МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Направление специальности 1-40 01 01 10 Программное обеспечение информационных технологий (программирование интернет-изданий)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

по дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования»

Тема «Магазин автозапчастей»

Исполнитель

студент 2 курса группы 5 Шестопалов Денис Александрович

Руководитель работы ассистент Демещик А.С.

Курсовой проект защищён с оценкой

Председатель Пацей Н.В.

Минск 2020

**Содержание**

[ВВЕДЕНИЕ 3](#_Toc41633565)

[1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству 4](#_Toc41633566)

[2. Анализ требований к программному средству и разработка функциональных требований 6](#_Toc41633567)

[2.1. Описание инструментов для разработки программного средства 6](#_Toc41633568)

[2.2. Описание функций, реализуемых программой 7](#_Toc41633569)

[3. Проектирование программного средства 8](#_Toc41633570)

[3.1. Модель базы данных 8](#_Toc41633571)

[3.2. Архитектура системы 11](#_Toc41633572)

[3.3. Авторизация и регистрация 11](#_Toc41633573)

[3.4. Окна 12](#_Toc41633574)

[4. Создание программного средства 13](#_Toc41633575)

[4.1. Структура проекта 13](#_Toc41633576)

[4.2. Уровень бизнес-логики 13](#_Toc41633577)

[4.3. Уровень представлений 14](#_Toc41633578)

[4.4. Уровень модели представлений 15](#_Toc41633579)

[4.5. Dependency injection 15](#_Toc41633580)

[5. Тестирование, проверка работоспособности и анализ полученных результатов 16](#_Toc41633581)

[6. Руководство по установке и использованию 18](#_Toc41633582)

[ЗАКЛЮЧЕНИЕ 21](#_Toc41633583)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 22](#_Toc41633584)

[Приложение А 23](#_Toc41633585)

[Приложение Б 24](#_Toc41633586)

[Приложение В 25](#_Toc41633587)

[Приложение Г 26](#_Toc41633588)

[Приложение Д 28](#_Toc41633589)

[Приложение Е 29](#_Toc41633590)

[Приложение Ж 30](#_Toc41633591)

[Приложение З 32](#_Toc41633592)

# ВВЕДЕНИЕ

В современном мире автомобиль – незаменимая вещь для большинства населения. Однако любая техника имеет свойство ломаться. Чтобы отремонтировать автомобиль, необходимо преобретать запчасти. Данной целью занимаются магазины по продаже автозапчастей.

На сегодняшний день существует много магазинов по продаже автомобильных запчастей и офлайн- и онлайн-мире. Однако почти все магазины имеют свои онлайн-площадки.

Они отличаются дизайном, количеством поставщиков, наличием автомобильных запчастей, ценой товара и так далее.

Целью данного курсового проекта является разработка программного средства для продажи автозапчастей.

Данное программное средство позволяет пользователям найти интересующие запчасти для автомобиля, а владельцам – осуществлять продажу автозапчастей, находящихся в наличии. Также программное средство должно представлять пользователю информацию о заказе и уведомлять людей, зарегистрированных в данном программном средстве о поступлении на склад интересующихи хи запчастей.

Задачами курсового проекта являются:

* Разработка графических интерфейсов для взаимодействия с пользователем
* Разработка базы данных для хранения необходимой информации
* Разработка программного средства для продажи автозапчастей

Приложение имеет 2 типа пользователей: покупатель и администратор. Каждая роль имеет свой функционал. Основой задачей покупателя является заказ товара, а у администратора – пополнение ассортимента и управление заказами.

Далее в частях будут описаны процессы создания программного средства от обзора и анализа прототипов до создания готового продукта.

# Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству

В процессе выполнения курсовой работы были использованы дополнительные источники. На сайте https://metanit.com[1] имеется много необходимой информации для решения проблем, возникших в процессе программирования программного средства. Также данный ресурс имеет огромное количество статей по языку программирования C#. На сайте http://professorweb.ru[2] была взята информация по концепциям библиотеки Windows Presentation Foundation. Большое количество информации было взято с оффициального сайта Microsoft https://docs.microsoft.com[3] по архитектурному паттерну MVVM(Model-View-ViewModel). Ресурс https://refactoring.guru/ru/design-patterns[4] хорошо описывает назначение и реализацию шаблонов проектирования.

В качестве аналога был выбран сайт http://amr.by/. Для посетителей на главной странице представлен поиск по интересующему автомобилю (рис. 1.1.)

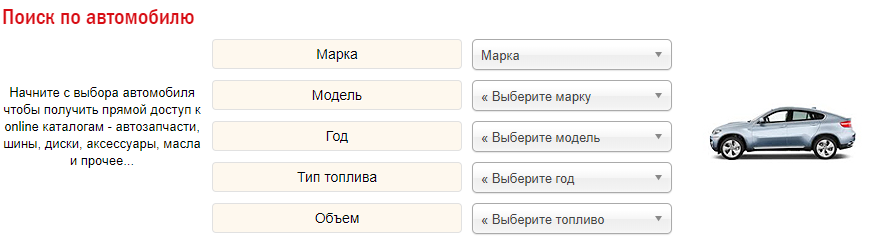


Рисунок 1.1 – меню выбора автомобиля

После ввода данных появляется окно с выбором категории запчастей (рис.1.2.)

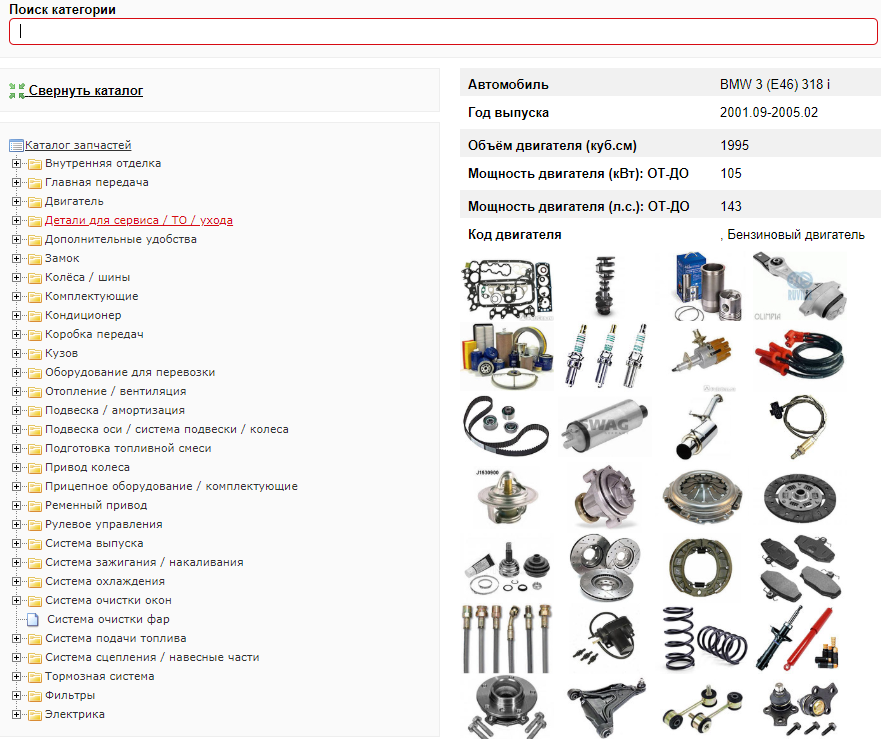


Рисунок 1.2 – выбор категории запчасти

Далее выбирается необходимая запчасть и выбирается по производителю(рис 1.3.)

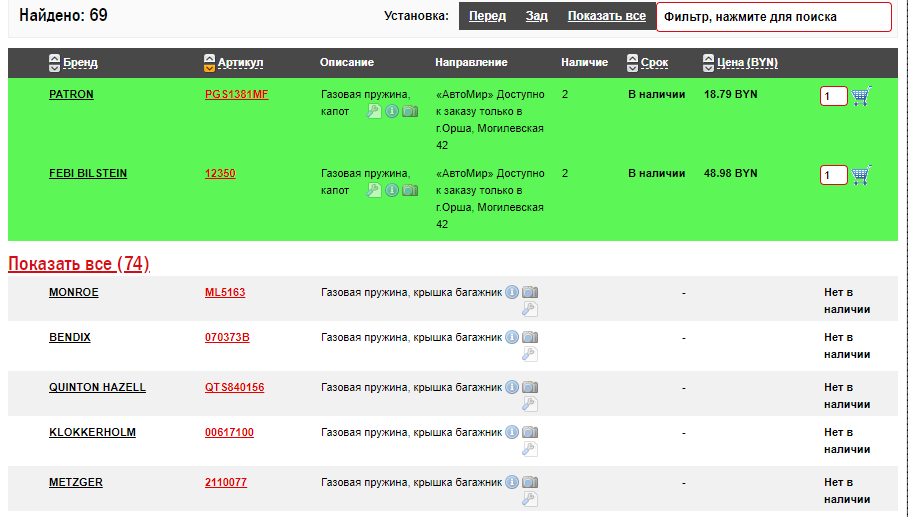


Рисунок 1.3 – список деталей, выбранных по категории

После добавления в корзину пользователь может заказать товар, указав свои контактные данные (рис. 1.4.)

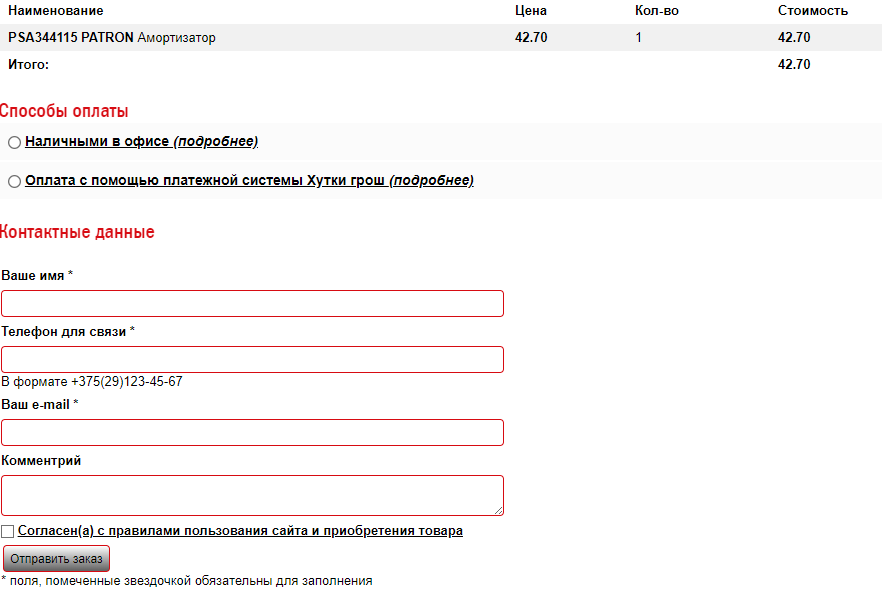


Рисунок 1.4 – окно заказа товара

Я решил сделать программное средство на данную тему, потому что запчасти для автомобилей имеют спрос среди автомобилистов и автосервисов. Ещё одной причиной является фактическое отсутствие приложений для продажи автомобильных запчастей на персональных компьютерах.

