|  |
| --- |
| **МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ** |
| **ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕХНОЛОГИЙ И УПРАВЛЕНИЯ ИМЕНИ К.Г. РАЗУМОВСКОГО (ПЕРВЫЙ КАЗАЧИЙ УНИВЕРСИТЕТ)»**  **(ФГБОУ ВО «МГУТУ ИМ. К.Г. РАЗУМОВСКОГО (ПКУ)»)** |
|  |
| **УНИВЕРСИТЕТСКИЙ КОЛЛЕДЖ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ** |

|  |
| --- |
| **КУРСОВОЙ ПРОЕКТ** |
|  |
| по междисциплинарному курсу: МДК.02.02. Технология разработки и защиты баз данных. |
|  |
| на тему: Разработка системы автоматизации работы автосервиса. |
|  |
| студента группы 090203-9о-20/1  специальности 09.02.03 Программирование в компьютерных системах |
| Пономарёва Евгения Романовича |

|  |  |  |
| --- | --- | --- |
| Студент |  | Е.Р. Пономарёв |
| Руководитель курсового проекта |  | Е.А. Ларионова |
| Председатель ПЦК специальности 09.02.03 Программирование в компьютерных системах |  | А.И. Глускер |

|  |  |  |
| --- | --- | --- |
| Дата защиты «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2023 г. | | |
| Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | |
| Заведующий отделением № 1 |  | И.А. Миланова |

Москва

2023

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc148199942)

[1. ОБЩАЯ ЧАСТЬ 4](#_Toc148199943)

[1.1 Назначение и цели создания 4](#_Toc148199944)

[1.2 Обзор и анализ предметной области 4](#_Toc148199945)

[1.3 Жизненный цикл базы данных 4](#_Toc148199946)

[1.4 Выбор и характеристика СУБД 5](#_Toc148199947)

[1.5 Выбор и характеристика среды разработки приложения 6](#_Toc148199948)

[2. СПЕЦИАЛЬНАЯ ЧАСТЬ 8](#_Toc148199949)

[2.1 Постановка задачи 8](#_Toc148199950)

[2.2 Архитектура информационной системы 8](#_Toc148199951)

[2.3 Логическая модель базы данных 9](#_Toc148199952)

[2.4 Нормализация таблиц 10](#_Toc148199953)

[2.5 Описание таблиц 10](#_Toc148199954)

[2.6 Другие объекты базы данных (представления, хранимые процедуры, триггеры, имена входа, пользователи) 14](#_Toc148199955)

[2.7 Разработка приложения 14](#_Toc148199956)

[2.7.1 Диаграмма вариантов использования 14](#_Toc148199957)

[2.7.2 Решение главной задачи проекта 15](#_Toc148199958)

[2.7.3 Тестирование приложения 16](#_Toc148199959)

[2.7.4 Защита информационной системы 18](#_Toc148199960)

[2.7.5 Общие сведения об информационной системе 18](#_Toc148199961)

[2.7.6 Требования к техническим средствам 19](#_Toc148199962)

[2.7.7 Требования к программным средствам 19](#_Toc148199963)

[2.7.8 Формы ввода 19](#_Toc148199964)

[2.7.9 Отчёты 20](#_Toc148199965)

[ЗАКЛЮЧЕНИЕ 21](#_Toc148199966)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 22](#_Toc148199967)

[ПРИЛОЖЕНИЕ А. SQL скрипты 23](#_Toc148199968)

[ПРИЛОЖЕНИЕ Б. Код программы 27](#_Toc148199969)

# **ВВЕДЕНИЕ**

В настоящее время нельзя недооценивать важность информационных систем в обществе. Задачи, связанные с информацией в бизнесе, производстве и образовании сильно зависят от баз данных. Благодаря им, возможны значительные экономии времени и ресурсов при поиске нужной информации. Они также способствуют получению объективного понимания проблемы путем объединенного доступа к обширным источникам информации.

Целью курсового проекта является разработка системы автоматизации работы автосервиса. Она должна решать проблему с обслуживанием автомобиля.

**Задачи проекта:**

* Анализ предметной области информационной системы для автосервисов;
* Проектирование базы данных CASE-средствами
* Разработка приложения с помощью программы «Visual Studio 2022»;
* Защита приложения с помощью системы прав доступа.

# **1. ОБЩАЯ ЧАСТЬ**

## 1.1 Назначение и цели создания

Назначение: разработка системы автоматизации работы автосервиса.

Цели создания: реализация учебного плана специальности 09.02.03 «Программирование в компьютерных системах» и освоение общих и профессиональных компетенций по модулю 2.

## 1.2 Обзор и анализ предметной области

Система автоматизации работы автосервиса — это программное обеспечение, которая позволяет выбирать запчасти необходимые для починки и обслуживания машины. В основе программы лежит база данных, в которой хранится информация о клиентах, услугах, запчастях, мастеров и машинах.

Основная цель программы - упростить и автоматизировать процесс обслуживания своей машины, удобно приобретая необходимые запчасти.

Создание приложения было для операционной системы Windows, в котором клиентам будет доступен удобный и интуитивно понятный интерфейс для работы с приложением.

Функциональные возможности приложения:

1. Просмотр актуальных услуг и запчастей.
2. Просмотр актуальных заказов, и истории заказов. Также актуальный заказ можно напечатать, где будет отображена машина, услуга и мастер, который занимался заказом.
3. Мастер специалист может выбрать понравившейся актуальный заказ, для его выполнения.

## 1.3 Жизненный цикл базы данных

Жизненный цикл базы данных состоит из нескольких стадий:

1. Планирование (сбор и анализ информации, которая будет храниться в базе данных)
2. Описание требования (требования к базе данных, информационные потребности, требования к оборудованию и программному обеспечению)
3. Логическое проектирование (создание логической модели базы данных, отображающая всю необходимую информацию для решения поставленной задачи *(См. рис. 2))*
4. Физическое проектирование (окончательное решение о способе физической реализации логического проекта базы данных)
5. Разработка (Создание приложения на языке C# с использованием API-интерфейса WPF).
6. Тестирование (проверка функционирования программы *(См. таб. 10)*)
7. Эксплуатация (использование программы пользователями)
8. Сопровождение (расширение базы данных, расширение функциональных возможностей программы, внесение изменений, поддержка проекта в актуальном состоянии).

## 1.4 Выбор и характеристика СУБД

Существует множество различных систем управления базами данных (СУБД), включая такие как Microsoft SQL Server, MySQL и PostgreSQL. Для разработки базы данных был выбран Microsoft SQL Server, такой выбор сделан из-за его простоты использования и наличия опыта работы с ним. Если рассматривать другие СУБД, то у них есть свои существенные недостатки. Например, MySQL имеет проблемы с обработкой транзакций, в то время как PostgreSQL может быть более сложным для понимания.

«Microsoft SQL Server - система управления реляционными базами данных (РСУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов - Transact-SQL, создан совместно Microsoft и Sybase. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия.»

Преимущества:

1. Масштабирование системы. Взаимодействовать с ней можно как на простых ноутбуках, так и на ПК с мощным процессором,
2. Размер страниц - до 8 Кб. Данные извлекаются быстро, а сложную информацию удобнее хранить,
3. Автоматизация рутинных административных задач. Управление блокировками и памятью, редактура размеров файлов,
4. Удобный поиск. Его можно осуществлять по фразам, словам, тексту либо создавать ключевые индексы.

Недостатки:

1. Зависимость от операционной системы. Доступна только для Windows,
2. Высокая стоимость программы.

## 1.5 Выбор и характеристика среды разработки приложения

Для разработки приложения было решено воспользоваться Visual Studio 2022, при анализе таких доступных интегрированных сред, как Rider, Visual Studio Code и Eclipse. Этот выбор обусловлен имеющимися навыками работы с данной IDE и удобством работы с API-интерфейсом WPF. Если рассматривать другие варианты инструментов разработки, то стоит учесть, что Rider обладает платной моделью распространения, в то время как Eclipse представляет определенные трудности в использовании.

Visual Studio 2022 - интегрированная среда разработки для создания многофункциональных, кроссплатформенных приложений для Windows, Mac, Linux, iOS и Android.

Преимущества:

1. Удобный интерфейс,
2. Поддержка почти всех языков программирования,
3. Бесплатная для любого пользователя.

Недостаток:

1. Зависимость от Microsoft и Windows.

Языком программирования для разработки приложения был выбран C#, так как это требовалось от курсового проекта.

В рамках курсового проекта предоставлялся выбор, среди ADO.NET и EntityFramework.

Для обработки базы данных был выбран EntityFramework, фреймворк языка C#. Он отличается высокой степенью абстракции, что облегчает взаимодействие с базой данных по сравнению с использованием ADO.NET. Одной из основных преимуществ этого фреймворка является возможность автоматической генерации кода и структуры базы данных согласно подходам Code First, Model First и Database First, что отсутствует в ADO.NET. Кроме того, EntityFramework позволяет применять LINQ для работы с объектами в базе данных.

Чтобы манипулировать данными использовался пользовательский метод GetContext(), с помощью которого можно создать взаимосвязь между проектом и SQL Server.

После установки подключения и открытия соединения, можно выполнять любые взаимодействия с данными, например: редактирование данных в таблицах, удаление данных из таблиц, и тому подобные действия.

# **СПЕЦИАЛЬНАЯ ЧАСТЬ**

## 2.1 Постановка задачи

Необходимо разработать базу данных пользуясь средствами MS SQL SERVER, которая позволяет выбрать одну или несколько запчастей для своего автомобиля и оформить заказ, также создать приложение средствами Visual Studio 2022, используя язык программирования C# и API-интерфейс WPF.

Предусмотреть поддержку базы данных в актуальном состоянии.

Предусмотреть работу 2-ух пользователей.

Программа должна выдавать:

* + - 1. Список актуальных услуг и запчастей,
      2. Печатную версию актуального заказа,
      3. Список заказов всех клиентов.

Система должна позволять:

1. Выбрать машину, услугу, запчасть,
2. Пополнить список машин, услуг, запчастей,
3. Просмотр профиля клиента,
4. Просматривать степень выполнения заказа,
5. Оформить заказ.

## 2.2 Архитектура информационной системы

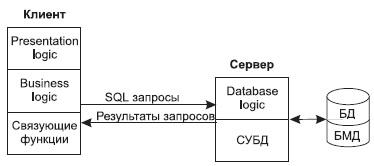


Рисунок 1 Архитектура информационной системы

Данная информационная система работает по модели клиент-сервер, где клиентское приложение взаимодействует с сервером, на котором хранится база данных. Эта модель обеспечивает быстрое выполнение операций и уменьшает нагрузку на сеть.

Сервер выполняет функции:

1. Хранения и обработки запросов от клиента,
2. Получения результатов,
3. Управление транзакциями,
4. Поддержка языков баз данных,
5. Управление данными во внешней памяти.

Клиент-серверная архитектура:

1. Обслуживание большого числа пользователей,
2. Улучшает гибкость системы,
3. Обеспечивает стабильность в случае сбоев. Даже если у клиента возникнут проблемы, целостность данных и доступность для других клиентов не пострадают.

