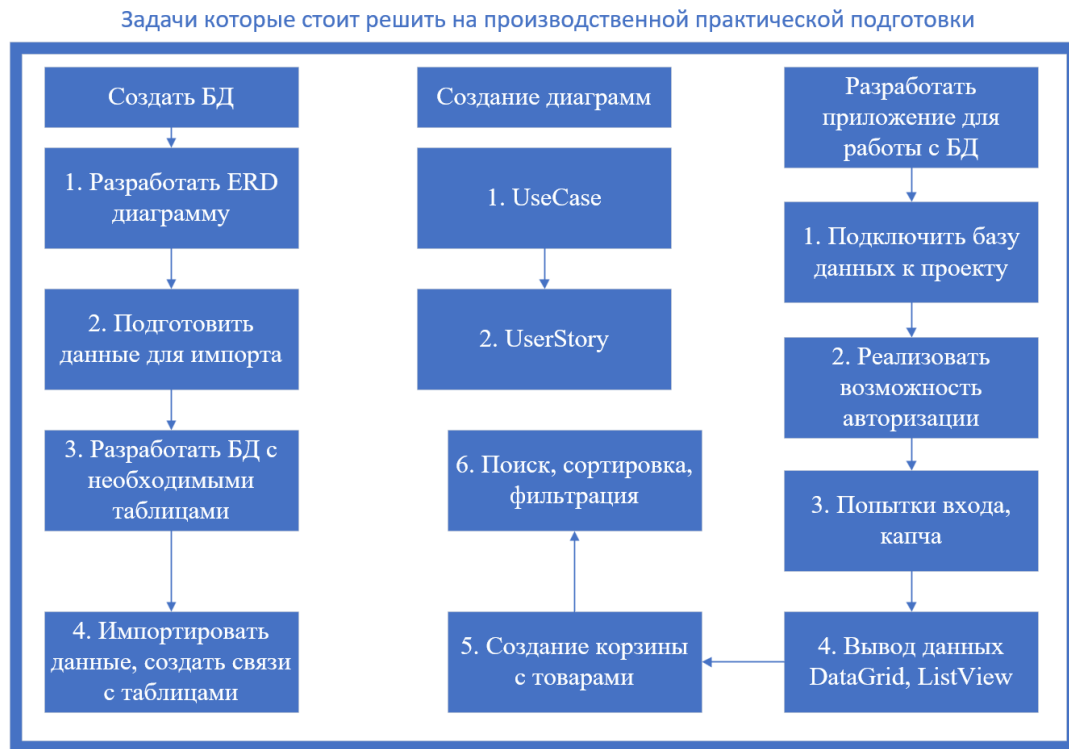


Рисунок 1. Набор задач на производственную практическую подготовку

Необходимо выполнить следующие задачи:

1. Создать БД → Разработать ERD диаграмму → Подготовить данные для импорта → Разработать БД с нужными таблицами → Импортировать данные.
2. Создание диаграмм → UseCase → UserStory
3. Разработать WPF приложение → Подключить БД к проекту → Реализовать возможность авторизации → Попытки входа и капча → Вывод данных, DataGrid, ListView → Создание корзины с товарами → Поиск, сортировка, фильтрация.

Установка и настройка инструментального ПО

Microsoft SQL Server Management Studio

В качестве инструментального ПО, необходимого для предприятия ООО «Авто-сфера», нам необходимо установить **SQL Server Management Studio**, интегрированная разработка для управления и администрирования баз данных SQL Server, которую разработала компания Microsoft. SSMS позволяет разработчикам и администраторам баз данных управлять базами данных, создавать и изменять таблицы, хранимые процедуры, функции и другие объекты базы данных.

SSMS позволяет выполнять множество задач, таких как:

- Создание и управление базами данных.
- Создание и изменение таблиц, представлений, индексов, хранимых процедур, триггеров и других объектов базы данных.
- Импорт и экспорт данных.
- Выполнение запросов к базе данных и просмотр результатов.
- Управление безопасностью и аутентификацией пользователей и групп.
- Настройка параметров сервера и баз данных.
- Мониторинг и анализ производительности сервера и базы данных.

SSMS является важным инструментом для разработчиков и администраторов баз данных, которые работают с Microsoft SQL Server. Он облегчает работу с базами данных, позволяя пользователям создавать и изменять объекты базы данных, а также мониторить и настраивать производительность сервера и баз данных.

Скачать приложение можно с официального [сайта](#) (Рисунок 2).

Download SQL Server Management Studio (SSMS)

Article • 03/13/2023 • 6 minutes to read • 48 contributors

[Feedback](#)

Applies to: [SQL Server](#) [Azure SQL Database](#) [Azure SQL Managed Instance](#) [Azure Synapse Analytics](#)

SQL Server Management Studio (SSMS) is an integrated environment for managing any SQL infrastructure, from SQL Server to Azure SQL Database. SSMS provides tools to configure, monitor, and administer instances of SQL Server and databases. Use SSMS to deploy, monitor, and upgrade the data-tier components used by your applications and build queries and scripts.

Use SSMS to query, design, and manage your databases and data warehouses, wherever they are - on your local computer or in the cloud.

Download SSMS

[Free Download for SQL Server Management Studio \(SSMS\) 19.0.2](#)

SSMS 19.0.2 is the latest general availability (GA) version. If you have a *preview* version of SSMS 19 installed, you should uninstall it before installing SSMS 19.0.2. If you have SSMS 19.x installed, installing SSMS 19.0.2 upgrades it to 19.0.2.

- Release number: 19.0.2
- Build number: 19.0.20209.0
- Release date: March 13, 2023

Рисунок 2. Страница загрузки SQL Server Management Studio

Скачиваем установщик и запускаем его из папки с загрузками (Рисунок 3).

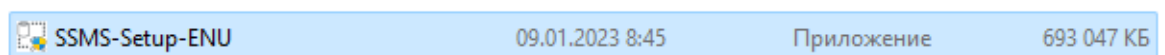


Рисунок 3. Установщик SQL Server Management Studio

После запуска, перед нами откроется главное окно установщика, где нам необходимо выбрать путь установки и нажать кнопку «Install» (Рисунок 4).

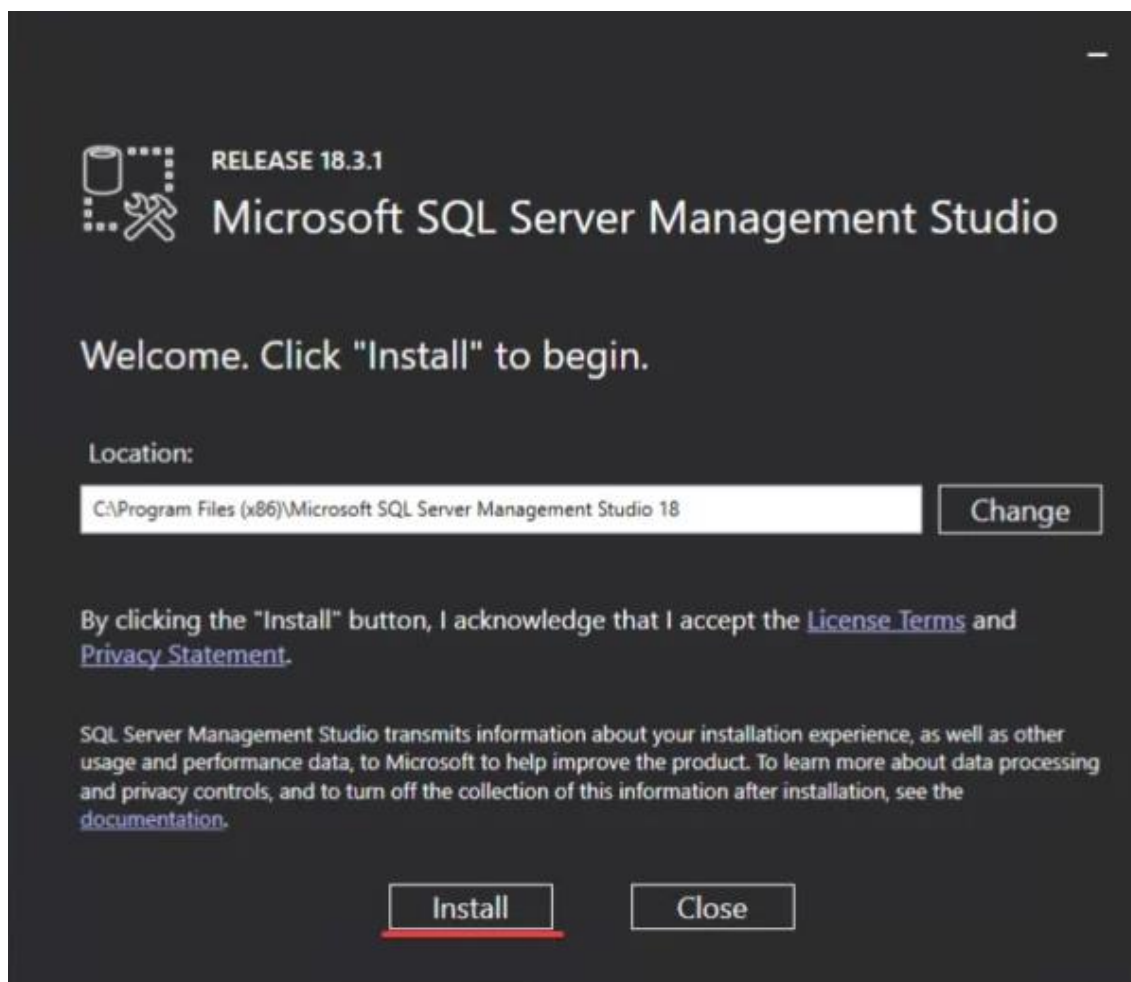


Рисунок 4. Окно установщика

После выбора места установки и перехода на следующий этап установки, нам необходимо дождаться завершения непосредственной установки программы (Рисунок 5).

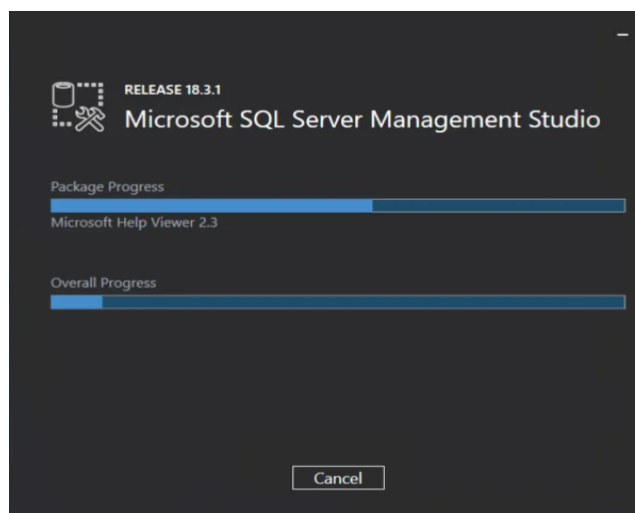


Рисунок 5. Процесс установки SQL Server Management Studio.

По завершении установки появится экран, предлагающий выполнить перезагрузку. Перезагружаемся, нажав на Restart (Рисунок 6).

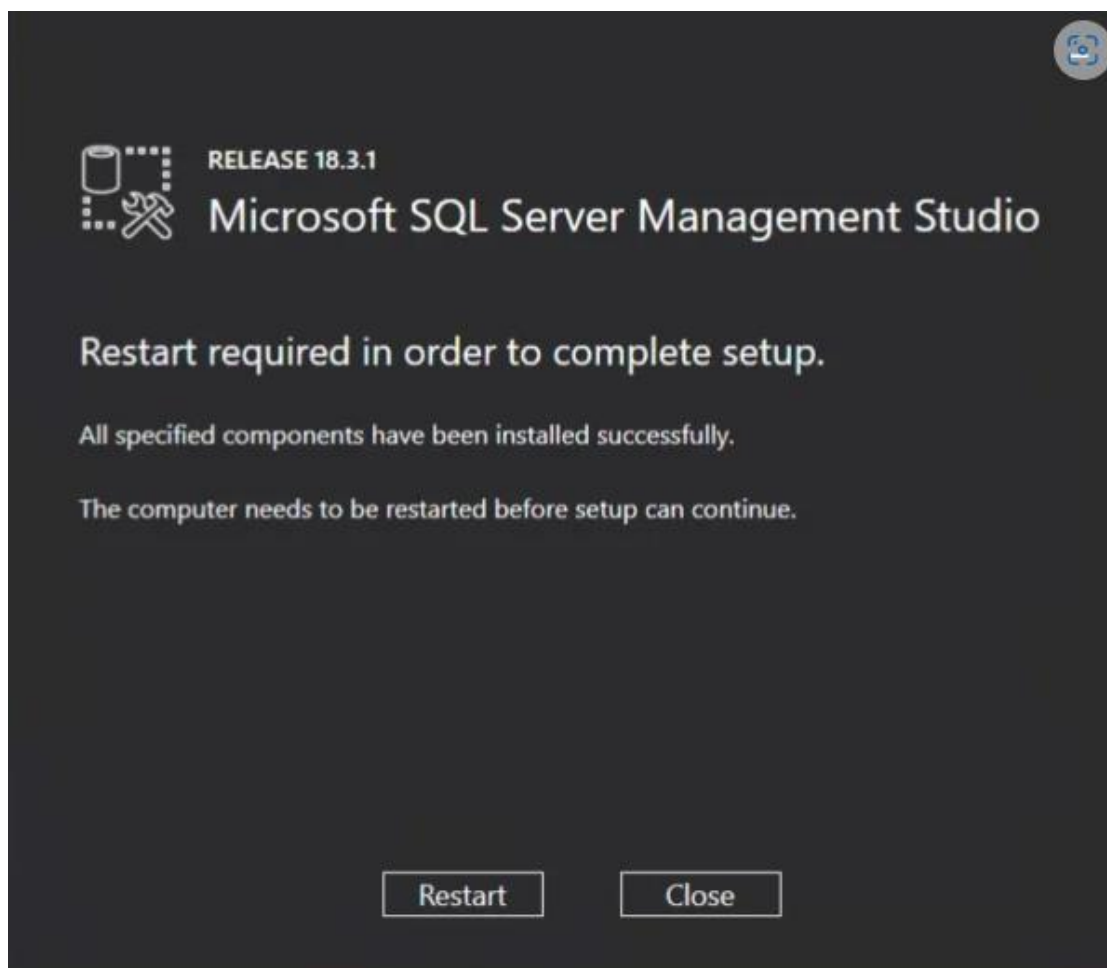


Рисунок 6. Завершение установки.

Запуск приложения осуществляется с помощью специальной иконки в панели, вызываемой с помощью нажатия кнопки «Windows» (Рисунок 7).

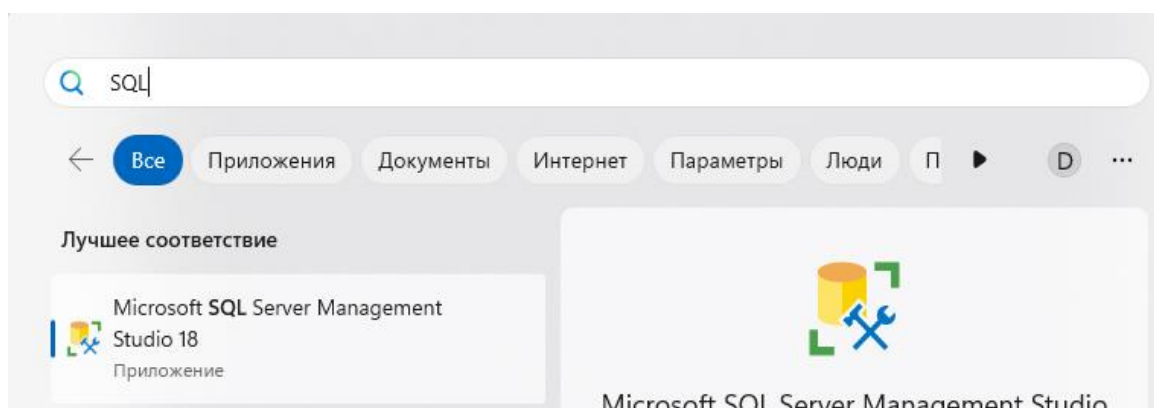


Рисунок 7. Иконка запуска приложения.

Приложение Sql Server Management Studio успешно установлено.

SQL EXPRESS


SQL Express (SQL Server Express) - это бесплатная версия системы управления реляционными базами данных Microsoft SQL Server. SQL Express предназначен для малых и средних приложений, которые не требуют высокой производительности и большого количества пользователей.

SQL Express имеет ограничения по размеру баз данных (до 10 Гб) и количеству доступных процессоров (до 4). Однако, она содержит все основные функции SQL Server, включая язык SQL, поддержку транзакций, процедуры и функции, агент службы SQL Server и многое другое. SQL Express поддерживает работу с различными языками программирования, включая C#, VB.NET, Java и PHP, и может использоваться в качестве хранилища данных для приложений на платформах Windows, Linux и macOS.

Для загрузки установщика перейдём на официальный сайт Microsoft на страницу SQL Express (Рисунок 8).

Microsoft® SQL Server® 2019 Express

Важно! Если выбрать язык ниже, содержимое страницы динамически отобразится на выбранном языке.

Выберите язык: 

[Скачать](#)

Microsoft® SQL Server® 2019 Express — мощная и надёжная бесплатная система управления данными, обеспечивающая функциональное и надёжное хранилище данных для веб-сайтов и настольных приложений.





-  [Сведения](#)
-  [Требования к системе](#)
-  [Инструкции по установке](#)
-  [Дополнительные сведения](#)

Рисунок 8. Страница загрузки SQL Express.

Скачиваем и запускаем установщик, где перед нами откроется окно, в котором будет выбор трёх вариантов установки, выбираем express (Рисунок 9).