# Анализ требований к программному средству и разработка функциональных требований

# 2.1. Описание инструментов для разработки программного средства

При разработке приложения используются:

* Платформа .NET Core;
* Объектно-ориентированный язык программирования C#;
* Система для построения клиентских приложений WPF;
* Расширяемый язык разметки XAML;
* Шаблон проектирования MVVM;
* Система управления реляционными базами данных Microsoft SQL Server;
* Фреймворк Entity Framework Core;
* Среда разработки Visual Studio 2019;
* Библиотека Material Design;
* Протокол SMTP для передачи e-mail сообщений;
* Библиотека Syncfusion Essential PDF для генерации pdf-документа;

.NET Core[1] — это модульная платформа для разработки программного обеспечения с открытым исходным кодом. .NET Core основана на .NET Framework. Платформа .NET Core отличается от неё модульностью, кроссплатформенностью, возможностью применения облачных технологий, и тем, что в ней произошло разделение между библиотекой CoreFX и средой выполнения CoreCLR.

.NET Core — модульная платформа. Каждый её компонент обновляется через менеджер пакетов NuGet, а значит можно обновлять её модули по отдельности, в то время как .NET Framework обновляется целиком. Каждое приложение может работать с разными модулями и не зависит от единого обновления платформы.

C#[1] — объектно-ориентированный язык программирования. Разработан в 1998 — 2001 под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework.

WPF (Windows Presentation Foundation) [2] — система для построения клиент-ских приложений Windows с визуально привлекательными возможностями взаи-модействия с пользователем, графическая (презентационная) подсистема в со-ставе .NET Framework (начиная с версии 3.0), использующая язык XAML.

XAML[2] — это декларативный язык разметки. С точки зрения модели про-граммирования .NET Framework язык XAML упрощает создание пользователь-ского интерфейса для приложения .NET Framework.

Model-View-ViewModel (MVVM)[3] — шаблон проектирования архитектуры приложения. Представлен в 2005 году Джоном Госсманом (John Gossman) как модификация шаблона Presentation Model. Используется для разделения модели и её представления, что необходимо для их изменения отдельно друг от друга. Например, разработчик задаёт логику работы с данными, а дизайнер работает с пользовательским интерфейсом.

Microsoft SQL Server [1]— система управления реляционными базами данных, разработанная корпорацией Microsoft. Основной используемый язык запросов — Transact-SQL. Он является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия.

Entity Framework Core [3] представляет собой объектно-ориентированную, легковесную и расширяемую технологию от компании Microsoft для доступа к данным. EF Core является ORM-инструментом. То есть EF Core позволяет работать базами данных, но представляет собой более высокий уровень абстракции: EF Core позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными независимо от типа хранилища. Entity Framework Core поддерживает множество различных систем баз данных. Таким образом, мы можем через EF Core работать с любой СУБД, если для нее имеется нужный провайдер.

Material Design Library[4] — это исчерпывающая кросс-платформенная кон-цепция визуального оформления приложений, затрагивающая правила отображения анимации, общий стиль приложения и правила взаимодействия с управляю-щими элементами.

SMTP (Simple Mail Transfer Protocol [1]) — это широко используемый сетевой протокол, предназначенный для передачи электронной почты в сетях TCP/IP.

Syncfusion Essential PDF[1] – библиотека, позволяющая генерировать документы в pdf-формате. Имеет удобный функционал и возможность гибкой настройки генерации pdf-файла.

# Описание функций, реализуемых программой

При разработке курсового проекта были определены требования к программному средству. Оно должно иметь:

* Разделение посетителей на пользователей и администраторов;
* Хеширование паролей в формате SHA-256;
* Возможность авторизации и регистрации;
* Просмотр ассортимента;
* Уведомление пользователей о поступлении интересующего товара и смены статуса заказа по e-mail;
* Добавление и удаление товара из корзины;
* Система оплаты заказоа;
* Изменение статуса заказа;
* Уведомление пользователей о поступлении интересующего товара и смены статуса заказа по e-mail;
* Возможность просмотра подробностей заказа через чек в pdf-формате;

Диаграмма использования показана в Приложении А.

# Проектирование программного средства

# Модель базы данных

Для реализации задачи была создана локальная база данных CarPartShopDB. Она состоит из 11 таблиц. Их структура была прописана в исходном коде программы по принципу Model First. Связи между таблицами были созданы с помощью Fluent API. На поля были наложены ограничения целостности при помощи аннотаций.

Описание таблиц базы данных представлено в таблицах 3.1-3.11.

Таблица 3.1 – описание таблицы Accounts

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| id | int | Хранит номер пользователя |
| username | nvarchar(20) | Хранит имя пользователя |
| pass | nvarchar(max) | Хранит захешированный пароль |
| email | nvarchar(max) | Хранит электронную почту |
| creation\_date | datetime2(7) | Хранит дату создания аккаунта |
| balance | float | Хранит баланс аккаунта |
| role | int | Хранит роль пользователя |

Таблица 3.2 – описание таблицы Address

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| id | int | Хранит номер адреса |
| city | nvarchar(max) | Хранит город |
| street | nvarchar(max) | Хранит улицу |
| house\_number | int | Хранит номер дома |
| apartament\_number | int | Хранит квартиру (если есть) |

Таблица 3.3 – описание таблицы Cards

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| number | bigint | Хранит номер карты |
| balance | float | Хранит баланс |
| pin | nvarchar(max) | Хранит захешированный пин |
| attempts | int | Хранит число |
| creation\_date | datetime2(7) | Хранит дату создания |
| finish\_date | datetime2(7) | Хранит дату окончания |

Таблица 3.4 – описание таблицы CarParts

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| car\_id | int | Хранит номер машины |
| part\_id | int | Хранит номер запчасти |

Таблица 3.5 – описание таблицы Cars

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| id | int | Хранит номер машины |
| mark | nvarchar(max) | Хранит марку автомобиля |
| model | nvarchar(max) | Хранит модель автомобиля |
| fuel\_type | nvarchar(max) | Хранит тип топлива |
| creation\_year | int | Хранит год создания |

Таблица 3.6 – описание таблицы Cart

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| id | int | Хранит номер корзины |
| account\_id | int | Хранит номер аккаунта |
| part\_id | int | Хранит номер запчасти |
| provider\_id | int | Хранит номер поставщика |
| car\_id | int | Хранит номер машины |
| amount | int | Хранит общее количество запчастей |

Таблица 3.7 – описание таблицы OrderParts

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| order\_id | int | Хранит номер заказа |
| part\_id | int | Хранит номер запчасти |
| amount | int | Хранит число запчасти в заказе |
| price | float | Хранит цену за запчать |
| provider\_id | int | Хранит номер поставщика |

Таблица 3.8 – описание таблицы Orders

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| id | int | Хранит номер заказа |
| creation\_time | datetime2(7) | Хранит дату создания заказа |
| finish\_time | datetime2(7) | Хранит дату окончания заказа |
| status | int | Хранит статус заказа |
| account\_id | int | Хранит номер аккаунта |

Таблица 3.9 – описание таблицы PartProviders

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| part\_id | int | Хранит номер запчасти |
| provider\_id | int | Хранит номер поставщика |
| amount | int | Хранит количество делатей запчасти |
| price | float | Хранит цену запчасти |

Таблица 3.10 – описание таблицы Parts

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| id | int | Хранит номер запчасти |
| name | nvarchar(max) | Хранит наименование запчасти |
| description | nvarchar(max) | Хранит описание |
| color | nvarchar(max) | Хранит цвет |
| article | int | Хранит артикул запчасти |
| category | nvarchar(max) | Хранит категорию |

Таблица 3.11 – описание таблицы Providers

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| id | int | Хранит номер поставщика |
| name | nvarchar(max) | Хранит имя поставщика |

Структура базы данных показана в Приложении Б.

# Архитектура системы

В архитектуру программного средства входит шаблон проектирования MVVM, который отделяет уровень представления от уровня логики.

Элементы привязки значительно упрощают работы с кодом, сокращая его, однако требуют большое количество реализаций различных свойств, классов и интерфейсов.

Для работы с базой данных был выбран фреймворк Entity Framework Core, потому что он позволяет упросить работу, а также защищает от SQL-инъекций. Также был применён шаблон проектирования Репозиторий, который является промежуточным слоем между базой данных и остальной программой.

Смысл приложения заключается в том, что любой пользователь, воспользовавшись данной программой, мог посмотреть ассортимент, который добавляет администратор, и если он хочет что-то купить либо отложить в корзину, то он регистрируется либо авторизуется.

На рисунке 3.1 показана диаграмма последовательности при работе приложения со стороны пользователя.

