МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий Кафедра Программной инженерии Специальность 1-40 05 01 Информационные системы и технологии   
Направление специальности 1-40 05 01 Информационные системы и технологии

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

по дисциплине «Объектно-ориентированное программирование»   
Тема «Программное средство по подбору автозапчастей»

Исполнитель

студент (ка) 2 курса группы 1 Юшкевич Илья Николаевич

(Ф.И.О.)

Руководитель работы ст. препод. Сухорукова И.Г.

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

Курсовой проект защищен с оценкой

Председатель Пацей Н.В.

(подпись)

Минск 2020

**ВВЕДЕНИЕ**

В наше время автомобили стали неотъемлемой частью жизни каждого человека. Они позволяют преодолевать большие расстояния без больших затрат по времени и ресурсов человека, что значительно ускоряет и упрощает нашу жизнь. Свою популярность они приобрели за счет своей стоимости и мобильности по сравнению с другими видами транспорта. Однако автомобили, как и любой другой инструмент, не идеальны и могут выходить из строя. Несмотря на то, что с каждым годом автомобили становятся все надежнее, все еще есть потребность в их ремонте, для которого необходимы запасные детали. Актуальность темы поиска автозапчастей связана именно с тем, что автомобили являются наиболее популярным видом транспорта после велосипеда. Их также выпускает огромное количество заводов в различных комплектациях, видах кузова, что очень сильно затрудняет поиск запасных деталей к конкретной модели.

Целью данного курсового проекта является разработка программного средства, которое позволит быстро и удобно найти необходимую деталь для конкретного автомобиля. Продавцу запчастей – быстро продавать их, а покупателям – продавать.

Программное средство должно позволять продавцу создавать новые товары, регистрировать заказы, оповещать клиента о выполнении заказа, а клиенту - отслеживать статус заказа, позволять пользователю искать магазины.

В разделе «Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству» рассматриваются аналоги данного программного средства, а также формируются требования для выполнения в курсовом проекте.

В разделе «Моделирование предметной области и разработка функциональных требований» сформулированы функциональные требования для проектирования программного средства.

В разделе «Проектирование программного средства» описана структура программного средства.

В разделе «Создание программного средства» реализуется техническое решение и собирается программное средство.

В разделе «Тестирование, проверка работоспособности и анализ полученных результатов» выполнено тестирование программного средства в соответствии с требованиями, составленными ранее.

В разделе «Руководство по установке и использованию» подробно описаны правила по установке и использованию программного средства.

1. **Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству**

В данном курсовом проекте будут использоваться дополнительные материалы из таких источников, как <https://metanit.com/> [1] для получения дополнительной информации о WPF, <https://refactoring.guru/> [2] для дополнительной информации о внедрении паттернов проектирования в программное средство с целью создания более гибкого продукта, а также официальный сайт Microsoft <https://docs.microsoft.com/> [3] для получения общей информации о C# и Microsoft SQL Server.

В качестве аналогов были выбраны сайты <https://motorland.by/> [4], <https://www.autozone.com/> [5].

1. <https://motorland.by/>

Для клиентов главная страница содержит поиск комплектующих по марке и модели автомобиля (рис. 1.1):

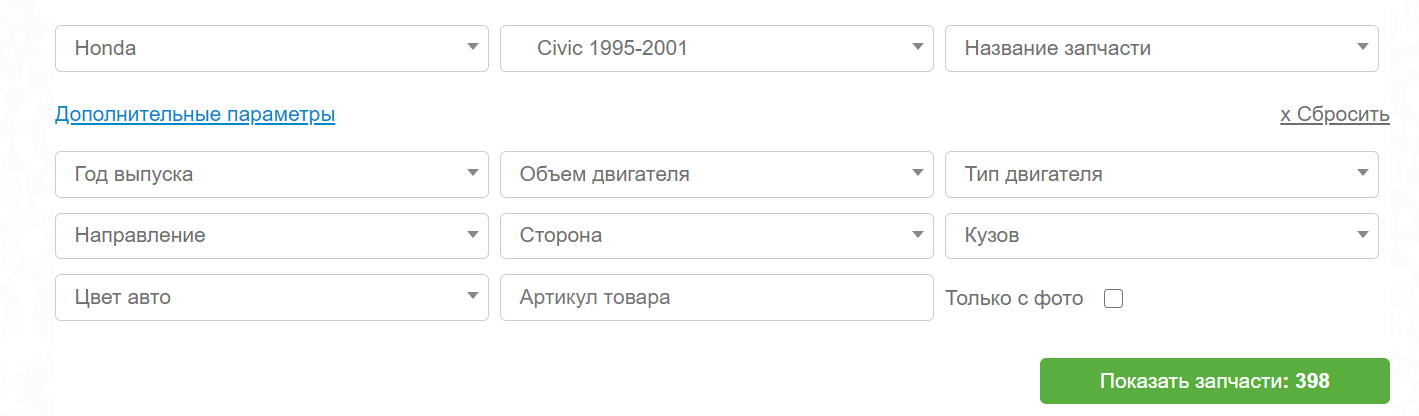


Рис. 1.1 – Поиск по марке и модели

После выбора автомобиля отображаются все запчасти для данного автомобиля. Есть возможность дальнейшей фильтрации по запчастям, а также сортировка (рис. 1.2):

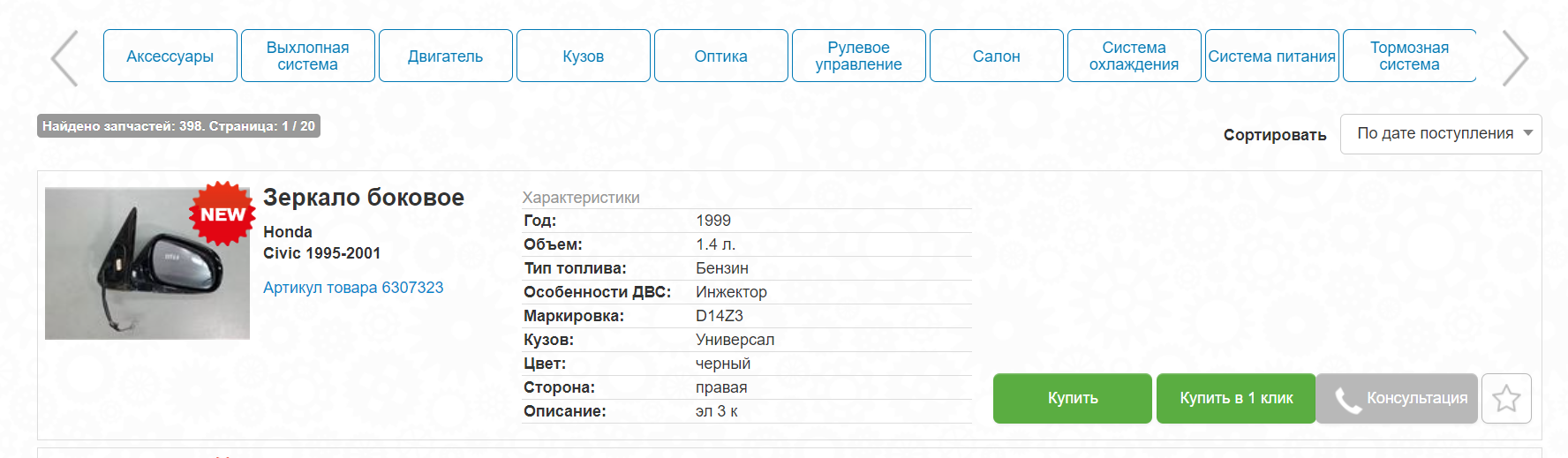


Рис. 1.2 – Список запчастей по поиску

Карта товара содержит его фотографии, а также описание автомобиля, к которому этот товар подходит (рис 1.3):