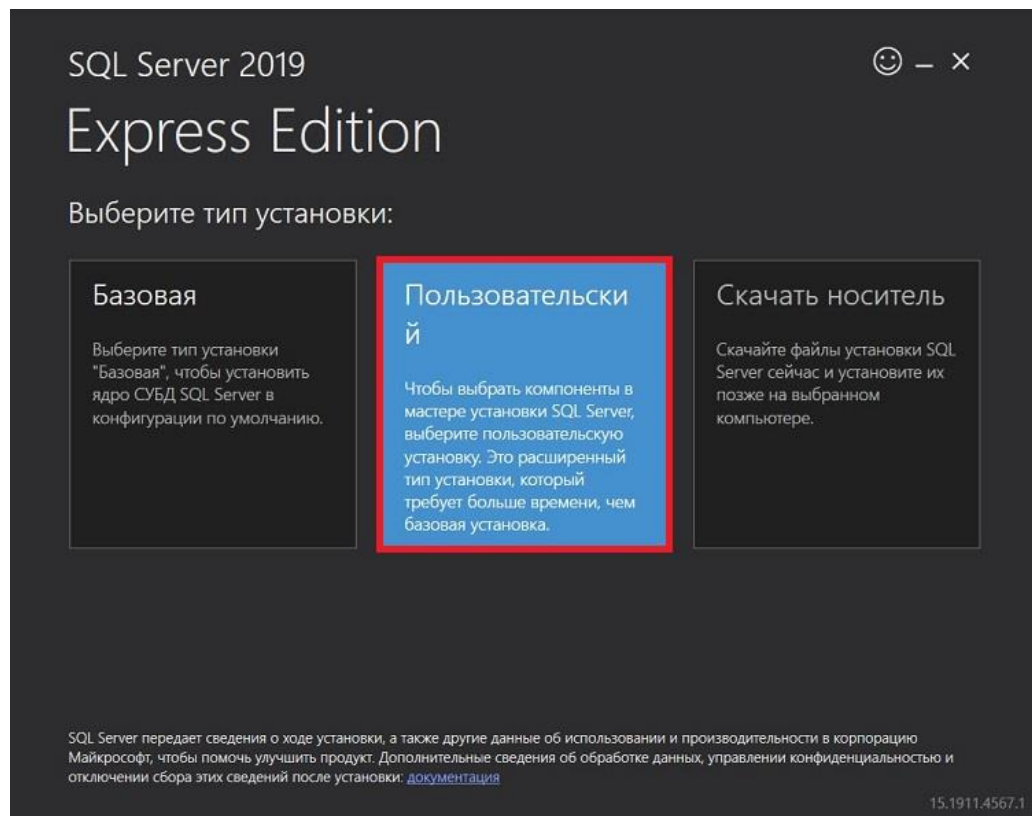


Рисунок 9. Выбор варианта установки.

После сделанного выбора, выбираем путь установки и нажимаем кнопку «Установить»(Рисунок 10).

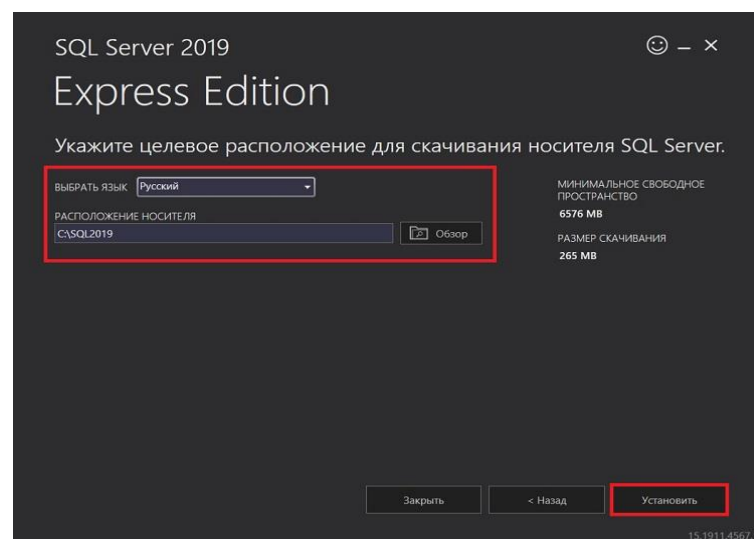


Рисунок 10. Указание места установки.

После этого, начнётся сам процесс установки(Рисунок 11).

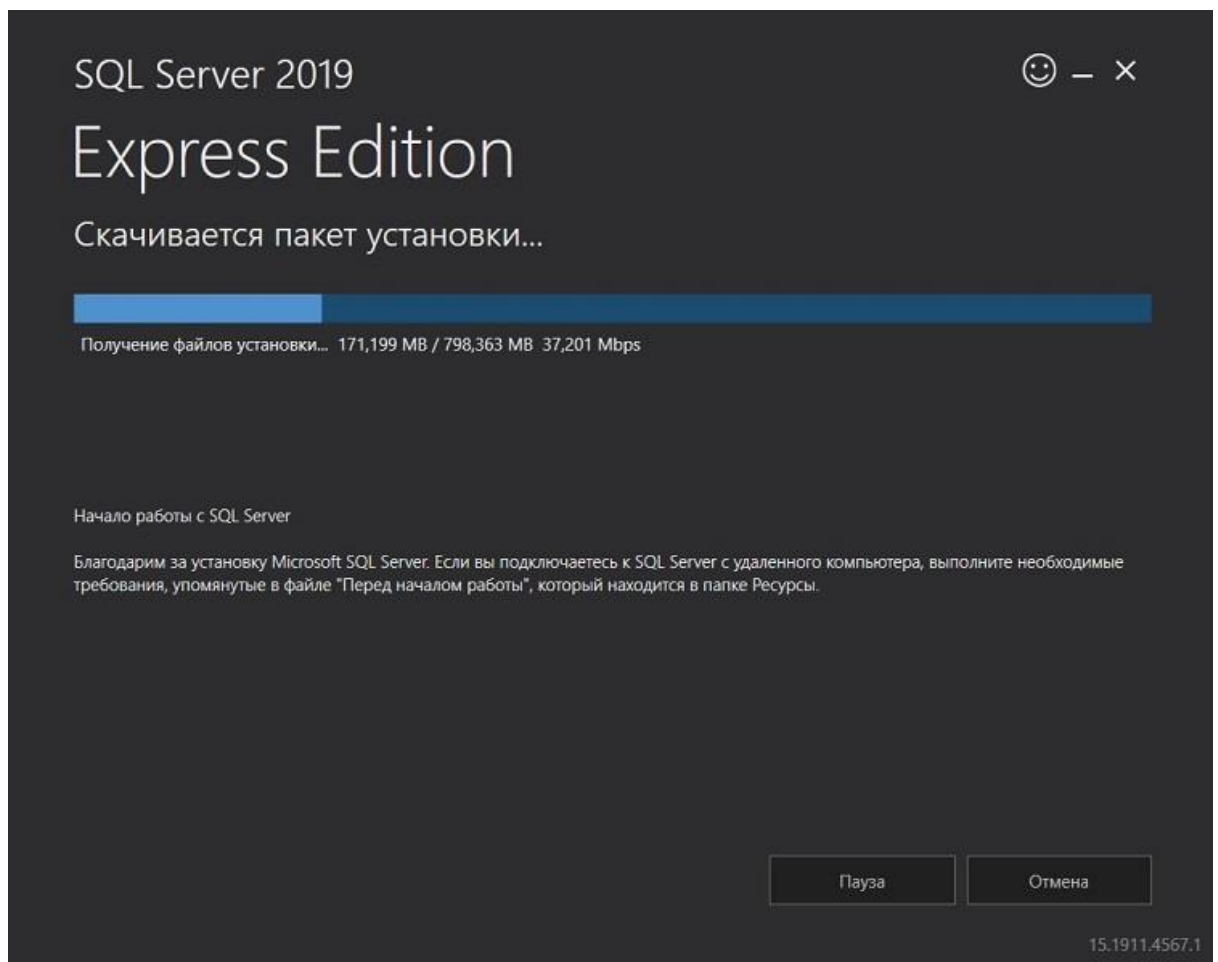


Рисунок 11. Процесс установки SQL Server Express.

Sql Server Express успешно установлен.

Visual Studio 2022

Visual Studio - это интегрированная среда разработки (IDE) от компании Microsoft, которая используется для создания программного обеспечения для различных платформ, включая Windows, macOS, Linux, Android и iOS. Visual Studio предоставляет разработчикам множество инструментов и функций, которые позволяют быстро создавать и отлаживать приложения, включая возможности для написания кода, визуального проектирования пользовательских интерфейсов, автоматической проверки кода, отладки приложений и тестирования.

Среди языков программирования, поддерживаемых Visual Studio, наиболее популярными являются C++, C#, Visual Basic, Python, JavaScript и TypeScript. Visual Studio также включает в себя инструменты для создания веб-приложений, включая ASP.NET и Razor, а также возможность создания приложений для платформы Windows Universal, мобильных устройств и игр.

Visual Studio имеет много полезных функций для коллективной работы, включая системы контроля версий и средства совместной разработки, что позволяет разработчикам работать в команде и синхронизировать свои работы.

Для загрузки Visual Studio перейдём на официальный сайт Microsoft, на страницу установщика Visual Studio 2022 Community Edition (Рисунок 12).

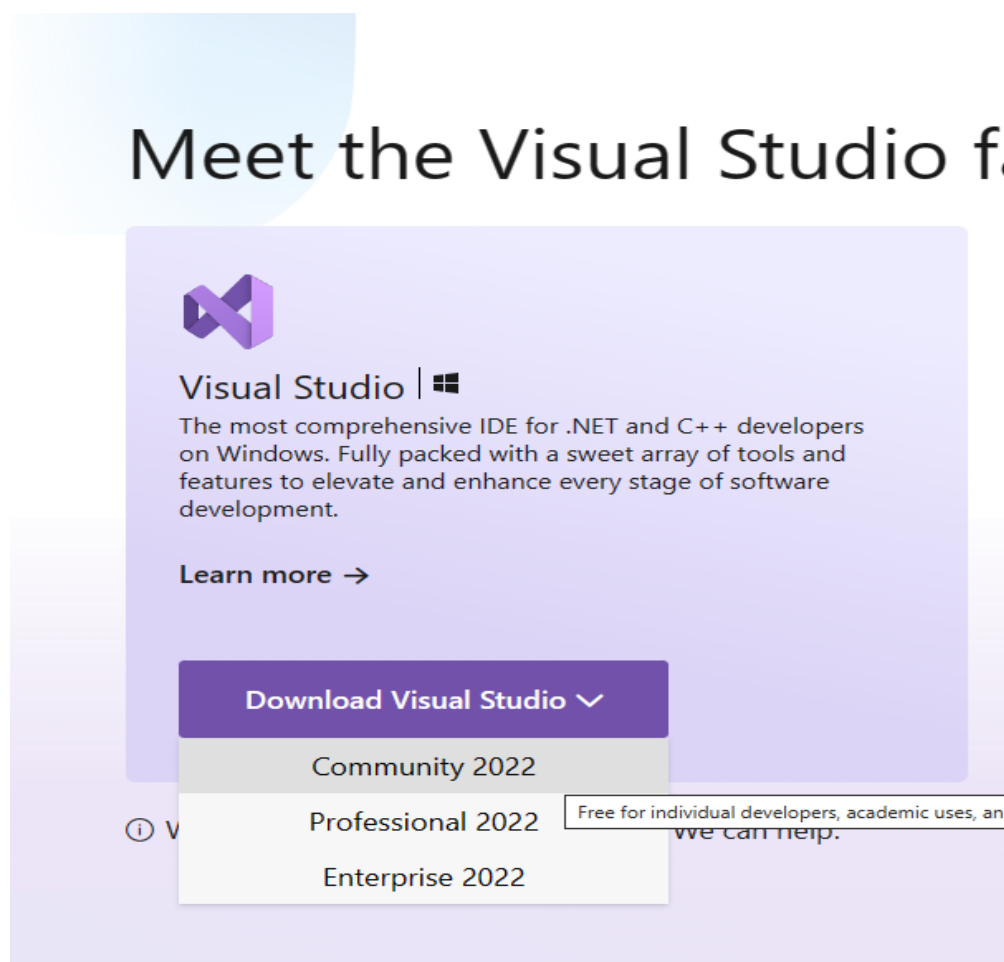


Рисунок 12. Страница загрузки инсталлятора Visual Studio 2022 Community Edition.

Дважды щёлкаем по иконке установщика, принимаем условия лицензионного соглашения по нажатию кнопки «Continue» (Рисунок 13).

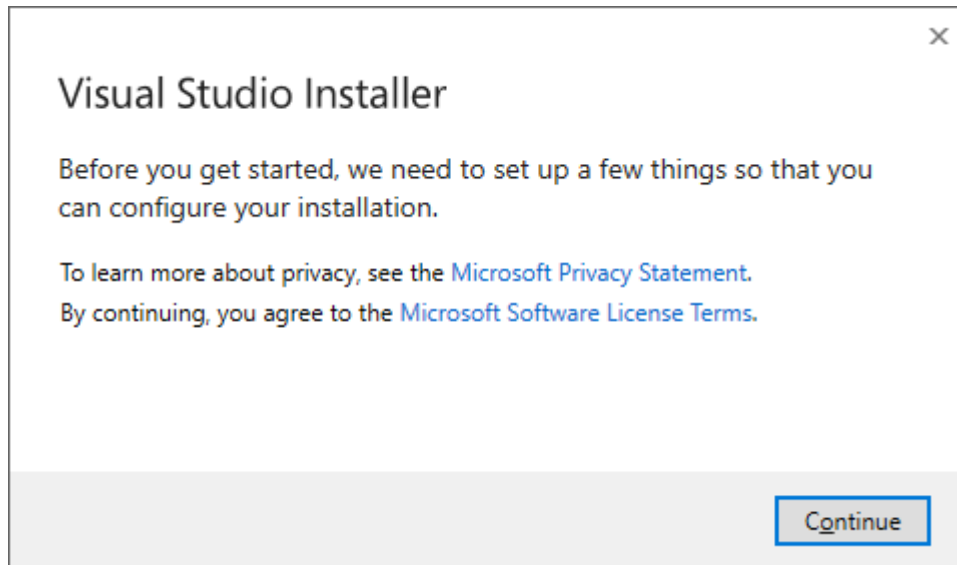


Рисунок 13. Принятие лицензионного соглашения.

После установки Visual Studio Installer его можно использовать для настройки установки, выбрав нужные наборы компонентов или рабочие нагрузки. Вот как это сделать (Рисунок 14).

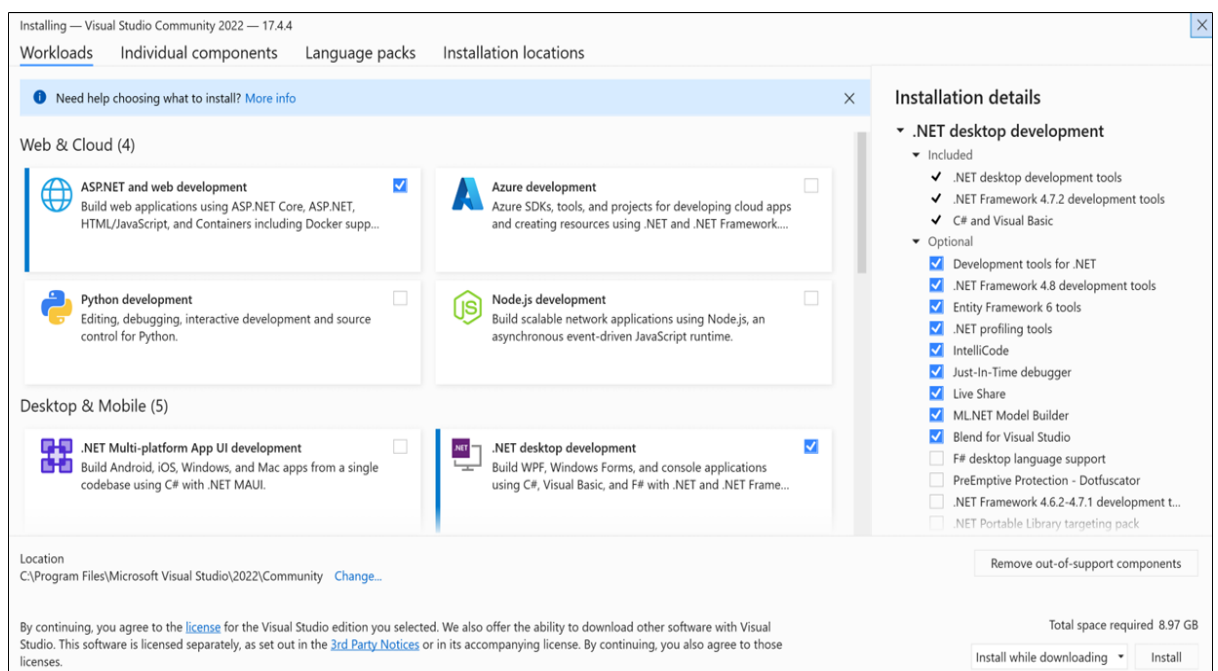


Рисунок 14. Выбор компонентов для установки.

Выбираем необходимые нам для работы компоненты и нажимаем «Install». Далее будут отображаться экраны состояния, на которых демонстрируется ход установки Visual Studio (Рисунок 15).

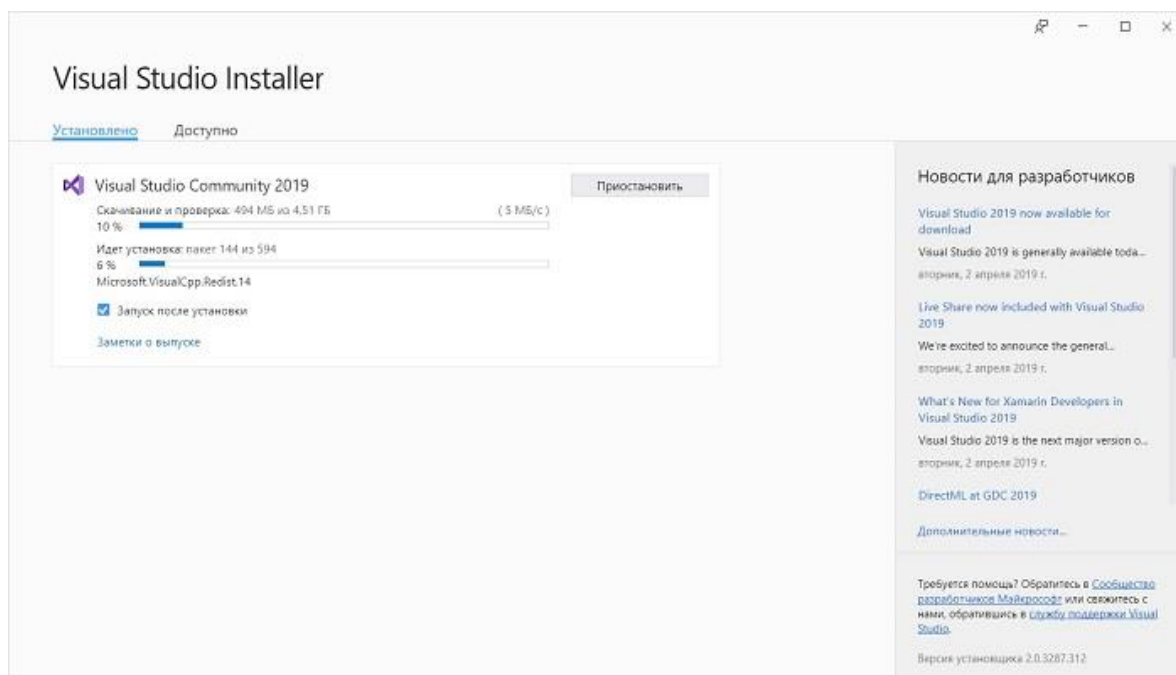


Рисунок 15. Процесс установки Visual Studio 2022.