## 2.3 Логическая модель базы данных

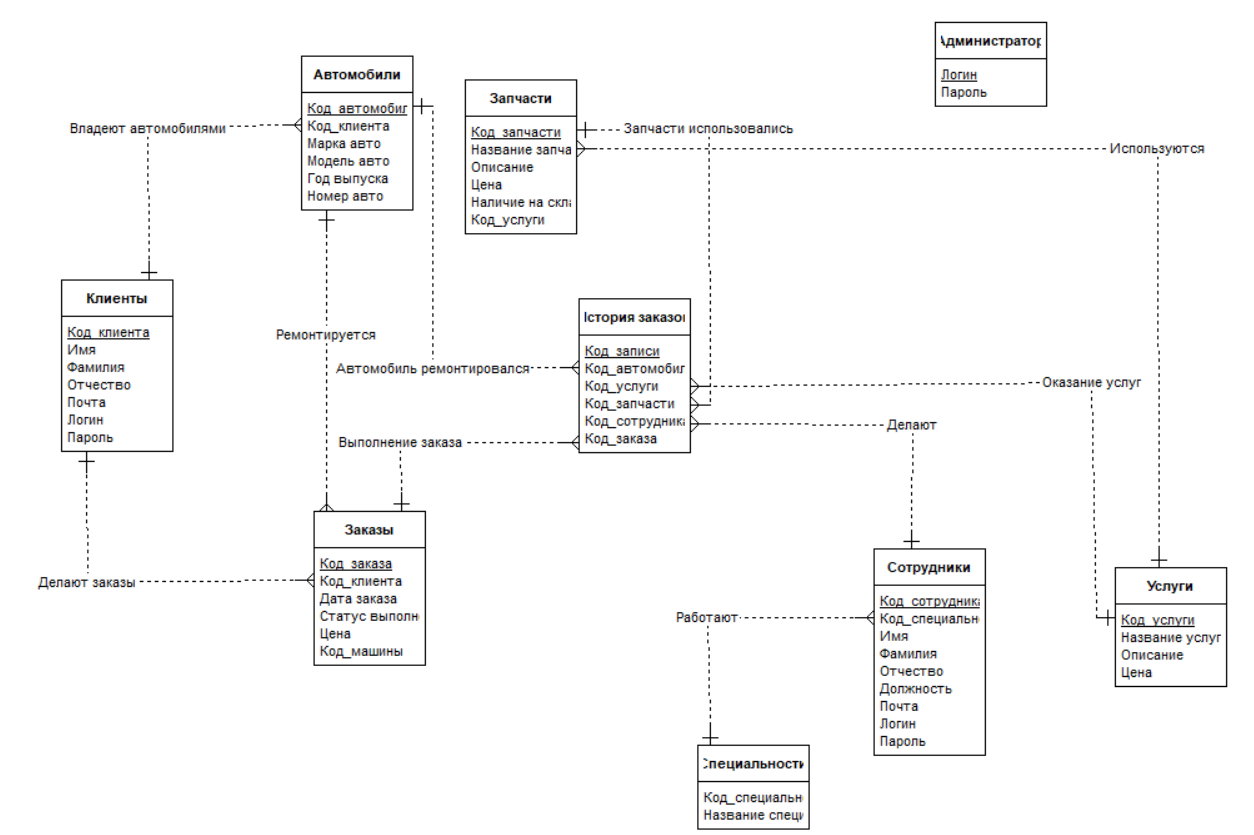


Рисунок 2 Логическая модель базы данных

## 2.4 Нормализация таблиц

1. Таблицы приведены к первой нормальной форме (все атрибуты просты, далее неделимы);
2. Таблицы приведены ко второй нормальной форме (каждый не ключевой атрибут функционально полно зависит от ключа);
3. Таблицы находятся в третьей нормальной форме (отсутствие транзитивной зависимости).

## 2.5 Описание таблиц

Таблица 1 - Admin (Администратор)

|  |  |  |  |
| --- | --- | --- | --- |
| **Название столбца** | **Описание** | **Тип** | **Примечание** |
| login | Логин | varchar(50) | PK, NOT NULL |
| password | Пароль | varchar (50) | NOT NULL |

Таблица 2 - Cars (Машины)

|  |  |  |  |
| --- | --- | --- | --- |
| **Название столбца** | **Описание** | **Тип** | **Примечание** |
| id\_car | Код машины | Int | PK, NOT NULL |
| Id\_client | Код клиента | Int | FK, NOT NULL |
| Car\_brand | Бренд машины | Varchar (50) | NOT NULL |
| Car\_model | Модель машины | Varchar (50) | NOT NULL |
| Year\_of\_manufacture | Год выпуска | Int | NOT NULL |
| Number\_car | Номер машины | Varchar (10) | NOT NULL |
| photo | Фото | VARBINARY (MAX) | NOT NULL |

Таблица 3 - Clients (клиенты)

|  |  |  |  |
| --- | --- | --- | --- |
| **Название столбца** | **Описание** | **Тип** | **Примечание** |
| Id\_client | Код клиента | int | PK, NOT NULL |
| Name | Имя | Varchar (50) | NOT NULL |
| Surname | Фамилия | Varchar (50) | NOT NULL |
| Patronymic | Отчество | Varchar (50) | NULL |
| Login | Логин | Varchar (50) | NOT NULL |
| Password | Пароль | Varchar (50) | NOT NULL |
| email | Почта | Varchar (50) | NOT NULL |

Таблица 4 – Employees (Мастера)

|  |  |  |  |
| --- | --- | --- | --- |
| **Название столбца** | **Описание** | **Тип** | **Примечание** |
| Id\_employee | Код мастера | Int | PK, NOT NULL |
| Id\_specialties | Код специальности | Int | FK, NOT NULL |
| Name | Имя | Varchar (50) | NOT NULL |
| Surname | Фамилия | Varchar (50) | NOT NULL |
| Patronymic | Отчество | Varchar (50) | NULL |
| Job\_title | Должность | Varchar (50) | NOT NULL |
| Login | Логин | Varchar (50) | NOT NULL |
| Password | Пароль | Varchar (50) | NOT NULL |
| email | Почта | Varchar (50) | NOT NULL |

Таблица 5 – History\_Orders (История заказов)

|  |  |  |  |
| --- | --- | --- | --- |
| **Название столбца** | **Описание** | **Тип** | **Примечание** |
| Id\_record | Код записи | Int | PK, NOT NULL |
| Id\_client | Код клиента | Int | FK, NOT NULL |
| Id\_car | Код машины | Int | FK, NOT NULL |
| Id\_service | Код сервиса | Int | FK, NULL |
| Id\_part | Код запчасти | Int | FK, NULL |
| Id\_employer | Код мастера | Int | FK, NULL |

Таблица 6 – Orders (Заказы)

|  |  |  |  |
| --- | --- | --- | --- |
| **Название столбца** | **Описание** | **Тип** | **Примечание** |
| Id\_order | Код заказа | Int | PK, NOT NULL |
| Id\_client | Код клиента | Int | FK, NOT NULL |
| Id\_service | Код сервиса | Int | FK, NOT NULL |
| Id\_car | Код машины | Int | FK, NOT NULL |
| Date\_Order | Дата заказа | Date | NOT NULL |
| Execution\_Status | Статус выполнения | Int | NOT NULL |
| Price | Цена | Int | NOT NULL |

Таблица 7 - Services (Услуги)

|  |  |  |  |
| --- | --- | --- | --- |
| **Название столбца** | **Описание** | **Тип** | **Примечание** |
| Id\_service | Код услуги | Int | PK, NOT NULL |
| Name\_service | Название | Varchar (25) | NOT NULL |
| Description | Примечание | Varchar (25) | NOT NULL |
| Price | Цена | Varchar (25) | NULL |

Таблица 8 – Spare\_Parts (Запчасти)

|  |  |  |  |
| --- | --- | --- | --- |
| **Название столбца** | **Описание** | **Тип** | **Примечание** |
| Id\_part | Код запчасти | Int | PK, NOT NULL |
| Id\_service | Код услуги | Int | FK, NOT NULL |
| Name\_parts | Название | Varchar (50) | NOT NULL |
| Photo | Фото | VARBINARY (MAX) | NOT NULL |
| Description | Примечание | Varchar (50) | NOT NULL |
| Price | Цена | Int | NOT NULL |
| Presence | Количество на складе | Int | NOT NULL |

Таблица 9 – Specialities (Специальности)

|  |  |  |  |
| --- | --- | --- | --- |
| **Название столбца** | **Описание** | **Тип** | **Примечание** |
| Id\_specialties | Код специальности | Int | PK, NOT NULL |
| Name\_specialties | Название | Varchar (50) | NOT NULL |

## 2.6 Другие объекты базы данных (представления, хранимые процедуры, триггеры, имена входа, пользователи)

Для приложения не использовались представления, хранимые процедуры и триггеры.

Имена для входа: MSI\SQLEXPRESS

## 2.7 Разработка приложения

### **2.7.1 Диаграмма вариантов использования**

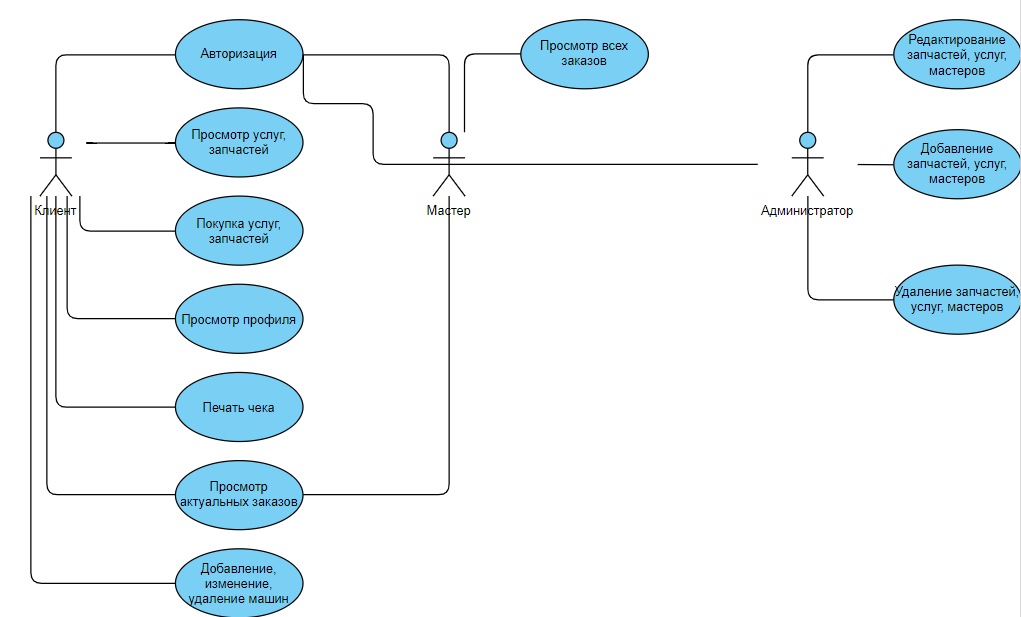


Рисунок 4 Диаграмма вариантов использования

### **2.7.2 Решение главной задачи проекта**

Корзина:

1. Выбор понравившейся запчасти (*См. рис. 4*),
2. Оформление заказа (*См. рис. 5*).

Алгоритм работы корзины: создается список cartItems, на ListView есть событие MouseDoubleClick, при двойном нажатии на выбранную запчасть, она записывается в список, и выводится в новом ListView. Также присутствуют две кнопки, очистить корзину и оформить заказ. При очищении – очищается список с элементами. При оформлении – записывается запись в таблицу History\_Orders и очищается список.

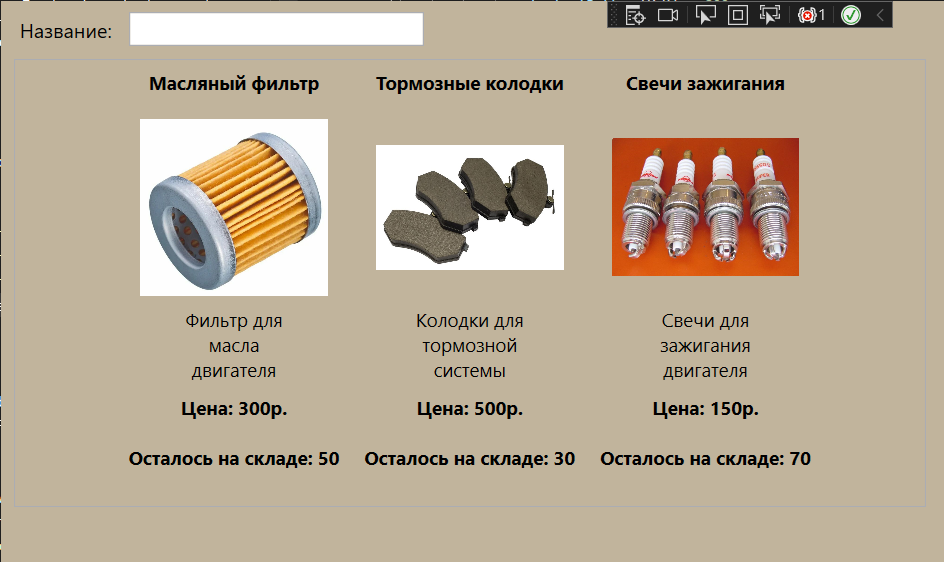


Рисунок 4 Актуальные запчасти

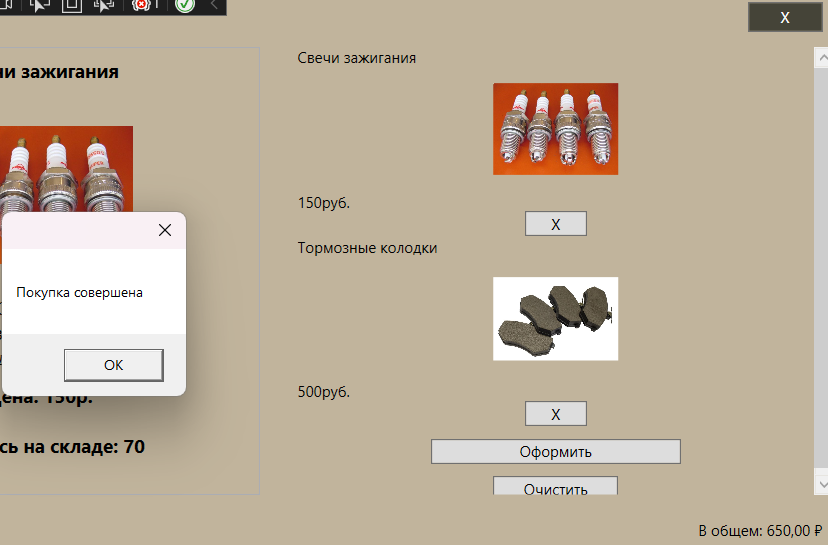


Рисунок 4 Оформление заказа

//Оформление заказа

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

{

// Создание записи в таблице History\_Order

using (var context = new auto\_repair\_shopEntities())