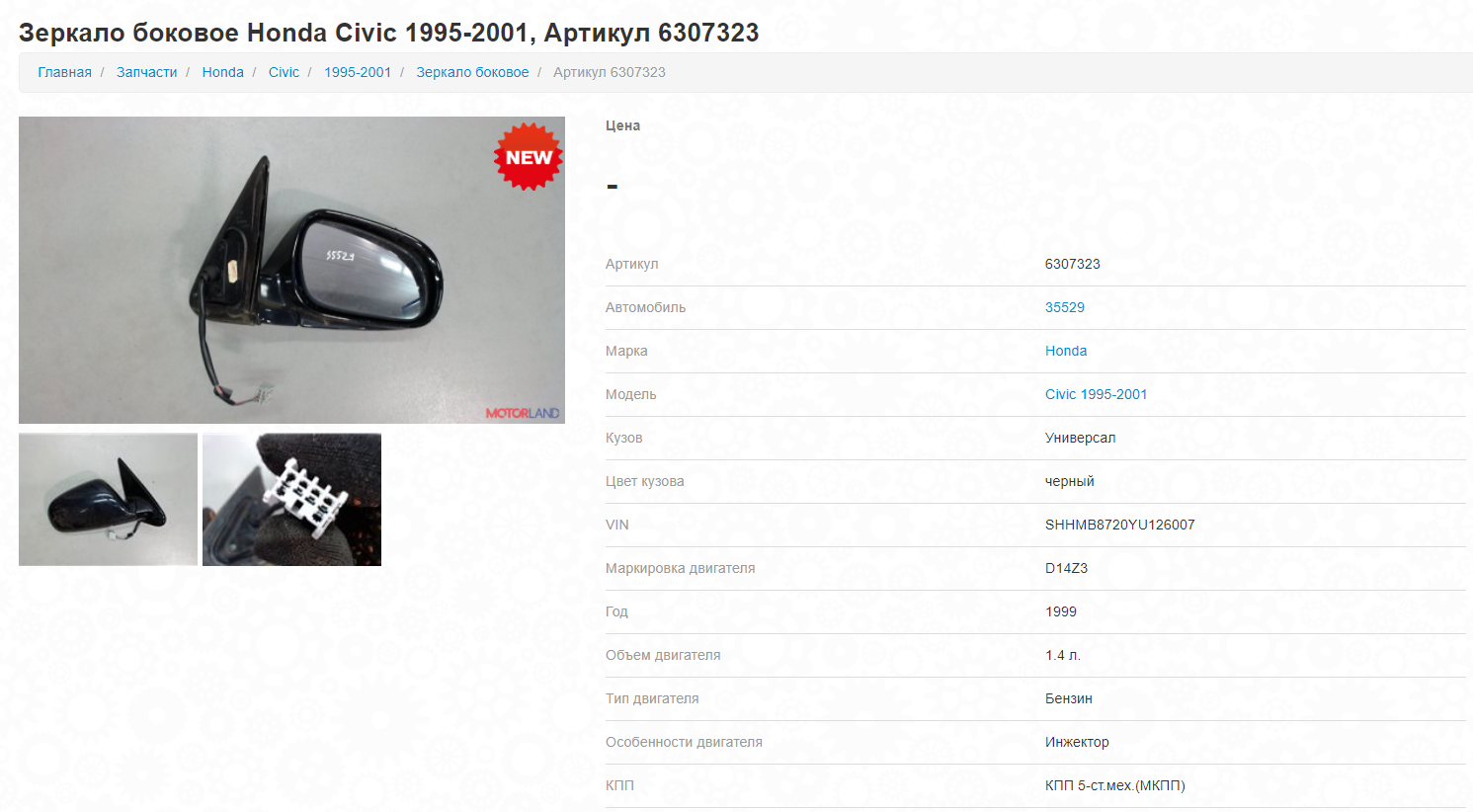


Рис. 1.3 – Карта товара

1. [https://www.autozone.com/](%20https://www.autozone.com/)

Главная страница содержит кнопки для поиска по модели авто и по категориям запчастей (рис. 1.4):

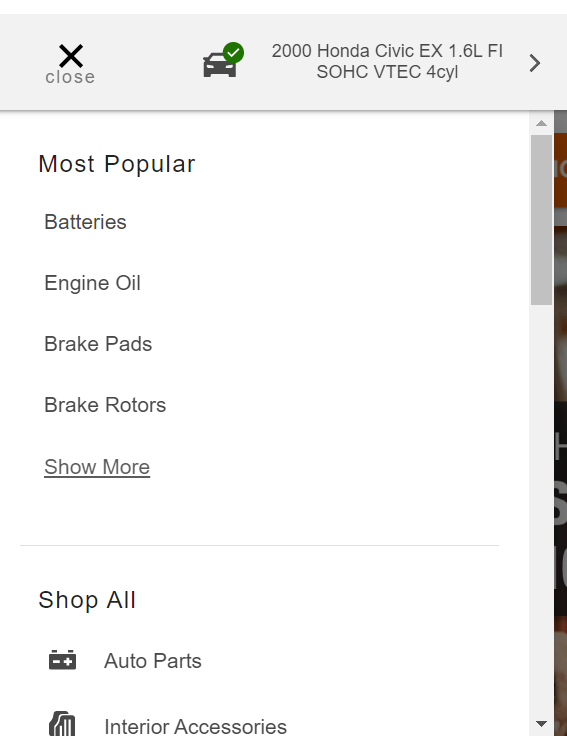


Рис. 1.4 – Поиск по категориям

Категории скрыты в боковом меню, а по нажатию на кнопку выбора автомобиля открывается окно с фильтрами (рис. 1.5):

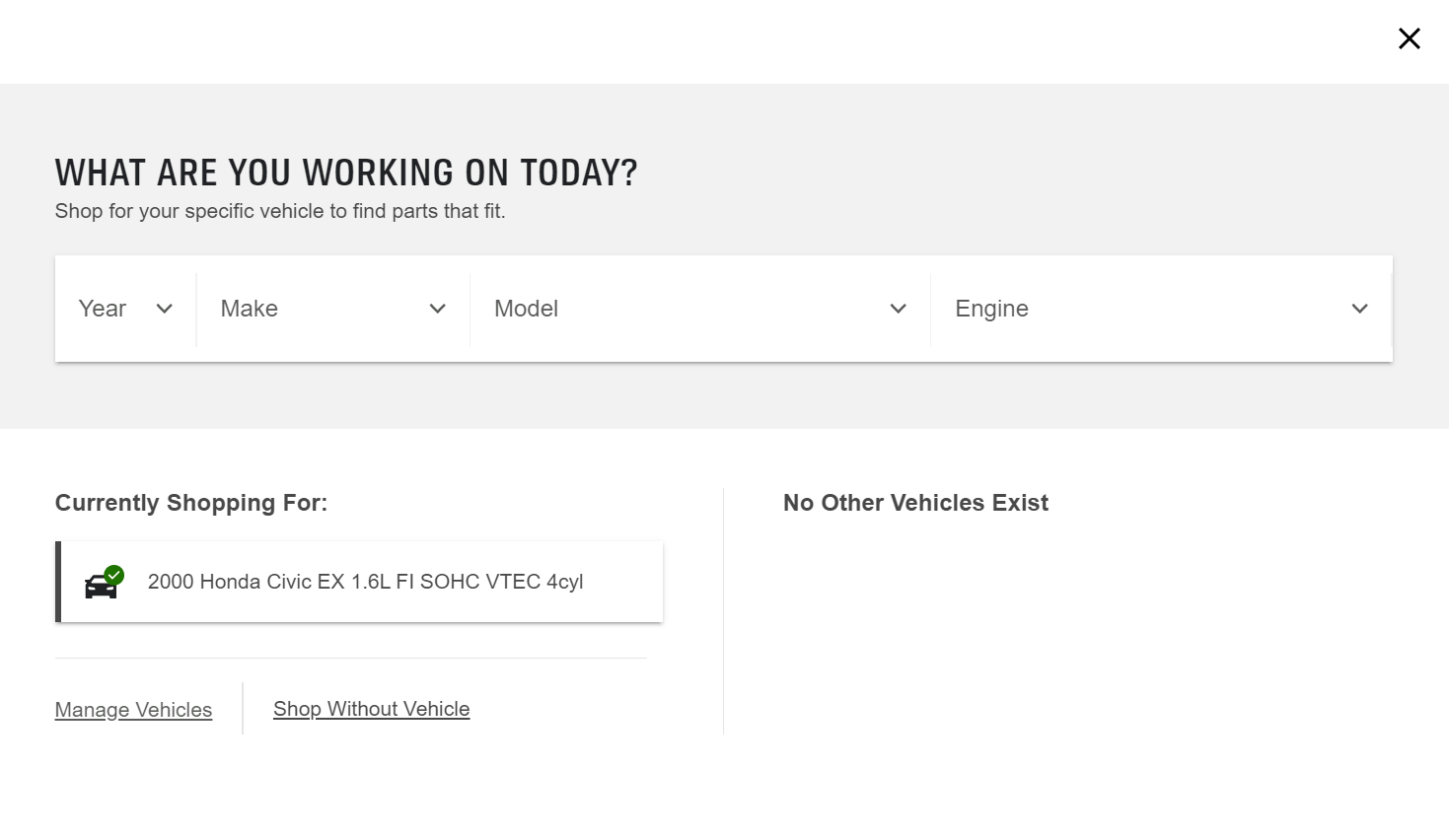


Рис. 1.5 – Фильтры для поиска по модели

После выбора автомобиля всплывает уведомление о том, что в каталоге будут отображаться товары, подходящие выбранному автомобилю (рис. 1.6):