Как только Visual Studio Installer закончит загрузку, необходимо будет перезагрузить компьютер, перезагружаемся (Рисунок 16).

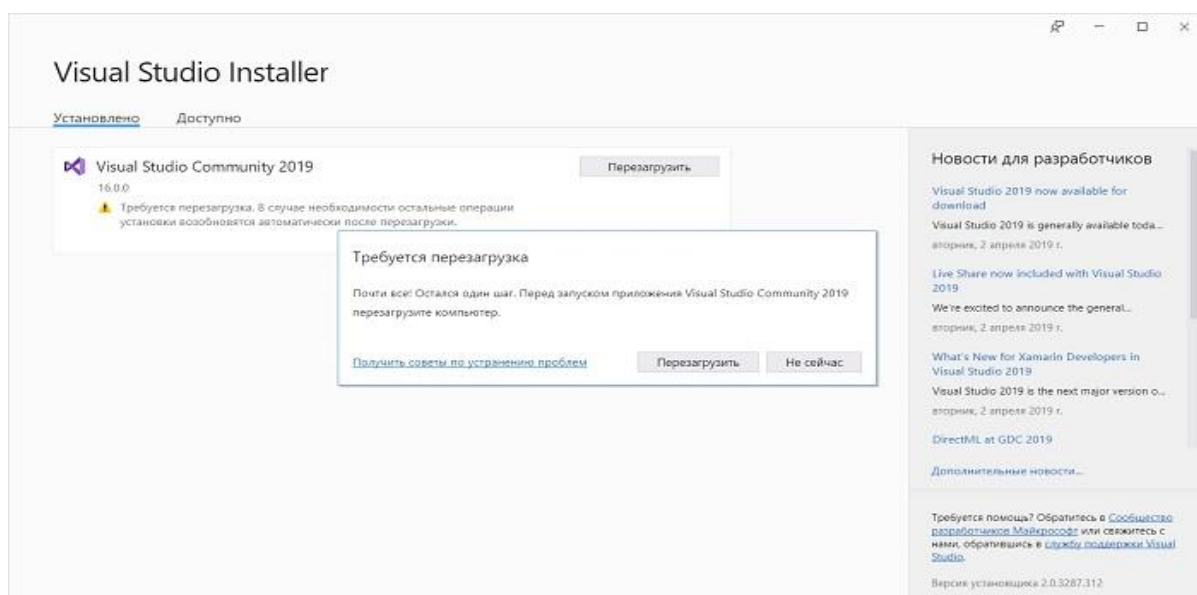


Рисунок 16. Завершение установки.

Visual Studio 2022 Успешно установлен.

Создание ERD-диаграммы в Visio

В Visio необходимо создать представление информационной системы автосалона, для этого нужно запустить программу и выбрать нужный шаблон (Рисунок 17).

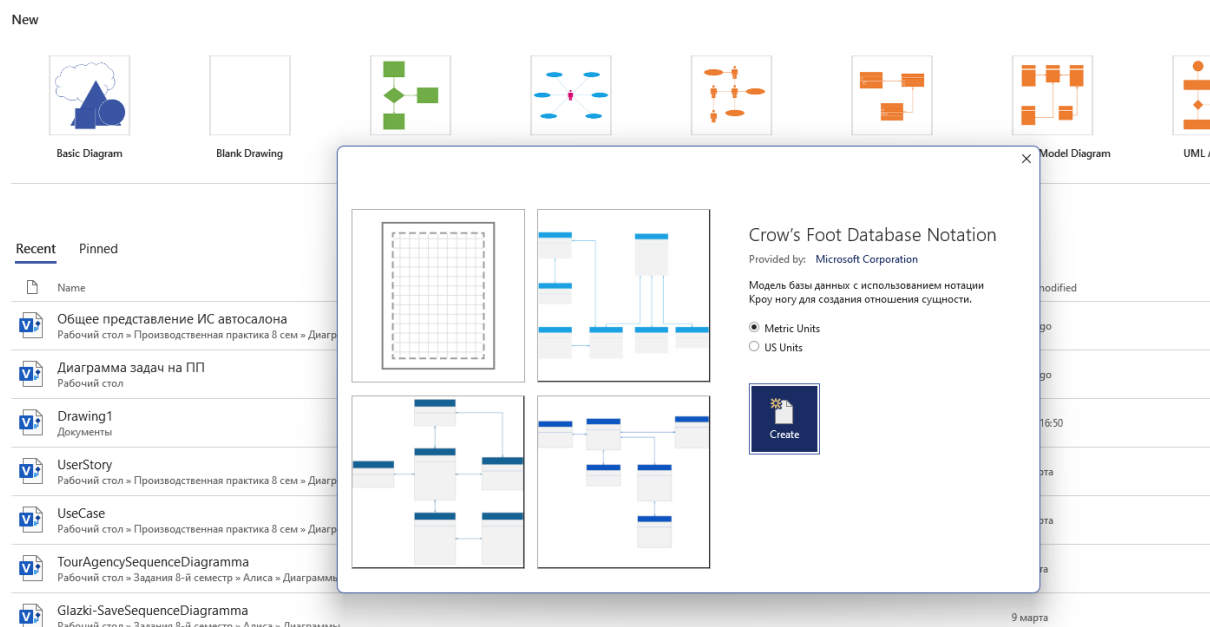


Рисунок 17. Выбор шаблона.

С помощью представленных наборов фигур составим таблицы, необходимые для работы с информационной системой Автосалона (Рисунок 18).

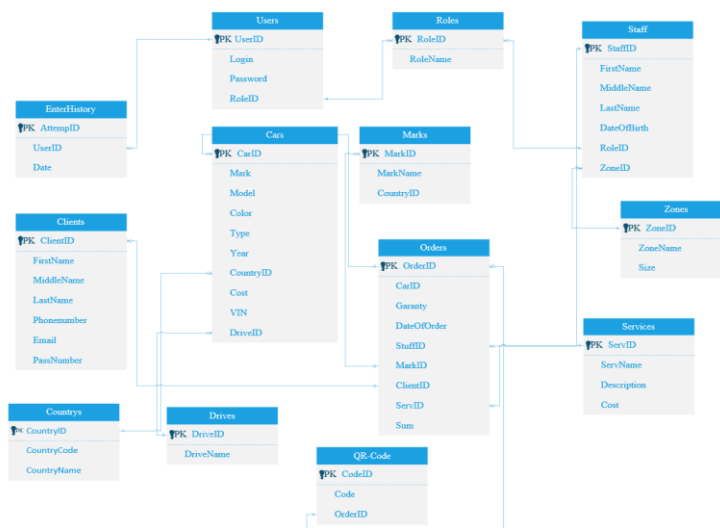


Рисунок 18. Готовая ERD диаграмма автосалона.

Таблица приводов (передний, задний, полный): Таблица приводов используется для хранения информации о типах приводов автомобилей, которые предлагаются в автосалоне. Эта таблица обычно содержит список всех возможных типов приводов, таких как передний, задний и полный, и используется для классификации автомобилей и поиска по типу привода.

Таблица заказов: Таблица заказов используется для хранения информации о заказах клиентов на автомобили. Эта таблица может содержать информацию о дате и времени заказа, автомобиле, на который сделан заказ, статусе заказа и информации о клиенте.

Таблица машин: Таблица машин используется для хранения информации о каждом автомобиле, который продается в автосалоне. Эта таблица может содержать информацию о марке и модели автомобиля, годе выпуска, типе кузова, типе привода, количестве мест и другой информации, которая поможет описать автомобиль.

Таблица пользователей: Таблица пользователей используется для хранения информации о пользователях информационной системы автосалона. Эта таблица может содержать информацию о имени пользователя, электронной почте, пароле и других аутентификационных данных.

Таблица сотрудников: Таблица сотрудников используется для хранения информации о сотрудниках автосалона. Эта таблица может содержать информацию о имени, фамилии, должности, зарплате и другой информации, которая поможет управлять персоналом автосалона.

Таблица зон: Таблица зон используется для хранения информации о зонах работы сотрудников. Эта таблица может содержать информацию о названии зоны и описании ее границ.

Таблица ролей: Таблица ролей используется для хранения информации о ролях пользователей в информационной системе автосалона. Эта таблица может содержать информацию о ролях, таких как

администратор, менеджер и продавец, и об их правах доступа к функционалу системы.

Таблица истории входа: Таблица истории входа используется для хранения информации о входах пользователей в информационную систему автосалона. Эта таблица может содержать информацию о времени входа, имени пользователя и IP-адресе, с которого был сделан вход.

Таблица клиентов: Таблица клиентов используется для хранения информации о клиентах автосалона.

User Story

User Story (продуктовая история) - это короткая и простая форма описания требований к функциональности продукта или сервиса, написанная в виде истории из глаз пользователя. User Story помогает команде разработчиков продукта понять, какую задачу нужно выполнить и для какой целевой аудитории. На рисунке 19 представлен User Story для автосалона.

[User Stories: руководство для разработчиков

При разработке User Stories вам необходимо использовать технологию разработки программного обеспечения BDD (сокр. от англ. Behavior-driven development, дословно «разработка через поведение») — это методология разработки программного обеспечения, являющаяся ответвлением от методологии разработки через тестирование (TDD)).

User story вкратце описывает:

- человека, использующего систему (заказчик);
- то, что должно содержаться в данной системе (примечание);
- то, для чего она нужна пользователю (цель).

Для разработки используйте следующий шаблон:

История: Автомобили в наличии

Как Клиент

Я хочу иметь возможность просматривать автомобили в наличии

Чтобы я мог увидеть автомобили в наличии, готовые к продаже

Сценарий: Данные о доступных автомобилях получена

Дано: клиент открыл программу

Когда страница с автомобилями открыта

Тогда Информация о доступных автомобилях получена

Сценарий: Данные о доступных автомобилях не получены

Дано: открыл программу

Когда страница с автомобилями вернула ошибку

Тогда не удалось вывести автомобили в наличии

Рисунок 19. User Story клиента.

Use case

Use Case (случай использования) - это сценарий или описание того, как конкретный пользователь может использовать определенное приложение, систему или продукт для достижения определенной цели или решения определенной проблемы.

В рамках разработки программного обеспечения, Use Case является частью методологии разработки, которая позволяет описывать, как система будет использоваться в реальной жизни. Use Case может содержать информацию о том, кто будет использовать систему, как они будут ее использовать, какую информацию они вводят и получают из системы, и какие результаты они ожидают.

Use Case также может быть использован для описания бизнес-процессов, чтобы определить, какие роли и акторы будут участвовать в процессе, какие шаги они будут предпринимать и какие решения они будут принимать. Это позволяет бизнес-аналитикам и разработчикам создать систему, которая эффективно поддерживает бизнес-процессы и требования пользователей.

Ниже приведён пример Use Case диаграммы, на которой показаны возможности администратора, менеджера и клиента

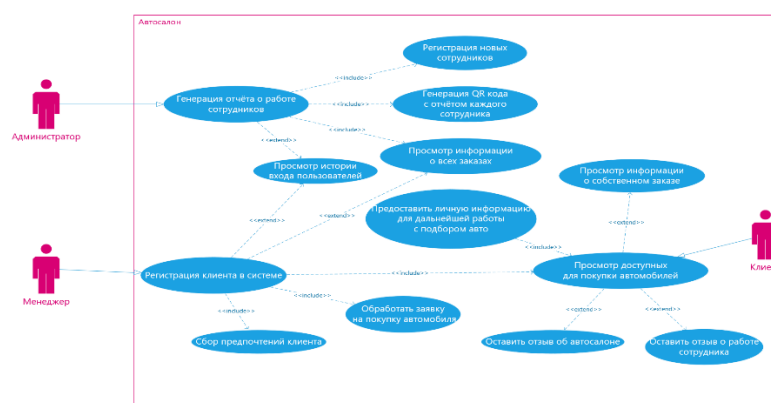


Рисунок 20. Use case автосалона.

Разработка базы данных автосалона

База данных автосалона представляет собой систему хранения и организации информации о продаваемых автомобилях, клиентах, заказах и других связанных с продажей автомобилей данных.

У нас уже есть ERD диаграмма которая является важным ключём в создании БД. На её основе мы создадим таблицы, поставим нужные ключи и типы данных, для этого запустим Microsoft SQL Server Management Studio с помощью специального значка (Рисунок 21).

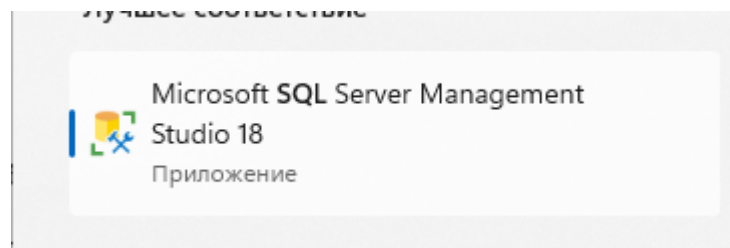


Рисунок 21. Значёк запуска приложения

Вводим данные для подключения, которые мы получили при установке SQL Server Express (Рисунок 22).

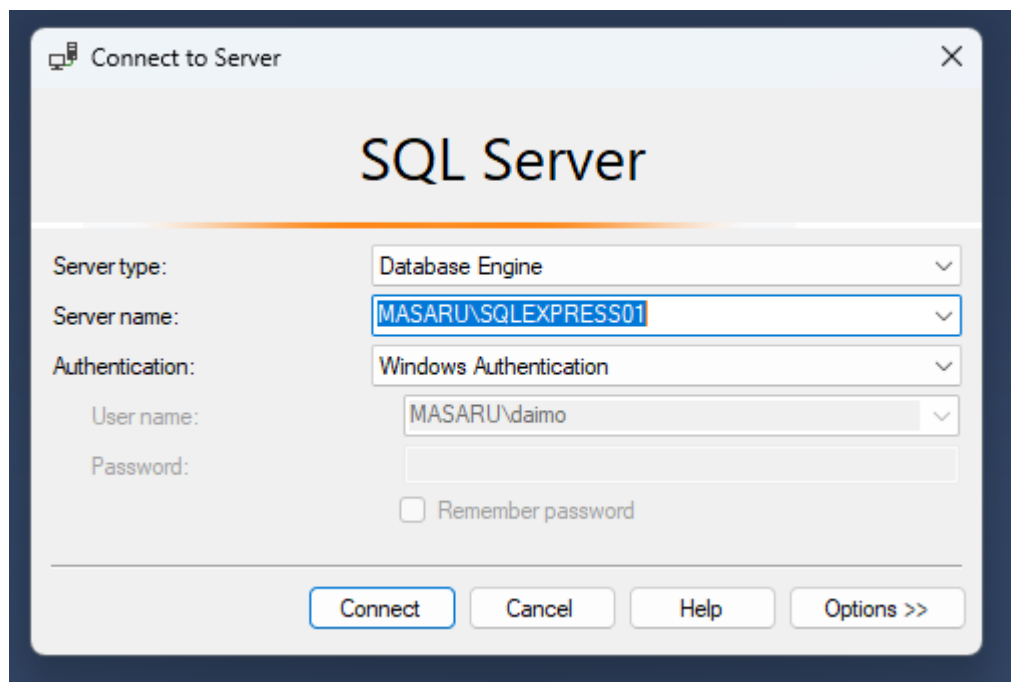


Рисунок 22. Подключение к ядру БД.

После подключения нажимаем правой кнопкой мыши по папке «DataBases» и выбираем «New Database» (Рисунок 23).

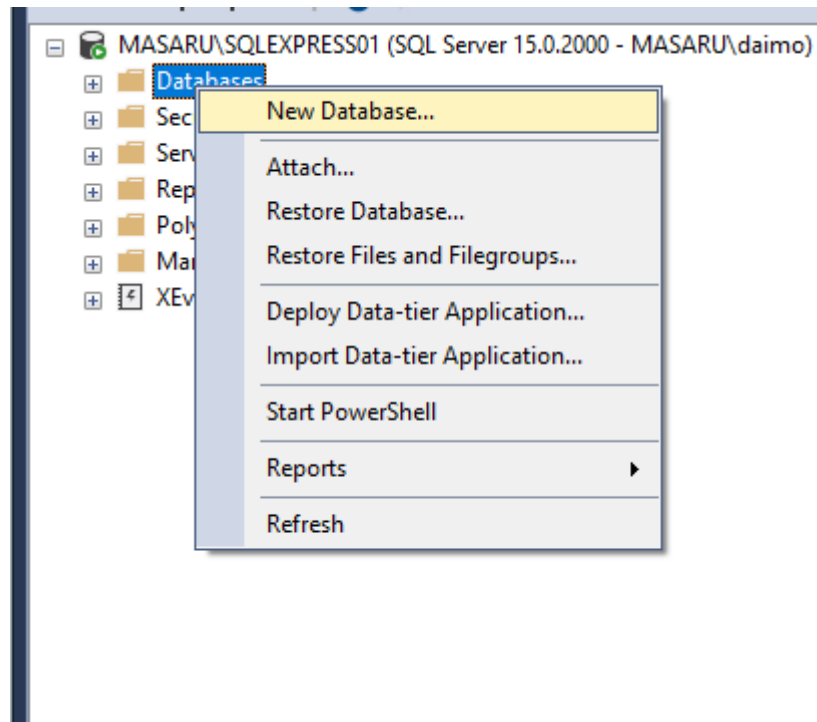


Рисунок 23. Создание новой базы данных.