{

foreach (var item in cartItems)

{

var order = new History\_Orders

{

id\_client = \_client.id\_client,

id\_car = \_car.id\_car,

id\_part = item.id\_part,

};

context.History\_Orders.Add(order);

}

context.SaveChanges();

MessageBox.Show("Покупка совершена");

}

cartItems.Clear();

cartPanel.Children.Clear();

UpdateCartUI();

}

### **2.7.3 Тестирование приложения**

Таблица 10 - Тестирование

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Действие** | **Ожидаемый результат** | **Результат** |
| 1 | Запуск приложения | Открытие окна выбора авторизации | Открытие окна выбора авторизации |
| 2 | Авторизация клиента | Переход зарегистрированного клиента на окно пользователя | Зарегистрированный клиент переходит на окно пользователя |
| 3 | Авторизация мастера | Переход зарегистрированного мастера на окно мастеров | Зарегистрированный мастер переходит на окно преподавателя |
| 4 | Редактирование, добавление и удаление записей таблиц администратором | Успешное выполнение всех действий | Успешное выполнение всех действий |
| 5 | Оформление заказа | Заказ оформляется без ошибок, в последствии можно отследить в профиле | Заказ оформляется без ошибок и записывается в таблицу «Истории заказов», в последствии можно отследить в профиле |
| 6 | Заполнение полей некорректными данными | Сообщение в виде уведомления, в котором находиться описание ошибки | При попытке ввести некорректные значения для полей выводится сообщение о некорректных полях |
| 7 | Пустые формы для заполнения | Сообщение о обязательном заполнении нужной формы | При не заполнении формы, для которой требуется заполнение, выводится сообщение об ошибке. |

### **2.7.4 Защита информационной системы**

Информационная система защищена от неопытного пользователя благодаря проверкам всех входных данных на корректность ввода и обработке любых возникших исключений.

**Инструкция пользователю**

Для того чтобы начать работать с СУБД auto\_repair\_shop, необходимо запустить скрипт (под названием auto\_repair\_shop.sql) на создание SQL базы данных. Далее следует запуск приложения и авторизация клиента.

Существует уже 3 зарегистрированный человека:

1. Админ (логин: admin пароль: admin);
2. Клиент (логин: 123 пароль: 123)
3. Мастер (логин: ivan пароль: ivanpass)

Для авторизации необходимо ввести зарегистрированный логин и пароль. Если мастер, то при имеющемся аккаунте можно пройти авторизацию и в последствии использовать приложение. Если клиент, то нужно ввести логин и пароль, при правильном вводе, можно будет воспользоваться приложением.

После авторизации клиента ему доступен просмотр запчастей, услуг, своих машин, просмотр профиля и печать чека.

Авторизовавшись в роли мастера, предоставляется возможность отслеживать активные заказы, и выполнять их.

### **2.7.5 Общие сведения об информационной системе**

Вся информационная система построена на клиент-серверной модели, где присутствует взаимодействие пользователя через сервер к базе данных. В качестве базы данных выступает MS SQL SERVER Express 2019.

В качестве пользовательского приложения выступает информационная система работы автосервиса. Сервер – компьютер, на котором находится сервер.

Система предназначена для покупки запчастей и услуг для обслуживания своих машин.

### **2.7.6 Требования к техническим средствам**

В состав технических средств должен входить компьютер, или ноутбук, включающий:

1. Оперативную память 2 Гб. и более,
2. Процессор с частотой 2 ГГц. и более,
3. Клавиатуру,
4. Компьютерную мышь или тачпад,
5. Видеокарту GeForce 8400 GS и более,
6. Монитор,
7. SSD или HDD диск.

### **2.7.7 Требования к программным средствам**

В состав программных средств должно входить:

1. Операционная система Windows 7 и более,
2. IDE с .NET Framework,
3. MS SQL Server 2018 и более.

### **2.7.8 Формы ввода**

Все поля доступные для ввода, проверяют корректность ввода и наличие данных в них. Если данные не проходят проверку, выводится сообщение об ошибке (*См. рис. 5*).

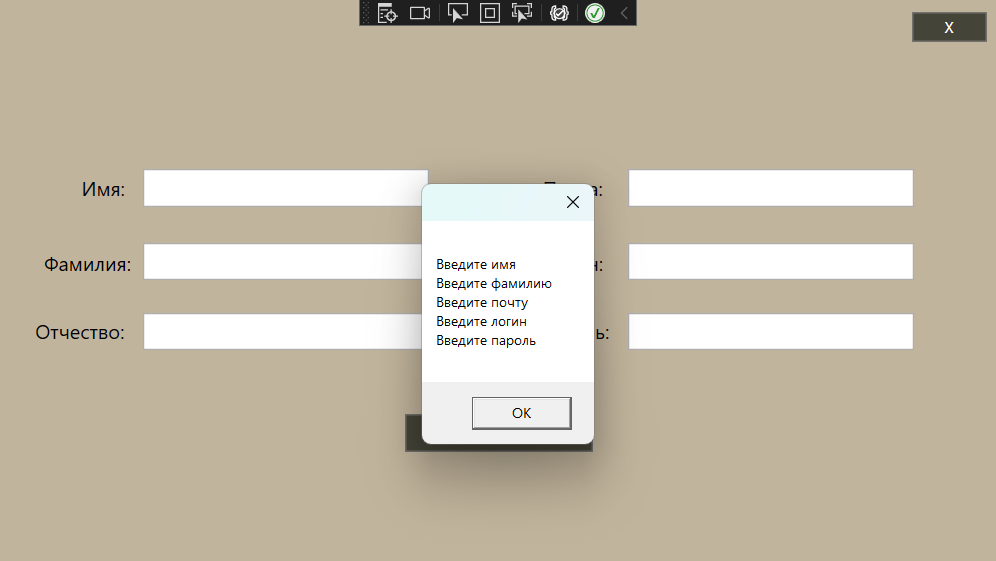


Рисунок 5 Вывод сообщения об ошибке заполнения форм

### **2.7.9 Отчёты**

При желании, клиент может создать печатную версию заказа, которая будет содержать автомобиль, услугу, дату заказа, статус выполнения и цену. Данный чек будет иметь название и будет находится в том месте, в которому будет удобно клиенту. (*См. рис. 6*).

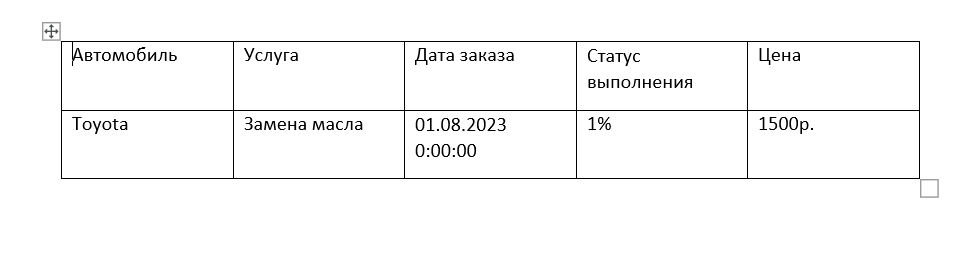


Рисунок 6 Чек

# **ЗАКЛЮЧЕНИЕ**

В ходе курсового проекта было разработано приложение, позволяющее в удобной форме записать машину на обслуживание, или купить необходимые запчасти.

Программа выполнена с учётом всех требований предметной области, таблицы соответствуют правилам нормализации, что позволяет обеспечить целостность и безопасность данных.

Приложение было разработано с помощью языка программирования C#, API-интерфейса WPF и СУБД MS SQL Server.

Поставленные задачи были выполнены:

1. Разработка логической модели базы данных,
2. Разработка физической модели базы данных,
3. Создание WPF приложения на языке C# с использованием технологии Entity Framework,
4. Защита приложения с помощью прав доступа.

Выполнение курсового проекта существенно обогатило мои навыки в области программирования на языке C#. Благодаря этому проекту, я не только укрепил свои предыдущие знания, но и приобрёл новые навыки, такие как умение работать с LINQ для эффективной обработки данных, а также овладел навыками создания документов в формате Word из приложения. Одним из важных достижений стало успешное взаимодействие между пользовательским приложением и системой управления базами данных, что расширило мои возможности в разработке более сложных и функциональных программных продуктов. Результаты этого проекта не только подтвердили мои компетенции, но и вдохновили меня на дальнейшее самосовершенствование в области программирования.

# **СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Базы данных и системы управления базами данных: Учебное пособие / Лазицкас Е.А., Загумённикова И.Н., Гилевский П.Г. - Мн.: РИПО, 2016. URL: <http://znanium.com/bookread2.php?book=946561>. (Дата обращения: 12.02.2023).
2. Определение Microsoft SQL Server. - URL: <https://ru.wikipedia.org/wiki/Microsoft_SQL_Server>. (Дата обращения: 03.04.2023).
3. Описание преимуществ и недостатков Microsoft SQL Server. - URL: <https://muzeon.ru/medicina/2912-chto-takoe-sql-server-plyusy-i-minusy-ispolzovaniya.html>. (Дата обращения: 03.05.2023).
4. Описание среды разработки Visual Studio. - URL: <https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio>. (Дата обращения: 03.02.2023).
5. Полное руководство по языку программирования С# 9.0 и платформе .NET 5. - URL: <https://metanit.com/sharp/tutorial>. (Дата обращения: 10.05.2023).
6. Руководство по WPF. - URL: <https://metanit.com/sharp/wpf/>. (Дата обращения: 12.01.2023).
7. Руководство по MS SQL Server 2018. - URL: <https://metanit.com/sql/sqlserver/>. (Дата обращения: 15.04.2023).
8. Руководство по Entity Framework: <https://learn.microsoft.com/ru-ru/ef/> (Дата обращения: 19.03.2023).

# **ПРИЛОЖЕНИЕ А. SQL скрипты**

USE [master]

IF DB\_ID('auto\_repair\_shop') IS NOT NULL

DROP DATABASE auto\_repair\_shop

GO

CREATE DATABASE auto\_repair\_shop

GO

USE auto\_repair\_shop

IF OBJECT\_ID(N'dbo.Clients', N'U') IS NOT NULL

DROP TABLE Clients

IF OBJECT\_ID(N'dbo.Cars', N'U') IS NOT NULL

DROP TABLE Cars

IF OBJECT\_ID(N'dbo.Orders', N'U') IS NOT NULL

DROP TABLE Orders

IF OBJECT\_ID(N'dbo.Spare\_Parts', N'U') IS NOT NULL

DROP TABLE Spare\_Parts

IF OBJECT\_ID(N'dbo.Services', N'U') IS NOT NULL

DROP TABLE Services

IF OBJECT\_ID(N'dbo.Specialities', N'U') IS NOT NULL

DROP TABLE Specialities

IF OBJECT\_ID(N'dbo.Employees', N'U') IS NOT NULL

DROP TABLE Employees

IF OBJECT\_ID(N'dbo.History\_Orders', N'U') IS NOT NULL

DROP TABLE History\_Orders

GO

CREATE TABLE Clients(

id\_client int IDENTITY(1, 1) PRIMARY KEY,

name varchar(50) NULL,

surname varchar(50) NULL,

patronymic varchar(50) NULL,

login varchar(50) NULL,

password varchar(50) NULL,

email varchar(100) NULL,

)

GO

CREATE TABLE Cars(

id\_car int IDENTITY(1, 1) PRIMARY KEY,

id\_client int NULL,

car\_brand varchar(50) NULL,

car\_model varchar(50) NULL,

year\_of\_manufacture int NULL,

number\_car varchar(10) NULL,

photo varbinary(MAX) NULL,

FOREIGN KEY(id\_client) REFERENCES Clients(id\_client) ON DELETE CASCADE ON UPDATE CASCADE