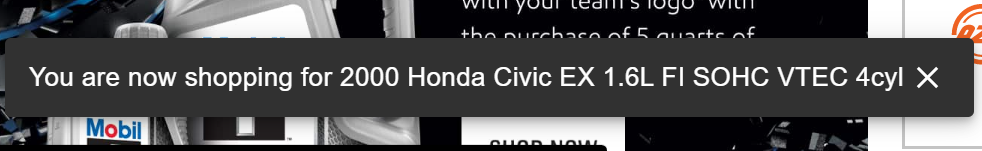


Рис. 1.6 – Уведомление о выборе автомобиля

В каталоге отображается список доступных деталей, также есть поиск по производителю (рис. 1.7):

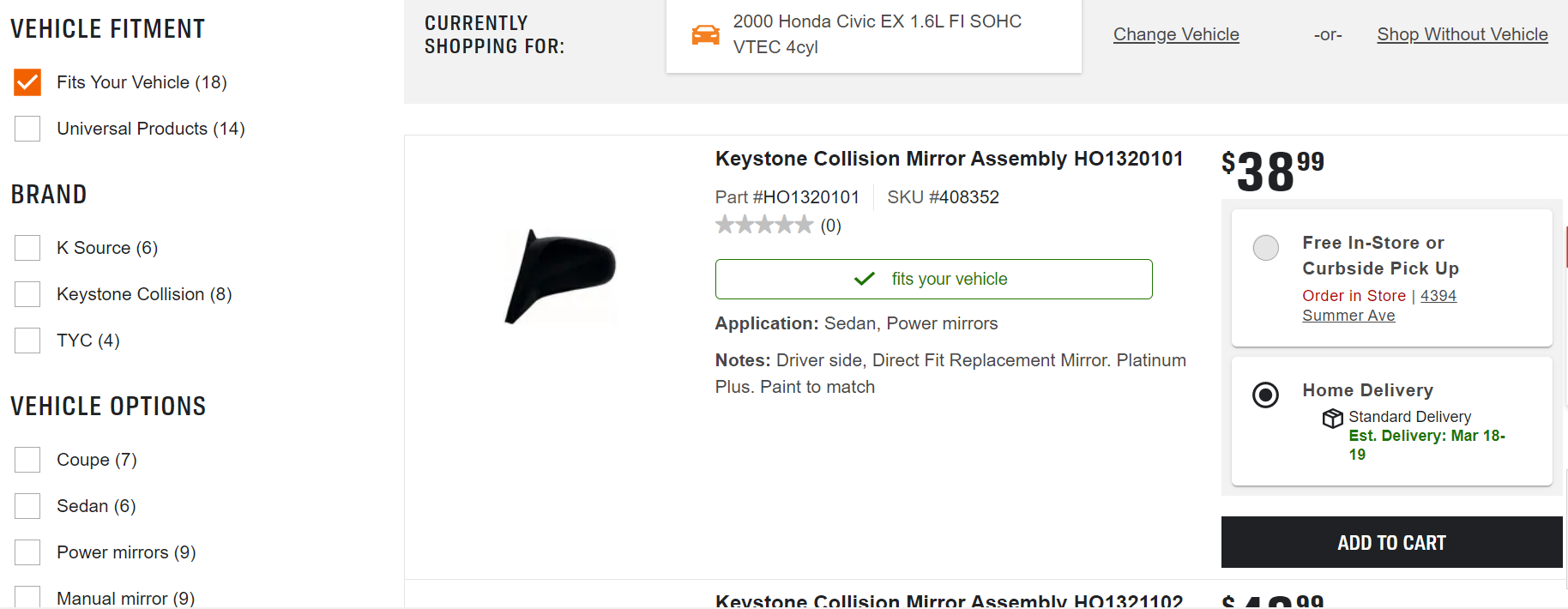


Рис. 1.7 – Список товаров

Карта товара содержит фотографии товара (рис. 1.8), а также его характеристики (рис. 1.9):