Вводим название БД и переходим к созданию таблиц (Рисунок 24).

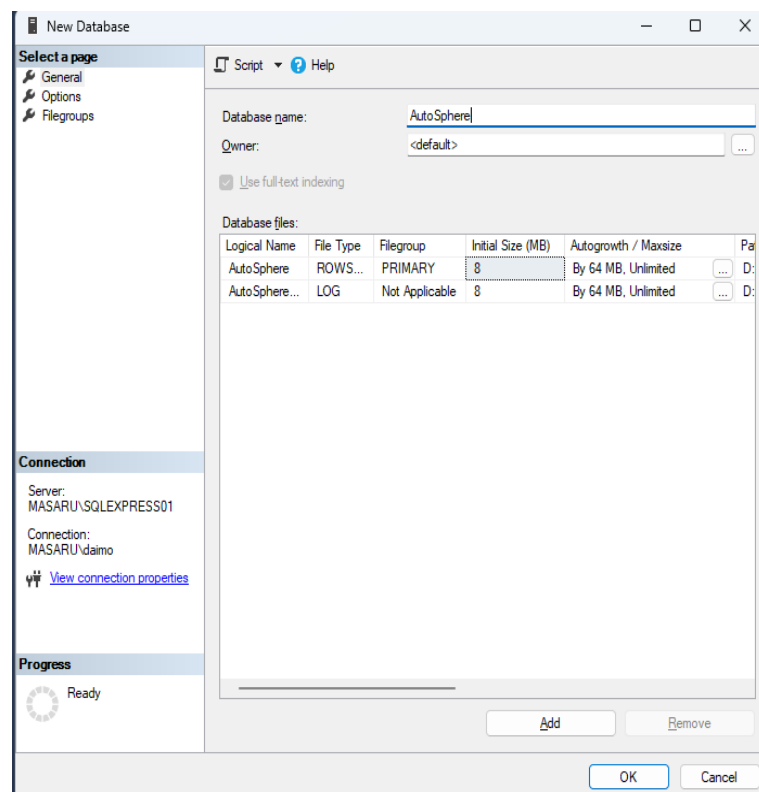


Рисунок 24. Выбор имени базы данных.

После успешного создания БД нажимаем правой кнопкой мыши по папке «Tables» и выбираем пункт «New Table» (Рисунок 25).

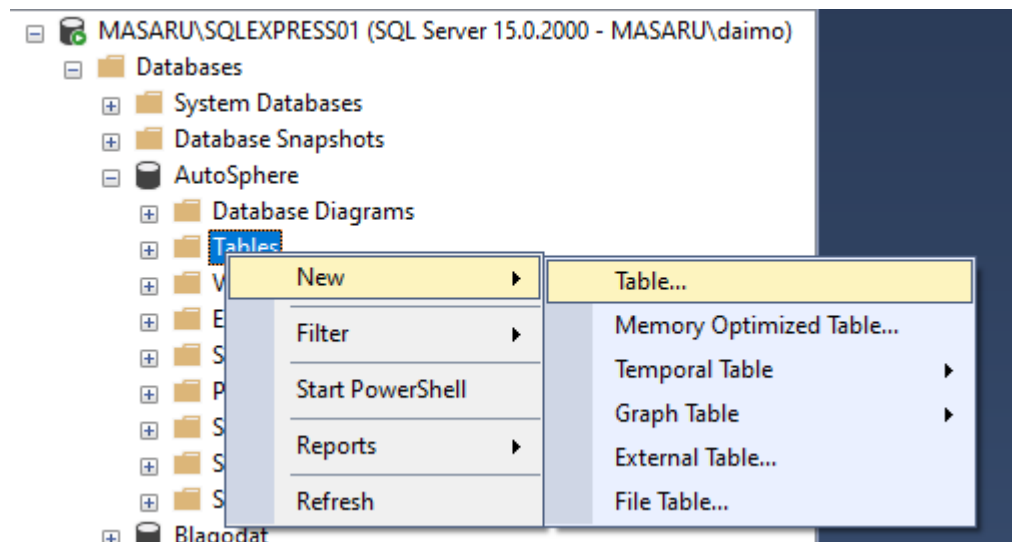


Рисунок 25. Создание новой таблицы.

Перед нами открывается окно с созданием таблицы, в которой нам необходимо выбрать название столбца и его тип данных (Рисунок 26).

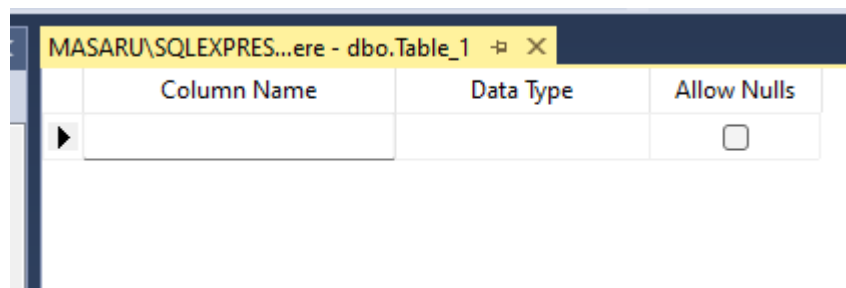


Рисунок 26. Создание таблицы.

Создадим таблицу машин, для этого определим названия столбцов и их типов данных. Первый столбец в основном используется для нумерации объекта и этому столбцу присваивается первичный ключ (Рисунок 27).

Column Name	Data Type	Allow Nulls
CarID	int	<input type="checkbox"/>
Mark	nvarchar(50)	<input checked="" type="checkbox"/>
Model	nvarchar(50)	<input checked="" type="checkbox"/>
Color	nvarchar(50)	<input checked="" type="checkbox"/>
Type	nvarchar(50)	<input checked="" type="checkbox"/>
Year	int	<input checked="" type="checkbox"/>
CountryID	int	<input checked="" type="checkbox"/>
Cost	money	<input checked="" type="checkbox"/>
VIN	nvarchar(50)	<input checked="" type="checkbox"/>
DriveID	int	<input checked="" type="checkbox"/>
Actuality	nvarchar(50)	<input checked="" type="checkbox"/>

Рисунок 27. Таблица машин.

Мы определили такие столбцы как:

- Номер машины
- Марка
- Модель
- Цвет
- Тип кузова
- Год выпуска
- Номер страны производства
- Цена
- ВИН номер
- Номер привода
- Наличие

В созданной нами таблице будет храниться информации об автомобилях, год выпуска, марка, модель и так далее. В дальнейшем, пользователь сможет просматривать записи из этой таблицы, в том числе и изображения, а так же производить выборку по автомобилям, например вывести список только актуальных автомобилей.

Аналогичным образом были созданы оставшиеся таблицы в БД, которые были представлены на ERD диаграмме (Рисунок 28).

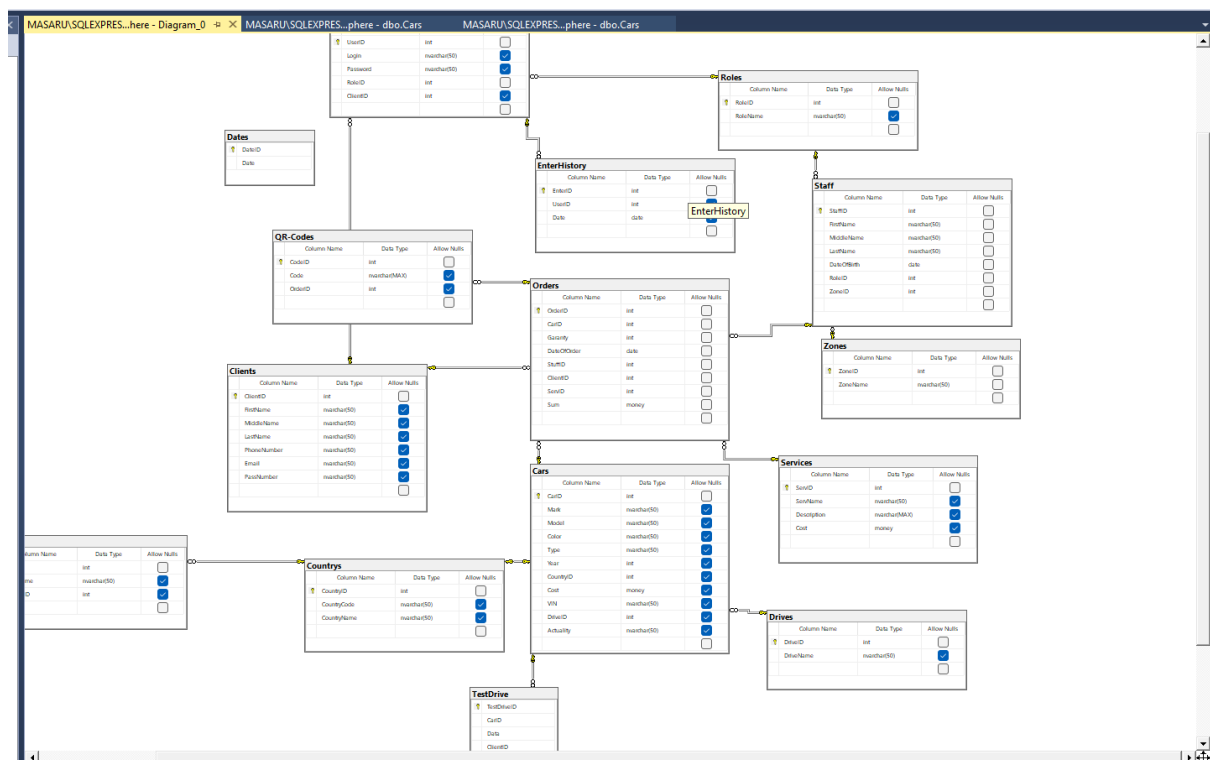


Рисунок 28. Диаграмма базы данных созданная в SQL Server Management Studio.

База данных автосалона успешно создана

Разработка WPF приложения

WPF (Windows Presentation Foundation) - это технология разработки графического интерфейса пользователя (GUI) для Windows приложений, которая предоставляет широкие возможности для создания интерактивных и multifunctional приложений.

1. Для работы с базой данных автосалона, WPF приложение может быть использовано для реализации следующих функций:
2. Отображение списка автомобилей и их характеристик из базы данных, таких как модель, год выпуска, цена и т.д.
3. Добавление новых автомобилей в базу данных.
4. Редактирование и обновление информации об уже существующих автомобилях в базе данных.
5. Удаление автомобилей из базы данных.

6. Поиск и фильтрация автомобилей по различным параметрам, таким как модель, год выпуска, цена и т.д.

Для создания WPF приложения для работы с базой данных автосалона можно использовать Visual Studio, интегрированную с Microsoft SQL Server для хранения и управления данными.

Для начала необходимо создать проект WPF в Visual Studio и настроить подключение к базе данных автосалона. Для этого можно использовать инструменты Server Explorer и Data Source Configuration Wizard в Visual Studio.

Затем можно создать различные элементы управления, такие как таблицы, списки, формы и кнопки, для отображения и редактирования данных из базы данных. В коде приложения можно использовать язык SQL для выполнения запросов к базе данных, а также LINQ (Language Integrated Query) для работы с данными и объектами приложения.

Для улучшения пользовательского интерфейса и удобства работы с приложением можно использовать стили и темы оформления WPF, а также добавить функции автозаполнения, валидации вводимых данных и диалоговые окна для подтверждения действий пользователя.

После создания приложения и проверки его работоспособности можно развернуть его на целевом устройстве, таком как ПК или планшет, для использования сотрудниками автосалона.

Перейдём в ранее установленное приложение Visual Studio 2022 для создания приложения, создадим новый проект с шаблоном WPF (Рисунок 29).

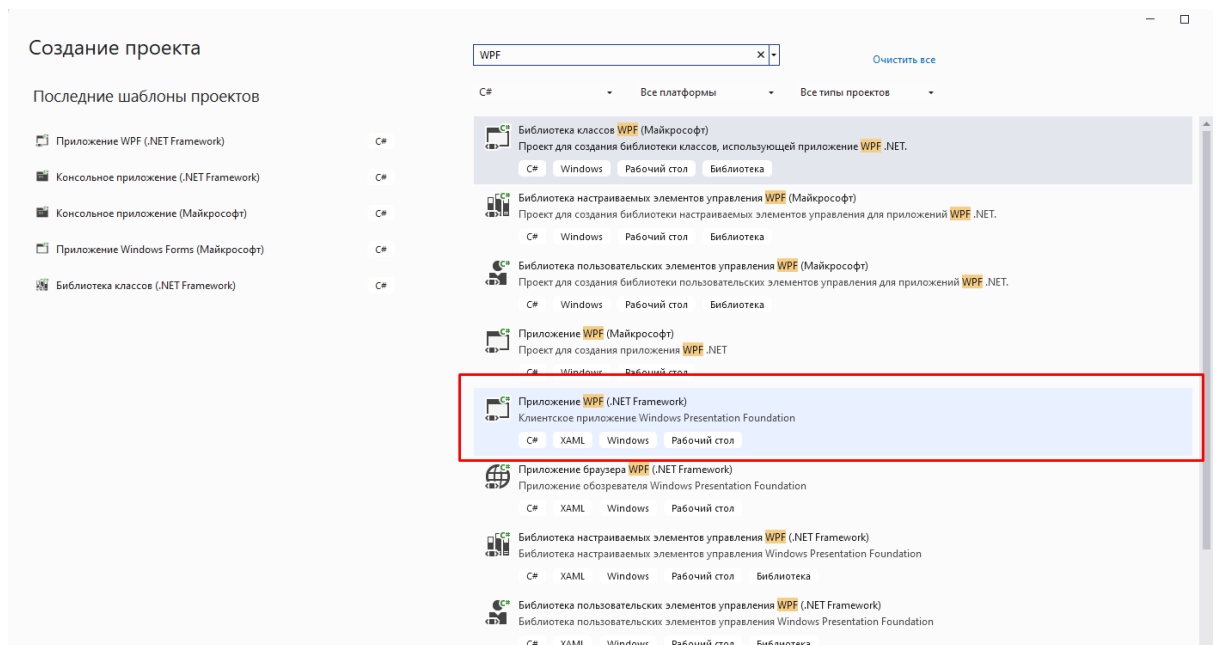


Рисунок 29. Создание нового WPF проекта

Вводим название проекта и жмём кнопку «Создать» (Рисунок 30).

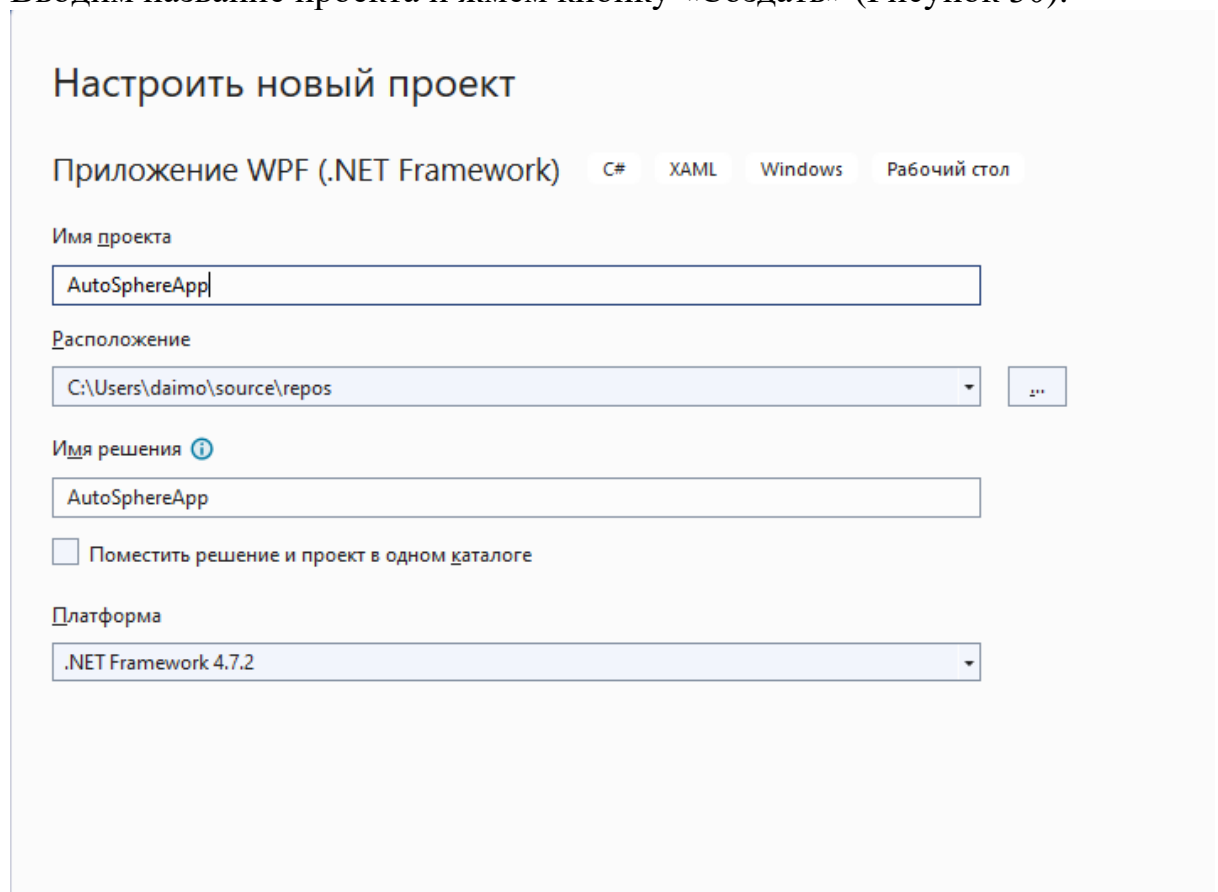


Рисунок 30. Название проекта

После загрузки нужных компонентов перед нами откроется главное окно приложения для разработки пользовательского интерфейса. С

помощью XAML разметки создадим главное окно приложения, «шапку» и «футер», как на сайтах (Рисунок 31).