)

GO

CREATE TABLE Services(

id\_service int IDENTITY(1, 1) PRIMARY KEY,

name\_service varchar(50) NULL,

description varchar(100) NULL,

price int NULL,

)

GO

CREATE TABLE Orders(

id\_order int IDENTITY(1, 1) PRIMARY KEY,

id\_client int NULL,

id\_service int NULL,

id\_car int NULL,

date\_order date NULL,

execution\_status int NULL,

price int NULL,

FOREIGN KEY(id\_client) REFERENCES Clients(id\_client) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY(id\_service) REFERENCES Services(id\_service),

FOREIGN KEY(id\_car) REFERENCES Cars(id\_car),

)

GO

CREATE TABLE Spare\_Parts(

id\_part int IDENTITY(1, 1) PRIMARY KEY,

id\_service int NULL,

name\_parts varchar(50) NULL,

photo varbinary(MAX) NULL,

description varchar(100) NULL,

price int NULL,

presence int NULL,

FOREIGN KEY(id\_service) REFERENCES Services(id\_service) ON DELETE CASCADE ON UPDATE CASCADE,

)

GO

CREATE TABLE Specialities(

id\_specialties int IDENTITY(1, 1) PRIMARY KEY,

name\_specialties varchar(50) NULL,

)

GO

CREATE TABLE Employees(

id\_employee int IDENTITY(1, 1) PRIMARY KEY,

id\_specialties int NULL,

name varchar(50) NULL,

surname varchar(50) NULL,

patronymic varchar(50) NULL,

job\_title varchar(50) NULL,

login varchar(50) NULL,

password varchar(50) NULL,

email varchar(50) NULL,

FOREIGN KEY (id\_specialties) REFERENCES Specialities(id\_specialties) ON DELETE CASCADE ON UPDATE CASCADE

)

GO

CREATE TABLE History\_Orders(

id\_record int IDENTITY(1, 1) PRIMARY KEY,

id\_client int NULL,

id\_car int NULL,

id\_services int NULL,

id\_part int NULL,

id\_employer int NULL,

FOREIGN KEY(id\_client) REFERENCES Clients(id\_client),

FOREIGN KEY(id\_car) REFERENCES Cars(id\_car),

FOREIGN KEY(id\_services) REFERENCES Services(id\_service),

FOREIGN KEY(id\_part) REFERENCES Spare\_Parts(id\_part),

FOREIGN KEY(id\_employer) REFERENCES Employees(id\_employee),

)

GO

CREATE TABLE Admin(

login varchar(50) PRIMARY KEY,

password varchar(50) NULL,

)

GO

INSERT INTO Clients (name, surname, patronymic, login, password, email)

VALUES

('Иван', 'Иванов', 'Иванович', '123', '123', 'ivan@example.com'),

('Мария', 'Петрова', 'Александровна', 'maria456', 'mariapass', 'maria@example.com'),

('Алексей', 'Сидоров', 'Павлович', 'alex789', 'alexpass', 'alex@example.com');

INSERT INTO Cars (id\_client, car\_brand, car\_model, year\_of\_manufacture, number\_car)

VALUES

(1, 'Toyota', 'Camry', 2018, 'A123BC'),

(2, 'Ford', 'Focus', 2020, 'X456YZ'),

(3, 'Chevrolet', 'Cruze', 2019, 'K789LM');

INSERT INTO Services (name\_service, description, price)

VALUES

('Замена масла', 'Замена моторного масла и фильтра', 800),

('Ремонт тормозов', 'Работы по ремонту тормозной системы', 1200),

('Замена свечей зажигания', 'Замена свечей зажигания в двигателе', 300);

INSERT INTO Orders (id\_client, id\_service, id\_car, date\_order, execution\_status, price)

VALUES

(1, 1, 1, '2023-08-01', 1, 1500),

(2, 2, 2, '2023-08-05', 2, 2000),

(3, 3, 3, '2023-08-10', 1, 1800);

INSERT INTO Spare\_Parts (id\_service, name\_parts, description, price, presence)

VALUES

(1, 'Масляный фильтр', 'Фильтр для масла двигателя', 300, 50),

(2, 'Тормозные колодки', 'Колодки для тормозной системы', 500, 30),

(3, 'Свечи зажигания', 'Свечи для зажигания двигателя', 150, 70);

INSERT INTO Specialities (name\_specialties)

VALUES

('Механик'),

('Электрик'),

('Слесарь');

INSERT INTO Employees (id\_specialties, name, surname, patronymic, job\_title, login, password, email)

VALUES

(1, 'Петр', 'Смирнов', 'Игоревич', 'Механик', 'petrmech', 'petrpass', 'petr@example.com'),

(2, 'Елена', 'Васильева', 'Андреевна', 'Электрик', 'elenaelec', 'elenapass', 'elena@example.com'),

(3, 'Иван', 'Зайцев', 'Алексеевич', 'Слесарь', 'ivan', 'ivanpass', 'ivan@example.com');

INSERT INTO History\_Orders (id\_client, id\_car, id\_services, id\_part, id\_employer)

VALUES

(1, 1, 1, 1, 1),

(2, 2, 3, 2, 2),

(3, 3, 2, 3, 3);

INSERT INTO Admin (login, password)

VALUES

('admin', 'admin')

UPDATE Cars

SET photo = (SELECT \* FROM OPENROWSET(BULK N'C:\Users\nb msi\Desktop\auto\_repair\_shopProject\photo\toyota.jpg', SINGLE\_BLOB) AS image)

WHERE id\_client = 1

UPDATE Cars

SET photo = (SELECT \* FROM OPENROWSET(BULK N'C:\Users\nb msi\Desktop\auto\_repair\_shopProject\photo\Ford\_Focus.jpg', SINGLE\_BLOB) AS image)

WHERE id\_client = 2

UPDATE Cars

SET photo = (SELECT \* FROM OPENROWSET(BULK N'C:\Users\nb msi\Desktop\auto\_repair\_shopProject\photo\chevrolet\_cruze.jpg', SINGLE\_BLOB) AS image)

WHERE id\_client = 3

UPDATE Spare\_Parts

SET photo = (SELECT \* FROM OPENROWSET(BULK N'C:\Users\nb msi\Desktop\auto\_repair\_shopProject\photo\maslo.jpg', SINGLE\_BLOB) AS image)

WHERE name\_parts = 'Масляный фильтр'

UPDATE Spare\_Parts

SET photo = (SELECT \* FROM OPENROWSET(BULK N'C:\Users\nb msi\Desktop\auto\_repair\_shopProject\photo\tormoz.jpg', SINGLE\_BLOB) AS image)

WHERE name\_parts = 'Тормозные колодки'

UPDATE Spare\_Parts

SET photo = (SELECT \* FROM OPENROWSET(BULK N'C:\Users\nb msi\Desktop\auto\_repair\_shopProject\photo\svechi.jpg', SINGLE\_BLOB) AS image)

WHERE name\_parts = 'Свечи зажигания'

# **ПРИЛОЖЕНИЕ Б. Код программы**

// Представляет окно администратора в GUI.

public partial class AdminWind : Window

{

// Конструктор класса AdminWind. Инициализирует компоненты окна.

public AdminWind()

{

InitializeComponent();

}

// Обработчик события нажатия на кнопку employee.

// Отображает страницу деталей сотрудников в основном фрейме.

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

{

mainFrame.Content = new EmployeeAdmin();

}

// Обработчик события нажатия на кнопку service.

// Отображает страницу деталей услуг в основном фрейме.

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

{

mainFrame.Content = new ServicesAdmin();

}

// Обработчик события нажатия на кнопку parts.

// Отображает страницу деталей запчастей в основном фрейме.

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

{

mainFrame.Content = new SparePartsAdmin();

}

// Обработчик события нажатия на кнопку specialities.

// Отображает страницу деталей специализаций в основном фрейме.

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

{

mainFrame.Content = new SpecialitiesAdmin();

}

// Обработчик события нажатия на кнопку close.

// Закрывает текущее окно и открывает окно входа/выхода.

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

{

LogWind wind = new LogWind();

wind.Show();

Close();

}

}

}

public partial class CreateCarWind : Window

{

// Объявляет объекты Cars и Clients.

Cars \_cars = new Cars();

Clients \_client;

// Конструктор класса CreateCarWind, который инициализирует компоненты приложения.

public CreateCarWind(Clients client, Cars selectedCars)

{

InitializeComponent();

// Проверке на null переданного клиента. Если клиента нет - функция будет прекращена до обращения к неопределенному значению.

if (client == null)

return;

// Если передана конкретная машина, она устанавливается как текущая машина.

if (selectedCars != null)

\_cars = selectedCars;

// Установка максимальной длины для полей ввода деталей автомобиля.

brand.MaxLength = 20;

model.MaxLength = 20;

year.MaxLength = 4;

number.MaxLength = 10;

// Установка клиента.

\_client = client;

// Установка контекста данных для текущего окна.

DataContext = \_cars;

}

// Обработчик нажатия кнопки "Войти".

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

{

// Создание объекта StringBuilder для сбора ошибок.

StringBuilder errors = new StringBuilder();

// Проверка данных, введенных во все поля, на их наличие.

if (string.IsNullOrWhiteSpace(\_cars.car\_brand))

errors.AppendLine("Введите бренд");

if (string.IsNullOrWhiteSpace(\_cars.car\_model))

errors.AppendLine("Введите модель");

if (string.IsNullOrWhiteSpace(\_cars.year\_of\_manufacture.ToString()))

errors.AppendLine("Введите год выпуска");

if (string.IsNullOrWhiteSpace(\_cars.number\_car))

errors.AppendLine("Введите номер машины");

// Проверка, выбрано ли изображение автомобиля.

if (\_cars.photo == null || \_cars.photo.Length == 0)

errors.AppendLine("Выберите изображение");

// Если есть ошибки, они отображаются пользователю в диалоговом окне, а функция прекращается.

if (errors.Length > 0)

{

MessageBox.Show(errors.ToString());

return;

}

// Если ID автомобиля равен 0, то это новый автомобиль, и он добавляется в базу данных.

if (\_cars.id\_car == 0)

{

\_cars.id\_client = \_client.id\_client;

auto\_repair\_shopEntities.GetContext().Cars.Add(\_cars);

}

// Завершается обновление базы данных.

try

{

auto\_repair\_shopEntities.GetContext().SaveChanges();

// Затем открывается главное окно.

MainWindow wind = new MainWindow(\_client);

wind.Show();

Close();

}

// В случае ошибки в сообщении отображается информация о ней.

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

// Если затем необходимо обновить изображение машины.

if (\_cars.photo != null || \_cars.photo.Length > 0)

{

var car = auto\_repair\_shopEntities.GetContext().Cars.Find(\_cars.id\_car);

if (car != null) car.photo = \_cars.photo;

auto\_repair\_shopEntities.GetContext().SaveChanges();

}

}// Обработчик события нажатия кнопки addImage.

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

{

// Создается диалоговое окно OpenFileDialog чтобы позволить пользователю выбрать изображение.

var openFile = new OpenFileDialog();

openFile.Filter = "Image files (\*.png;\*.jpeg;\*.jpg)|\*.png;\*.jpeg;\*.jpg";

if (openFile.ShowDialog() == true)

{

// Затем изображение преобразуется в массив байтов и сохраняется в объекте \_cars.

\_cars.photo = File.ReadAllBytes(openFile.FileName);

photo.Source = LoadImage(\_cars.photo);

}

}

// Этот метод используется для преобразования массива байтов в изображение BitmapImage, которое можно отобразить в интерфейсе.

private BitmapImage LoadImage(byte[] imageData)

{

if (imageData == null || imageData.Length == 0)

return null;

var image = new BitmapImage();

using (var stream = new MemoryStream(imageData))

{

stream.Position = 0;

image.BeginInit();

image.CacheOption = BitmapCacheOption.OnLoad;

image.StreamSource = stream;

image.EndInit();

}

return image;

}

// Обработчики событий для проверки входных данных в полях для бренда, года и номера автомобиля.

private void brand\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

Regex rgx = new Regex("[^A-Za-z]");

e.Handled = rgx.IsMatch(e.Text);

}

private void year\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

Regex rgx = new Regex("[^0-9]");

e.Handled = rgx.IsMatch(e.Text);

}

private void number\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

Regex rgx = new Regex("[^А-Яа-я0-9]");

e.Handled = rgx.IsMatch(e.Text);

}

// Обработчик закрытия текущего окна.