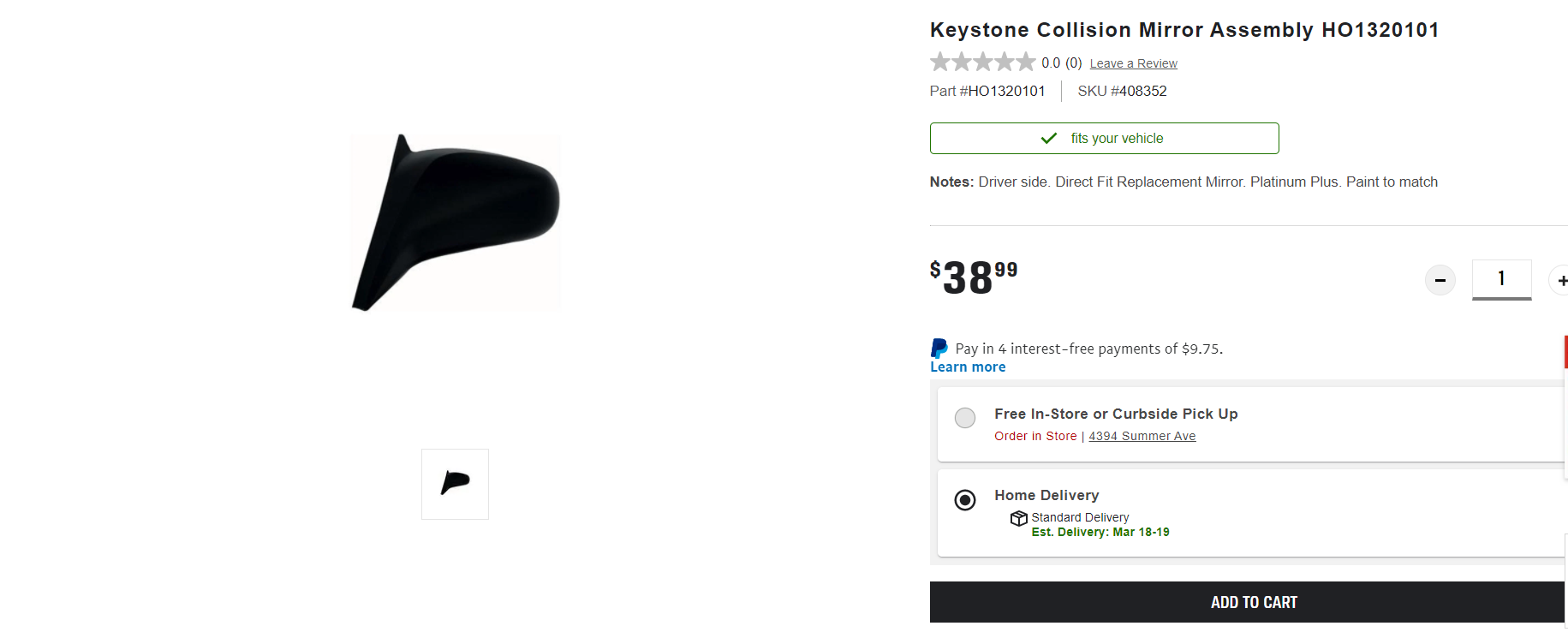


Рис. 1.8 – Фотография товара и выбор доставки

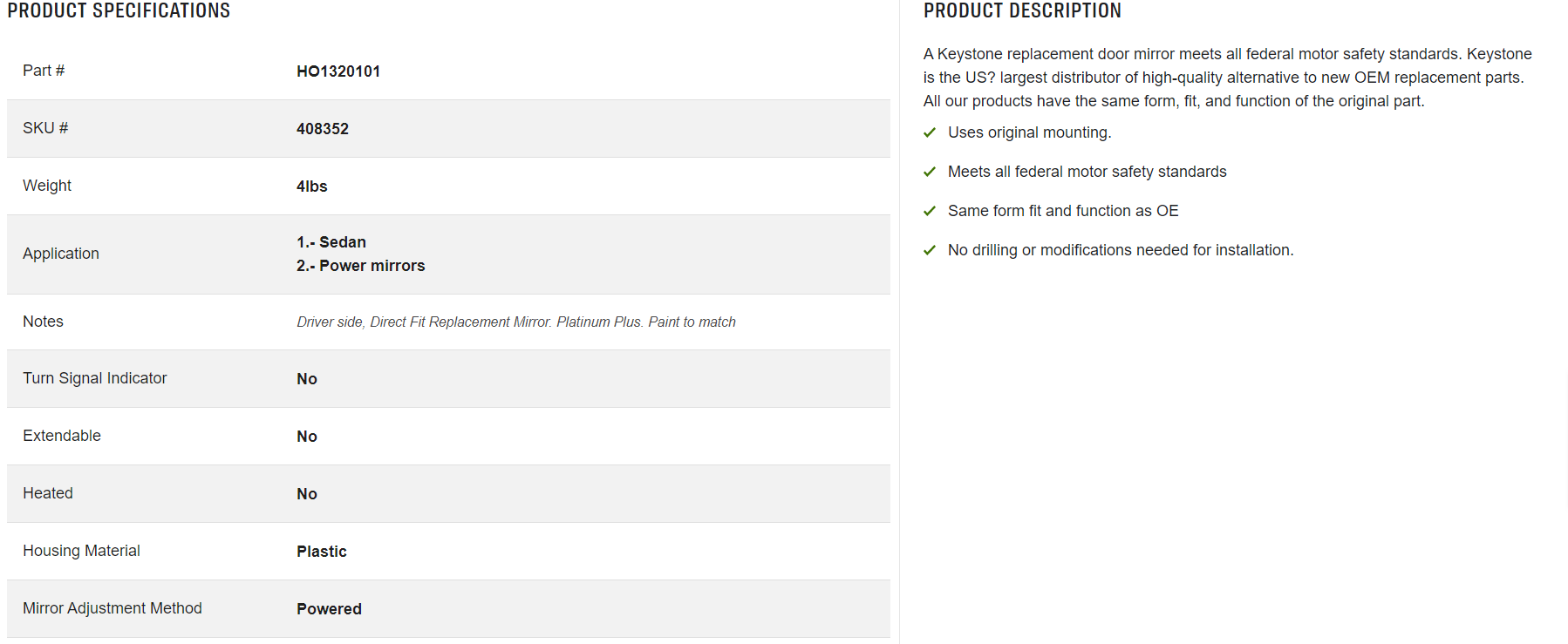


Рис. 1.9 – Описание продукта и его характеристики

Исходя из анализа аналогов, были выявлены сильные и слабые стороны обоих проектов.

Сильные стороны:

* Удобный поиск по модели автомобиля на обоих сайтах;
* Возможность как быстро заказать товар, так и добавить его в корзину;
* Краткое описание товара в каталоге;

Слабые стороны:

* Устаревший интерфейс на сайте <https://motorland.by/>;
* Неудобная фильтрация товаров по категориям в виде карусели на сайте <https://motorland.by/>;

Исходя из анализа аналогов, были выделены требования к программному обеспечению:

* Удобный и понятный интерфейс для пользователей с любой компьютерной грамотностью;
* Разделение бизнес-логики приложения на клиентскую часть и часть администратора;

1. **Моделирование предметной области и разработка функциональных требований.**
   1. **Описание инструментов для разработки программного средства**

В ходе разработки данного программного средства используются следующие инструменты:

* Объектно-ориентированный язык программирования C#;
* Платформа для кроссплатформенной разработки с открытым исходным кодом .NET Core;
* Расширяемый язык разметки XAML;
* Система управления базами данных Microsoft SQL Server;
* Шаблон проектирования MVVM;
* Фреймворк для работы с базой данных Entity Framework Core;
* Программный интерфейс приложения Bing Maps.

C# [6]— это язык программирования, предназначенный для разработки самых разнообразных приложений, предназначенных для выполнения в среде .NET Framework. Visual C# — это реализация языка C# корпорацией Майкрософт. Поддержка Visual C# в Visual Studio обеспечивается с помощью полнофункционального редактора кода, компилятора, шаблонов проектов, конструкторов, мастеров кода, мощного и удобного отладчика и многих других средств. Библиотека классов .NET Framework предоставляет доступ ко многим службам операционной системы и другим полезным, правильным классам, что существенно ускоряет цикл разработки.

.NET — это модульная платформа для разработки программного обеспечения с открытым исходным кодом, основанная на .NET Framework, но разделенная на модули, например, с интегрированной библиотекой CoreFX, а также со средой выполнения CoreCLR, включающей в себя JIT-компилятор, сборщик мусора и другие компоненты.

XAML — это декларативный язык разметки. С точки зрения модели программирования .NET Core язык XAML упрощает создание пользовательского интерфейса для приложения .NET Core. Можно создать видимые элементы пользовательского интерфейса в декларативной XAML-разметке, а затем отделить определение пользовательского интерфейса от логики времени выполнения, используя файлы кода программной части, присоединенные к разметке с помощью определений разделяемых классов. Язык XAML напрямую представляет создание экземпляров объектов в конкретном наборе резервных типов, определенных в сборках.

Microsoft SQL Server — система управления реляционными базами данных (РСУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов — Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL 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; конкурирует с другими СУБД в этом сегменте рынка.

SQL Server — это основа платформы обработки данных Майкрософт, которая предоставляет надежную и устойчивую производительность (в том числе благодаря технологиям обработки данных в памяти) и помогает быстрее извлечь ценную информацию из любых данных, расположенных как в локальной среде, так и в облаке.

Программный интерфейс приложения Bing Maps позволяет встроить приложение Bing Maps в приложение WPF.

MVVM — шаблон проектирования архитектуры приложения, который позволяет отделить логику приложения от визуальной части, что упрощает тестирование и поддержку приложения.

* 1. **. Описание функциональности программного средства**

На основе анализа были составлены следующие функциональные требования для программного средства для клиента:

* Поиск запчастей по модели автомобиля;
* Фильтрация запчастей по категориям;
* Возможность заказа товара;
* Просмотр статуса заказа;
* Просмотр магазинов.

Для администратора:

* Обработка заказов;
* Добавление и редактирование товаров;
* Оповещение клиента о выполнении заказа.

Далее приведена схема Use-case, описывающая функциональность программного средства для каждого из пользователей (рис 2.1):

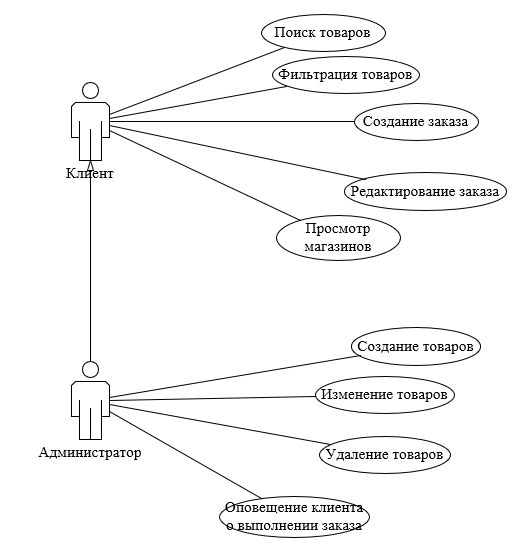


Рисунок 2.2 – Диаграмма деятельностей

Из данной схемы видно, что клиент может выполнять операции по поиску и фильтрации товаров, по созданию и редактированию заказов и просмотру магазинов на карте, а администратор в свою очередь наследует все функции клиента, а также может создавать, изменять и удалять товары в каталоге, а также оповещать клиента о выполнении заказа.