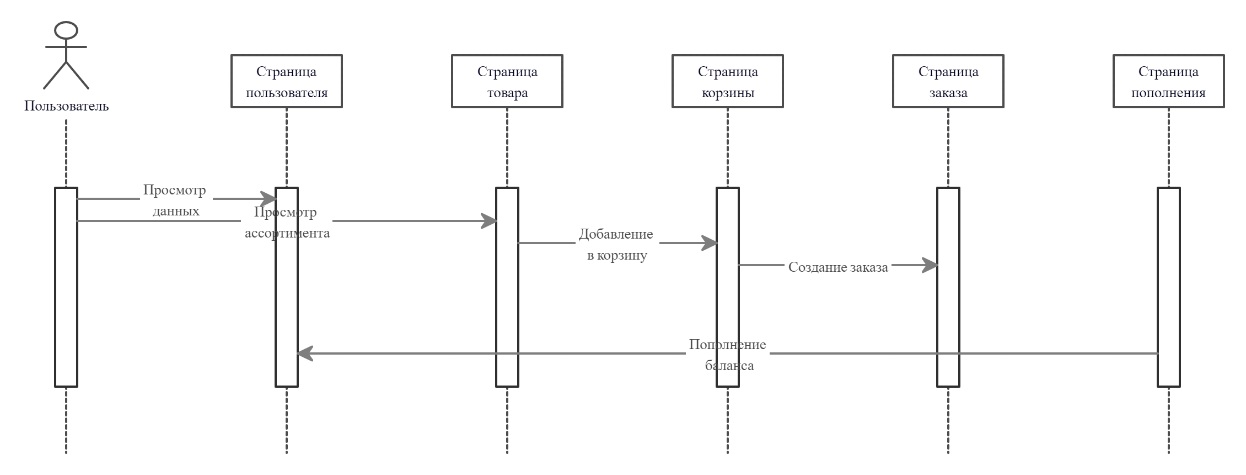


Рисунок 3.1 – диаграмма последовательности для пользователя

Диаграмма отображает возможные последовательные действия пользователя, которые он может совершить в процессе работы с приложением. После процедуры авторизации пользователь попадает на страницу пользователя. Далее он может пополнить баланс, либо перейти на страницу товара. Со страницы товара после добавления в корзину, он может просмотреть содержимое. Со страницы корзины пользователь создаёт заказ.

Диаграмма классов uml приведена в Приложении В.

# Авторизация и регистрация

Чтобы полноценно пользоваться программным средством, необходимо авторизоваться. Если пользователь ещё не пользовался программным средством, то необходимо зарегистрироваться. При регистрации логин и электронная почта должны быть уникальны и соответствовать формату, а пароль – не менее 8 и не более 15 символов. Если хоть одно правило не выполнится, то пользователь будет уведомлён. После нажатия на кпопку регистрации будет произведена запись в таблицу Accounts базы данных. Перед этим пароль будет захеширован. Блок-схемы алгоритов приведены в Приложении Г.

# Окна

Рассмотрим необходимые окна

Так как в приложении есть система авторизации и регистрации, то необходимо запускать приложение имеено с этих окон. После процесса авторизации необходимо в зависимости от роли пользователя открывать окно пользователя или панель адмиинистратора. У пользователя доступна страница заказов, корзина, страница пользователя(личный кабинет). На странице пользователя должна быть панель заказов, окно пополения счёта. Администратор на своей панели должен иметь окна добавления автозапчастей, автомобилей, поставщиков, карт, а также окно заказов. Общая схема приведена на рисунке 3.2.

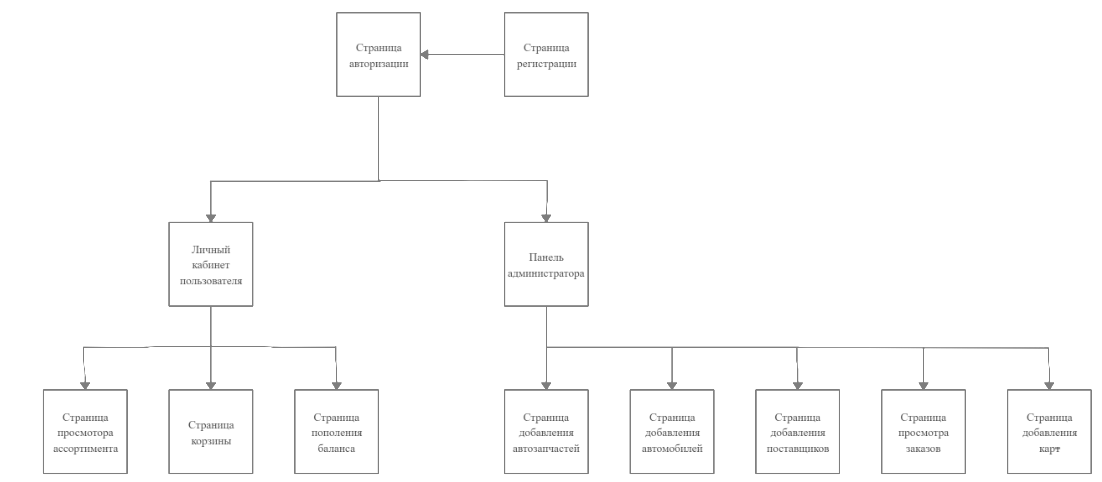


Рисунок 3.2 – общая схема окон

# Создание программного средства

# Структура проекта

Программное решение представляет собой 3 проекта: WPF приложение и 2 библиотеки классов: в первом содержатся сущности и интерфейсы репозиториев, а во втором – миграции и сервисы для работы с базой данных.

Содержимое каждого проекта представлено в Приложении Д.

В таблицах 4.1-4.3 содержится описание каждого пакета.

Таблица 4.1 – структура проекта PartShop.Domain

|  |  |
| --- | --- |
| Имя пакета | Описание |
| Model | Содержатся сущности, которые хранятся в базе данных |
| Services | Содержатся репозитории сервисов для работы с базой данных |

Таблица 4.2 – структура проекта PartShop.EntityFramework

|  |  |
| --- | --- |
| Имя пакета | Описание |
| Migrations | Содержатся миграции |
| Services | Содержатся сервисы для работы с базой данных |

Таблица 4.3 – структура проекта PartShop.WPF

|  |  |
| --- | --- |
| Имя пакета | Описание |
| Commands | Содержит команды для взаимодейстия сервисов и отображением |
| Controls | Содержит необходимые элементы управления |
| Converters | Содержит преобразователи данных для передачи в команды |
| State | Содержит классы для авторизации и навигации |
| ViewModels | Содержит слой взаимодействия моделей и отображений |
| Views | Содержит отображения |

Следует отметить, что в папке ViewModels содержится папка Factories, в которой содержится реализация шаблона проектирования Фабрика для создания ViewModels.

# Уровень бизнес-логики

Перед программированием основного проекта была сгенерирована база данных. Для этого сначала были созданы сущности, которые в будущем будут представлять таблицы в базе данных. Связи были созданы с помощью Fluent API, которые позволяет генерировать их с помощью кода. Валидацию сущностей было решено делать через аннотации. Далее с помощью миграций была создана готовая база данных.

Далее был создан интерфейс репозитория IDataService<T>, где T – сущность, котоая наследуется от класса DomainObject.

При разработке уровня сервисов были созданы классы, которые непосредственно работают с базой данных. Для повышения работоспособности был создан обобщённый класс NonQueryDataService, который отвечал за создание, обновление и удаление данных. Для получения данных необходимо также получать зависимые объекты. Следует отметить, что Entity Framework Core использует технологию linq для работы с данными, что увеличивает читабельность кода. Также в сервисах была осуществлена частичная валидация входных данных. Код интерфейса репозитория и класса NonQueryDataService приведён в Приложении Е.

# Уровень представлений

Уровень представления представлен XAML-документами, находящимися в папке View. Каждый документ описывает отдельную, сменяющуюся часть окна. При создании документов использовался Grid, так как с помощью него удобнее позиционировать элементы. Для ввода информации использовались TextBox и ComboBox. Для вывода информации из базы данных использовался DataGrid. Использовался инструмент привязки для отображения данных из другого уровня. Главным окном является MainWindow, на котором находится панель навигации и окно, которое будет появляться при нажатии на определённые кнопки. Навигационное меню является всплывающим, что является следствием использования анимации и триггеров. Логика навигатора находится в Приложении Ж. В таблице 4.4 содержится описание всех представлений.

Таблица 4.4 – описание отображений

|  |  |  |
| --- | --- | --- |
| Имя представления | Описание | Уровень доступности |
| 1 | 2 | 3 |
| AddBalanceView | Окно для пополнения баланса | Покупатель |
| AddCardView | Окно добавления карты | Администратор |
| AddCarView | Окно добавления автомобиля | Администратор |
| AddPartView | Окно добавления автомобиля | Администратор |
| AddProviderView | Окно добавления поставщика | Администратор |
| AddressView | Окно ввода адреса для заказа | Покупатель |
| AdminView | Пользовательский элемент администрирования | Администратор |
| AuthView | Пользовательский элемент авторизации | Покупатель, администратор |
| CartView | Пользовательский элемент корзины | Покупатель |
| HomeView | Пользовательский элемент пользователя | Покупатель |
| OrderView | Пользовательский элемент заказов | Администратор |
|  |  |  |
| 1 | 2 | 3 |
| PartView | Пользовательский элемент просмотра ассортимента | Покупатель |
| RegisterView | Пользовательский элемент регистрации | Покупатель, администратор |

# Уровень модели представлений

В папке ViewModels располагаются файлы, названия которых аналогичны советующим им XAML-документам в папке Views. В каждом из таких файлов есть общая часть, которое идентична для всех. Все классы были наследованы от класса ViewModelBase, который реализует интерфейс INotifyPropertyChanged, благодаря которому и осуществляется привязка данных из уровня представлений и соотвестствующей модели. Для связи уровня представления и уровня модели представлений в конструкторе соответствующего представления используется свойство DataContext. Каждая модель отображения содержит поле с модификатором private и свойство, которое при получении данных вызывает метод NotifyPropertyChanged, а также ссылки на команды.

# Dependency injection

Для того, чтобы связать уровень сервисов, моделей представления и комманд, был использован шаблон проектирования dependency property. Я выбрал его, потому что он позволяет писать слабосвязанный код и сократить потребление оперативной памяти. В нём были зарегистрированы сервисы, модели отображения и их фабрики. Подробнее зарегистрированные сервисы можно посмотреть в Приложении З.

# Тестирование, проверка работоспособности и анализ полученных результатов

При тестировании данного программного средства, были применены сценарии, которые могли привести процесс выполения у исключительным ситуациям.

При регистрации может возникнуть ситуация, при которой пользователь может ввести некорректные данные, а именно логин, пароль и электронную почту. Пример такого сценария приведён на рисунке 5.1.