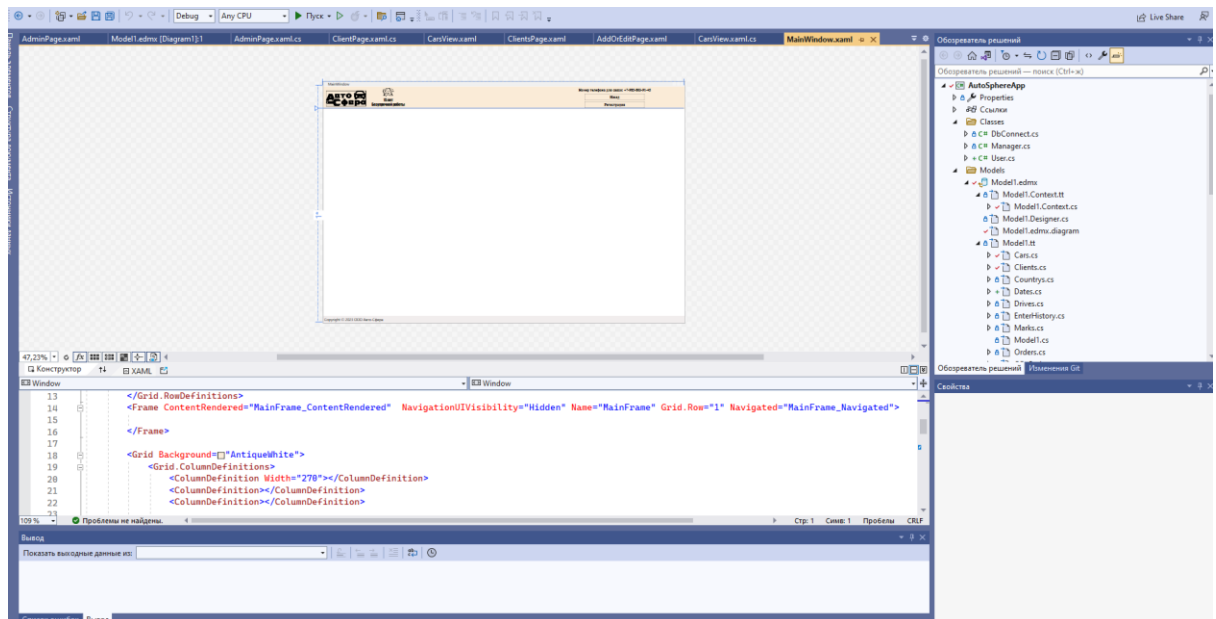


Рисунок 31. Главное окно Visual Studio 2022

С помощью XAML разметки создаём Grid и Frame, нужны кнопки, добавляем логотип кампании. Ниже приведён полный код страницы:

```
<Window x:Class="AutoSphereApp.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:local="clr-namespace:AutoSphereApp"
mc:Ignorable="d"
Title="Main Window" Height="800" Width="1200">
<Grid Background="White">
<Grid.RowDefinitions>
<RowDefinition Height="75" x:Name="Head"></RowDefinition>
<RowDefinition Height="*"></RowDefinition>
</Grid.RowDefinitions>
<Frame ContentRendered="MainFrame_ContentRendered" NavigationUIVisibil-
ity="Hidden" Name="MainFrame" Grid.Row="1" Navigated="MainFrame_Navigated">
</Frame>
<Grid Background="AntiqueWhite">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="270"></ColumnDefinition>
<ColumnDefinition></ColumnDefinition>
<ColumnDefinition></ColumnDefinition>
</Grid.ColumnDefinitions>
</Grid>
</Window>
```

```

        <Button Name="QuitButton" Visibility="Collapsed" Content="Выход"
Grid.Column="1" Click="Button_Click"></Button>
        <Image HorizontalAlignment="Left" Source="\Resources\Autosphere-
Logotip.png" Width="150" RenderTransformOrigin="0.548,-0.008" Margin="0,0,0,-
7"/>

        <StackPanel Height="auto" HorizontalAlignment="Right" Grid.Col-
umn="0">

            <Image Source="Resources\guaranty.png" HorizontalAlignment="Cen-
ter" Height="37" Width="60" Stretch="Fill"></Image>
            <TextBlock HorizontalAlignment="Center" Grid.Row="1" TextWrap-
ping="Wrap" Text="15 лет" VerticalAlignment="Top" FontFamily="Impact"/>
            <TextBlock HorizontalAlignment="Left" Grid.Row="1" TextWrap-
ping="Wrap" Text="Безупречной работы" VerticalAlignment="Top" FontFamily="Im-
pact"/>

        </StackPanel>

        <StackPanel Grid.Column="3">
            <Label Grid.Column="3" VerticalAlignment="Top" Width="NaN" Hori-
zontalAlignment="Center" Content="Номер телефона для связи: +7-985-803-91-45"
FontFamily="Times New Roman" Height="25" FontWeight="Bold"/>
            <Button FontWeight="Bold" FontFamily="Times New Roman"
Name="BackButton" Background="AntiqueWhite" Width="200" Content="Назад"
Height="20" Click="BackButton_Click" ></Button>
            <Button FontWeight="Bold" FontFamily="Times New Roman" Back-
ground="AntiqueWhite" Name="RegistrationButton" Width="200" Con-
tent="Регистрация" Height="20" Click="RegistrationButton_Click" Margin="5"
></Button>
        </StackPanel>
    </Grid>
    <DockPanel Grid.Row="1" VerticalAlignment="Bottom" Background="Antique-
White">
        <StatusBar DockPanel.Dock="Bottom">
            <TextBlock Text="Copyright © 2023 ООО Авто-Сфера" />

        </StatusBar>
    </DockPanel>
    <!-- Основное содержимое окна -->
    </Grid>
</Grid>
</Window>

```

Этот код является частью XAML-файла главного окна (MainWindow) WPF-приложения для работы с автосалоном.

На первых строках определяются пространства имен (xmlns), которые будут использоваться в дальнейшем, включая пространства имен для основных элементов управления в WPF (PresentationFramework) и для локальных ресурсов приложения.

Затем определяется Grid-контейнер (элемент управления, который упорядочивает другие элементы в виде сетки), содержащий две строки.

Первая строка содержит Grid.RowDefinition для определения размера первой строки, содержащей элементы управления, такие как логотип автосалона.

Вторая строка имеет высоту "*", что означает, что она займет всю оставшуюся высоту окна.

В этой строке находятся три элемента управления:

1. **Frame** - элемент управления, который позволяет встраивать страницы в приложение.
2. **Grid** - элемент управления, который содержит другие элементы управления, включая логотип, кнопку выхода, номер телефона для связи и кнопки навигации.

Теперь, нам необходимо настроить навигацию, для этого мы будем использовать класс «Manager», в котором мы определим ранее созданную в XAML разметке рамку, в которой будет происходить «прорисовка» контента с других страниц. Для создания класса выберем правой кнопкой мыши наш проект и создадим новую папку «Classes» и уже в неё добавим новый класс «Manager» (Рисунок 32).

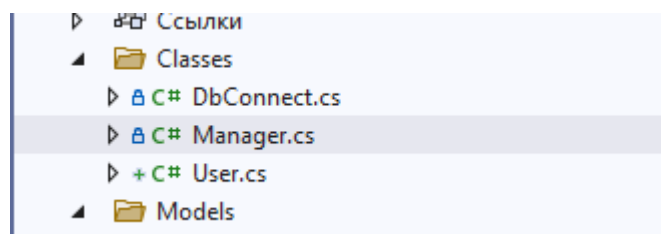


Рисунок 32. Папка Classes и класс Manager

Напишем следующий код в класс Manager:

```
class Manager
{
    public static Frame MainFrame { get; set; }
}
```

Этот код определяет класс `Manager`, в котором объявляется статическое свойство `MainFrame` типа `Frame`. Это свойство позволяет получить доступ к `Frame` в различных частях приложения, таких как окна и страницы, и использовать его для навигации между различными страницами в приложении. Благодаря использованию статического свойства, `MainFrame` может быть доступен для всех экземпляров класса `Manager` без необходимости создания новых экземпляров.

Теперь нам необходимо создать подключение к БД и класс подключения. Добавляем класс `Dbconnect` в папку `Classes` и создаём публичный метод:

```
public static Models.AutoSphereEntities modeldb;
```

Эта строка отвечает за подключение добавленной модели к нашему проекту. Теперь нам необходимо создать папку `Models` и добавить туда модель нашей БД. Для этого нажимаем правой кнопкой мыши по папке `Models` → Добавить

В поиске вводим «ADO» и нажимаем «Модель ADO.NET EDM» (Рисунок 33).

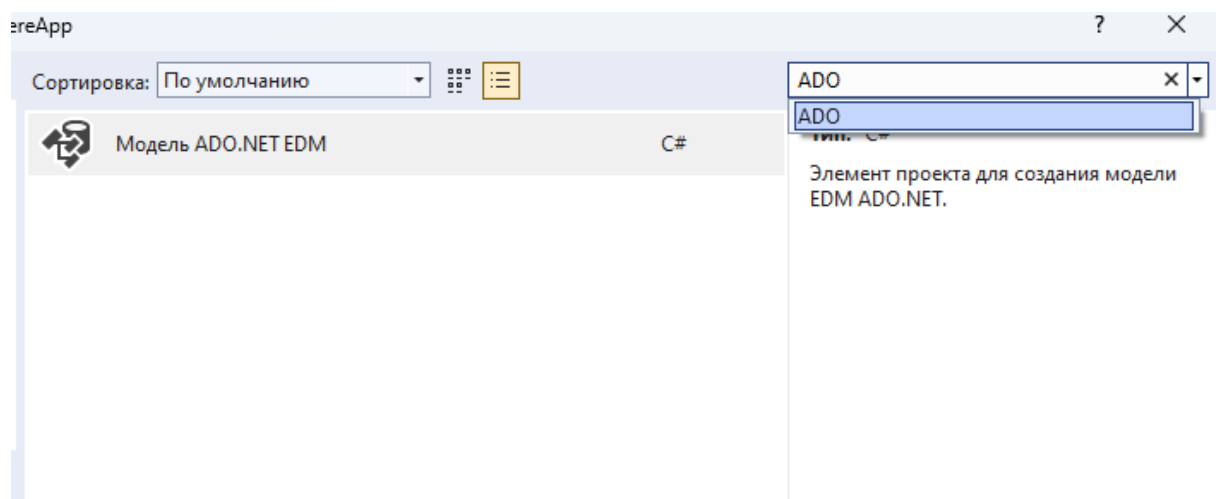


Рисунок 33. Добавление модели БД.

Следуя всем инструкциям, вводим название нашей БД и добавляем в папку `Models` (Рисунок 34).

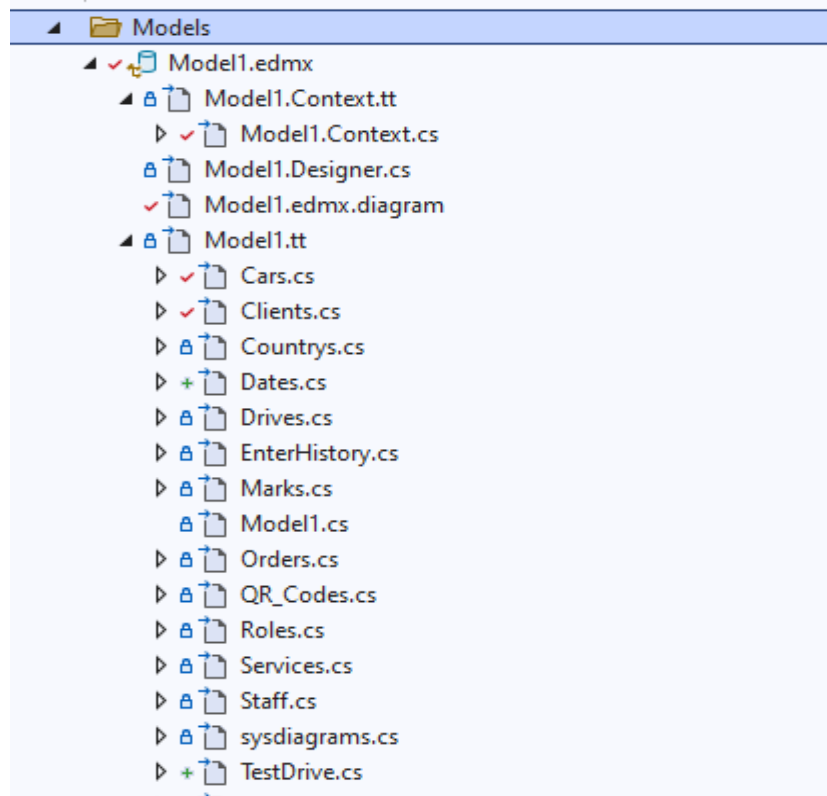


Рисунок 34. Подключенная модель

Теперь, написанный нами ранее код подключения, а именно класс DbConnect будет работать исправно и на страницах далее, нам нужно будет обращаться именно к этому классу для подключения к БД.

Следующим шагом в подключении БД является редактирование файла Model1.Context.cs. Переходим к странице кода и дописываем следующий код:

```
public partial class AutoSphereEntities : DbContext
{
    private static AutoSphereEntities _context;
    public AutoSphereEntities()
        : base("name=AutoSphereEntities")
    {
    }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        throw new UnintentionalCodeFirstException();
    }
    public static AutoSphereEntities GetContext()
    {
        if (_context == null)
            _context = new AutoSphereEntities();
        return _context;
    }

    public virtual DbSet<Cars> Cars { get; set; }
}
```

```

public virtual DbSet<Clients> Clients { get; set; }
public virtual DbSet<Country> Country { get; set; }
public virtual DbSet<Drives> Drives { get; set; }
public virtual DbSet<EnterHistory> EnterHistory { get; set; }
public virtual DbSet<Marks> Marks { get; set; }
public virtual DbSet<Orders> Orders { get; set; }
public virtual DbSet<QR_Codes> QR_Codes { get; set; }
public virtual DbSet<Roles> Roles { get; set; }
public virtual DbSet<Services> Services { get; set; }
public virtual DbSet<Staff> Staff { get; set; }
public virtual DbSet<sysdiagrams> sysdiagrams { get; set; }
public virtual DbSet<Users> Users { get; set; }
public virtual DbSet<Zones> Zones { get; set; }
public virtual DbSet<TestDrive> TestDrives { get; set; }
public virtual DbSet<Dates> Dates { get; set; }
public static Users currentuser = null;
}

```

Этот код определяет класс `AutoSphereEntities`, который наследуется от класса `DbContext` в Entity Framework. Этот класс представляет контекст базы данных и содержит свойства `DbSet` для каждой таблицы в базе данных, а также статический метод `GetContext()`, который возвращает экземпляр контекста базы данных. Конструктор класса инициализирует базовый класс `DbContext` и задает имя подключения к базе данных. Метод `OnModelCreating` используется для настройки модели базы данных. Строка `throw new UnintentionalCodeFirstException();` генерирует исключение, если модель базы данных была создана неправильно.

Теперь наша БД полностью подключена к проекту.

Код приложения

Страница авторизации

Страница авторизации в WPF приложении представляет собой графический интерфейс, который позволяет пользователю ввести свои учетные данные (например, логин и пароль) для входа в приложение. Она обычно содержит следующие элементы:

1. Поля ввода: текстовые поля для ввода имени пользователя (логина) и пароля.

2. Кнопка "Войти": пользователь может нажать на эту кнопку, чтобы отправить данные авторизации на сервер и войти в приложение.
3. Чек бокс «Показать пароль»

Страница авторизации может быть разработана в соответствии с корпоративным стилем приложения или в соответствии с общепринятыми стандартами дизайна пользовательского интерфейса. Она должна быть интуитивно понятной, легко доступной и удобной для использования. Также важно обеспечить безопасность передачи данных пользователя (например, через протокол HTTPS) и защитить приложение от возможных атак на авторизацию (например, через использование капчи или двухфакторной аутентификации).

Начнём с XAML разметки страницы авторизации. Создаём новую страницу и прописываем следующий код:

```
<Grid>
    <StackPanel Width="300">
        <Image Source="Resources\loginImage.png" Height="150" Width="234"
Margin="35" ></Image>
        <Label FontSize="20" Content="Авторизация" HorizontalAlignment="Cen-
ter" FontFamily="Impact"></Label>
        <Label HorizontalAlignment="Center" Content="Логин:" FontFamily="Im-
pact"></Label>
        <TextBox Name="LoginTB" HorizontalAlignment="Center" Width="200"
></TextBox>
        <Label FontFamily="Impact" Content="Пароль" HorizontalAlignment="Cen-
ter" ></Label>
        <TextBox Name="TxbPassword" Width="200" Visibility="Collapsed"
></TextBox>

        <PasswordBox Width="200" Name="Password"></PasswordBox>
        <CheckBox Name="CheckBoxPass" FontFamily="Impact" Unchecked="Check-
Box_Unchecked" HorizontalAlignment="Center" Content="Показать пароль" Margin="5"
Checked="CheckBox_Checked"></CheckBox>
        <Canvas Name="Noises" Visibility="Collapsed"></Canvas>
        <StackPanel Visibility="Collapsed" Width="200" Height="150" Orienta-
tion="Horizontal" HorizontalAlignment="Center" VerticalAlignment="Bottom"
Name="SymbolsGen">

            </StackPanel>
        <TextBox Width="200" Name="GetCapcha" Visibility="Collapsed"></Text-
Box>

        <Button Name="UpdateCapcha" Visibility="Collapsed" Content="Обновить
капчу" Width="200 " Margin="5" Click="Button_Click_1"></Button>
```

```
<Button Name="ConfirmCapcha" Width="200" Height="20" Margin="5" Content="Подтвердить капчу" Visibility="Collapsed" ></Button>
<Button FontFamily="Impact" Content="Войти" Width="200" Click="Button_Click" ></Button>

</StackPanel>

</Grid>
```

Этот код определяет графический интерфейс для страницы авторизации в WPF-приложении.

Код определяет элемент Grid, который содержит StackPanel, который в свою очередь содержит несколько элементов управления, таких как изображение (Image), заголовки (Label), текстовые поля (TextBox), поле для ввода пароля (PasswordBox), флажок (CheckBox) и кнопки (Button).

Некоторые элементы скрыты (свойство Visibility="Collapsed"), такие как текстовое поле для ввода пароля (TextBox Name="TxbPassword") и элементы для генерации капчи.

Код также содержит обработчики событий для кнопок, которые будут выполняться при нажатии пользователем на них. Например, обработчик события Button_Click будет вызываться при нажатии на кнопку "Войти", и обработчик события Button_Click_1 будет вызываться при нажатии на кнопку "Обновить капчу" (Рисунок 35).