Диаграмма деятельностей пользователей указана на рисунке 2.2

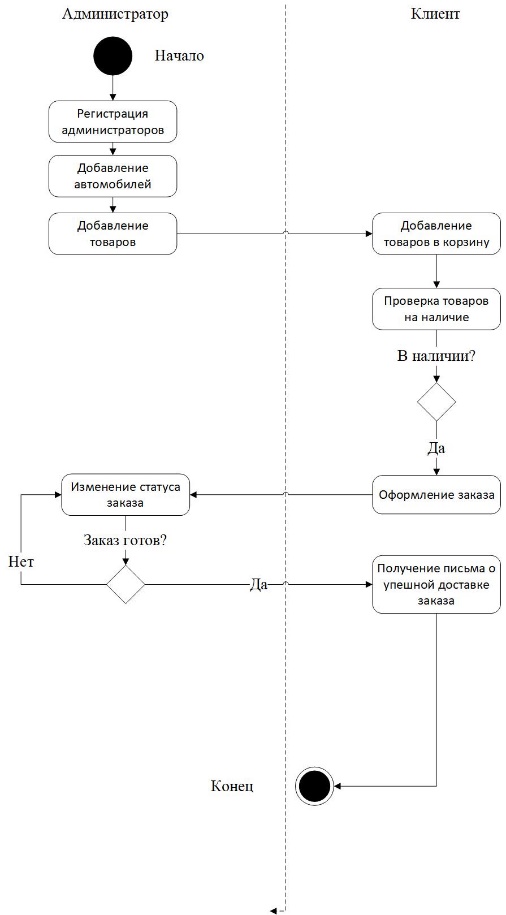


Рисунок 2.2 – Диаграмма деятельностей

1. **Проектирование программного средства**
   1. **Модель базы данных**

Для создания программного средства была разработана база данных Shop, состоящая из 6 таблиц. Описания таблиц базы данных представлены в таблицах 3.1 – 3.6:

Таблица 3.1 – таблица Addresses

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| AddressID | int | Содержит номер адреса |
| City | nvarchar(max) | Содержит название города |
| Street | nvarchar(max) | Содержит название улицы |
| House\_Num | int | Содержит номер дома |
| Flat\_Num | int | Содержит номер квартиры |

Таблица 3.2 – таблица Car

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| CarID | int | Содержит номер машины |
| Manufacturer | nvarchar(max) | Содержит название марки машины |
| Model | nvarchar(max) | Содержит название модели машины |
| Year | int | Содержит год выпуска машины |

Таблица 3.3 – таблица Clients

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| Login | Nvarchar(450) | Содержит логин пользователя |
| ClientID | int | Содержит номер пользователя |
| Name | nvarchar(max) | Содержит ФИО пользователя |
| Phone\_Num | nvarchar(max) | Содержит номер телефона пользователя |
| AddressID | int | Содержит номер адреса пользователя |
| Email | nvarchar(max) | Содержит Email пользователя |
| PasswordHash | nvarchar(max) | Содержит хешированный пароль пользователя |

Продолжение таблицы 3.3

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| IsAdmin | bit | Опреледяет, является ли пользователь администратором |

Таблица 3.4 – таблица Orders

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| OrderID | int | Содержит номер заказа |
| ClientID | int | Содержит номер пользователя, совершившего заказ |
| AddressID | int | Содержит номер адреса пользователя, совершившего заказ |
| Price | decimal(18, 2) | Содержит стоимость заказа |
| Status | bit | Содержит статус заказа |
| OrderDate | Datetime2(7) | Содержит дату и время совершения заказа |
| ClientLogin | Nvarchar(450) | Содержит логин пользователя, совершившего заказ |

Таблица 3.5 – таблица Parts

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| PartID | int | Содержит номер детали |
| CarID | int | Содержит номер машины, которой подходит данная деталь |
| Name | nvarchar(max) | Содержит название товара |
| Type | nvarchar(max) | Содержит тип товара |
| Availability | bit | Содержит статус товара |
| Image | nvarchar(max) | Содержит картинку товара |
| Amount | int | Содержит количество товара на складе |

Таблица 3.6 – таблица Parts

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| PartID | int | Содержит номер заказанной детали |
| OrderID | int | Содержит номер заказа |

Продолжение таблицы 3.6

|  |  |  |
| --- | --- | --- |
| Имя столбца | Тип данных | Описание |
| Amount | int | Содержит количество заказанного товара |

Диаграмма базы данных, отображающая связи между таблицами, представлена на рисунке 3.1

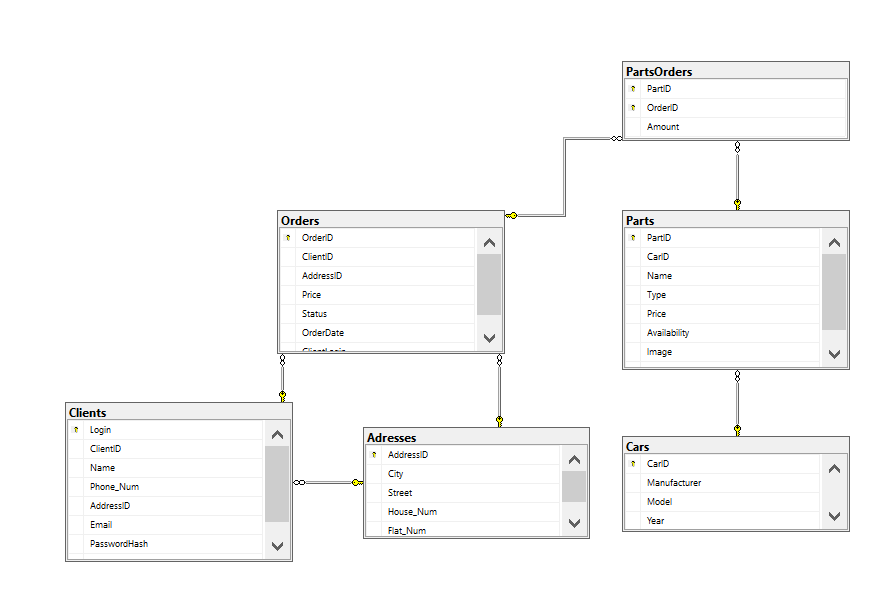


Рисунок 3.1 – Диаграмма базы данных

* 1. **Структура проекта**

Программное средство выполнено одним проектом, структура которого показана на рисунке 3.2

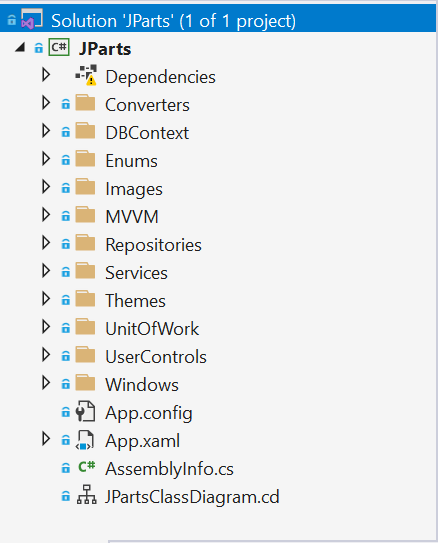


Рисунок 3.2 – Структура проекта

Таблица 3.7 – Описание структуры проекта

|  |  |
| --- | --- |
| Элемент | Описание |
| Dependencies | Содержит зависимости проекта от других сборок и библиотек |
| Папка «Converters» | Содержит конвертеры данных, применяемые при биндинге данных модели представления к представлению |
| Папка «DBContext» | Содержит в себе контекст данных JPartsContext, необходимый для работы с базой данных |
| Папка «Enums» | Содержит перечисления |
| Папка «Images» | Содержит картинки используемые в программе |
| Папка «MVVM» | Содержит папки, в которых хранятся классы, представляющие шаблон проектирования архитектуры приложения, а именно модели, модели представления и представления |
| Папка «Repositories» | Содержит интерфейсы и классы, реализующие шаблон базы данных Repository |
| Папка «Services» | Содержит сервис авторизации пользователя в приложение |
| Папка «Themes» | Содержит визуальные настройки приложения |
| Папка «UnitOfWork» | Содержит классы, реализующие шаблон для работы с базой данных Unit Of Work |
| Папка «UserControls» | Содержит пользовательские элементы управления, использованные в проекте |
| Папка «Windows» | Содержит классы, представляющие окна программы |
| App.config | Содержит строки конфигурации приложения |
| App.xaml | Содержит стартовую точку, с которой начинается выполнение приложения |

* 1. **Структура классов программного средства**

Структура классов программного средства представлена в виде диаграммы классов.