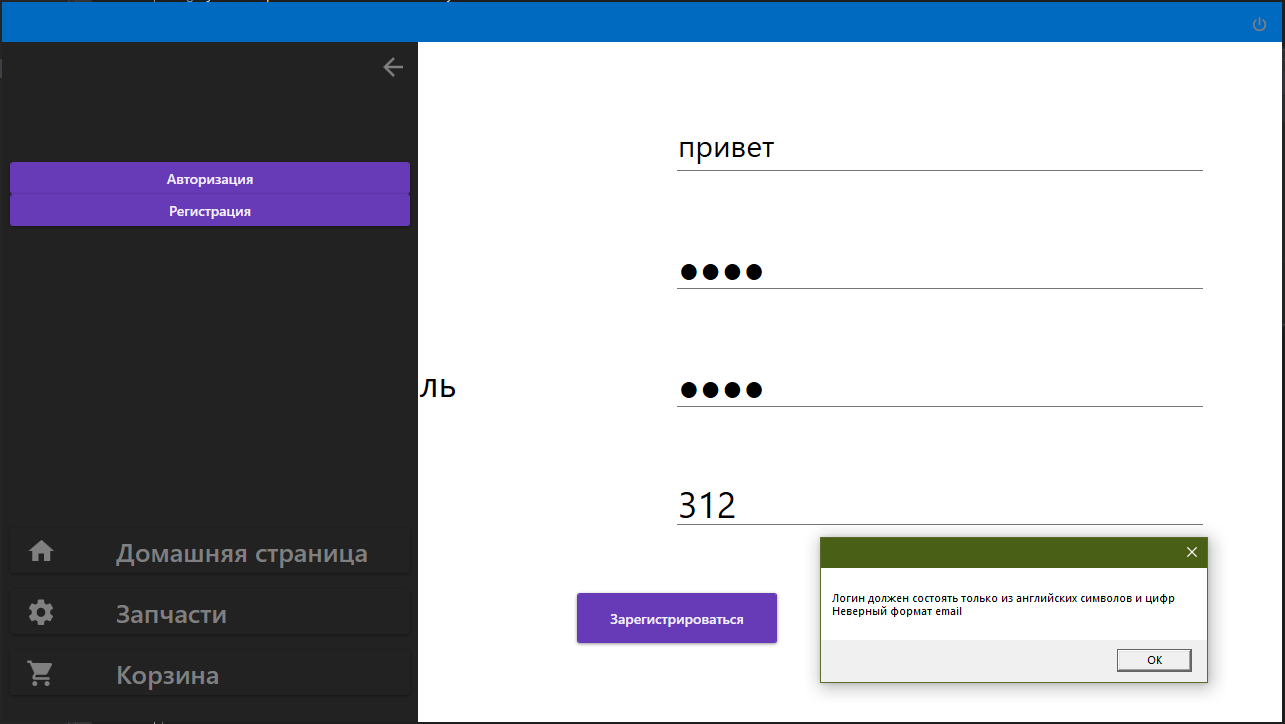


Рисунок 5.1 – пример некорректного ввода данных при регистрации

При авторизации может возникнуть ситуация, когда пользователь ввёл неверный пароль. Результат неверного ввода пароля показан на рисунке 5.2.

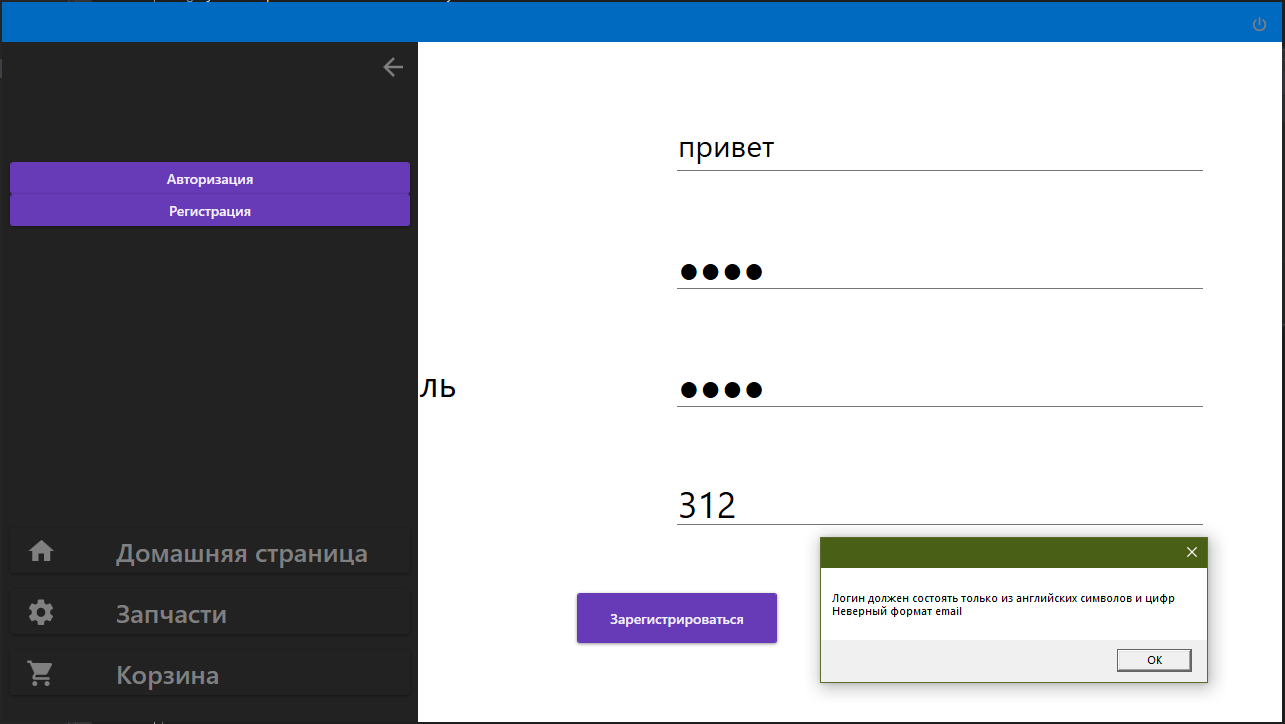


Рисунок 5.2 – пример ввода некорректного ввода пароля

Так как в программном средстве присутствуют DataGrid, то следует выводить сообщения о том, что пользователь не выбрал элемент перед некотором действии с ним. Результат такого действия показан на рисунке 5.3.

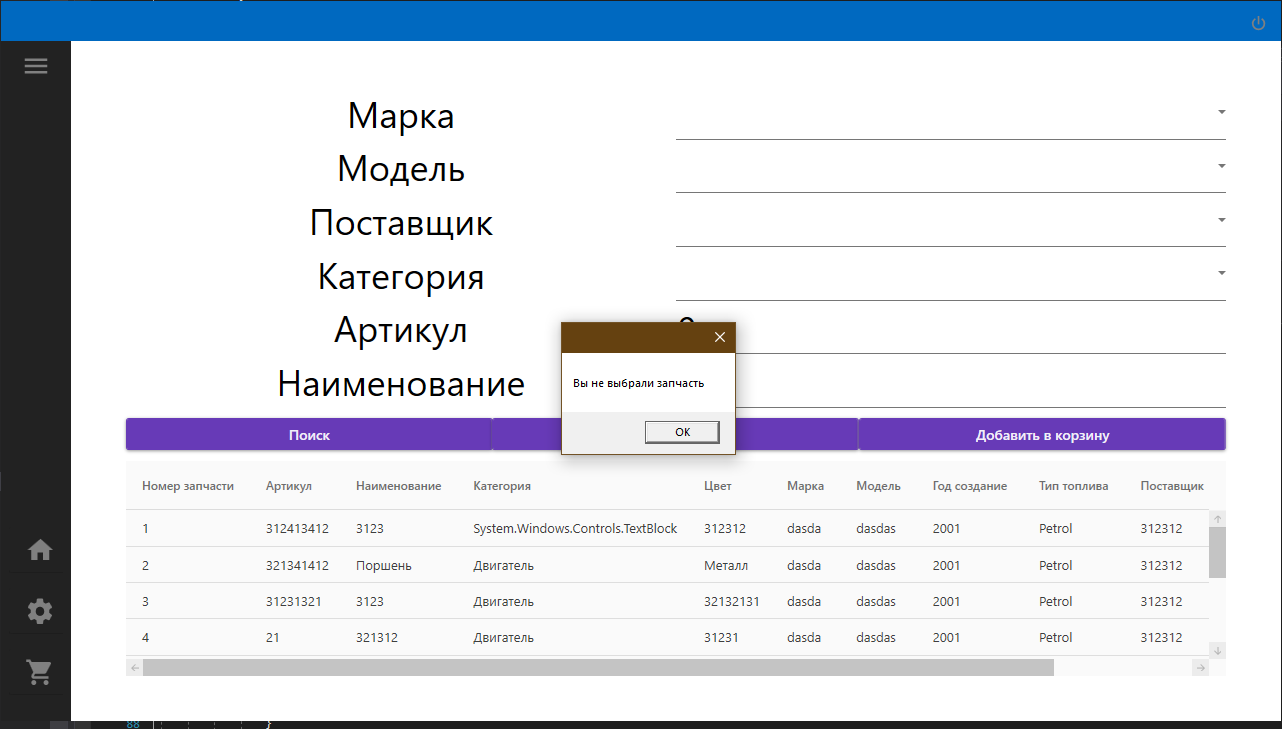


Рисунок 5.3 – результат нажатия кнопки Добавить в корзину без выбранной запчасти

Также была создана проверка при добавлении данных. Присером является ситуация, показанная на рисунке 5.4.