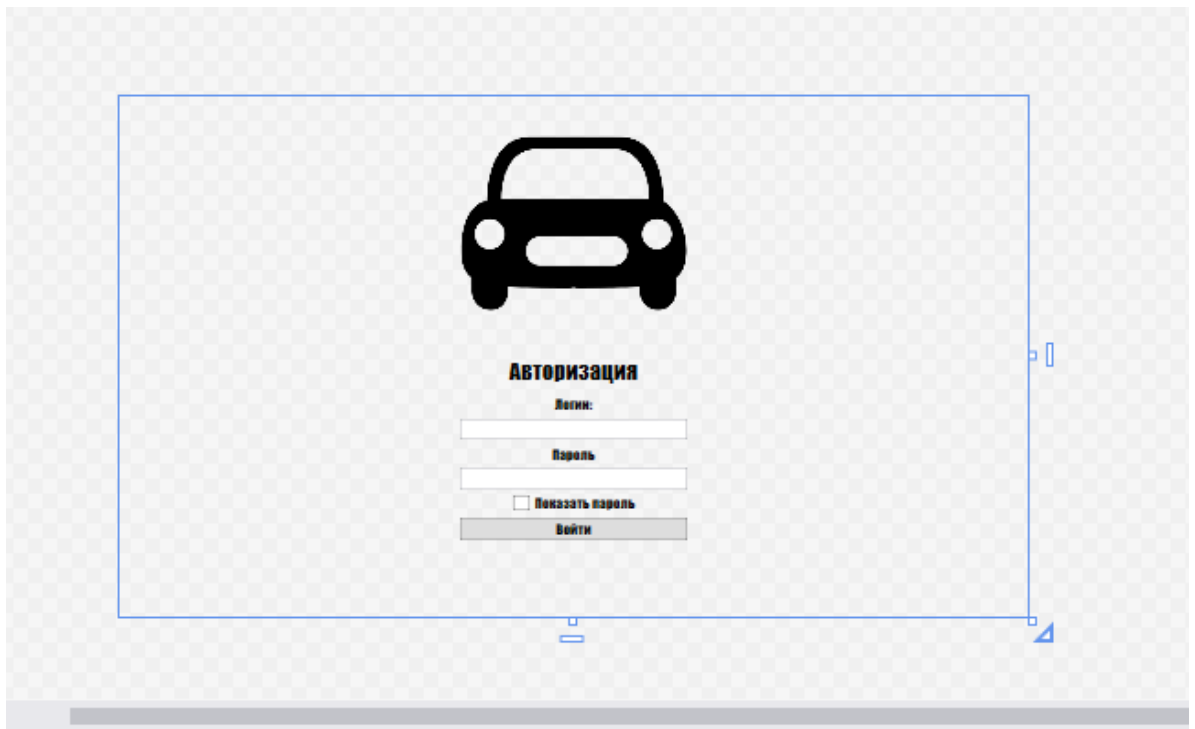


Рисунок 35. Страница авторизации

Теперь перейдём к коду работы страницы. Нам нужно подключить БД, сверить введённый пароль, прописать логику «показать пароль», логику кнопки «Войти», а так же попытки входа, после которых будет выскакивать просьба пройти капчу.

Код страницы:

```
public partial class LogINPage : Page
{
    private int attempts = 0;
    Random _random = new Random();
    private TextBlock myTextBlock;

    public LogINPage()
    {
        InitializeComponent();

        Classes.DbConnect.modeldb = new Models.AutoSphereEntities();
    }

    private void CheckBox_Checked(object sender, RoutedEventArgs e)
    {
        if (string.IsNullOrEmpty>Password.Password)
        {
            MessageBox.Show("Для начала, введите пароль!", "Уведомление");

            CheckBoxPass.IsChecked = false;
        }
    }
}
```

```

    }
    else
    {
        TxbPassword.Visibility = Visibility.Visible;
        Password.Visibility = Visibility.Collapsed;
        TxbPassword.Text = Password.Password;
    }

}

private void CheckBox_UnChecked(object sender, RoutedEventArgs e)
{

    TxbPassword.Visibility = Visibility.Collapsed;
    Password.Visibility = Visibility.Visible;
    TxbPassword.Text = Password.Password;

}

private void LogIN()
{
    try
    {
        var userObj = Classes.DbConnect.modeldb.Users.FirstOrDefault(x=>
x.Login == LoginTB.Text && x.Password == Password.Password);

        if (userObj != null)
        {
            Models.AutoSphereEntities.currentuser = userObj;
            MessageBox.Show("Здравствуйте " + userObj.Roles.RoleName + ",
" + userObj.Login, "Warning", MessageBoxButton.OK, MessageBoxImage.Information);

            switch (userObj.RoleID)
            {
                case 1:
                    Manager.MainFrame.Navigate(new AdminPage());
                    break;
                case 2:
                    Manager.MainFrame.Navigate(new ManagerPage());
                    break;
                case 3:
                    var clientdata = Classes.DbConnect.modeldb.Cli-
ents.FirstOrDefault(c => c.ClientID == userObj.ClientID);
                    if (clientdata != null)
                    {

                        User user = new User
                        {
                            FirstName = clientdata.FirstName,
                            MiddleName = clientdata.MiddleName,
                            LastName = clientdata.LastName,
                            Email = clientdata.Email,
                            PhoneNumber = clientdata.PhoneNumber,
                            PassNumber = clientdata.PassNumber,
                            ClientID = clientdata.ClientID

                        };
                        Manager.MainFrame.Navigate(new ClientPage(user));
                    }
                    else
                    {
                        MessageBox.Show("Данные клиента не найдены!",
"Уведомление", MessageBoxButton.OK, MessageBoxImage.Warning);

```



```

    }

    break;

    default:
        MessageBox.Show("Данные не обнаружены!",
"Уведомление", MessageBoxButton.OK, MessageBoxImage.Warning);
        break;
    }

}
else
{
    MessageBox.Show("Не верный логин или пароль", "Уведомление");
}

}
catch (Exception ex)
{
    MessageBox.Show("Ошибка: " + ex.Message.ToString(), "Критическая
работа приложения",
    MessageBoxButton.OK, MessageBoxImage.Warning);
}

}

private void Button_Click(object sender, RoutedEventArgs e)
{
    var userObj = Classes.DbConnect.modeldb.Users.FirstOrDefault(x =>
x.Login == LoginTB.Text && x.Password == Password.Password);

    if (userObj == null)
    {
        MessageBox.Show("Неверный логин или пароль", "Ошибка при
авторизации",
            MessageBoxButton.OK, MessageBoxImage.Error);
        attemps++;
        CheckAttemps();
    }
    else
    {
        LogIN();
    }
}

private void CheckAttemps()
{
    if (attemps == 2)
    {
        MessageBox.Show("Слишком много неудачных попыток! Подтвердите,
что вы человек", "Не удастся войти!", MessageBoxButton.OK,
MessageBoxImage.Warning);
        Noises.Visibility = Visibility.Visible;
        SymbolsGen.Visibility = Visibility.Visible;
        GetCapcha.Visibility = Visibility.Visible;
        UpdateCapcha.Visibility = Visibility.Visible;
        ConfirmCapcha.Visibility = Visibility.Visible;
        GenerateNoisesForCapcha(30);
        GenerateSymbols(3);
    }
}

```

```

else
{
    if (attempts == 3)
    {
        MessageBox.Show("Подтвердите что вы человек", "Не удастся
войти! Попытка 3", MessageBoxButtons.OK, MessageBoxImage.Warning);

    }

}

}

private void GenerateNoisesForCapcha(int volumeNoise)
{
    Noises.Visibility = Visibility.Visible;
    for (int i = 0; i < volumeNoise; i++)
    {
        Ellipse ellipse = new Ellipse
        {
            Fill = new SolidColorBrush(Color.FromArgb((byte)_random.Next(100, 200),
                (byte)_random.Next(0, 256),
                (byte)_random.Next(0, 256),
                (byte)_random.Next(0, 256))),

        };

        ellipse.Height = ellipse.Width = _random.Next(20, 60);

        Canvas.SetLeft(ellipse, _random.Next(0, 290));
        Canvas.SetTop(ellipse, _random.Next(0, 100));

        Noises.Children.Add(ellipse);
    }
}

public string Symbols = "";
private void GenerateSymbols(int count)
{
    string alphabet = "ABCDEFGHJKLMN123456789";
    for (int i = 0; i < count; i++)
    {
        string symbol = alphabet.ElementAt(_random.Next(0, alphabet.Length)).ToString();
        TextBlock lbl = new TextBlock();

        lbl.Text = symbol;
        lbl.FontSize = _random.Next(20, 40);
        lbl.RenderTransform = new RotateTransform(_random.Next(-45, 45));
        lbl.Margin = new Thickness(20, 20, 20, 20);

        Noises.Visibility = Visibility.Visible;
        SymbolsGen.Children.Add(lbl);
        Symbols = Symbols + symbol;
        myTextBlock = lbl;
    }
}

```

```

private void Button_Click_1(object sender, RoutedEventArgs e)
{
    Symbols = "";
    SymbolsGen.Children.Clear();
    Noises.Children.Clear();

    GenerateSymbols(3);
    GenerateNoisesForCapcha(25);
}

}

```

Этот код представляет собой часть пользовательского интерфейса для страницы входа в систему. При нажатии кнопки "Войти" происходит проверка правильности введенных данных пользователя (логин и пароль) в базе данных. Если данные правильные, пользователь перенаправляется на соответствующую страницу, в зависимости от роли в системе (администратор, менеджер, клиент). Если данные неверные, пользователю предлагается попробовать еще раз или подтвердить, что он является человеком с помощью CAPTCHA.

Также код реализует возможность отображения пароля в текстовом виде по нажатию на соответствующий флажок.

Полный список функций:

- Проверка правильности введенных данных пользователя
- Перенаправление пользователя на соответствующую страницу, в зависимости от роли
- Отображение пароля в текстовом виде по нажатию на соответствующий флажок
- Проверка количества неудачных попыток входа в систему и возможность ввода CAPTCHA

Отображение пароля в текстовом виде по нажатию на соответствующий флажок

Проверка количества неудачных попыток входа в систему и возможность ввода CAPTCHA

Страница авторизации и работа чек бокса «показать пароль» (Рисунок 36).

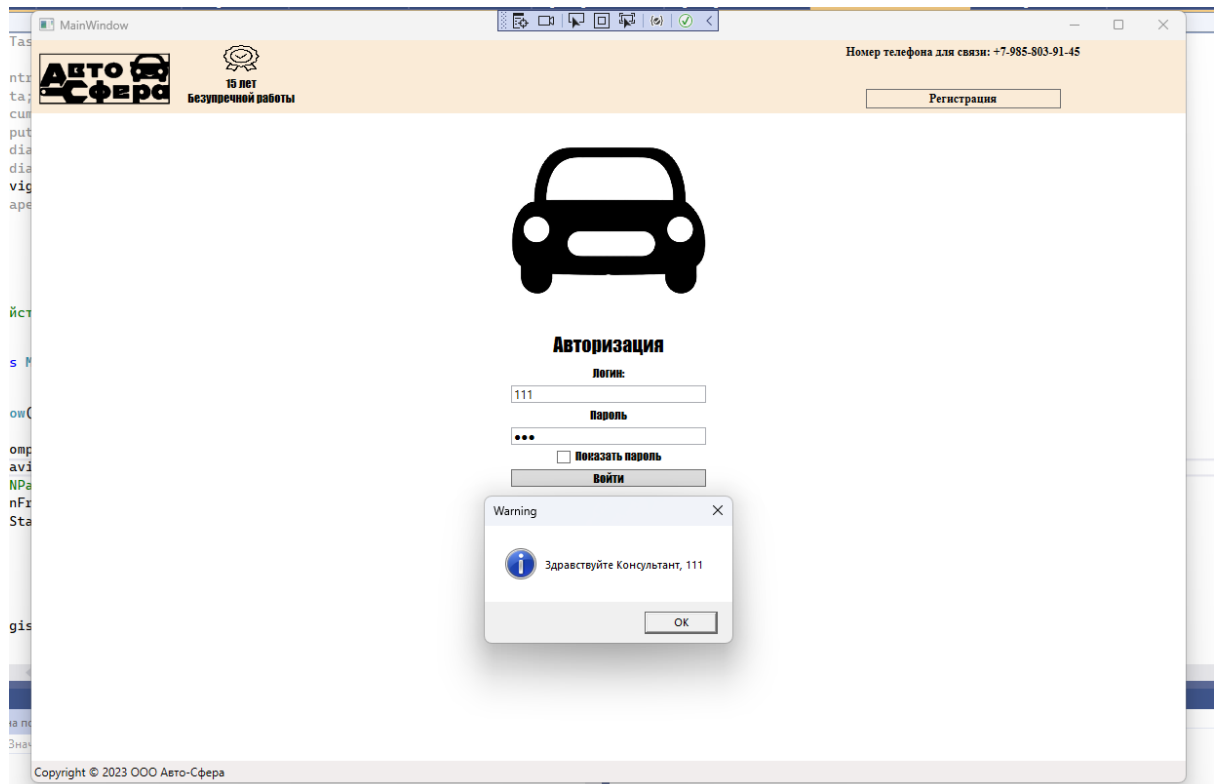


Рисунок 36. Успешная авторизация в качестве консультанта
Код авторизации рассмотрен

Вывод данных в DataGrid

DataGrid - это элемент управления, который отображает данные в виде таблицы в WPF (Windows Presentation Foundation). DataGrid позволяет пользователям взаимодействовать с данными, редактировать их, сортировать и фильтровать. Это очень удобно для отображения большого объема данных в удобочитаемом формате, а также для выполнения операций с этими данными.

DataGrid в WPF поддерживает связывание данных, множественный выбор, сортировку, фильтрацию, редактирование и многие другие функции.

Он также предоставляет множество событий, которые можно использовать для обработки действий пользователя.

XAML Код страницы с выводом данных в DataGrid представлен ниже:

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition Height="50"/>
        <RowDefinition Height="50"/>
    </Grid.RowDefinitions>
    <StackPanel Orientation="Horizontal" Grid.Row="1">
        <Button Background="AntiqueWhite" Content="Клиенты" FontFamily="Times
New Roman" Margin="5" Click="Button_Click_2"></Button>
        <Button Background="AntiqueWhite" Content="Заказы" FontFamily="Times
New Roman" Margin="5"></Button>
        <Button Background="AntiqueWhite" Content="Машины" FontFamily="Times
New Roman" Margin="5" Click="Button_Click_3"></Button>
        <Button Background="AntiqueWhite" Content="История входа" FontFam-
ily="Times New Roman" Margin="5"></Button>
        <Button Background="AntiqueWhite" Content="Сотрудники" FontFam-
ily="Times New Roman" Margin="5"></Button>
        <Button Content="Список машин" FontFamily="Times New Roman" Mar-
gin="5" Background="Green" BorderBrush="Red"></Button>
        <Button Background="AntiqueWhite" Content="Сложить стоимость всех
автомобилей" FontFamily="Times New Roman" Margin="5" Click="Button_Click_4"
Width="57"></Button>
        <Button Width="100" Click="Button_Click_5" Background="AntiqueWhite"
Content="Создать таблицу" Margin="5"></Button>

    </StackPanel>
    <StackPanel Orientation="Horizontal" HorizontalAlignment="Right"
Width="auto" Grid.Row="1">
        <Button Name="BtnEdit" Content="Добавить данные" Click="BtnRed-
Data_Click" FontFamily="Times New Roman" HorizontalAlignment="Right" Margin="5"
Background="AntiqueWhite" />
        <Button Name="BtnDel" Content="Удалить запись" Click="BtnDel_Click"
FontFamily="Times New Roman" HorizontalAlignment="Right" Margin="5" Back-
ground="AntiqueWhite" />

    </StackPanel>

    <StackPanel Grid.Row="2" Orientation="Horizontal" Margin="0 0 0 25"
Grid.RowSpan="2">
        <Button Margin="3" Width="20" Name="BtnFirstPage" Click="Btn-
FirstPage_Click" Content="&lt;&lt;" FontFamily="Impact" Background="Antique-
White" />
        <Button Margin="3" Width="20" Name="BtnPreviousPage" Click="BtnPrevi-
ousPage_Click" Content="&lt;" FontFamily="Impact" Background="AntiqueWhite"/>
        <Label Name="LblPages" VerticalAlignment="Center" FontSize="16"
FontFamily="Impact" Height="28" >2/5</Label>

        <Button Margin="3" Width="20" Name="BtnNextPage" Click="Btn-
NextPage_Click" Content="&gt;" FontFamily="Impact" Background="AntiqueWhite"/>
        <Button Margin="3" Width="20" Name="BtnLastPage"
Click="BtnLastPage_Click" Content="&gt;&gt;" FontFamily="Impact" Background="An-
tiqueWhite"/>
        <Button Content="123123" Margin="3" Click="Button_Click_1" ></Button>
```

```

</StackPanel>

<DataGrid Name="CarsView" Visibility="Visible" IsReadOnly="True" AutoGenerateColumns="False" MouseDoubleClick="CarsList_MouseDoubleClick">
    <DataGrid.Columns>
        <DataGridTextColumn Header="Номер машины" Binding="{Binding CarID}"></DataGridTextColumn>
        <DataGridTextColumn Header="Марка" Binding="{Binding Mark}"></DataGridTextColumn>
        <DataGridTextColumn Header="Модель" Binding="{Binding Model}"></DataGridTextColumn>
        <DataGridTextColumn Header="Цвет" Binding="{Binding Color}"></DataGridTextColumn>
        <DataGridTextColumn Header="Тип кузова" Binding="{Binding Type}"></DataGridTextColumn>
        <DataGridTextColumn Header="Год выпуска" Binding="{Binding Year}"></DataGridTextColumn>
        <DataGridTextColumn Header="Номер страны" Binding="{Binding CountryID}"></DataGridTextColumn>
        <DataGridTextColumn Header="Цена" Binding="{Binding Cost, StringFormat='{{0:N0}} руб. {{1:C}}'"></DataGridTextColumn>
        <DataGridTextColumn Header="VIN номер" Binding="{Binding VIN}"></DataGridTextColumn>
        <DataGridTemplateColumn>
            <DataGridTemplateColumn.CellTemplate>
                <DataTemplate>
                    <Button Name="BtnRedData" Content="Редактировать данные" Click="BtnRedData_Click" FontFamily="Times New Roman" HorizontalAlignment="Right" Margin="5" Background="AntiqueWhite" />
                </DataTemplate>
            </DataGridTemplateColumn.CellTemplate>
        </DataGridTemplateColumn>
    </DataGrid.Columns>
</DataGrid>