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

{

MainWindow wind = new MainWindow(\_client);

wind.Show();

Close();

}

}

}

public partial class AllOrdersWind : Window

{

// Объевление переменной для хранения текущего сотрудника.

private Employees \_employee;

// Конструктор класса AllOrdersWind, который инициализирует компоненты приложения и загружает заказы на ремонт.

public AllOrdersWind(Employees employee)

{

InitializeComponent();

// Установка переданного сотрудника.

\_employee = employee;

// Получение списка всех сотрудников из БД.

var employ = auto\_repair\_shopEntities.GetContext().Employees.ToList();

// Добавление опции "Все мастера" в начало списка.

employ.Insert(0, new Employees

{

surname = "Все мастера"

});

// Установка источника данных для комбо-бокса сотрудников.

comboEmpl.ItemsSource = employ;

// Установка начального выбора в комбо-боксе на "Все мастера".

comboEmpl.SelectedIndex = 0;

// Обновление представления с заказами.

UpdateGrid();

}

// Метод обновления элемента управления gridOrder, отображающего все заказы.

private void UpdateGrid()

{

// Перезагрузка значений всех элементов ChangeTracker в контексте БД.

auto\_repair\_shopEntities.GetContext().ChangeTracker.Entries().ToList().ForEach(p => p.Reload());

// Получение всех заказов у которых указан сотрудник и их вывод в gridOrder.

gridOrder.ItemsSource = auto\_repair\_shopEntities.GetContext().History\_Orders.Where(p => p.id\_employer != null).ToList();

}

// Обработчик события изменения выбранного элемента в комбо-боксе сотрудников.

private void comboEmpl\_SelectionChanged(object sender, System.Windows.Controls.SelectionChangedEventArgs e)

{

// Если выбран пункт "Все мастера", выводятся все доступные заказы.

if (comboEmpl.SelectedIndex == 0)

{

gridOrder.ItemsSource = GetAviableStudents();

}

// Если выбран конкретный сотрудник, выводятся заказы, относящиеся к этому сотруднику.

else

{

var employ = comboEmpl.SelectedItem as Employees;

if (employ != null)

{

ICollectionView view = CollectionViewSource.GetDefaultView(gridOrder.ItemsSource);

view.Filter = s => ((History\_Orders)s).id\_employer == employ.id\_employee;

gridOrder.ItemsSource = view;

}

}

}

// Функция возвращает список всех доступных заказов.

private List<History\_Orders> GetAviableStudents()

{

return auto\_repair\_shopEntities.GetContext().History\_Orders.Where(p => p.id\_employer != null).ToList();

}

// Обработчик события нажатия кнопки закрытия текущего окна.

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

{

EmployeeWind wind = new EmployeeWind(\_employee);

wind.Show();

Close();

}

}

}

public partial class EmployeeAdd : Page

{

// Обьявление переменной для нового сотрудника

private Employees \_employee = new Employees();

// Конструктор, который инициализирует компоненты страницы и загружает данные о сотрудниках.

public EmployeeAdd(Employees selectedEmployee)

{

InitializeComponent();

// Загрузка специальностей из БД и установка в качестве источника данных для комбо-бокса.

var listSpec = auto\_repair\_shopEntities.GetContext().Specialities.ToList();

comboSpec.ItemsSource = listSpec;

// Если выбран сотрудник, его данные загружаются в соответствующие поля.

if (selectedEmployee != null)

{

\_employee = selectedEmployee;

var spec = listSpec.Where(g => g.id\_specialties.Equals(selectedEmployee.id\_specialties));

var specEnumerable = spec as Specialities[] ?? spec.ToArray();

if (specEnumerable.Any())

comboSpec.SelectedValue = specEnumerable.First();

}

// Установка максимальной длины для каждого из полей ввода.

name.MaxLength = 20;

surname.MaxLength = 30;

patronymic.MaxLength = 40;

jobtitle.MaxLength = 40;

email.MaxLength = 60;

login.MaxLength = 20;

password.MaxLength = 20;

// Установка контекста данных для этой страницы.

DataContext = \_employee;

}

// Обработчик события клика по кнопке "Сохранить".

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

{

StringBuilder errors = new StringBuilder();

// Объявление паттерна и регулярного выражение для проверки вводимого email.

string pattern = @"^([\w\.\-]+)@([\w\-]+)((\.(\w){2,3})+)$";

Regex regex = new Regex(pattern);

// Проверка на введенность и корректность значений в поле email и других полей.

if (string.IsNullOrWhiteSpace(\_employee.name))

errors.AppendLine("Введите имя");

if (string.IsNullOrWhiteSpace(\_employee.surname))

errors.AppendLine("Введите фамилию");

if (string.IsNullOrWhiteSpace(\_employee.job\_title))

errors.AppendLine("Введите должность");

if (string.IsNullOrWhiteSpace(\_employee.email))

errors.AppendLine("Введите почту");

else

{

if (!regex.IsMatch(\_employee.email))

errors.AppendLine("Почта введена неккоректно");

}

if (string.IsNullOrWhiteSpace(\_employee.login))

errors.AppendLine("Введите логин");

if (string.IsNullOrWhiteSpace(\_employee.password))

errors.AppendLine("Введите пароль");

// Присваивание специальности выбранной сотруднику

if (comboSpec.SelectedItem is Specialities selectedSpec)

\_employee.id\_specialties = selectedSpec.id\_specialties;

// Проверка, была ли выбрана специальность.

if (comboSpec.SelectedItem == null)

errors.AppendLine("Выберите специальность");

// Если есть ошибки, показать их и прекратить выполнение функции.

if (errors.Length > 0)

{

MessageBox.Show(errors.ToString());

return;

}

// Если id сотрудника равен 0, добавить нового сотрудника в базу данных.

if (\_employee.id\_employee == 0)

auto\_repair\_shopEntities.GetContext().Employees.Add(\_employee);

// Попытка сохранить изменения в БД.

try

{

auto\_repair\_shopEntities.GetContext().SaveChanges();

NavigationService.GoBack();

}

// Обработка исключений и вывод сообщения об ошибке.

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

// Обработчик события валидации вводимых значений для имени сотрудника.

private void name\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

Regex rgx = new Regex("[^А-Яа-я]+");

e.Handled = rgx.IsMatch(e.Text);

}

// Обработчик события валидации вводимых значений для email сотрудника.

private void email\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

Regex rgx = new Regex("[^a-zA-Z0-9@.-]+");

e.Handled = rgx.IsMatch(e.Text);

}

// Обработчик события валидации вводимых значений для логина сотрудника.

private void login\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

Regex rgx = new Regex("[^a-zA-Z0-9]+");

e.Handled = rgx.IsMatch(e.Text);

}

// Обработчик события клика по кнопке "Закрыть".

// Закрытие текущей страницы и возврат на предыдущую.

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

{

NavigationService.GoBack();

}

}

}

public partial class EmployeeAdmin : Page

{

// Конструктор по умолчанию, который инициирует все визуальные компоненты страницы.

public EmployeeAdmin()

{

InitializeComponent();

}

// Метод обновления данных работников: выполняется перезагрузка записей из области уведомлений о изменениях

// и затем обновляется источник данных для GridView с работниками.

private void UpdateEmployee()

{

auto\_repair\_shopEntities.GetContext().ChangeTracker.Entries().ToList().ForEach(p => p.Reload());

gridEmployee.ItemsSource = auto\_repair\_shopEntities.GetContext().Employees.ToList();

}

// Обработчик события нажатия на кнопку удаления.

// Если работник выбран, отображается диалоговое окно с запросом на подтверждение.

// При подтверждении, выбранный работник удаляется из базы данных.

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

{

if (gridEmployee.SelectedItem != null)

{

if (MessageBox.Show("Удалить специалиста?", "Уведомление", MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)

{

if (gridEmployee.SelectedItem is Employees selectedEmployee)

{

auto\_repair\_shopEntities.GetContext().Employees.Remove(selectedEmployee);

auto\_repair\_shopEntities.GetContext().SaveChanges();

UpdateEmployee();

}

}

}

else

MessageBox.Show("Выберите запись для удаления");

}

// Обработчик события нажатия на кнопку добавления.

// Пользователь перенаправляется на страницу добавления нового работника.

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

{

NavigationService.Navigate(new EmployeeAdd(null));

}

// Обработчик события нажатия на кнопку редактирования.

// Если работник выбран, пользователь перенаправляется на страницу редактирования данных этого работника.

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

{

if (gridEmployee.SelectedItem != null)

{

if (gridEmployee.SelectedItem is Employees selectedEmployee)

NavigationService.Navigate(new EmployeeAdd(selectedEmployee));

}

else

MessageBox.Show("Выберите запись для изменения");

}

// Обработчик события изменения видимости страницы.

// Когда страница становится видимой, обновляются данные работников.

private void Page\_IsVisibleChanged(object sender, DependencyPropertyChangedEventArgs e)

{

UpdateEmployee();

}

}

}

// Класс окна сотрудника.

public partial class EmployeeWind : Window

{

// Приватное поле \_employee типа Employees (сотрудники).

private Employees \_employee;

// Конструктор для EmployeeWind, принимающий сотрудника (подразумевается пользователь, который авторизован в системе).

public EmployeeWind(Employees employee)

{

InitializeComponent();

// Инициализация \_employee этим сотрудником.

\_employee = employee;

// Обновление заказов.

UpdateOrders();

}

// Метод для обновления заказов.

private void UpdateOrders()

{

// Перезагрузка всех записей из ChangeTracker (контроль изменений).

auto\_repair\_shopEntities.GetContext().ChangeTracker.Entries().ToList().ForEach(p => p.Reload());

// Задать источник данных для viewNewOrder как все заказы из контекста данных (то есть из базы данных напрямую).

viewNewOrder.ItemsSource = auto\_repair\_shopEntities.GetContext().Orders.ToList();

}

// Обработчик двойного клика мыши по элементу в viewNewOrder.

private void viewNewOrder\_MouseDoubleClick(object sender, MouseButtonEventArgs e)

{

if (viewNewOrder.SelectedItem != null)

{

if (viewNewOrder.SelectedItem is Orders selectedOrder)

{

// Показывается сообщение с подтверждением завершения заказа.

if (MessageBox.Show("Вы хотите завершить заказ?", "Уведомление", MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)

{

// Найти клиента по id в выбранном заказе.

var client = auto\_repair\_shopEntities.GetContext().Clients.FirstOrDefault(p => p.id\_client == selectedOrder.id\_client);

// Создать новый завершенный заказ.

var complitedOrder = new History\_Orders

{

id\_client = client.id\_client,

id\_services = selectedOrder.id\_service,

id\_car = selectedOrder.id\_car,

id\_employer = \_employee.id\_employee,

};

// Добавить завершенный заказ в контекст данных и сохранить изменения.

auto\_repair\_shopEntities.GetContext().History\_Orders.Add(complitedOrder);

auto\_repair\_shopEntities.GetContext().SaveChanges();

MessageBox.Show("Заказ завершен");

}

// Удалить выбранный заказ из контекста данных и сохранить изменения.

auto\_repair\_shopEntities.GetContext().Orders.Remove(selectedOrder);

auto\_repair\_shopEntities.GetContext().SaveChanges();

// Обновить заказы.

UpdateOrders();

}

}

}

// Обработчик клика по кнопке "Закрыть".

// Показывает окно входа и закрывает текущее окно.

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

{

LogWind wind = new LogWind();

wind.Show();

Close();

}

// Обработчик клика по кнопке "Показать все заказы".

// Показывает окно со всеми заказами и закрывает текущее окно.