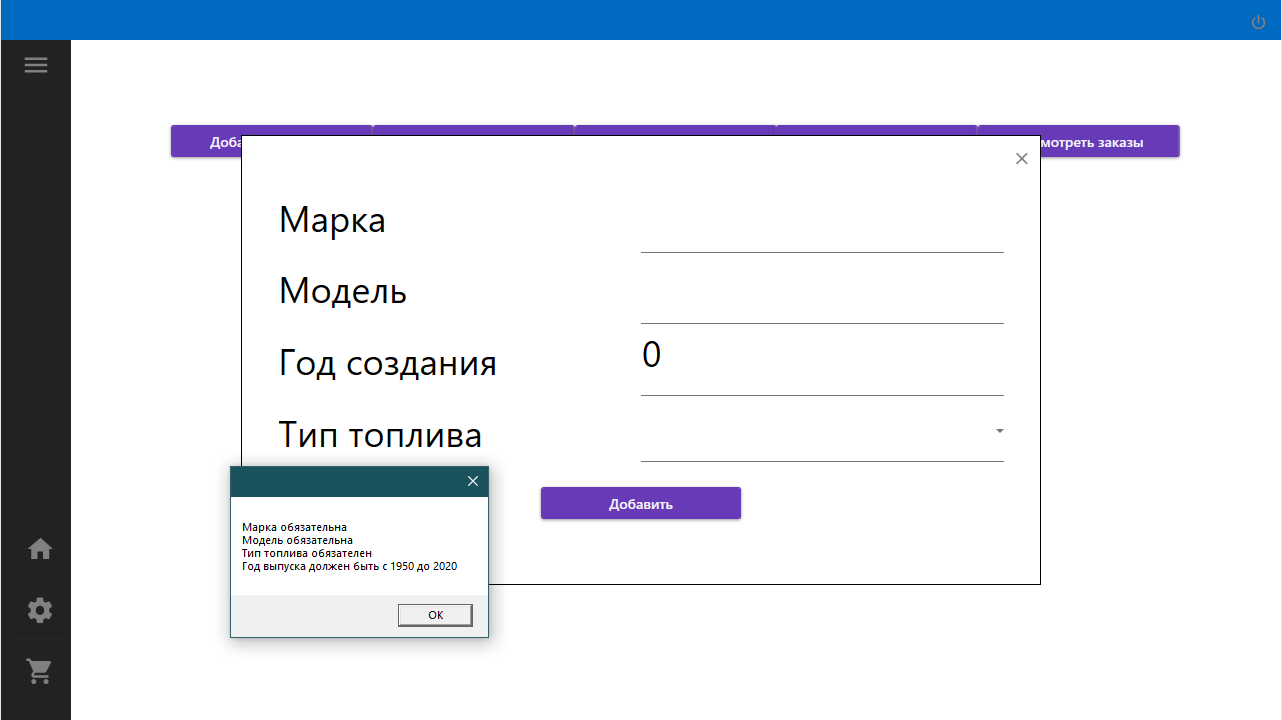


Рисунок 5.4 – исключение, возникшее из-за невведённых данных в поля

# Руководство по установке и использованию

После запуска программы пользователю предлагается авторизоваться либо зарегистрироваться. В случае удачной авторизации пояится страница пользователя (рисунок 6.1).

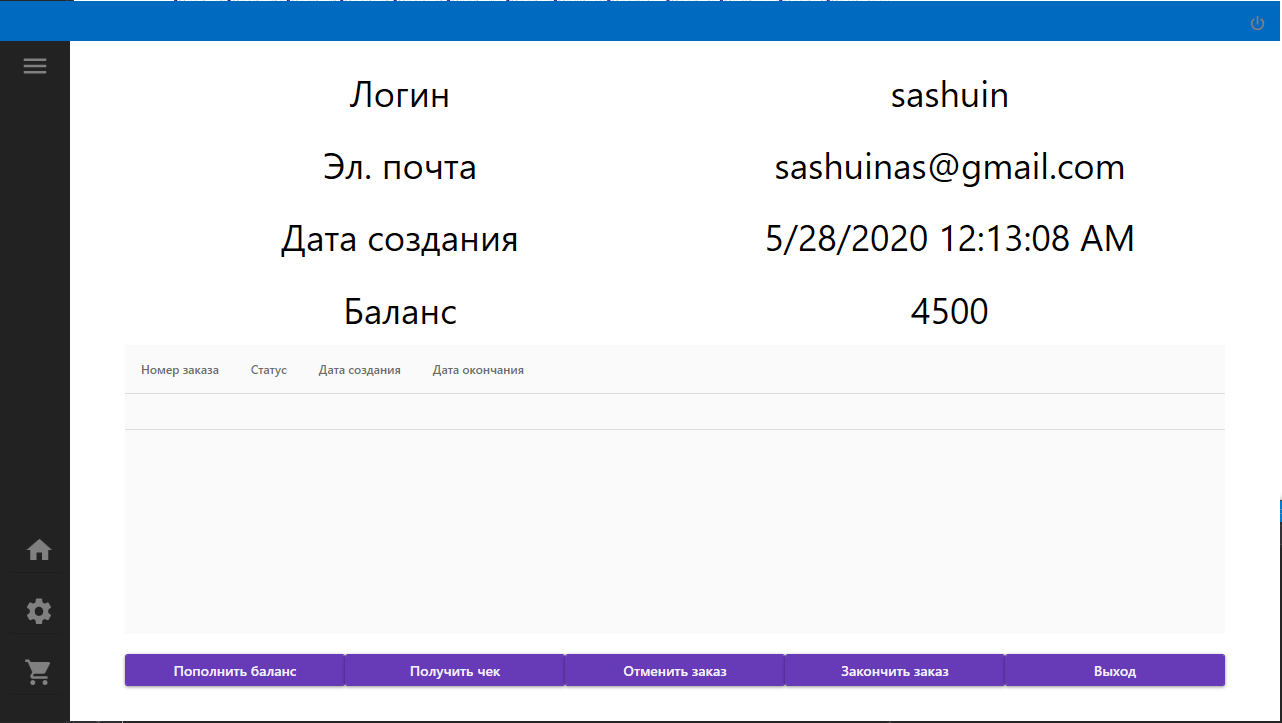


Рисунок 6.1 – страница пользователя

Далее необходимо выбрать запчасти, которые интересуют пользователя. Для этого необходимо выбрать критерии, по которым будет искаться запчасть для интересующего автомобиля. (рисунок 6.2)

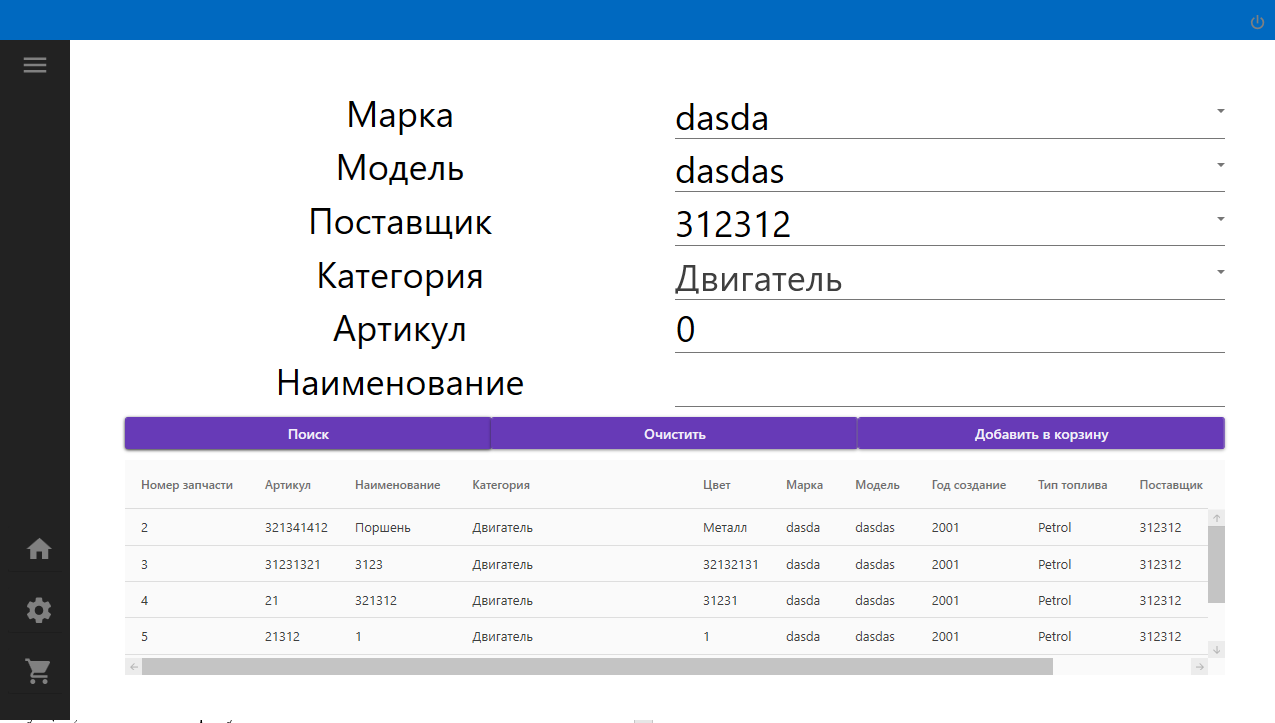


Рисунок 6.2 – страница просмотра ассортимента

Следующим шагом необходимо выбрать запчать и нажать на кнопку Добавить в корзину. Добавится одна единица товара и отобразится в корзине. Чтобы создать заказ, необходимо перейти во вкладку Корзина (рисунок 6.3), выбрать интересующие детали и нажать кнопку Создать заказ.

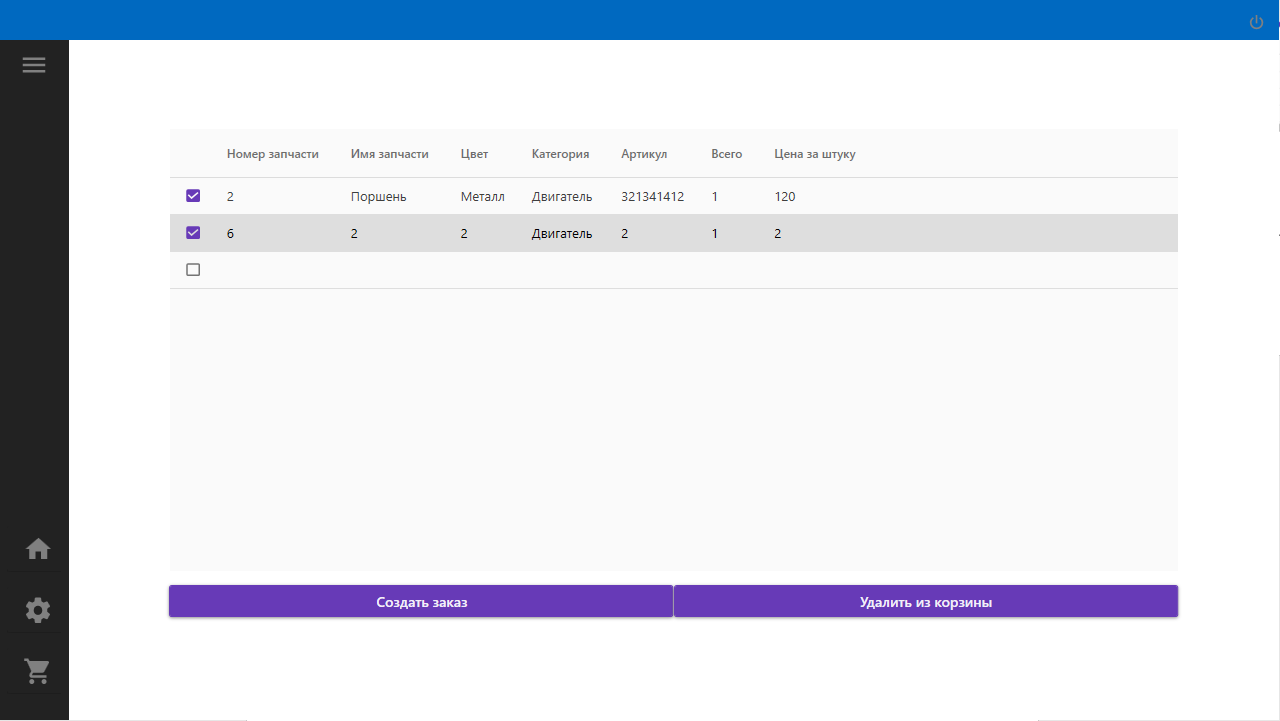


Рисунок 6.3 – окно корзины

После нажатия появится окно с вводом адреса, где необходимо указать свои данные. (рисунок 6.4)