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами. Вид и интерпретация диаграммы классов существенно зависит от точки зрения (уровня абстракции): классы могут представлять сущности предметной области (в процессе анализа) или элементы программной системы (в процессах проектирования и реализации).

Основными элементами являются классы и связи между ними. Классы характеризуются при помощи атрибутов и операций.

Атрибуты описывают свойства объектов класса. Большинство объектов в классе получают свою индивидуальность из-за различий в их атрибутах и взаимосвязи с другими объектами. Однако, возможны объекты с идентичными значениями атрибутов и взаимосвязей. Т.е. индивидуальность объектов определяется самим фактом их существования, а не различиями в их свойствах. Имя атрибута должно быть уникально в пределах класса. За именем атрибута может следовать его тип и значение по умолчанию [7].

В данном пункте будут рассматриваться только классы модели приложения без структуры модели представления.

Модель приложения представлена 6 основными классами.

Класс Car определят автомобиль, для которого будут добавляться детали, и содержит такие свойства, как номер автомобиля в базе данных, производитель, модель, год производства, а также детали, связанные с ним.

Класс Address определяет адреса для клиентов и содержит такие свойства, как город, номер дома, номер квартиры.

Класс Client определяет клиента и содержит свойства Еmail, логин, имя пользователя, список заказов пользователя, номер телефона пользователя, номер адреса пользователя в базе данных, хешированный пароль пользователя и имеет ли пользователь права администратора.

Класс Part определяет деталь, которая будет продаваться в магазине и содержит в себе такие поля, как название детали, номер машины, которой она подходит, изображение детали, ее стоимость, тип, количество в наличии, а также ссылку на класс PartsOrders, который сопоставляет заказанные детали заказам.

Класс Order определяет заказ, совершенный пользователем в магазине. Содержит поле с номером адреса из базы данных, номером клиента из базы данных, датой заказа, стоимостью, статусом заказа, а также ссылку на класс PartsOrders.

Класс CartPart содержит ссылку на класс Part и имеет свойство количества, определяющее количество товара в наличии.

Класс PartsOrders устанавливает связь между заказом и заказанными товарами.

На основании данных классов была создана база данных и ее таблицы. База данных с таблицами создавалась с помощью фреймворка Entity Framework Core.

Диаграмма классов представлена на рисунке 3.3

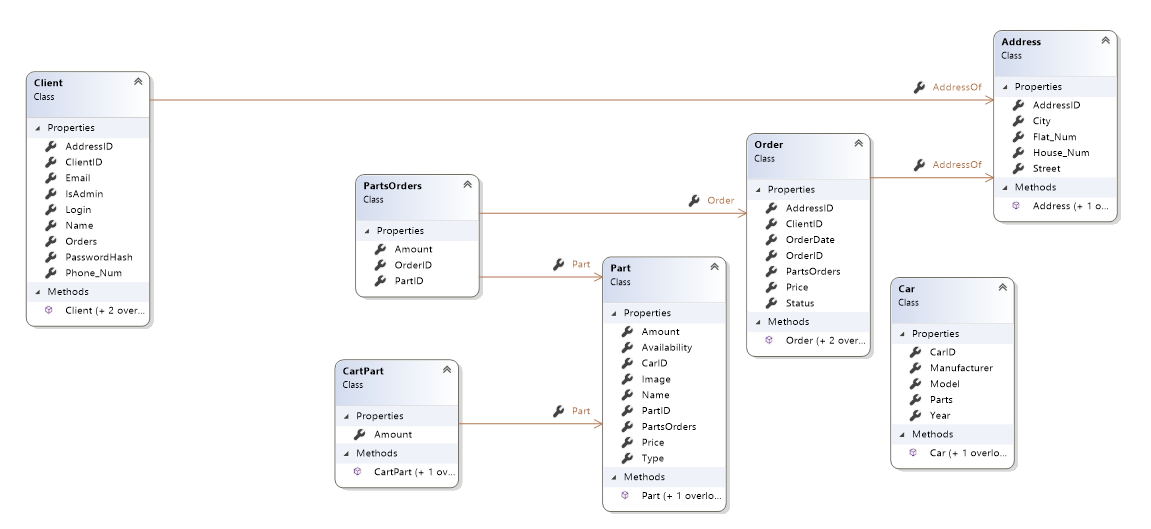


Рисунок 3.3 – Диаграмма классов ПО

* 1. **Работа с базой данных**

Работа с базой данных будет вестись с помощью фреймворка Entity Framework Core. Данный фреймворк позволил разработать базу данных с ее таблицами с помощью подхода Code-first, т. е. сначала будут созданы классы, а затем по ним будет построена база данных с таблицами в автоматическом режиме.

Взаимодействие с базой данных будет реализовано с помощью паттернов Repository и Unity Of Work.

Для каждого класса модели будет создан репозиторий, который реализует обобщенный класс Repository<T>, содержащий базовые функции работы с базой данных. Все репозитории будут объединены в Unit Of Work для удобного обращения к ним.

Диаграмма классов для работы с базой данных представлена в приложении 1.

* 1. **Аутентификация пользователей в системе**

Программное средство подразумевает использование несколькими пользователями с разными правами. Для этого необходима регистрация и аутентификация пользователей в системе. В системе может зарегистрироваться каждый, однако администраторов может регистрировать только другой администратор. Данные, введенные пользователями, должны проходить валидацию, а пароль – хешироваться для безопасности.

Для разрабатывается сервис, который будет регистрировать и аутентифицировать пользователей. Схема работы алгоритма регистрации приведена в приложении 2, а схема работы алгоритма авторизации – в приложении 3.

* 1. **Окна приложения**

Набор окон приложения доступных каждому пользователю, будет отличаться, т. к. отличаются их возможности в системе. Для этого необходимо разработать схему переходов для каждого пользователя. Запуск ПО будет начинаться с окна логина, откуда пользователь может перейти к регистрации, или залогиниться и перейти к основному меню. Далее оба пользователя могут просматривать информацию о товарах, добавлять товары в корзину, делать заказ, а администратор может также изменять статус заказов и добавлять как товары, так и автомобили. Схема администратора представлена на рисунке 3.4, а пользователя - на рисунке 3.5.