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

{

AllOrdersWind wind = new AllOrdersWind(\_employee);

wind.Show();

Close();

}

}

}

// Класс окна авторизации

public partial class LogWind : Window

{

public LogWind()

{

InitializeComponent();

// Установка максимальной длины ввода для полей ввода логина и пароля

LoginBox.MaxLength = 20;

PswdBox.MaxLength = 20;

Password.MaxLength = 20;

}

// Обработчик для события клика на кнопку "checkPas"

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

{

// Если флажок "checkPas" установлен, то отображаем открытый текст в PswdBox и скрываем закрытый в Password. Делаем наоборот, если флажок снят.

if (checkPas.IsChecked != null && checkPas.IsChecked.Value)

{

PswdBox.Text = Password.Password;

PswdBox.Visibility = Visibility.Visible;

Password.Visibility = Visibility.Hidden;

}

else

{

Password.Password = PswdBox.Text;

PswdBox.Visibility = Visibility.Hidden;

Password.Visibility = Visibility.Visible;

}

}

// Обработчик для события клика на кнопку входа ("BtnLog")

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

{

// Если поля логина и пароля пусты, то выводим сообщение об ошибке

if (string.IsNullOrWhiteSpace(LoginBox.Text) || string.IsNullOrWhiteSpace(Password.Password))

{

MessageBox.Show("Введите все данные", "Уведомление");

}

else

{

// Ищем совпадающий логин для клиента, сотрудника и администратора

Clients client = auto\_repair\_shopEntities.GetContext().Clients.FirstOrDefault(p => p.login == LoginBox.Text);

Employees employee = auto\_repair\_shopEntities.GetContext().Employees.FirstOrDefault(p => p.login == LoginBox.Text);

Admin admin = auto\_repair\_shopEntities.GetContext().Admin.FirstOrDefault(p => p.login == LoginBox.Text);

// Если данные совпадают с данными клиента, открываем окно клиента

if (client != null && client.password == Password.Password || client != null && client.password == PswdBox.Text)

{

CreateCarWind wind = new CreateCarWind(client, null);

wind.Show();

Close();

}

// Если данные совпадают с данными сотрудника, открываем окно сотрудника

else if (employee != null && employee.password == Password.Password || employee != null && employee.password == PswdBox.Text)

{

EmployeeWind wind = new EmployeeWind(employee);

wind.Show();

Close();

}

// Если данные совпадают с данными администратора, открываем окно администратора

else if (admin != null && admin.password == Password.Password || admin != null && admin.password == PswdBox.Text)

{

AdminWind wind = new AdminWind();

wind.Show();

Close();

}

else

{

// Если данные не совпадают, показываем сообщение об ошибке

MessageBox.Show("Проверьте корректность данных", "Уведомление");

}

}

}

// Обработчик события ввода в поле LoginBox, позволяет вводить только цифры и латинские буквы

private void LoginBox\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

Regex rgx = new Regex("[^a-zA-Z0-9]+");

e.Handled = rgx.IsMatch(e.Text);

}

// Обработчик для события клика на кнопку регистрации ("BtnReg"), открывает окно регистрации

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

{

RegWind wind = new RegWind(null, 0);

wind.Show();

Close();

}

// Обработчик для события клика на кнопку "close", закрывает текущее окно

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

{

Close();

}

}

}

public partial class SparePartsAdd : Page

{

// Объявление приватного поля для хранения информации о части

private Spare\_Parts \_parts = new Spare\_Parts();

// Конструктор класса

public SparePartsAdd(Spare\_Parts parts)

{

InitializeComponent();

// Получение списка сервисов и привязка его к выпадающему списку

var listService = auto\_repair\_shopEntities.GetContext().Services.ToList();

comboService.ItemsSource = listService;

// Если используется существующая часть, то заполняются соответствующие поля

if (parts != null)

{

\_parts = parts;

// Выбор сервиса, к которому относится запасная часть, и установка его в выпадающем списке

var service = listService.Where(g => g.id\_service.Equals(parts.id\_service));

var serviceEnumerable = service as Services[] ?? service.ToArray();

if (serviceEnumerable.Any())

comboService.SelectedValue = serviceEnumerable.First();

}

// Установка максимальной длины вводимого текста для соответствующих полей

name.MaxLength = 30;

desc.MaxLength = 60;

price.MaxLength = 5;

presence.MaxLength = 5;

// Установка контекста данных этой страницы на рассматриваемую запасную часть

DataContext = \_parts;

}

// Обработчик события нажатия кнопки "Save"

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

{

StringBuilder errors = new StringBuilder();

// Проверка на валидность введенных данных и, при необходимости, добавление ошибок в StringBuilder

if (string.IsNullOrWhiteSpace(\_parts.name\_parts))

errors.AppendLine("Введите название");

if (string.IsNullOrWhiteSpace(\_parts.description))

errors.AppendLine("Введите премечание");

if (string.IsNullOrWhiteSpace(\_parts.price.ToString()))

errors.AppendLine("Введите цену");

if (string.IsNullOrWhiteSpace(\_parts.presence.ToString()))

errors.AppendLine("Введите количество на складе");

// Установка выбранного сервиса для части

if (comboService.SelectedItem is Services selectedService)

\_parts.id\_service = selectedService.id\_service;

// Проверка, что сервис был выбран

if (comboService.SelectedItem == null)

errors.AppendLine("Выберите сервис");

// Если есть ошибки, выводим сообщение с ошибками и останавливаем выполнение метода

if (errors.Length > 0)

{

MessageBox.Show(errors.ToString());

return;

}

// Если это новая часть, то добавляем её в базу данных

if (\_parts.id\_part == 0)

auto\_repair\_shopEntities.GetContext().Spare\_Parts.Add(\_parts);

try

{

// Попытка сохранить изменения в базе данных и вернуться к предыдущей странице

auto\_repair\_shopEntities.GetContext().SaveChanges();

NavigationService.GoBack();

}

catch (Exception ex)

{

// Если произошла ошибка, выводим сообщение об ошибке

MessageBox.Show(ex.Message);

}

// Если есть изображение части, сохраняем его в базе данных

if (\_parts.photo != null || \_parts.photo.Length > 0)

{

var parts = auto\_repair\_shopEntities.GetContext().Spare\_Parts.Find(\_parts.id\_part);

if (parts != null) parts.photo = \_parts.photo;

auto\_repair\_shopEntities.GetContext().SaveChanges();

}

}

// Обработчик события нажатия кнопки "Add Image"

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

{

var openFile = new OpenFileDialog();

// Фильтр для типов файлов, которые можно выбрать

openFile.Filter = "Image files (\*.png;\*.jpeg;\*.jpg)|\*.png;\*.jpeg;\*.jpg";

// Если выбран файл, читаем его и присваиваем массиву байтов фотографии

if (openFile.ShowDialog() == true)

{

\_parts.photo = File.ReadAllBytes(openFile.FileName);

photo.Source = LoadImage(\_parts.photo);

}

}

// Метод для загрузки картинки из массива байтов

private BitmapImage LoadImage(byte[] imageData)

{

if (imageData == null || imageData.Length == 0)

return null;

var image = new BitmapImage();

using (var stream = new MemoryStream(imageData))

{

stream.Position = 0;

image.BeginInit();

image.CacheOption = BitmapCacheOption.OnLoad;

image.StreamSource = stream;

image.EndInit();

}

return image;

}

// Обработчик события ввода имени. Позволяет вводить только буквы.

private void name\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

Regex rgx = new Regex("[^А-Яа-я]");

e.Handled = rgx.IsMatch(e.Text);

}

// Обработчик события ввода цены. Позволяет вводить только цифры.

private void price\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

Regex rgx = new Regex("[^0-9]");

e.Handled = rgx.IsMatch(e.Text);

}

// Обработчик события нажатия на кнопку "Close". Возвращает нас к предыдущей странице.

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

{

NavigationService.GoBack();

}

}

}

public partial class SparePartsAdmin : Page

{

// Конструктор класса SparePartsAdmin - здесь инциализируются компоненты WPF страницы.

public SparePartsAdmin()

{

InitializeComponent();

}

// Функция, которая перезагружает записи из базы данных и обновляет источник привязки данных для таблицы на странице.

private void UpdateParts()

{

auto\_repair\_shopEntities.GetContext().ChangeTracker.Entries().ToList().ForEach(p => p.Reload());

gridParts.ItemsSource = auto\_repair\_shopEntities.GetContext().Spare\_Parts.ToList();

}

// Обработчик нажатия кнопки "Удалить". Если выбран элемент в таблице, он будет удалён после подтверждения от пользователя.

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

{

// проверка, выбран ли элемент для удаления

if (gridParts.SelectedItem != null)

{

// запрос на подтверждение удаления

if (MessageBox.Show("Удалить запчасть?", "Уведомление", MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)

{

// если пользователь подтвердил удаление, удаляем запись из базы данных и обновляем таблицу

if (gridParts.SelectedItem is Spare\_Parts selectedParts)

{

auto\_repair\_shopEntities.GetContext().Spare\_Parts.Remove(selectedParts);

auto\_repair\_shopEntities.GetContext().SaveChanges();

UpdateParts();

}

}

}

else

// если не выбран элемент для удаления, выводим сообщение об этом

MessageBox.Show("Выберите запись для удаления");

}

// Обработчик события нажатия на кнопку "Добавить". Открывает страницу для добавления новой запчасти.

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

{

NavigationService.Navigate(new SparePartsAdd(null));

}

// Обработчик события нажатия на кнопку "Изменить". Открывает страницу для изменения выбранной запчасти.

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

{

// проверка, выбран ли элемент для редактирования

if (gridParts.SelectedItem != null)

{

if (gridParts.SelectedItem is Spare\_Parts selectedParts)

NavigationService.Navigate(new SparePartsAdd(selectedParts));

}

}

// Обработчик, который вызывается, когда страница становится видимой или невидимой. Это используется для обновления таблицы при возвращении на страницу.

private void Page\_IsVisibleChanged(object sender, DependencyPropertyChangedEventArgs e)

{

UpdateParts();

}

}

}

// Объявление класса SparePartsWind, который наследует класс Window.

public partial class SparePartsWind : Window

{

// Здесь создаются закрытые члены данных для классов Clients и Cars, которые будут использоваться этим окном.

private Clients \_client;

private Cars \_car;

// Создание списка для отслеживания выбранного пользователем контента.

private List<Spare\_Parts> cartItems = new List<Spare\_Parts>();

private StackPanel cartPanel;

private TextBlock totalPriceLabel;

// Конструктор класса SparePartsWind.

public SparePartsWind(Clients client, Cars car)

{

InitializeComponent();

// Присваивание переданного объекта client к \_client

\_client = client;

// Присваивание переданного объекта car к \_car

\_car = car;

// Сортировка доступных запчастей

PartsSort();

// Инициализация пользовательского интерфейса

InitializeUI();

}

// Метод, сортирующий запчасти.

private void PartsSort()

{

// Очистка и перезагрузка элементов в контексте данных

auto\_repair\_shopEntities.GetContext().ChangeTracker.Entries().ToList().ForEach(p => p.Reload());

// Заполнение списка LViewPart всеми запчастями в базе данных

LViewPart.ItemsSource = auto\_repair\_shopEntities.GetContext().Spare\_Parts.ToList();

}

// Метод для поиска запчасти по имени.

private void Search()

{

// Получение всех запчастей из базы данных

var currentParts = auto\_repair\_shopEntities.GetContext().Spare\_Parts.ToList();

// Отбор запчастей, имена которых содержат введенный текст

currentParts = currentParts.Where(p => p.name\_parts.ToString().ToLower().Contains(name.Text.ToLower())).ToList();

// Заполнение списка LViewPart отобранными запчастями

LViewPart.ItemsSource = currentParts;

}

// Метод инициализации элементов пользовательского интерфейса

private void InitializeUI()

{

// Находит и присваивает элементы StackPanel и TextBlock в XAML к переменным cartPanel и totalPriceLabel.

cartPanel = FindName("CartPanel") as StackPanel;

totalPriceLabel = FindName("TotalPriceLabel") as TextBlock;

}

// Обрабатывает двойной клик по элементу в LViewPart, добавляет выбранную запчасть в корзину и обновляет UI.

private void LViewPart\_MouseDoubleClick(object sender, System.Windows.Input.MouseButtonEventArgs e)

{

// Проверяет выбранный элемент (выделенный элемент в списке), если это запчасть, то добавляет ее в корзину.

if (LViewPart.SelectedItem is Spare\_Parts selectedParts)

{

// Добавляет выбранную запчасть в список cartItems

cartItems.Add(selectedParts);

// Обновляет пользовательский интерфейс (UI), чтобы отразить изменения

UpdateCartUI();

}

}

// Обновляет UI корзины, чтобы показать товары в списке cartItems.

private void UpdateCartUI()

{

// Далее идут операции с UI для отображения элементов списка cartItems в пользовательский интерфейс.