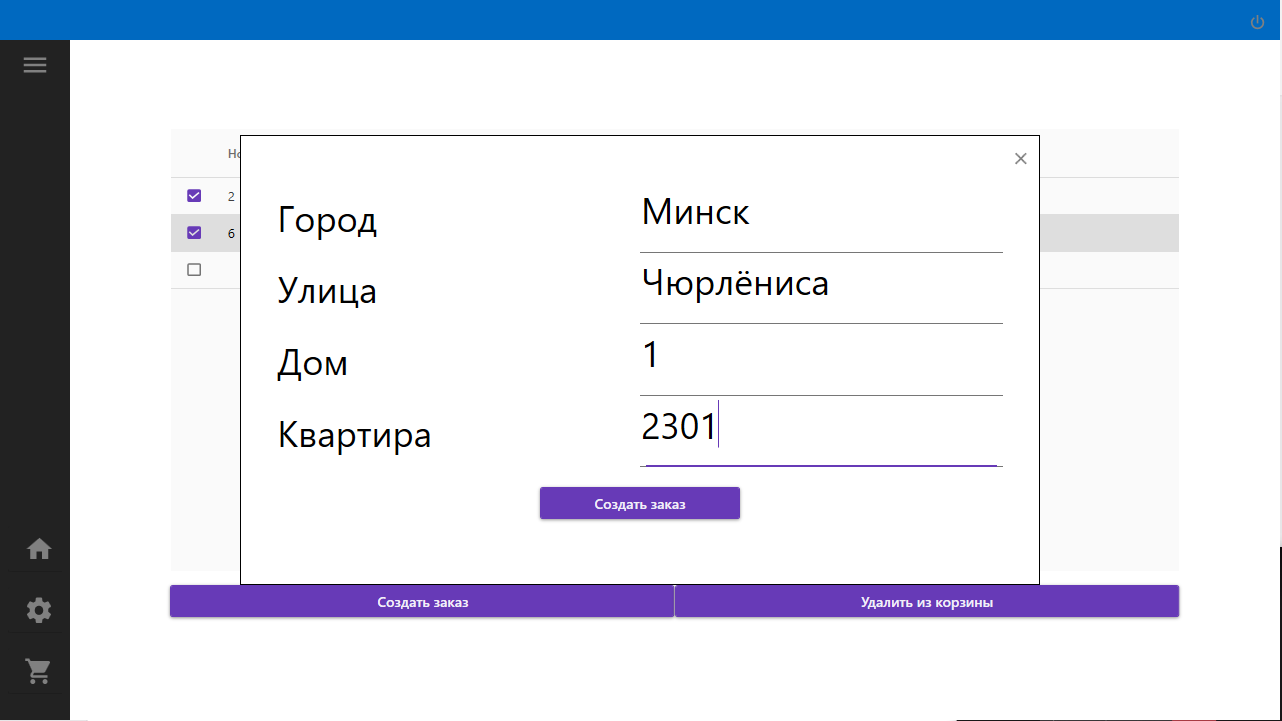


Рисунок 6.4 – окно ввода адреса

В случае успеха в окне пользователя появится заказ и у него будет статус PAYED.(рис.6.5) Это означает, что со счёта были сняты деньги и его ещё можно отменить, нажав на кнопку Отменить заказ.

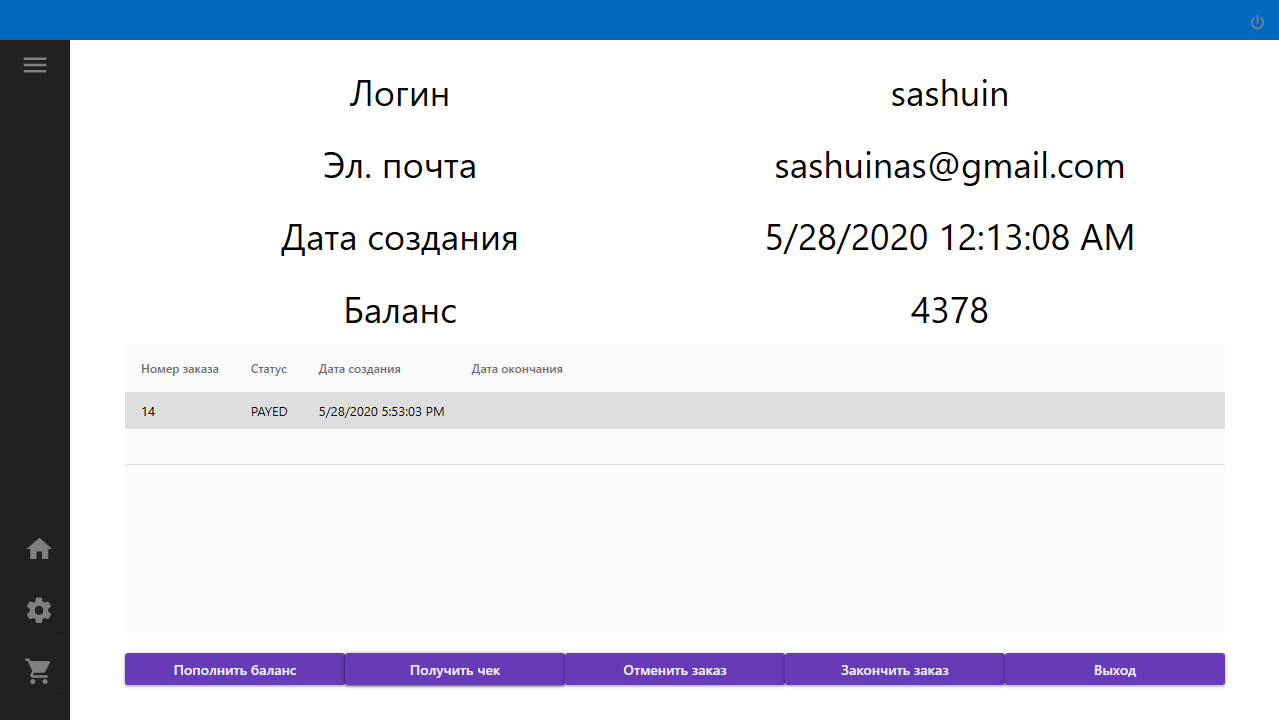


Рисунок 6.6 – окно пользователя после создания заказа

После смены заказа на DELIVERED, пользователь может нажать на кнопку Закончить заказ, подтверждая этим, что заказ завершён. При смене статуса заказа на e-mail пользователя приходит уведомление. Также пользователь в любой момент времени может посмотреть подробности своего заказа в виде документа в pdf-формате. (рисунок 6.6)

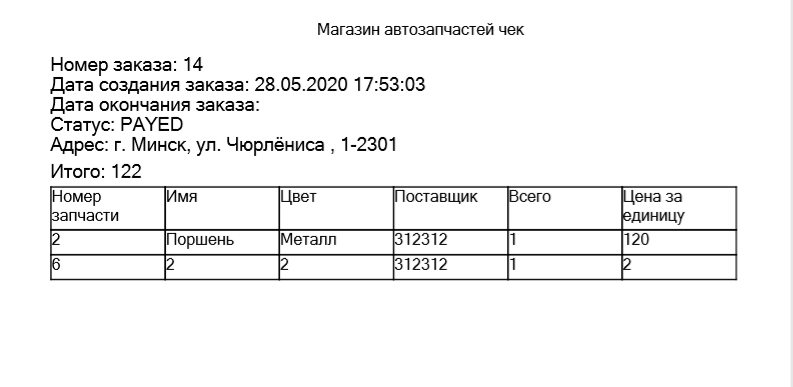


Рисунок 6.6 – чек заказа

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта было разработано приложение, реализующее основные функции магазина автозапчастей. В ходе анализа возможностей, разработанного ПС, был сделан вывод о том, что полноценная реализация подобного ПС требует более глубокой детализации предметной области.

Разработанное ПС реализовано с использованием платформы .NET Core, языка программирования C# и технологии WPF, применение базы данных в качестве хранилища информации позволяет оптимально и эффективно хранить информацию, структурировать ее.

В программном средстве были выполнены все пункты из указанного списка предполагаемого основного функционала приложения. Программное средство удовлетворяет поставленным задачам. В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

Для более корректной работы вся система построена на проверках валидации и связи объектов между собой, поэтому любое действие со стороны администратора должно быть обдуманным, поскольку данная система является автоматизированной.

Для разработки дизайна использовалась библиотека Material Design, которая имеет огромное количество красивых и удобных элементов управления.