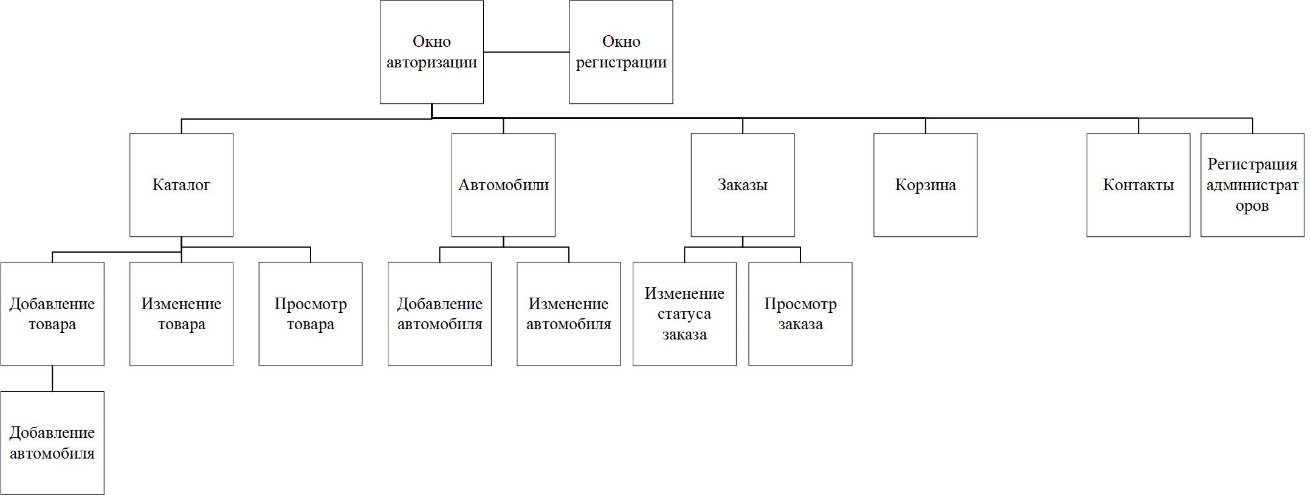


Рисунок 3.4 – Схема окон администратора

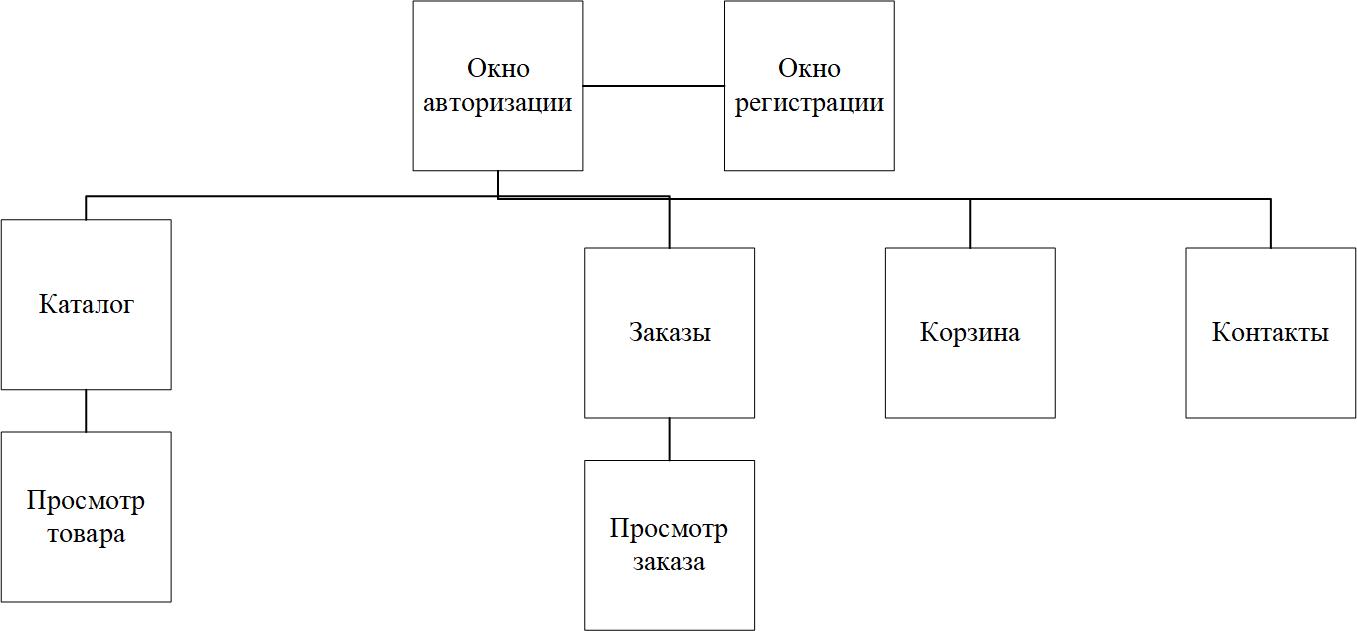


Рисунок 3.5 – Схема окон пользователя

1. **Реализация программного средства**
   1. **Реализация работы с базой данных**

В пункте 3.4 упоминалось, что работа с базой данных будет вестись с помощью фреймворка Entity Framework Core и паттернов базы данных Repository и Unit Of Work. База данных создается на основе спроектированных классов с помощью подхода Code-first.

Формирование дополнительной таблицы PartsOrders происходило с помощью Fluent API.

Данные получаются из контекста DbContext, который в свою очередь устанавливает подключение с базой данных с помощью строки подключения.

Контекст данных содержит в себе локальные таблицы, которые подключаются с помощью класса DbSet<T>.

Реализация паттернов Repository и Unit Of Work представлена в листингах 4.1 и 4.2 соответственно. Реализация класса Repository и Unit Of Work представлена в приложениях 4 и 5.

|  |
| --- |
| public interface IRepository<TEntity> where TEntity: class  {  TEntity Get(int id);  Task<TEntity> GetAsync(int id);  IEnumerable<TEntity> GetAll();  Task<IEnumerable<TEntity>> GetAllAsync();  IEnumerable<TEntity> Find(Expression<Func<TEntity, bool>> predicate);  void Add(TEntity entity);  void AddRange(IEnumerable<TEntity> entities);  void Remove(TEntity entity);  void RemoveRange(IEnumerable<TEntity> entities);  } |

Листинг 4.1 – Интерфейс IRepository

|  |
| --- |
| public interface IUnitOfWork : IDisposable  {  IAddressRepository Addresses { get; }  ICarRepository Cars { get; }  IClientRepository Clients { get; }  IOrderRepository Orders { get; }  IPartRepository Parts { get; }  IShopRepository Shops { get; }  int Complete();  Task<int> CompleteAsync();  } |

Листинг 4.2 – Интерфейс IUnitOfWork

* 1. **Реализация регистрации и аутентификации пользователей**

Для регистрации и аутентификации пользователей используется сервис AuthenticationService, который содержит в себе методы Register для регистрации пользователя по форме и Login для его аутентификации. Интерфейс IAuthenticationService приведен в листинге 4.3.

|  |
| --- |
| public interface IAuthenticationService  {  public RegistrationResult Register(string client\_ID, string name, string phone\_Num,  int? House\_Num, int? Flat\_Num, string Street, string City, string email, string login, string password, string confirmPassword, bool isAdmin);  Client Login(string username, string password);  } |

Листинг 4.3 – Интерфейс IAuthenticationService

Т. к. в проекте используется паттерн MVVM, то все операции выполняются в модели представления (Viewmodel) определенного представления (User Control или Window). Пример взаимодействия модели, модели представления и представления рассмотрим на примере регистрации.

|  |
| --- |
| public RegistrationResult Register(string client\_ID, string name, string phone\_Num,  int? House\_Num, int? Flat\_Num, string Street, string City, string email, string login, string password, string confirmPassword, bool isAdmin)  {  RegistrationResult result = RegistrationResult.Success;  if (\_unitOfWork.Clients.GetByUsername(login) != null)  {  result = RegistrationResult.LoginAlreadyExists;  }  else if (password == confirmPassword && result == RegistrationResult.Success)  {  Address addr = new Address(City, Street, House\_Num, Flat\_Num);  \_unitOfWork.Addresses.Add(addr);  \_unitOfWork.Complete();  string hashedPassword = \_passwordHasher.HashPassword(password);  Client client = new Client(name, phone\_Num, \_unitOfWork.Addresses.GetAddressByObj(addr).AddressID, email, login, hashedPassword, isAdmin);  \_unitOfWork.Clients.Add(client);  \_unitOfWork.Complete();  }  else if(password != confirmPassword)  {  result = RegistrationResult.PasswordDoNotMatch;  }  return result;  } |

Листинг 4.4 – Метод Register

Метод Register (листинг 4.4) регистрирует пользователя исходя из данных, введенных в форму. Здесь проверяется существование пользователя с идентичным логином и соответствие пароля и пароля подтверждения. Если пользователя с идентичным логином не существует, а пароли совпадают, то пароль хешируется с помощью хешера ASP.NET и создается новый пользователь, который записывается в базу данных с помощью описанного в пункте 4.1 Unit Of Work.

Данный метод вызывается в модели представления, которая получает данные из представления. Список свойств представлен в листинге 4.5

|  |
| --- |
| class RegisterViewModel : ViewModelBase, ICloseWindows, IDataErrorInfo  {  //Client  private string clientID;  public string ClientID { get => clientID; set { clientID = value; OnPropertyChanged(); } }  private string name;  public string Name { get => name; set { name = value; OnPropertyChanged(); } }  private string phone\_Num;  public string Phone\_Num { get => phone\_Num; set { phone\_Num = value; OnPropertyChanged(); } }  private string addressID;  public string AddressID { get => addressID; set { addressID = value; OnPropertyChanged(); } }  private string email;  public string Email { get => email; set { email = value; OnPropertyChanged(); } }  private string login;  public string Login { get => login; set { login = value; OnPropertyChanged(); } }  private string password;  public string Password { get => password; set { password = value; OnPropertyChanged(); } }  private string confirmPassword;  public string ConfirmPassword { get => confirmPassword; set { confirmPassword = value; OnPropertyChanged(); } }  private bool isAdmin;  public bool IsAdmin { get => isAdmin; set { isAdmin = value; OnPropertyChanged(); } } |

Листинг 4.5 – Свойства модели представления RegisterViewModel

Свойства модели представления проходят валидацию при помощи интерфейса IDataErrorInfo. Часть индексатора, который производит валидацию, для примера представлена в листинге 4.6. Полный код индексатора в приложении 6.