// Сюда попадают операции добавления изображений, прикрепления обработчиков событий к кнопкам и изменение текстовых полей.

cartPanel.Children.Clear();

int? totalCost = 0;

foreach (var item in cartItems)

{

var cartItemUI = new StackPanel();

var itemName = new TextBlock();

itemName.Text = item.name\_parts;

cartItemUI.Children.Add(itemName);

var itemImage = new Image();

itemImage.Source = ConvertByteArrayToImage(item.photo);

itemImage.Width = 100;

itemImage.Height = 100;

cartItemUI.Children.Add(itemImage);

var itemPrice = new TextBlock();

itemPrice.Text = item.price.ToString() + "руб.";

cartItemUI.Children.Add(itemPrice);

var deleteButton = new Button();

deleteButton.Content = "X";

deleteButton.Width = 50;

deleteButton.Tag = item;

deleteButton.Click += DeleteButton\_Click;

cartItemUI.Children.Add(deleteButton);

cartPanel.Children.Add(cartItemUI);

totalCost += item.price;

}

totalPriceLabel.Text = $"В общем: {totalCost:C}";

if (cartItems.Any())

{

var checkoutButton = new Button();

checkoutButton.Content = "Оформить";

checkoutButton.Width = 200;

checkoutButton.Margin = new Thickness(0, 10, 0, 0);

checkoutButton.Click += CheckoutButton\_Click;

cartPanel.Children.Add(checkoutButton);

var clearButton = new Button();

clearButton.Content = "Очистить";

clearButton.Width = 100;

clearButton.Margin = new Thickness(0, 10, 0, 0);

clearButton.Click += ClearCartButton\_Click;

cartPanel.Children.Add(clearButton);

}

}

// Метод для конвертации изображений из формата byte array в формат, который можно использовать в WPF Image control.

private ImageSource ConvertByteArrayToImage(byte[] byteArray)

{

// Здесь код для преобразования byteArray в объект BitmapImage и его возврат.

if (byteArray == null || byteArray.Length == 0)

return null;

using (MemoryStream memoryStream = new MemoryStream(byteArray))

{

BitmapImage bitmapImage = new BitmapImage();

bitmapImage.BeginInit();

bitmapImage.CacheOption = BitmapCacheOption.OnLoad;

bitmapImage.StreamSource = memoryStream;

bitmapImage.EndInit();

return bitmapImage;

}

}

// Обработчик нажатия на кнопку удаления в корзине. Удаляет соответствующий элемент из cartItems и обновляет UI.

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

{

// Код для удаления выбранной запчасти из корзины и обновления пользовательского интерфейса.

if (sender is Button deleteButton && deleteButton.Tag is Spare\_Parts itemToDelete)

{

cartItems.Remove(itemToDelete);

UpdateCartUI();

}

}

// Обработчик кнопки Оформить. Это финализирует заказ и сохраняет его в базе данных.

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

{

// Создание записи в таблице History\_Order. Здесь идет процесс сохранения заказа в базе данных и обновления UI.

using (var context = new auto\_repair\_shopEntities())

{

foreach (var item in cartItems)

{

var order = new History\_Orders

{

id\_client = \_client.id\_client,

id\_car = \_car.id\_car,

id\_part = item.id\_part,

};

context.History\_Orders.Add(order);

}

context.SaveChanges();

MessageBox.Show("Покупка совершена");

}

cartItems.Clear();

cartPanel.Children.Clear();

UpdateCartUI();

}

// Обработчик кнопки Очистить, который очищает корзину.

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

{

// Код для очистки корзины и обновления UI.

cartItems.Clear();

cartPanel.Children.Clear();

UpdateCartUI();

}

// Обработчик кнопки Закрыть, который возвращает пользователя на главное окно.

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

{

// Код для закрытия текущего окна и открытия главного окна.

MainWindow wind = new MainWindow(\_client);

wind.Show();

Close();

}

// Обработчик события изменения текста в поле поиска. Осуществляет прямой поиск запчастей по мере ввода.

private void name\_TextChanged(object sender, TextChangedEventArgs e)

{

// Здесь вызывается метод поиска каждый раз, когда пользователь меняет текст в поле поиска.

Search();

}

// Обработчик для проверки ввода, ограничивающий ввод только русскими буквами.

private void name\_PreviewTextInput(object sender, System.Windows.Input.TextCompositionEventArgs e)

{

// Регулярное выражение, которое отсеивает все символы кроме русских букв, и применяется каждый раз когда пользователь вводит текст.

Regex rgx = new Regex("[^А-Яа-я]");

e.Handled = rgx.IsMatch(e.Text);

}

}

}

public partial class Profile : System.Windows.Window

{

Clients \_client;

// Конструктор класса Profile. Принимает объект класса Clients в качестве параметра - информацию о клиенте

public Profile(Clients client)

{

InitializeComponent();

// Инициализирует поля данных о текущем клиенте

\_client = client;

// Вызов функции обновления инфороации о заказах

HistoryOrder();

}

// Функция обновления данных о истории заказов

private void HistoryOrder()

{

// Обновление данных из базы данных

auto\_repair\_shopEntities.GetContext().ChangeTracker.Entries().ToList().ForEach(p => p.Reload());

// Заполнение источников данных для разделов истории заказов и новых заказов

viewOrder.ItemsSource = auto\_repair\_shopEntities.GetContext().History\_Orders.Where(p => p.id\_client == \_client.id\_client).ToList();

viewNewOrder.ItemsSource = auto\_repair\_shopEntities.GetContext().Orders.Where(p => p.id\_client == \_client.id\_client).ToList();

// Обновление информации о клиенте

DataClient();

}

// Функция обновления данных о клиенте на интерфейсе

private void DataClient()

{

surname.Content = \_client.surname;

name.Content = \_client.name;

patronymic.Content = \_client.patronymic;

email.Content = \_client.email;

}

// Функция, которая срабатывает при двойном клике на новый заказ

private void viewNewOrder\_MouseDoubleClick(object sender, System.Windows.Input.MouseButtonEventArgs e)

{

// Если пользователь подтвердит действие печати

if (MessageBox.Show("Распечатать чек?", "Уведомление", MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)

{

if (viewNewOrder.SelectedItem is Orders selectedOrder)

{

Microsoft.Office.Interop.Word.Application wordApp = new Microsoft.Office.Interop.Word.Application();

Document doc = wordApp.Documents.Add();

// Создание содержимого для печати

string content = $"Заказ №{selectedOrder.id\_order}\n\n" +

$"Дата заказа: {selectedOrder.date\_order}\n" +

$"Статус выполнения: {selectedOrder.execution\_status}\n\n" +

"Детали заказа:\n";

// Добавление таблицы с данными заказа

Table table = doc.Tables.Add(doc.Range(), 1, 5); // 1 строка, 5 столбцов

table.Borders.Enable = 1; // Включение границ таблицы

// Заполнение заголовков столбцов

table.Cell(1, 1).Range.Text = "Автомобиль";

table.Cell(1, 2).Range.Text = "Услуга";

table.Cell(1, 3).Range.Text = "Дата заказа";

table.Cell(1, 4).Range.Text = "Статус выполнения";

table.Cell(1, 5).Range.Text = "Цена";

var car = auto\_repair\_shopEntities.GetContext().Cars.FirstOrDefault(p => p.id\_car == selectedOrder.id\_car);

var service = auto\_repair\_shopEntities.GetContext().Services.FirstOrDefault(p => p.id\_service == selectedOrder.id\_service);

// Заполнение данными

table.Rows.Add(); // Добавление новой строки

table.Cell(2, 1).Range.Text = car.car\_brand;

table.Cell(2, 2).Range.Text = service.name\_service;

table.Cell(2, 3).Range.Text = selectedOrder.date\_order.ToString();

table.Cell(2, 4).Range.Text = selectedOrder.execution\_status.ToString() + "%";

table.Cell(2, 5).Range.Text = selectedOrder.price.ToString() + "р.";

// Сохранение документа

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.Filter = "Документ Word (\*.docx)|\*.docx";

if (saveFileDialog.ShowDialog() == true)

{

string filePath = saveFileDialog.FileName;

doc.SaveAs2(filePath);

doc.Close();

wordApp.Quit();

MessageBox.Show("Документ успешно сохранен.", "Уведомление", MessageBoxButton.OK, MessageBoxImage.Information);

}

else

{

doc.Close();

wordApp.Quit();

}

}

}

}

// Обработчик событий нажатия кнопки редактирования. Создает новое окно регистрации с переданным клиентом.

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

{

RegWind wind = new RegWind(\_client, 1);

wind.Show();

Close();

}

// Обработчик событий нажатия кнопки удаления. Удаляет клиента из базы данных и переходит на страницу входа.

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

{

try

{

auto\_repair\_shopEntities.GetContext().Clients.Remove(\_client);

auto\_repair\_shopEntities.GetContext().SaveChanges();

LogWind wind = new LogWind();

wind.Show();

Close();

}

catch (Exception ex)

{

// В случае ошибки выводит сообщение

MessageBox.Show(ex.Message);

}

}

// Обработчик событий нажатия кнопки закрытия. Переходит на главное окно и передает информацию о текущем клиенте.

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

{

MainWindow wind = new MainWindow(\_client);

wind.Show();

Close();

}

}

}

// Определение класса RegWind (окно регистрации), наследуюющего от класса Window

public partial class RegWind : Window

{

//Приватные свойства класса.Объект клиента

private Clients \_client = new Clients();

// Счетчик

private int \_count;

// Конструктор принимает клиента и количество в качестве параметров

public RegWind(Clients client, int count)

{

InitializeComponent();

\_count = count;

// Если клиент не null, то \_client присваивает клиента

if (client != null)

\_client = client;

// Ограничение максимальной длины для полей ввода

name.MaxLength = 20;

surname.MaxLength = 20;

patronymic.MaxLength = 40;

email.MaxLength = 50;

log.MaxLength = 20;

pswd.MaxLength = 20;

// Установка DataContext в \_client

DataContext = \_client;

}

// Определение метода регистрации (обработчик события нажатия на кнопку регистрации)

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

{

// Создание экземпляра StringBuilder для накопления ошибок

StringBuilder errors = new StringBuilder();

// Ищет в базе данных клиента с таким же логином

var busyLog = auto\_repair\_shopEntities.GetContext().Clients.FirstOrDefault(p => p.login == log.Text);

// Паттерн для проверки корректности электронной почты

string pattern = @"^([\w\.\-]+)@([\w\-]+)((\.(\w){2,3})+)$";

Regex regex = new Regex(pattern);

// Проверка ввода каждого из полей и добавление в errors,

// если поле пустое или не соответствует требуемому формату

if (string.IsNullOrWhiteSpace(\_client.name))

errors.AppendLine("Введите имя");

if (string.IsNullOrWhiteSpace(\_client.surname))

errors.AppendLine("Введите фамилию");

if (string.IsNullOrWhiteSpace(\_client.email))

errors.AppendLine("Введите почту");

else

{

if (!regex.IsMatch(\_client.email))

errors.AppendLine("Почта введена неккоректно");

}

if (string.IsNullOrWhiteSpace(\_client.login))

errors.AppendLine("Введите логин");

if (string.IsNullOrWhiteSpace(\_client.password))

errors.AppendLine("Введите пароль");

// Если имеются ошибки,

// вывод сообщения об ошибке и прекращение выполнения функции

if (errors.Length > 0)

{

MessageBox.Show(errors.ToString());

return;

}

// Если идентификатор клиента равен 0,

// добавляем нового клиента в базу данных

if (\_client.id\_client == 0)

auto\_repair\_shopEntities.GetContext().Clients.Add(\_client);

try

{

// Сохраняет изменения в базе данных

auto\_repair\_shopEntities.GetContext().SaveChanges();

// Если \_count равен 0, то открывает окно входа

// Если нет, то открывает профиль с текущим клиентом

if (\_count == 0)

{

LogWind wind = new LogWind();

wind.Show();

Close();

}

else

{

Profile wind = new Profile(\_client);

wind.Show();

Close();

}

}

// Обработка исключений при попытке сохранения изменений в базе данных

catch (Exception ex)

{

// В случае ошибки показывает сообщение об ошибке

MessageBox.Show(ex.Message);

}

}

// Обработчик события нажатия на кнопку "Закрыть"

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

{

// Определение следующего окна,

// в зависимости от значения \_count

// 0 - окно входа, иначе - окно профиля

if (\_count == 0)

{

LogWind wind = new LogWind();

wind.Show();

Close();

}

else

{

Profile wind = new Profile(\_client);

wind.Show();

Close();

}

}

// Определение методов, ограничивающих ввод недопустимых символов в поля формы

private void name\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

Regex rgx = new Regex("[^А-Яа-я]+");

e.Handled = rgx.IsMatch(e.Text);

}

private void email\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

Regex rgx = new Regex("[^a-zA-Z0-9@.-]+");

e.Handled = rgx.IsMatch(e.Text);

}

private void log\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

Regex rgx = new Regex("[^a-zA-Z0-9]+");

e.Handled = rgx.IsMatch(e.Text);

}

}

}

// Определение класса SelectWind для окна выбора, наследует от класса Window

public partial class SelectWind : Window

{

// Приватные свойства класса

// \_client - объект клиента, что будет использоваться в окне

private Clients \_client;

// \_car - объект машины, что будет использоваться в окне

private Cars \_car;

// Конструктор класса, принимает клиента и машину

public SelectWind(Clients client, Cars car)

{

InitializeComponent();