Приложение было протестировано, поэтому возможность возникновения необработаных исключений сведена к нулю.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

[1]. METANIT.COM Сайт о программировании [Электронный ресурс] / Режим доступа: https://metanit.com

[2]. ProfessorWeb .NET & Web Programming [Электронный ресурс] / Режим доступа: <https://professorweb.ru>

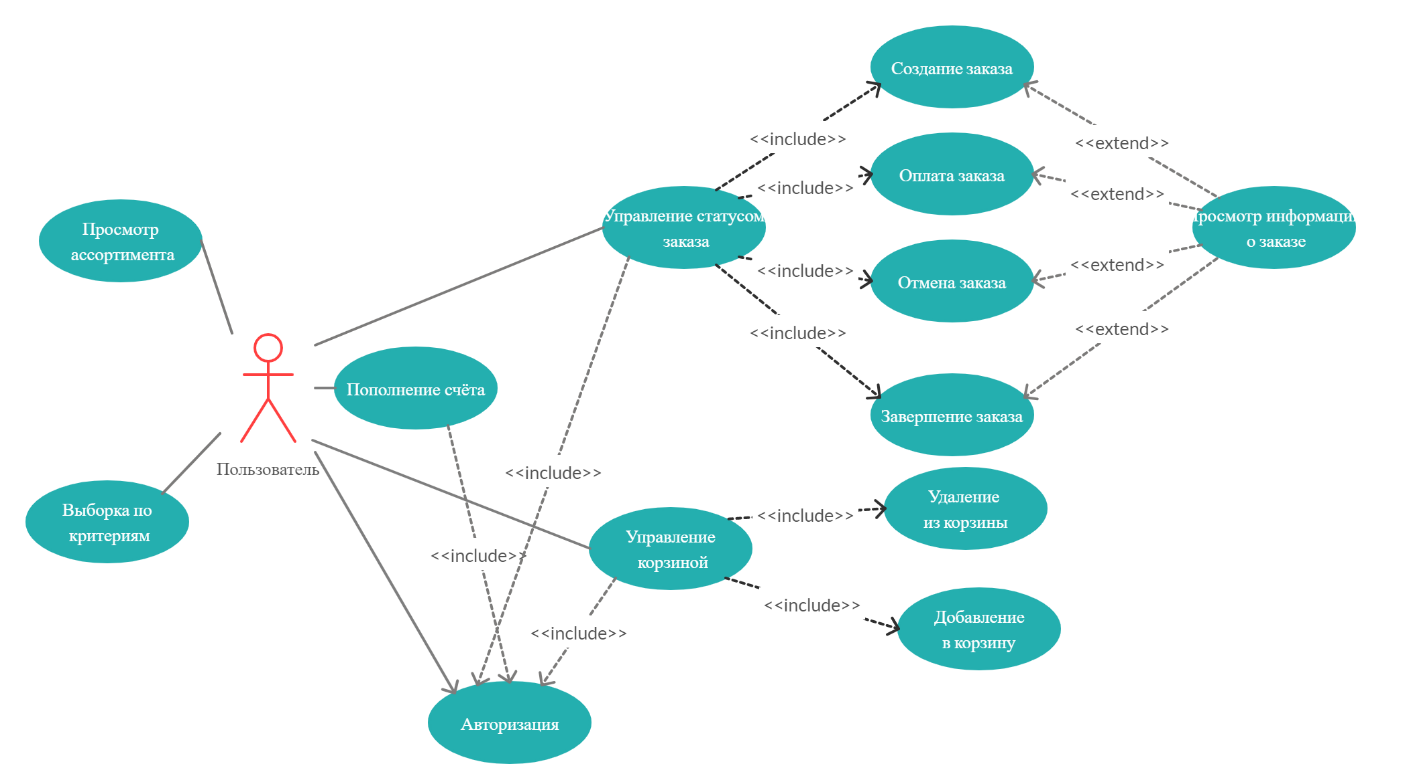
[3]. Docs Microsoft [Электронный ресурс] / Режим доступа: <https://docs.microsoft.com>

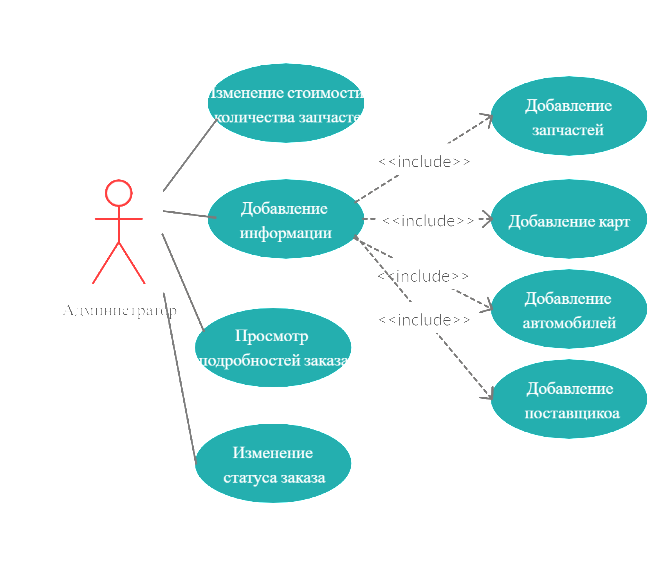
[4]. Refactoring Guru [Электронный ресурс] / Режим доступа: <https://refactoring.guru/ru/design-patterns>

[5]. Пацей, Н.В. Курс лекций по языку программирования С# / Н.В. Пацей. – Минск: БГТУ, 2018. – 175 с.