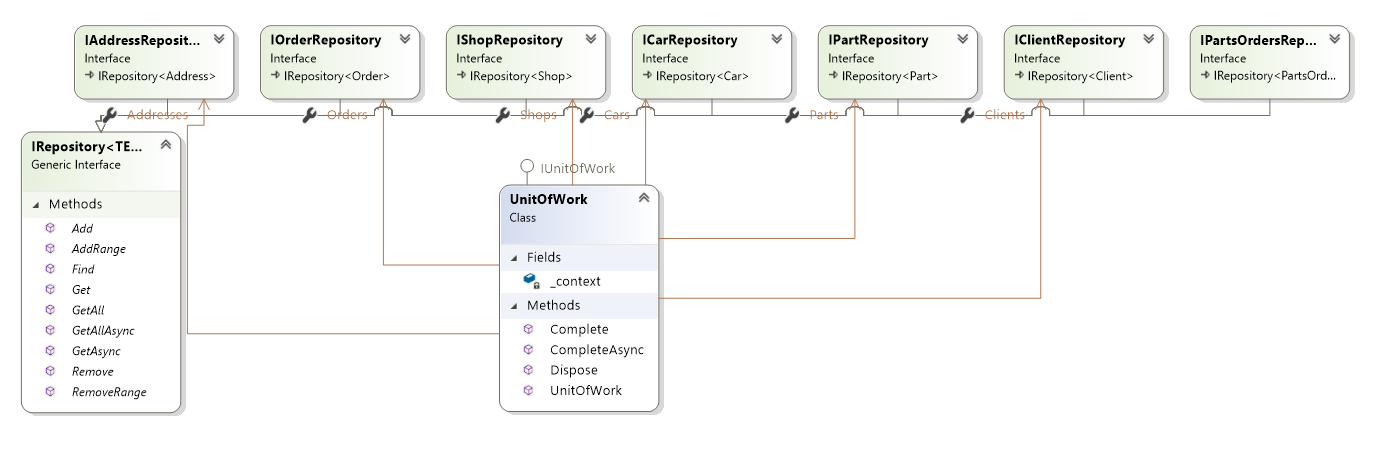
|  |
| --- |
| case "Email":  Regex eRegex = new Regex("^[\\w-\\.]+@([\\w-]+\\.)+[\\w-]{2,4}$");  if (!eRegex.IsMatch(Email))  {  error = "Введите корректный Email";  }  break; |

Листинг 4.6 – Индексатор для валидации полей

В данном примере индексатор проверяет свойство Email на соответствие требованиям электронного ящика с помощью регулярного выражения и выводит ошибку, если введен неверный Email.

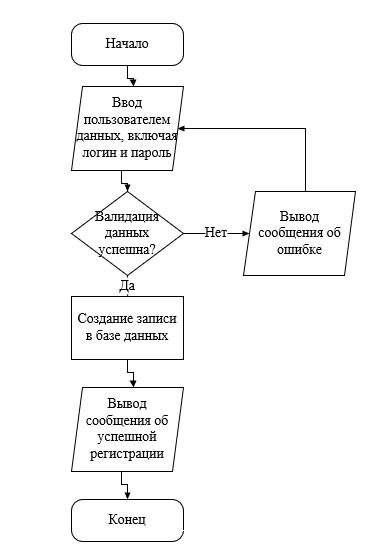
ПРИЛОЖЕНИЕ 1

UML диаграмма классов для работы с базой данных



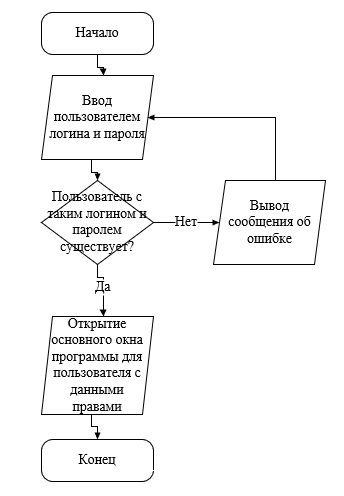
ПРИЛОЖЕНИЕ 2

Блок-схема алгоритма регистрации



ПРИЛОЖЕНИЕ 3

Блок-схема алгоритма авторизации



ПРИЛОЖЕНИЕ 4

Реализация Repository

|  |
| --- |
| public class Repository<TEntity> : IRepository<TEntity>  where TEntity : class  {  protected readonly JPartsContext Context;  public Repository(JPartsContext context)  {  Context = context;  }  public void Add(TEntity entity)  {  Context.Set<TEntity>().Add(entity);  }  public void AddRange(IEnumerable<TEntity> entities)  {  Context.Set<TEntity>().AddRange(entities);  }  public IEnumerable<TEntity> Find(Expression<Func<TEntity, bool>> predicate)  {  return Context.Set<TEntity>().Where(predicate);  }  public async Task<TEntity> GetAsync(int id)  {  return await Context.Set<TEntity>().FindAsync(id);  }  public async Task<IEnumerable<TEntity>> GetAllAsync()  {  return await Context.Set<TEntity>().ToListAsync();  }  public TEntity Get(int id)  {  return Context.Set<TEntity>().Find(id);  }  public IEnumerable<TEntity> GetAll()  {  return Context.Set<TEntity>().ToList();  }  public void Remove(TEntity entity)  {  Context.Set<TEntity>().Remove(entity);  }  public void RemoveRange(IEnumerable<TEntity> entities)  {  Context.Set<TEntity>().RemoveRange(entities);  }  } |

ПРИЛОЖЕНИЕ 5

Реализация Unit Of Work

|  |
| --- |
| public class UnitOfWork : IUnitOfWork  {  private readonly JPartsContext \_context;  public UnitOfWork(JPartsContext context)  {  \_context = context;  Addresses = new AddressRepository(\_context);  Cars = new CarRepository(\_context);  Clients = new ClientRepository(\_context);  Orders = new OrderRepository(\_context);  Parts = new PartRepository(\_context);  Shops = new ShopRepository(\_context);  }  public IAddressRepository Addresses { get; private set; }  public ICarRepository Cars { get; private set; }  public IClientRepository Clients { get; private set; }  public IOrderRepository Orders { get; private set; }  public IPartRepository Parts { get; private set; }  public IShopRepository Shops { get; private set; }  public int Complete()  {  return \_context.SaveChanges();  }  public async Task<int> CompleteAsync()  {  return await \_context.SaveChangesAsync();  }  public void Dispose()  {  \_context.Dispose();  }  } |

ПРИЛОЖЕНИЕ 6

Индексатор полей для регистрации

|  |
| --- |
| public string this[string columnName]  {  get  {  string error = String.Empty;  switch (columnName)  {  case "Phone\_Num":  Regex phRegex = new Regex("^(\\+375|80)(29|25|44|33)(\\d{3})(\\d{2})(\\d{2})$");  if (!phRegex.IsMatch(Phone\_Num))  {  error = "Введите корректный номер (прим. +375291111111)";  }  break;  case "Email":  Regex eRegex = new Regex("^[\\w-\\.]+@([\\w-]+\\.)+[\\w-]{2,4}$");  if (!eRegex.IsMatch(Email))  {  error = "Введите корректный Email";  }  break;  case "Login":  Regex lRegex = new Regex("^[A-Za-z0-9]+$");  if (!lRegex.IsMatch(Login))  {  error = "Используйте только символы латинского алфавита и цифры";  }  break;  case "Password":  Regex pRegex = new Regex("^(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\\d)[a-zA-Z\\d]{8,}$");  if (!pRegex.IsMatch(Password))  {  error = "Введите верный пароль (8 символов, из которых 2 - буквы латинского алфавита)";  }  break;  case "ConfirmPassword":  if (ConfirmPassword != Password)  {  error = "Пароли не совпадают";  }  break;  default:  error = null;  break;  }  return error;  }  } |