// Присвоение входящих значений свойствам

\_client = client;

\_car = car;

}

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

{

SparePartsWind wind = new SparePartsWind(\_client, \_car);

wind.Show();

Close();

}

// Метод, выполняющийся при нажатии на кнопку "service\_Click" (для услуг)

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

{

// Создание окна ServicesWind для отображения услуг

ServicesWind wind = new ServicesWind(\_client, \_car);

// Отображение созданного окна

wind.Show();

Close();

}

// Метод, выполняющийся при нажатии на кнопку "close\_Click" (закрыть)

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

{

// Создание нового главного окна MainWindow

MainWindow wind = new MainWindow(\_client);

// Закрытие текущего окна

wind.Show();

Close();

}

}

}

// Определение класса ServicesAdd для страницы добавления услуги, это наследуемый класс от класса Page

public partial class ServicesAdd : Page

{

// Создаем объект \_services типа Services

private Services \_services = new Services();

// Конструктор класса, он принимает услугу в качестве входного параметра

public ServicesAdd(Services services)

{

InitializeComponent();

// Если предоставлены услуги, обновляем \_services данными об услугах

if (services != null)

\_services = services;

// Устанавливаем максимальную длину для ввода в полях имя, описание и цена

name.MaxLength = 60;

desc.MaxLength = 60;

price.MaxLength = 3;

// Устанавливаем DataContext для \_services для привязки данных

DataContext = \_services;

}

// Обработчик события клика для кнопки save\_Click. Отвечает за сохранение новой или измененной услуги

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

{

// StringBuilder для сбора возможных ошибок

StringBuilder errors = new StringBuilder();

// Валидация полей формы

if (string.IsNullOrWhiteSpace(\_services.name\_service))

errors.AppendLine("Введите название");

if (string.IsNullOrWhiteSpace(\_services.description))

errors.AppendLine("Введите примечание");

if (string.IsNullOrWhiteSpace(\_services.price.ToString()))

errors.AppendLine("Введите цену");

// Если есть ошибки, выводим их в окне сообщений и затем возвращаемся из метода

if (errors.Length > 0)

{

MessageBox.Show(errors.ToString());

return;

}

// Если услуга новая (\_services.id\_service == 0), добавляем ее в контекст

if (\_services.id\_service == 0)

auto\_repair\_shopEntities.GetContext().Services.Add(\_services);

// Сохранения изменения, перехватываем исключения при возникновении ошибок

try

{

auto\_repair\_shopEntities.GetContext().SaveChanges();

// Возвращаемся на предыдущую страницу

NavigationService.GoBack();

}

catch (Exception ex)

{

// Выводим сообщение об ошибке

MessageBox.Show(ex.Message);

}

}

// Обработчик события ввода текста для поля name. Позволяет ввод только русских символов

private void name\_PreviewTextInput(object sender, System.Windows.Input.TextCompositionEventArgs e)

{

Regex rgx = new Regex("[^А-Яа-я]+");

e.Handled = rgx.IsMatch(e.Text);

}

// Обработчик события ввода текста для поля price. Позволяет ввод только цифр

private void price\_PreviewTextInput(object sender, System.Windows.Input.TextCompositionEventArgs e)

{

Regex rgx = new Regex("[^0-9]+");

e.Handled = rgx.IsMatch(e.Text);

}

// Обработчик события клика по кнопке close\_Click. Возвращает пользователя на предыдущую страницу

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

{

NavigationService.GoBack();

}

}

}

public partial class ServicesAdmin : Page

{

// Определение класса ServicesAdmin для страницы администрирования услуг, это наследуемый класс от класса Page

public ServicesAdmin()

{

InitializeComponent();

}

// Метод UpdateServices производит обновление данных в таблице услуг

private void UpdateServices()

{

// Операции смены трекера используются для перезагрузки данных из базы данных

auto\_repair\_shopEntities.GetContext().ChangeTracker.Entries().ToList().ForEach(p => p.Reload());

// Загружаем данные об услугах в gridService

gridService.ItemsSource = auto\_repair\_shopEntities.GetContext().Services.ToList();

}

// Обработчик события для кнопки delete\_Click. Удаляет выбранную услугу

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

{

// Проверяем, что пользователь выбрал услугу из таблицы

if (gridService.SelectedItem != null)

{

// Запрашиваем подтверждение у пользователя перед удалением услуги

if (MessageBox.Show("Удалить сервис?", "Уведомление", MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)

{

// Если выбранная услуга действительно является экземпляром Services

if (gridService.SelectedItem is Services selectedServices)

{

// Удаляем выбранную услугу из контекста и сохраняем изменения

auto\_repair\_shopEntities.GetContext().Services.Remove(selectedServices);

auto\_repair\_shopEntities.GetContext().SaveChanges();

// Обновляем список услуг в таблице

UpdateServices();

}

}

}

else

// Если пользователь не выбрал сервис для удаления, выводим сообщение

MessageBox.Show("Выберите запись для удаления");

}

// Обработчик события для кнопки add\_Click. Перенаправляет пользователя на страницу добавления новой услуги

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

{

NavigationService.Navigate(new ServicesAdd(null));

}

// Обработчик события для кнопки edit\_Click. Позволяет редактировать выбранную услугу

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

{

// Проверяем, что пользователь выбрал услугу из таблицы

if (gridService.SelectedItem != null)

{

// Если выбранная услуга действительно является экземпляром Services

if (gridService.SelectedItem is Services selectedService)

// Перенаправляем пользователя на страницу редактирования, передавая туда выбранную услугу

NavigationService.Navigate(new ServicesAdd(selectedService));

}

else

// Если пользователь не выбрал сервис для редактирования, выводим сообщение

MessageBox.Show("Выберите запись для изменения");

}

// Обработчик события для события IsVisibleChanged страницы. Обновляет данные о сервисах при смене видимости страницы

private void Page\_IsVisibleChanged(object sender, DependencyPropertyChangedEventArgs e)

{

UpdateServices();

}

}

}

// Класс ServicesWind, связанный с окном сервисной услуги

public partial class ServicesWind : Window

{

// приватные поля для объекта Clients и Cars

private Clients \_client;

private Cars \_car;

// Конструктор класса ServicesWind принимает в качестве параметров объекты Clients и Cars

public ServicesWind(Clients client, Cars car)

{

InitializeComponent();

// Приватному полю \_client присваивается пришедший объект client

\_client = client;

// Приватному полю \_car присваивается пришедший объект car

\_car = car;

// Вызывается метод для загрузки данных о сервисах из базы данных

DataService();

}

// Метод DataService загружает данные сервисов из базы данных

private void DataService()

{

// Обновление записей в контексте данных

auto\_repair\_shopEntities.GetContext().ChangeTracker.Entries().ToList().ForEach(p => p.Reload());

// Присваивание коллекции сервисов источнику данных LViewService

LViewService.ItemsSource = auto\_repair\_shopEntities.GetContext().Services.ToList();

}

// Обработчик двойного клика мышью по элементу списка сервисов

private void LViewService\_MouseDoubleClick(object sender, System.Windows.Input.MouseButtonEventArgs e)

{

// Получение контекста данных

var context = auto\_repair\_shopEntities.GetContext();

// Если выбранный элемент является объектом Services

if (LViewService.SelectedItem is Services selectedService)

{

// Создание нового объекта заказа

var order = new Orders

{

//Установка связи с клиентом

id\_client = \_client.id\_client,

//Установка связи с сервисом

id\_service = selectedService.id\_service,

//Установка связи с автомобилем

id\_car = \_car.id\_car,

//Установка даты заказа

date\_order = DateTime.Now,

//Установка статуса выполнения заказа

execution\_status = 0,

//Установка цены заказа

price = selectedService.price,

};

// Добавление нового заказа в контекст данных

context.Orders.Add(order);

// Сохранение изменений в базу данных

context.SaveChanges();

// Вывод сообщения о том, что заказ успешно создан

MessageBox.Show("Заказ успешно создан");

}

}

// Обработчик клика по кнопке закрытия окна

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

{

// Создание нового экземпляра главного окна с текущим клиентом

MainWindow wind = new MainWindow(\_client);

// Отображение главного окна

wind.Show();

Close();

}

}

}

// Класс страницы добавления новых специальностей.

public partial class SpecialitiesAdd : Page

{

// Приватное поле для объекта Specialities.

private Specialities \_specialities = new Specialities();

// Конструктор страницы SpecialitiesAdd.

public SpecialitiesAdd(Specialities selectedSpec)

{

InitializeComponent();

// Если передана конкретная специальность, объек будет инициализирован этой специальностью

if (selectedSpec != null)

\_specialities = selectedSpec;

// Ограничение максимальной длины названия специальности до 20 символов.

name.MaxLength = 20;

// Установка контекста данных для страницы.

DataContext = \_specialities;

}

// Обработчик события нажатия кнопки save.

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

{

// Использование StringBuilder для постройки сообщения об ошибках.

StringBuilder errors = new StringBuilder();

// Проверка заполнения поля названия.

if (string.IsNullOrWhiteSpace(\_specialities.name\_specialties))

errors.AppendLine("Введите название");

if (errors.Length > 0)

{

MessageBox.Show(errors.ToString());

return;

}

// Если специальность еще не добавлена в контекст данных, то добавление ее в контекст.

if (\_specialities.id\_specialties == 0)

auto\_repair\_shopEntities.GetContext().Specialities.Add(\_specialities);

// Выполнение сохранения изменений в контексте данных.

try

{

auto\_repair\_shopEntities.GetContext().SaveChanges();

// Если сохранение прошло успешно, переход на предыдущую страницу.

NavigationService.GoBack();

}

// Если сохранение вызвало исключение вывод сообщения об исключении.

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

// Обработчик события нажатия клавиш ввода данных в текстовое поле name.

private void name\_PreviewTextInput(object sender, TextCompositionEventArgs e)

{

// Проверка вводимых символов через регулярное выражение - разрешен ввод только кириллицы. Регулярное выражение для проверки кириллицы.

Regex rgx = new Regex("[^А-Яа-я]");

// Если вводимые символы не подходят под условия регулярного выражения, то они игнорируются.

e.Handled = rgx.IsMatch(e.Text);

}

// Обработчик события нажатия кнопки возврата ("отмены").

private void close\_Click(object sender, System.Windows.RoutedEventArgs e)

{

// При нажатии, возвращается на предыдущую страницу.

NavigationService.GoBack();

}

}

}

// Объявление класса SpecialitiesAdmin, который наследуюет класс Page.

public partial class SpecialitiesAdmin : Page

{

// Конструктор для класса SpecialitiesAdmin

public SpecialitiesAdmin()

{

InitializeComponent();

}

// Метод для обновления специальностей на странице.

private void UpdateSpecial()

{

// Очистка и перезагрузка элементов в контексте данных

auto\_repair\_shopEntities.GetContext().ChangeTracker.Entries().ToList().ForEach(p => p.Reload());

// Заполнение таблицы gridSpecial всеми специальностями из базы данных

gridSpecial.ItemsSource = auto\_repair\_shopEntities.GetContext().Specialities.ToList();

}

// Обработчик события нажатия кнопки удаления

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

{

// Проверяет, выбрана ли специальность для удаления

if (gridSpecial.SelectedItem != null)

{

// Запрашивает подтверждение на удаление специальности

if (MessageBox.Show("Удалить специальность?", "Уведомление", MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)

{

if (gridSpecial.SelectedItem is Specialities selectedSpecial)

{

// Удаляет выбранную специальность из базы данных и сохраняет изменения

auto\_repair\_shopEntities.GetContext().Specialities.Remove(selectedSpecial);

auto\_repair\_shopEntities.GetContext().SaveChanges();

// Обновляет список специальностей

UpdateSpecial();

}

}

}

else

// Если специальность не выбрана, показывает сообщение

MessageBox.Show("Выберите запись для удаления");

}

// Обработчик события нажатия кнопки добавления

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

{

// Навигирует на страницу добавления специальности

NavigationService.Navigate(new SpecialitiesAdd(null));

}

// Обработчик события нажатия кнопки редактирования

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

{

// Проверяет, выбрана ли специальность для редактирования

if (gridSpecial.SelectedItem != null)

{

if (gridSpecial.SelectedItem is Specialities selectedSpec)

// Навигирует на страницу редактирования специальности, передавая выбранную специальность

NavigationService.Navigate(new SpecialitiesAdd(selectedSpec));

}

}

// Обработчик события изменения видимости страницы

private void Page\_IsVisibleChanged(object sender, DependencyPropertyChangedEventArgs e)

{

// При каждом изменении видимости страницы обновляет список специальностей

UpdateSpecial();

}

}

}