</Grid>

```

Этот код представляет собой XAML-разметку пользовательского интерфейса (UI) для приложения WPF (Windows Presentation Foundation).

Внутри элемента Grid задаются определения строк (Grid.RowDefinitions) с различными значениями высоты (Height) для каждой строки, а также устанавливаются элементы управления в определенные строки (Grid.Row).

Первый StackPanel содержит ряд кнопок для перехода между различными разделами приложения, второй StackPanel содержит кнопки для редактирования и удаления данных из таблицы, а третий StackPanel содержит кнопки для переключения между страницами таблицы.

В конце разметки находится элемент управления DataGrid, который используется для отображения данных в виде таблицы. Внутри элемента DataGrid определены колонки таблицы (DataGrid.Columns) с соответствующими заголовками (Header) и привязками данных (Binding). Он также имеет обработчики событий, которые вызываются при двойном щелчке на строке таблицы или при нажатии на кнопки редактирования данных внутри таблицы.

Теперь перейдём к коду страницы:

```
public partial class AdminPage : Page
{
    int _currentPage = 1, _countInPage = 10, _maxPages;
    public AdminPage()
    {
        InitializeComponent();

        CarsView.ItemsSource = AutoSphereEntities.GetContext().Cars.ToList();

        CarsView.Visibility = Visibility.Visible;
    }

    private void Button_Click(object sender, RoutedEventArgs e)
    {
    }

    private void RefreshData()
    {
        var data = AutoSphereEntities.GetContext().Cars.ToList();

        // List<Models.Ingredient> listIngredients = _context.Ingredient.ToList();

        _maxPages = (int)Math.Ceiling(data.Count * 1.0 / _countInPage);
        data = data.Skip((_currentPage - 1) *
            _countInPage).Take(_countInPage).ToList();

        LblPages.Content = $"{_currentPage}/{_maxPages}";

        CarsView.ItemsSource = data;

        ManageButtonsEnable();
    }

    private void NavigateToSelectedPage(object sender, RoutedEventArgs e)
    {
        Button btn = sender as Button;
        string pageStr = btn.Content.ToString();
        int page = int.Parse(pageStr);
        _currentPage = page;
        RefreshData();
    }
}
```

```

private void ManageButtonsEnable()
{
    BtnLastPage.IsEnabled = BtnNextPage.IsEnabled = true;
    BtnFirstPage.IsEnabled = BtnPreviousPage.IsEnabled = true;

    if (_currentPage == 1)
    {
        BtnFirstPage.IsEnabled = BtnPreviousPage.IsEnabled = false;
    }

    if (_currentPage == _maxPages)
    {
        BtnLastPage.IsEnabled = BtnNextPage.IsEnabled = false;
    }
}

private void BtnLastPage_Click(object sender, RoutedEventArgs e)
{
    // if (CarsList.ItemsSource == AutoSphereEntities.GetCon-
text().Cars.ToList())
    // {

    // }
    _currentPage = _maxPages;
    RefreshData();
}

private void BtnNextPage_Click(object sender, RoutedEventArgs e)
{
    _currentPage++;
    RefreshData();
}

private void BtnPreviousPage_Click(object sender, RoutedEventArgs e)
{
    _currentPage--;
    RefreshData();
}

private void Button_Click_2(object sender, RoutedEventArgs e)
{
    Manager.MainFrame.Navigate(new ClientsPage());
}

private void Button_Click_3(object sender, RoutedEventArgs e)
{
    Manager.MainFrame.Navigate(new CarsView());
}

private void BtnRedData_Click(object sender, RoutedEventArgs e)
{
    Manager.MainFrame.Navigate(new AddOrEditPage((sender as But-
ton).DataContext as Cars));
}

private void BtnDel_Click(object sender, RoutedEventArgs e)
{
    var carsForRemoving = CarsView.SelectedItems.Cast<Cars>().ToList();
}

```



```

        if (MessageBox.Show($"Вы собираетесь удалить {carsForRemoving.Count()} записей", "Внимание", MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)
        {
            try
            {
                AutoSphereEntities.GetContext().Cars.RemoveRange(carsForRemoving);

                AutoSphereEntities.GetContext().SaveChanges();
                MessageBox.Show("Данные удалены");

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

        private void Button_Click_4(object sender, RoutedEventArgs e)
        {
            var selectedItems = CarsView.SelectedItems;
            var totalPrice = selectedItems.Cast<Cars>().Sum(c => c.Cost);
            MessageBox.Show(totalPrice.ToString());
        }

        private void BtnEdit_Click(object sender, RoutedEventArgs e)
        {
            Manager.MainFrame.Navigate(new AddOrEditPage((sender as Button).DataContext as Cars));
        }

        private void CarsList_MouseDoubleClick(object sender, MouseButtonEventArgs e)
        {
            // создаем окно редактирования и передаем ему выбранный объект данных
        }

        private void Button_Click_5(object sender, RoutedEventArgs e)
        {
            Microsoft.Office.Interop.Word.Application wordApp = new Microsoft.Office.Interop.Word.Application();
            wordApp.Visible = true;

            Microsoft.Office.Interop.Word.Document wordDoc = wordApp.Documents.Add();
            wordDoc.Activate();

            // Название таблицы
            Microsoft.Office.Interop.Word.Paragraph para = wordDoc.Content.Paragraphs.Add();
            para.Range.Text = "Список автомобилей";
            para.Range.Font.Bold = 1;
            para.Format.SpaceAfter = 10;
            para.Range.InsertParagraphAfter();

            // Таблица с данными

```

```

        Microsoft.Office.Interop.Word.Table table = wordDoc.Tables.Add(wordDoc.Bookmarks.get_Item("\\endofdoc").Range, CarsView.Items.Count + 1, 5);

        table.Cell(1, 1).Range.Text = "Марка";
        table.Cell(1, 2).Range.Text = "Модель";
        table.Cell(1, 3).Range.Text = "Цвет";
        table.Cell(1, 4).Range.Text = "Стоимость";
        table.Cell(1, 5).Range.Text = "Статус";

        int row = 2;
        foreach (var item in CarsView.Items)
        {
            Cars car = item as Cars;
            table.Cell(row, 1).Range.Text = car.Mark;
            table.Cell(row, 2).Range.Text = car.Model;
            table.Cell(row, 3).Range.Text = car.Color;
            table.Cell(row, 4).Range.Text = car.Cost.ToString();
            table.Cell(row, 5).Range.Text = car.Actuality;

            row++;
        }

        // Добавляем границы к таблице
        Microsoft.Office.Interop.Word.Borders borders = table.Borders;
        borders.InsideLineStyle = Microsoft.Office.Interop.Word.WdLineStyle.wdLineStyleSingle;
        borders.OutsideLineStyle = Microsoft.Office.Interop.Word.WdLineStyle.wdLineStyleSingle;
        borders.InsideColor = Microsoft.Office.Interop.Word.WdColor.wdColorBlack;
        borders.OutsideColor = Microsoft.Office.Interop.Word.WdColor.wdColorBlack;
    }

    private void BtnFirstPage_Click(object sender, RoutedEventArgs e)
    {
        _currentPage = 1;
        RefreshData();
    }

    private void Button_Click_1(object sender, RoutedEventArgs e)
    {
    }
}

```

Данный код является реализацией страницы администратора. В конструкторе происходит заполнение элемента CarsView данными из базы данных, затем настраивается отображение элементов страницы.

Далее идут методы, реализующие логику работы с пагинацией. RefreshData() обновляет данные на странице, учитывая текущую страницу и количество элементов на странице. Методы NavigateToSelectedPage() и ManageButtonsEnable() отвечают за навигацию по страницам и управление кнопками.

Методы BtnLastPage_Click(), BtnNextPage_Click() и BtnPreviousPage_Click() переключают на последнюю, следующую и предыдущую страницы соответственно.

Методы Button_Click_2(), Button_Click_3(), BtnRedData_Click(), BtnDel_Click(), Button_Click_4(), BtnEdit_Click(), Button_Click_5() реализуют действия, которые могут происходить при взаимодействии пользователя с интерфейсом (например, переход на другую страницу, редактирование, удаление, создание отчета).

Запускаем приложение и переходим на страницу AdminPage (Рисунок 37)

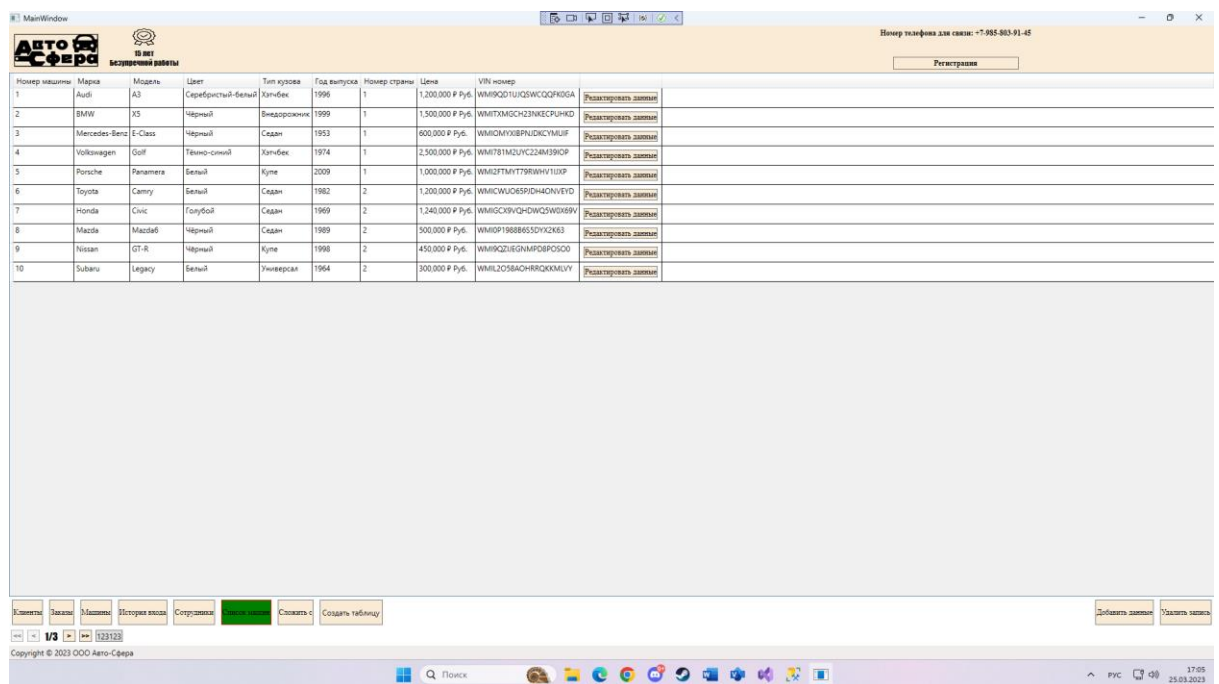


Рисунок 37. Вывод данных.

ListView

ListView в WPF (Windows Presentation Foundation) является элементом управления, который позволяет отображать данные в виде списка. Этот элемент управления предоставляет множество возможностей

для отображения данных, таких как сортировка, группировка, фильтрация и выборка.

ListView может отображать данные в различных форматах, включая текст, изображения, элементы управления и многое другое. Он также поддерживает свойство привязки данных, что позволяет связать данные с ListView и автоматически обновлять список при изменении данных.

ListView также предоставляет возможность настраивать внешний вид списка с помощью шаблонов элементов, что позволяет полностью контролировать отображение каждого элемента в списке.

В нашем WPF приложении будет выводиться список автомобилей, их описание, марка, модель и цена. Далее, будет добавлен поиск, сортировка и фильтрация.

Порядок действий схож с предыдущими, создаём страницу, создаём разметку, пишем код страницы. Начнём с кода XAML:

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="100"></RowDefinition>
        <RowDefinition Height="*"></RowDefinition>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="100"></ColumnDefinition>
        <ColumnDefinition Width="*"></ColumnDefinition>
        <ColumnDefinition Width="100"></ColumnDefinition>
    </Grid.ColumnDefinitions>

    <StackPanel Orientation="Horizontal" Grid.Column="1">
        <Label Content="Поиск машины" Height="25" FontFamily="Times New Roman" VerticalAlignment="Center"></Label>
        <TextBox Name="SearchList" Width="200" Height="25"
TextChanged="SearchList_TextChanged"></TextBox>
        <CheckBox Name="ActualCars" Content="В наличии" Cursor="Hand" Unchecked="ActualCars_Unchecked" Height="25" Width="83" VerticalContentAlignment="Center" Margin="5" FontFamily="Times New Roman" Checked="ActualCars_Checked"></CheckBox>
        <CheckBox Name="PodZakaz" Content="Под заказ" Cursor="Hand" Unchecked="PodZakaz_Unchecked" Height="25" Width="83" VerticalContentAlignment="Center" Margin="5" FontFamily="Times New Roman" Checked="PodZakaz_Checked"></CheckBox>
        <Label Content="Выберите модель" Height="25" FontFamily="Times New Roman"></Label>
        <ComboBox Name="MarksList" Width="200" Height="25" FontFamily="Times New Roman" Cursor="Hand" Margin="5" SelectionChanged="MarksList_SelectionChanged"></ComboBox>
        <Button Name="ResetFilters" Height="24" Width="126" Content="Сбросить фильтры" Click="ResetFilters_Click" Cursor="Hand" ></Button>
```

```

        <Button Content="Сформировать список автомобилей" Width="250" Margin="5" Height="25" Click="Button_Click"></Button>

    </StackPanel>
    <StackPanel Grid.Row="1" Grid.Column="0">
        <Image Source="\Resources\guaranty.png" Grid.Row="0"></Image>
        <Button Content="Корзина" Width="auto" Background="AliceBlue"></Button>
    </StackPanel>
    <DataGrid>
        <DataGrid.Columns>
            <DataGridTextColumn Header="Test"></DataGridTextColumn>
        </DataGrid.Columns>
    </DataGrid>
    </StackPanel>
    <WrapPanel Grid.Row="1" Grid.Column="1" Width="auto">
        <ListView Name="CarsViewPanel" Grid.Column="1" Grid.Row="1" ScrollViewer.HorizontalScrollBarVisibility="Disabled" VerticalAlignment="Center">
            <ListView.ItemsPanel>
                <ItemsPanelTemplate>
                    <WrapPanel Orientation="Horizontal" VerticalAlignment="Center"></WrapPanel>
                </ItemsPanelTemplate>
            </ListView.ItemsPanel>
            <ListView.ItemTemplate>
                <DataTemplate>
                    <Grid>
                        <Grid.RowDefinitions>
                            <RowDefinition Height = "75" ></RowDefinition >
                            <RowDefinition Height = "310" ></RowDefinition >
                            <RowDefinition Height = "auto" ></RowDefinition >
                            <RowDefinition Height = "auto" ></RowDefinition >
                            <RowDefinition Height = "auto" ></RowDefinition >
                        </Grid.RowDefinitions >
                        <Grid.ColumnDefinitions>
                            <ColumnDefinition Width="500"></ColumnDefinition>
                            <ColumnDefinition Width="auto"></ColumnDefinition>
                        </Grid.ColumnDefinitions>
                        <Image Width = "400" Grid.Row = "1" Stretch = "UniformToFill" HorizontalAlignment = "Center" Margin = "5" >
                            <Image.Source>
                                <ImageSource >\Resources\loginImage.png</ImageSource >
                            </Image.Source>
                        </Image >
                        <TextBlock Text = "{Binding CarID}" VerticalAlignment = "Center" TextAlignment = "Center" Width = "450"
                            TextWrapping = "Wrap" HorizontalAlignment = "Center" FontSize = "26" Grid.Row = "0" ></TextBlock >
                        <TextBlock Text = "{Binding Mark}" Grid.Row = "2" HorizontalAlignment = "Center" FontSize = "20" FontWeight = "Bold" ></TextBlock >
                        <TextBlock Text = "{Binding Model}" Grid.Row = "3" FontSize = "14" HorizontalAlignment = "Right" ></TextBlock >
                        <TextBlock Text = "{Binding Cost}" Grid.Row = "3" FontSize = "14" HorizontalAlignment = "Left" ></TextBlock >
                    </Grid>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </WrapPanel>
    </Grid>

```


Теперь приступим к рабочему коду, который будет отвечать за вывод информации в ListView:

```
public partial class CarsView : System.Windows.Controls.Page
{
    private List<Cars> allCars;

    public CarsView()
    {
        InitializeComponent();

        // Получаем все марки и привязываем к комбобоксу
        var allMarks = AutoSphereEntities.GetContext().Marks.ToList();
        MarksList.ItemsSource = allMarks;
        MarksList.DisplayMemberPath = "MarkName";
        MarksList.SelectedValue = "MarkName";

        // Получаем все автомобили и запоминаем их
        allCars = AutoSphereEntities.GetContext().Cars.ToList();

        // Заполняем список автомобилей по умолчанию
        UpdateCarsPage();
    }

    private void UpdateCarsPage()
    {
        // Создаем копию всех автомобилей, чтобы фильтровать только ее
        var currentCars = allCars.ToList();

        // Фильтруем список автомобилей по поисковому запросу
        string searchText = SearchList.Text.ToLower();
        currentCars = currentCars.Where(p => p.Model.ToLower().Contains(searchText) || p.Mark.ToLower().Contains(searchText)).ToList();

        // Фильтруем список автомобилей по состоянию "В наличии"
        if (ActualCars.IsChecked == true)
        {
            currentCars = currentCars.Where(p => p.Actuality == "В наличии").ToList();
        }

        // Фильтруем список автомобилей по состоянию "Под заказ"
        if (PodZakaz.IsChecked == true)
        {
            currentCars = currentCars.Where(p => p.Actuality == "Под заказ").ToList();
        }

        // Фильтруем список автомобилей по выбранной марке
        var selectedMark = MarksList.SelectedItem as Marks;
        if (selectedMark != null)
        {
            string selectedBrand = selectedMark.MarkName;
            currentCars = currentCars.Where(p => p.Mark == selectedBrand).ToList();
        }

        // Обновляем список автомобилей на странице
        CarsViewPanel.ItemsSource = currentCars;
    }
}
```

```

private void ActualCars_Unchecked(object sender, RoutedEventArgs e)
{
    UpdateCarsPage();
}

private void SearchList_TextChanged(object sender, TextChangedEventArgs
e)
{
    UpdateCarsPage();
}

private void MarksList_SelectionChanged(object sender, Selection-
ChangedEventArgs e)
{
    UpdateCarsPage();
}

private void ActualCars_Checked(object sender, RoutedEventArgs e)
{
    UpdateCarsPage();
}

private void PodZakaz_Unchecked(object sender, RoutedEventArgs e)
{
    UpdateCarsPage();
}

private void PodZakaz_Checked(object sender, RoutedEventArgs e)
{
    UpdateCarsPage();
}

private void ResetFilters_Click(object sender, RoutedEventArgs e)
{
    // Сбрасываем значения фильтров и поискового поля
    SearchList.Text = "";
    MarksList.SelectedIndex = -1;
    ActualCars.IsChecked = false;
    PodZakaz.IsChecked = false;

    // Сбрасываем сортировку
    CollectionViewSource.GetDefaultView(CarsViewPanel.Items-
Source).SortDescriptions.Clear();

    // Сбрасываем фильтры
    CollectionViewSource.GetDefaultView(CarsViewPanel.ItemsSource).Filter
= null;

    // Обновляем список автомобилей на странице
    UpdateCarsPage();
}

private void Button_Click(object sender, RoutedEventArgs e)
{
    Microsoft.Office.Interop.Word.Application wordApp = new Microsoft.Of-
fice.Interop.Word.Application();
    wordApp.Visible = true;

    Microsoft.Office.Interop.Word.Document wordDoc = wordApp.Docu-
ments.Add();
    wordDoc.Activate();
}