# Приложение А

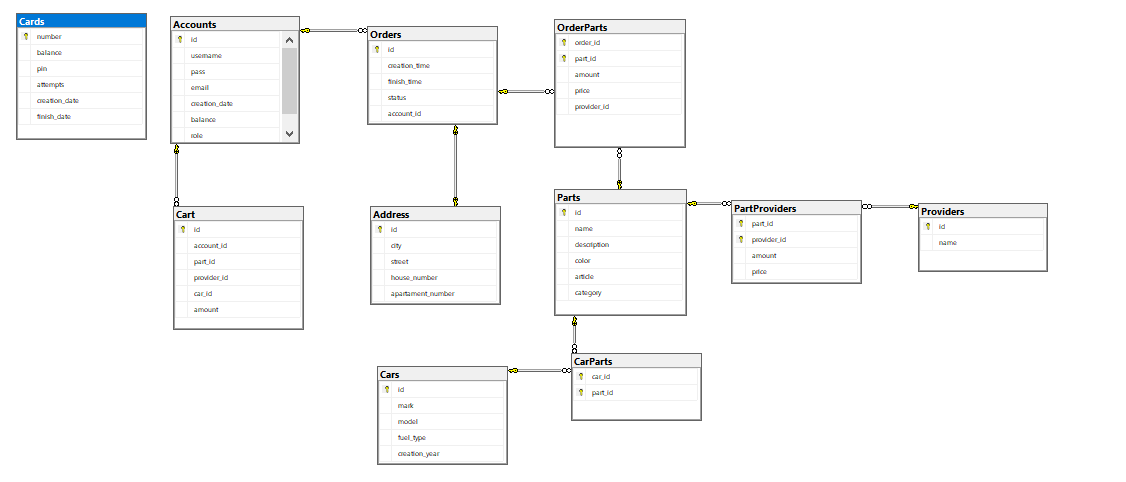
Диаграмма использования пользователя



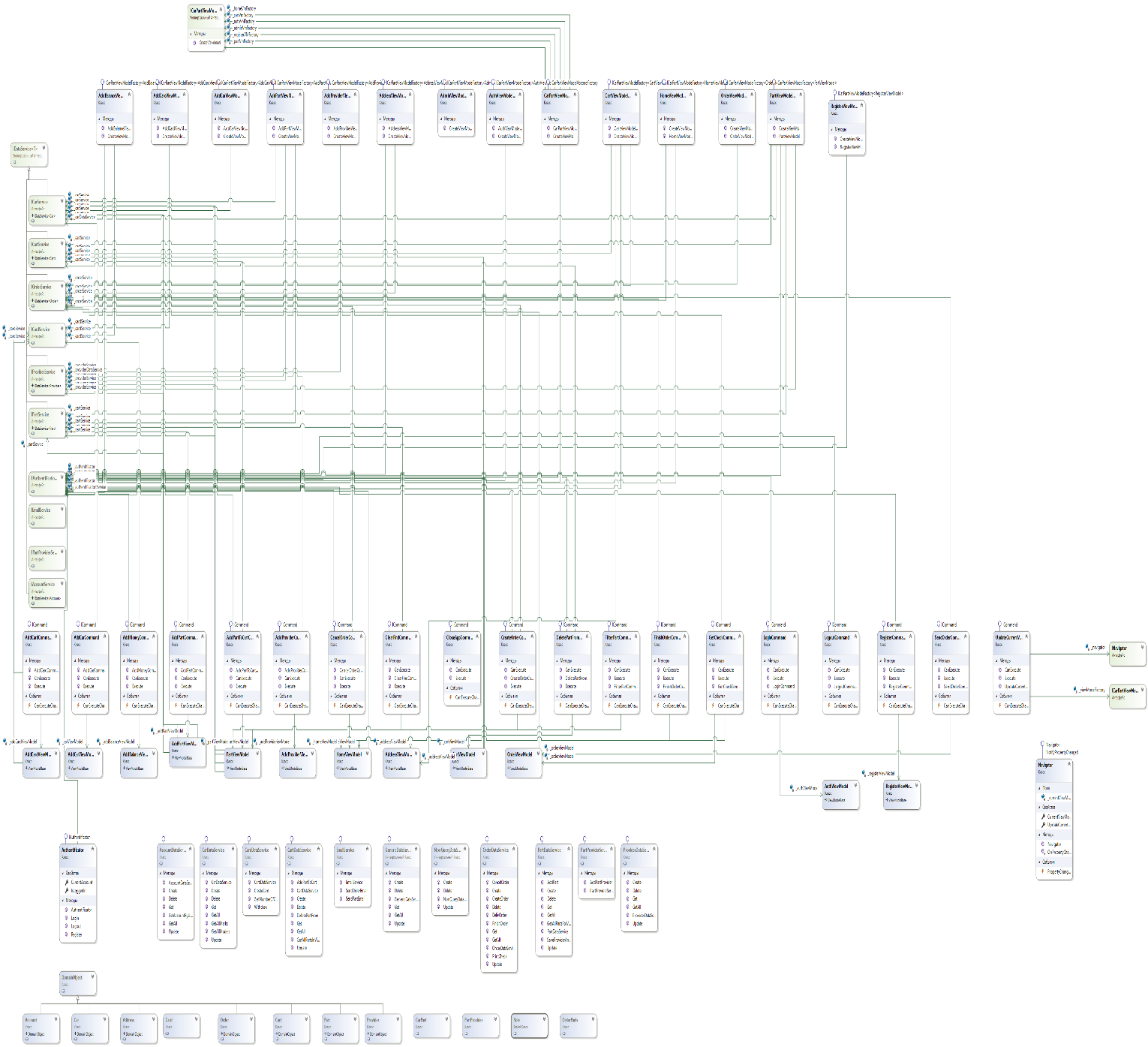


# Приложение Б

Структура базы данных

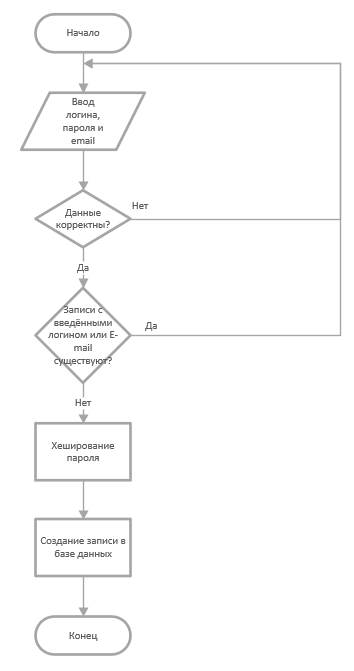


Приложение В

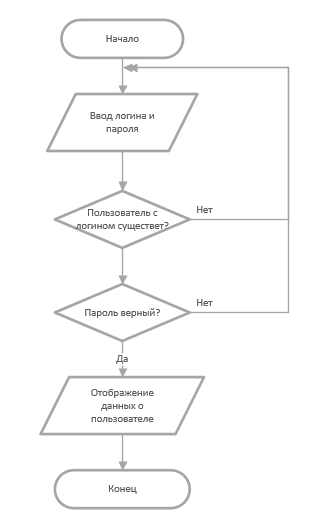


# Приложение Г

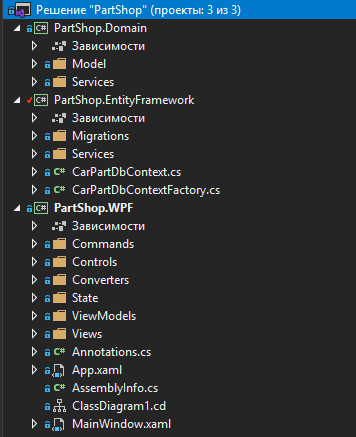
Блок-схема алгоритма регистрации



Блок-схема алгоритма авторизации



# Приложение Д



# Приложение Е

public interface IDataService<T>

{

Task<IEnumerable<T>> GetAll();

Task<T> Get(int id);

Task<T> Create(T entity);

Task<T> Update(int id, T entity);

Task<bool> Delete(int id);

}

Интерфейс IDataService<T>

public class NonQueryDataService<T> where T:DomainObject

{

private readonly CarPartDbContextFactory \_contextFactory;

public NonQueryDataService(CarPartDbContextFactory contextFactory)

{

\_contextFactory = contextFactory;

}

public async Task<T> Create(T entity)

{

using (CarPartDbContext context = \_contextFactory.CreateDbContext())

{

EntityEntry<T> createdResult = await context.Set<T>().AddAsync(entity);

await context.SaveChangesAsync();

return createdResult.Entity;

}

}

public async Task<T> Update(int id, T entity)

{

using (CarPartDbContext context = \_contextFactory.CreateDbContext())

{

entity.Id = id;

context.Set<T>().Update(entity);

await context.SaveChangesAsync();

return entity;

}

}

public async Task<bool> Delete(int id)

{

using (CarPartDbContext context = \_contextFactory.CreateDbContext())

{

T entity = await context.Set<T>().FirstOrDefaultAsync((e) => e.Id == id);

context.Set<T>().Remove(entity);

await context.SaveChangesAsync();

return true;

}

}

}

Класс NonQueryDataService

# Приложение Ж

public class CarPartViewModelAbstractFactory:ICarPartViewModelAbstractFactory

{

private readonly ICarPartViewModelFactory<AuthViewModel> \_authVMFactory;

private readonly ICarPartViewModelFactory<RegisterViewModel> \_registerVMFactory;

private readonly ICarPartViewModelFactory<HomeViewModel> \_homeVmFactory;

private readonly ICarPartViewModelFactory<PartViewModel> \_partVmFactory;

private readonly ICarPartViewModelFactory<AdminViewModel> \_adminVmFactory;

private readonly ICarPartViewModelFactory<CartViewModel> \_cartVmFactory;

private readonly IAuthentificator \_authentificator;

public CarPartViewModelAbstractFactory(

ICarPartViewModelFactory<RegisterViewModel> registerVmFactory,

ICarPartViewModelFactory<AuthViewModel> authVmFactory,

ICarPartViewModelFactory<HomeViewModel> homeVmFactory,

ICarPartViewModelFactory<PartViewModel> partVmFactory,

ICarPartViewModelFactory<AdminViewModel> adminVmFactory,

ICarPartViewModelFactory<CartViewModel> cartVmFactory,

IAuthentificator authentificator)

{

\_registerVMFactory = registerVmFactory;

\_authVMFactory = authVmFactory;

\_homeVmFactory = homeVmFactory;

\_partVmFactory = partVmFactory;

\_adminVmFactory = adminVmFactory;

\_cartVmFactory = cartVmFactory;

\_authentificator = authentificator;

}

public ViewModelBase CreateViewModel(ViewType viewType)

{

switch (viewType)

{

case ViewType.AUTH:

if (\_authentificator.IsLoggedIn) return \_homeVmFactory.CreateViewModel();

return \_authVMFactory.CreateViewModel();

case ViewType.REGISTER:

if (\_authentificator.IsLoggedIn) return \_homeVmFactory.CreateViewModel();

return \_registerVMFactory.CreateViewModel();

case ViewType.HOME:

if (\_authentificator.IsLoggedIn == false) return \_authVMFactory.CreateViewModel();

if (\_authentificator.CurrentAccount.UserRole == Role.ADMIN) return \_adminVmFactory.CreateViewModel();

return \_homeVmFactory.CreateViewModel();

case ViewType.PARTS:

return \_partVmFactory.CreateViewModel();

case ViewType.ADMIN:

return \_adminVmFactory.CreateViewModel();

case ViewType.CART:

if (\_authentificator.IsLoggedIn == false) return \_authVMFactory.CreateViewModel();

return \_cartVmFactory.CreateViewModel();

default:

throw new ArgumentException("Uncorrect viewType parametr", "viewType");

}

}

}

public class Navigator:INavigator, INotifyPropertyChanged

{

private ViewModelBase \_currentViewModel;

public ViewModelBase CurrentViewModel

{

get => \_currentViewModel;

set

{

\_currentViewModel=value;

OnPropertyChanged(nameof(CurrentViewModel));

}

}

public ICommand UpdateCurrentViewModelCommand { get; set; }

public event PropertyChangedEventHandler PropertyChanged;

[NotifyPropertyChangedInvocator]

protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

public Navigator(ICarPartViewModelAbstractFactory viewModelFactory)

{

UpdateCurrentViewModelCommand=new UpdateCurrentViewModelCommand(this, viewModelFactory);

}

}

public class UpdateCurrentViewModelCommand:ICommand

{

public event EventHandler CanExecuteChanged;

private INavigator \_navigator;

private readonly ICarPartViewModelAbstractFactory \_viewModelFactory;

public UpdateCurrentViewModelCommand(INavigator navigator, ICarPartViewModelAbstractFactory viewModelFactory)

{

\_navigator = navigator;

\_viewModelFactory = viewModelFactory;

}

public bool CanExecute(object parameter)

{

return true;

}

public void Execute(object parameter)

{

if (parameter is ViewType)

{

ViewType viewType = (ViewType) parameter;

\_navigator.CurrentViewModel = \_viewModelFactory.CreateViewModel(viewType);

}

}

}

# Приложение З

private IServiceProvider CreateServiceProvider()

{

IServiceCollection service=new ServiceCollection();

service.AddSingleton<INavigator, Navigator>();

service.AddScoped<MainViewModel>();

service.AddSingleton<CartViewModel>();

service.AddSingleton<MainWindow>(s=>new MainWindow(s.GetRequiredService<MainViewModel>()));

service.AddSingleton<IAuthentificator, Authentificator>();

service.AddSingleton<CarPartDbContextFactory>();

service.AddSingleton<IAuthentificationService, AuthentificationService>();

service.AddSingleton<IDataService<Account>, AccountDataService>();//МБ УДАЛИТЬ

service.AddSingleton<IAccountService, AccountDataService>();

service.AddSingleton<ICarService, CarDataService>();

service.AddSingleton<IPartService, PartDataService>();

service.AddSingleton<ICartService,CartDataService>();

service.AddSingleton<IOrderService,OrderDataService>();

service.AddSingleton<ICardService,CardDataService>();

service.AddSingleton<IEmailService,EmailService>();

service.AddSingleton<IProviderService, ProviderDataService>();

service.AddSingleton<ICarPartViewModelAbstractFactory, CarPartViewModelAbstractFactory>();

service.AddSingleton<ICarPartViewModelFactory<AuthViewModel>, AuthViewModelFactory>();

service.AddSingleton<ICarPartViewModelFactory<RegisterViewModel>, RegisterViewModelFactory>();

service.AddSingleton<ICarPartViewModelFactory<HomeViewModel>, HomeViewModelFactory>();

service.AddSingleton<ICarPartViewModelFactory<PartViewModel>, PartViewModelFactory>();

service.AddSingleton<ICarPartViewModelFactory<AdminViewModel>, AdminViewModelFactory>();

service.AddSingleton<ICarPartViewModelFactory<AddCarViewModel>, AddCarViewModelFactory>();

service.AddSingleton<ICarPartViewModelFactory<AddProviderViewModel>, AddProviderViewModelFactory>();

service.AddSingleton<ICarPartViewModelFactory<AddPartViewModel>, AddPartViewModelFactory>();

service.AddSingleton<ICarPartViewModelFactory<CartViewModel>, CartViewModelFactory>();

service.AddSingleton<ICarPartViewModelFactory<OrderViewModel>, OrderViewModelFactory>();

service.AddSingleton<ICarPartViewModelFactory<AddCardViewModel>, AddCardViewModelFactory>();

service.AddSingleton<ICarPartViewModelFactory<AddBalanceViewModel>, AddBalanceViewModelFactory>();

service.AddSingleton<ICarPartViewModelFactory<AddressViewModel>, AddressViewModelFactory>();

return service.BuildServiceProvider();

}