```



```

        // Название таблицы
        Microsoft.Office.Interop.Word.Paragraph para = wordDoc.Content.Paragraphs.Add();
        para.Range.Text = "Список автомобилей";
        para.Range.Font.Bold = 1;
        para.Format.SpaceAfter = 10;
        para.Range.InsertParagraphAfter();

        // Таблица с данными
        Microsoft.Office.Interop.Word.Table table = wordDoc.Tables.Add(wordDoc.Bookmarks.get_Item("\\endofdoc").Range, CarsViewPanel.Items.Count + 1, 5);
        table.Cell(1, 1).Range.Text = "Марка";
        table.Cell(1, 2).Range.Text = "Модель";
        table.Cell(1, 3).Range.Text = "Цвет";
        table.Cell(1, 4).Range.Text = "Стоимость";
        table.Cell(1, 5).Range.Text = "Статус";

        int row = 2;
        foreach (var item in CarsViewPanel.Items)
        {
            Cars car = item as Cars;
            table.Cell(row, 1).Range.Text = car.Mark;
            table.Cell(row, 2).Range.Text = car.Model;
            table.Cell(row, 3).Range.Text = car.Color;
            table.Cell(row, 4).Range.Text = car.Cost.ToString();
            table.Cell(row, 5).Range.Text = car.Actuality;

            row++;
        }

        // Добавляем границы к таблице
        Microsoft.Office.Interop.Word.Borders borders = table.Borders;
        borders.InsideLineStyle = Microsoft.Office.Interop.Word.WdLineStyle.wdLineStyleSingle;
        borders.OutsideLineStyle = Microsoft.Office.Interop.Word.WdLineStyle.wdLineStyleSingle;
        borders.InsideColor = Microsoft.Office.Interop.Word.WdColor.wdColorBlack;
        borders.OutsideColor = Microsoft.Office.Interop.Word.WdColor.wdColorBlack;

        // Сохранение документа
    }
}

```

Этот код представляет пользовательский интерфейс для отображения автомобилей на странице Windows. Страница содержит несколько фильтров, которые пользователь может использовать для поиска автомобилей по различным критериям. Когда пользователь изменяет любой из фильтров, отображение автомобилей на странице обновляется в соответствии с выбранными критериями.

Класс `CarsView` наследует класс `Page` из пространства имен `System.Windows.Controls` и содержит методы и события, которые позволяют отображать и фильтровать список автомобилей на странице.

При создании экземпляра класса `CarsView`, метод `InitializeComponent()` инициализирует компоненты страницы, а затем выполняется инициализация переменных и обновление списка автомобилей на странице с помощью метода `UpdateCarsPage()`.

Метод `UpdateCarsPage()` обновляет список автомобилей на странице в соответствии с выбранными фильтрами. Он создает копию всех автомобилей и затем фильтрует этот список по поисковому запросу, состоянию "В наличии", состоянию "Под заказ" и выбранной марке. В результате получается список автомобилей, который соответствует выбранным критериям, и этот список отображается на странице.

Методы `ActualCars_Unchecked`, `SearchList_TextChanged`, `MarksList_SelectionChanged`, `ActualCars_Checked`, `PodZakaz_Unchecked` и `PodZakaz_Checked` обновляют список автомобилей на странице при изменении соответствующего фильтра.

Метод `ResetFilters_Click` сбрасывает все значения фильтров и поискового поля, а также сбрасывает сортировку и фильтры. Затем он вызывает метод `UpdateCarsPage()` для обновления списка автомобилей на странице.

Метод `Button_Click` открывает приложение Microsoft Word и создает таблицу со списком автомобилей на странице. Затем он заполняет таблицу данными из списка автомобилей на странице и добавляет границы к таблице.

Запускаем приложение и смотрим результат:

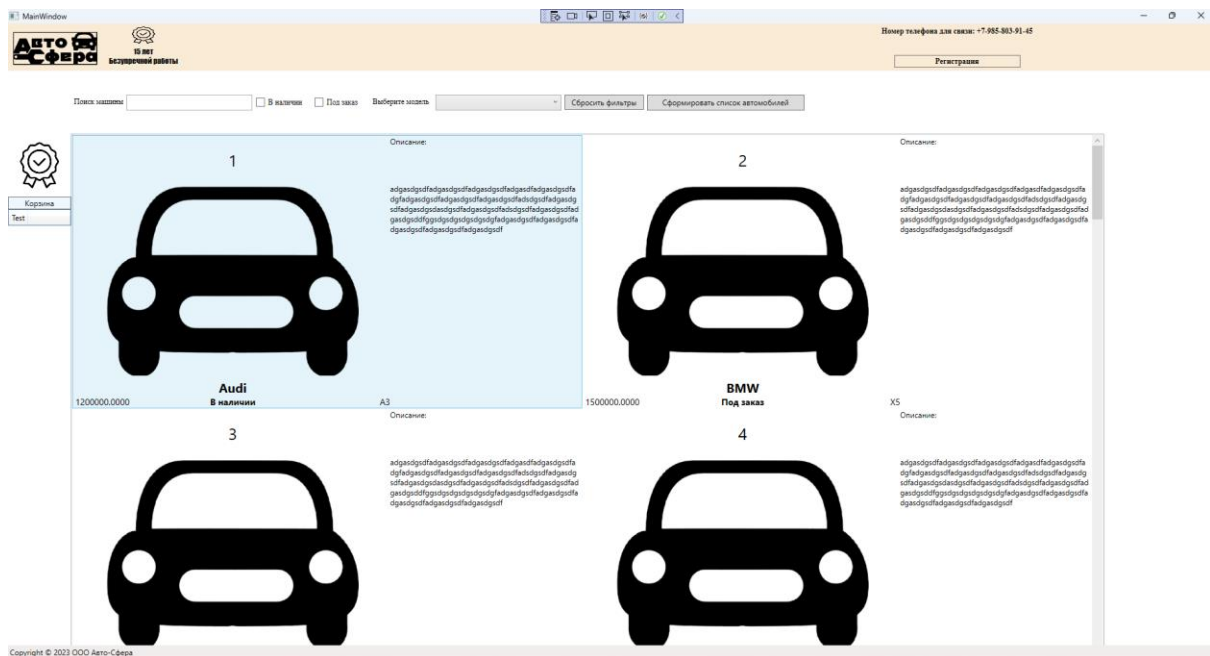


Рисунок 38. ListView всех автомобилей.

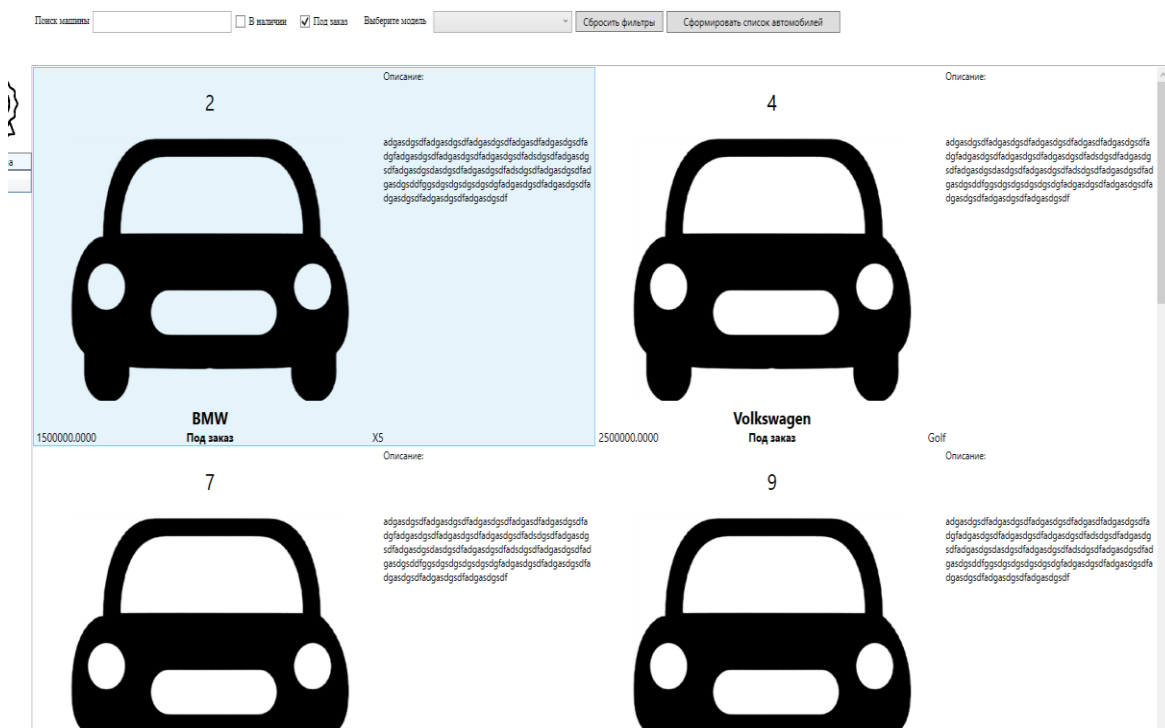


Рисунок 39. Автомобили только под заказ

Корзина

Корзина будет представлять из себя DataGrid, в который будут выводиться заказанные тест драйвы конкретного пользователя. Клиент заказывает тест драйв и в базу данных заносится номер машины, номер клиента, дата и стоимость, которая будет составлять 0.5% от всей стоимости автомобиля.

По нажатию кнопки «Заказать тест драйв» будет происходить выполнение следующего кода:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    var selected = CarsViewPanel.SelectedItem as Cars;
    var selectedDate = DatesComboBox.SelectedItem as Dates;
    DateTime? date = selectedDate?.Date;
    decimal discount = selected.Cost.Value * (decimal)0.005;
    decimal discountedPrice = selected.Cost.Value - discount;

    // Выводим данные в MessageBox с учетом скидки
    string message = "Марка: " + selected.Mark + "\n" +
                    "Модель: " + selected.Model + "\n" +
                    "Год выпуска: " + selected.Year + "\n" +
                    "Цвет: " + selected.Color + "\n" +
                    "Цена: " + discount.ToString("#.##") + " руб. (Цена
теста-драйва составляет 0.5% от общей стоимости автомобиля ";

    MessageBoxResult result = MessageBox.Show(message, "Подтверждение
данных", MessageBoxButton.YesNo, MessageBoxImage.Question);
    var currentUser = AutoSphereEntities.GetContext().Users.FirstOrDefault(u => u.ClientID == user.ClientID);
    if (currentUser == null)
    {
        return;
    }
    var newTestDrive = new TestDrive
    {
        CarID = selected.CarID,
        //Data = selectedDate.Value,
        ClientID = currentUser.ClientID,
        Price = discount,
        Data = date // явное преобразование типа
    };
}
```

1. Этот код относится к событию клика на кнопке и выполняет несколько действий:
2. Получает выбранный элемент из панели просмотра автомобилей (предполагается, что в этой панели отображаются объекты типа "Cars").

3. Получает выбранную дату из выпадающего списка "DatesComboBox" (предполагается, что в этом списке отображаются объекты типа "Dates").
4. Вычисляет скидку на автомобиль в размере 0,5% от его стоимости.
5. Вычисляет цену тест-драйва автомобиля с учетом скидки.
6. Выводит сообщение о выбранном автомобиле и цене тест-драйва в диалоговом окне "MessageBox".
7. Получает текущего пользователя из базы данных (предполагается, что в базе данных есть таблица "Users" с полями "ClientID" и другими).
8. Если текущий пользователь не найден в базе данных, то выходит из метода.
9. Создает новый объект типа "TestDrive" с заполнением его полей:
 - идентификатор выбранного автомобиля ("CarID"),
 - идентификатор текущего пользователя ("ClientID"),
 - цена тест-драйва с учетом скидки ("Price"),
 - выбранная дата тест-драйва ("Data").
 - Записывает созданный объект "TestDrive" в базу данных.

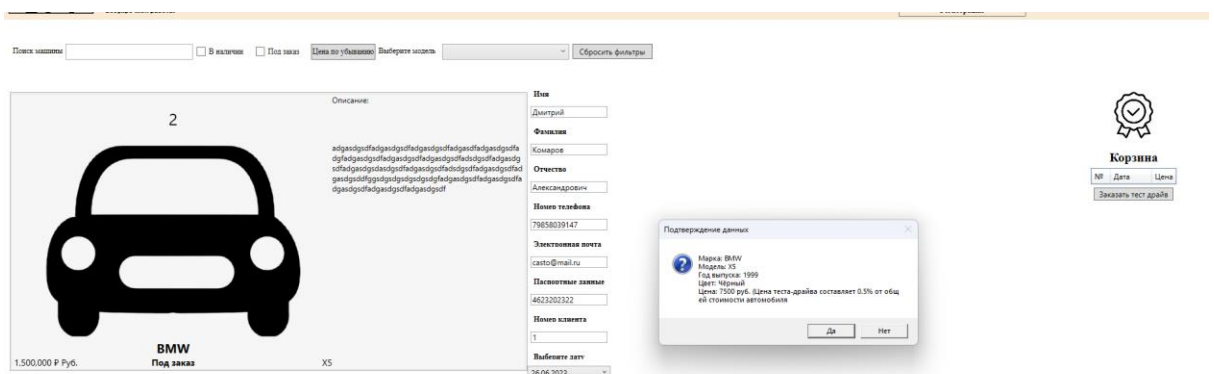


Рисунок 42. Подтверждение данных



Корзина

№	Дата	Цена
2	6/26/2023	7,500 Р Руб.

Заказать тест драйв

Рисунок 43. Корзина клиента

После подтверждения данных, срабатывает следующий код:

```
AutoSphereEntities.GetContext().TestDrives.Add(newTestDrive);  
AutoSphereEntities.GetContext().SaveChanges();  
Cart.ItemsSource = AutoSphereEntities.GetContext().Test-  
Drives.Where(td => td.ClientID ==
```

В DataGridView происходит вывод данных только конкретного клиента.

Заключение

В ходе прохождения производственной (преддипломной) практической подготовки в ООО Авто-Сфера мною было выполнено индивидуальное задание по производственной практике и достигнута цель по приобретению знаний, умений и навыков по ПМ. 01 Разработка модулей программного обеспечения для компьютерных систем, ПМ. 02 Осуществление интеграции программных модулей:

ПК 1.1. Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием.

ПК 1.2. Разрабатывать программные модули в соответствии с техническим заданием.

ПК 1.3. Выполнять отладку программных модулей с использованием специализированных программных средств.

ПК 1.4. Выполнять тестирование программных модулей.

ПК 1.5. Осуществлять рефакторинг и оптимизацию программного кода.

ПК 1.6. Разрабатывать модули программного обеспечения для мобильных платформ.

ПК 2.1. Разрабатывать требования к программным модулям на основе анализа проектной и технической документации на предмет взаимодействия компонент.

ПК 2.2. Выполнять интеграцию модулей в программное обеспечение.

ПК 2.3. Выполнять отладку программного модуля с использованием специализированных программных средств.

ПК 2.4. Осуществлять разработку тестовых наборов и тестовых сценариев для программного обеспечения.

ПК 2.5. Производить инспектирование компонент программного обеспечения на предмет соответствия стандартам кодирования

Мною освоены следующие общие компетенции:

ОК 01. Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам.

ОК 02. Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности.

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие.

ОК 04. Работать в коллективе и команде, эффективно взаимодействовать с коллегами, руководством, клиентами.

ОК 05. Осуществлять устную и письменную коммуникацию на государственном языке с учетом особенностей социального и культурного контекста.

ОК 06. Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных общечеловеческих ценностей.

ОК 07. Содействовать сохранению окружающей среды, ресурсосбережению, эффективно действовать в чрезвычайных ситуациях.

ОК 08. Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания необходимого уровня физической подготовленности.

ОК 09. Использовать информационные технологии в профессиональной деятельности.

ОК 10. Пользоваться профессиональной документацией на государственном и иностранном языке.

ОК 11. Планировать предпринимательскую деятельность в профессиональной сфере.

Список литературы

1. "WPF в действии: мастерство создания приложений с использованием Windows Presentation Foundation" - Адам Нейт, Джош Смит, Иэн Гриффитс (2019)
2. "Создание Windows-приложений с использованием WPF и XAML" - Андрей Лунин (2019)
3. "WPF. Разработка приложений для Windows" - Андрей Карпов (2020)
4. "Современные технологии программирования баз данных" - Михаил Голуб, Александр Зубков, Алексей Ярошевич (2019)
5. "Основы проектирования баз данных" - Лоренцо Дженнаро (2020)
6. "SQL Server 2019. Руководство разработчика" - Дэвид Рамел (2021)
7. "Проектирование баз данных. Учебное пособие" - Иван Иванов (2022)