МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка веб-приложения для выбора моторного масла для автомобиля

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.306-01-077.ВКР

Научный руководитель,
доцент кафедры СП, к.ф.-м.н.
_____ А.Б.А. Алаасам

Автор работы,
студент группы КЭ-433
_____ А.А. Володин

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-433
Володину Антону Алексеевичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)

Разработка веб-приложения для выбора моторного масла для автомобиля.

2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.

3. **Исходные данные к работе**

3.1. Симан М., Дерсен С., Внедрение зависимостей на платформе .NET. – 2 изд. – СПб.: Питер, 2021. – 608 с.

3.2. Руководство по ASP.NET Core 8 | METANIT.COM. [Электронный ресурс] URL: https://metanit.com/sharp/aspnet6/ (дата обращения: 14.02.2024 г.).

3.3. Полное руководство по языку программирования C# 12 и платформе .NET 8 | METANIT.COM. [Электронный ресурс] URL: https://metanit.com/sharp/tutorial/ (дата обращения: 14.02.2024 г.).

4. **Перечень подлежащих разработке вопросов**

4.1. Проведение обзора аналогичных приложений.

4.2. Проектирование архитектуры приложения и требуемой структуры базы данных.

4.3. Проектирование пользовательского интерфейса.

4.4. Разработка серверной части.

4.5. Тестирование приложения, выявление возможных проблем и их решение.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.ф.-м.н. А.Б.А. Алаасам

Задание принял к исполнению А.А. Володин

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

С начала 20 столетия в мире и в последние 20 лет в России число автовладельцев значительно выросло. По данным ГИБДД, в России к концу 2022 года насчитывалось около 60 миллионов автомобилей [1]. Для хорошей работы двигателя автомобиля требуется использование моторного масла. Поскольку автомобилистов – много, значит, многим из них требуется обслуживание автомобиля. Поскольку моторное масло рекомендуется менять примерно каждые 3–6 месяцев [2] – перед автомобилистами часто встает вопрос о том, какое моторное масло им следует выбрать. Как минимум, этот вопрос встает перед ними при покупке нового автомобиля, как максимум – каждые 3–6 месяцев при активном использовании автомобиля. Формально для того, чтобы масло считалось подходящим для автомобиля, оно должно иметь необходимый, установленный производителем автомобиля, допуск. Однако поскольку стоимость получения допуска для производителя масла может быть высокой, не все масла, фактически подходящие для автомобиля, обладают нужным допуском. Масла, обладающие нужным допуском, могут стоить дороже масел, не обладающих нужным допуском и при этом также подходящих для автомобиля. Поэтому, как правило, при выборе моторных масел покупатель подбирает масло, основываясь на его классе качества и вязкости [3]. Чтобы покупателю не приходилось при подборе масла тратить время на чтение документации о своем автомобиле и переплачивать за масла с допусками, целесообразно разработать систему по подбору масла на основании модели автомобиля.

Постановка задачи

Целью выпускной квалификационной работы является разработка веб-приложения для подбора моторного масла для автомобиля. Для достижения поставленной цели необходимо решить следующие задачи:

1) произвести обзор аналогичных приложений;

2) спроектировать архитектуру приложения и требуемую структуру базы данных;

3) спроектировать пользовательский интерфейс;

4) разработать серверную часть;

5) протестировать приложение, выявить возможные проблемы и их решение.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 52 страницы с учетом приложений, без учета приложений – 43, объем списка литературы – 25 источников.

В первой главе «Анализ предметной области» рассмотрены технологии для реализации веб-приложения, веб-сервисы по подбору товаров.

Во второй главе «Проектирование системы» описываются требования и архитектура приложения.

В третьей главе «Реализация приложения» представлены детали и технические подробности реализации приложения.

В четвертой главе «Тестирование» было проведено тестирование приложения.

В заключении приведены основные результаты работы.

В листингах приложения А приведены листинги кода некоторых компонентов приложения.

В приложении Б приведен скриншот окна редактирования условий подбора масел.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Технологии реализации веб-приложений

Django – бесплатный и свободный бэкэнд-фреймворк для веб-приложений, использующий Python [4]. Django имеет встроенную защиту от SQL-инъекций и хорошо подходит для работы с базами данных за счет ORM-технологии. Из достоинств Django можно выделить такие особенности, как создание условий для соблюдения принципа DRY (don’t repeat yourself – не повторяй себя), расширяемая система шаблонов, встроенный интерфейс администратора, интернационализация. Поскольку Django использует Python, у разработчиков сервисов, использующих нейронные сети, например, для подбора контента, значительно облегчается задача по подключению нейронных сетей к веб-приложению. С использованием Django работают такие веб-сервисы, как Google, YouTube, Instagram, Dropbox, The Washington Post, Reddit и другие [5].

ASP.NET Core – кросс-платформенный бэкенд-фреймворк разработки веб-приложений от компании Microsoft, использующий C# [6, 7]. Он предоставляет разработчикам мощные инструменты для создания высокоэффективных и масштабируемых веб-приложений. К его преимуществам можно отнести высокую производительность, кросс-платформенность, встроенную поддержку различных форматов данных для обмена данными с другими приложениями, наличие встроенных механизмов безопасности, интеграцию с современными фронтэнд-фреймворками. С помощью ASP.NET были написаны веб-сайты для компаний Microsoft, DELL, написано веб-приложение Stack Overflow [8].

Laravel – это бесплатный PHP-фреймворк с открытым исходным кодом, специально разработанный для создания сложных сайтов и веб-приложений [9]. Позволяет упростить аутентификацию, маршрутизацию, сессии, кэширование, архитектуру приложения, работу с базой данных за счет ORM. В нем также предусмотрены встроенные механизмы защиты от SQL-инъекций и XSS-атак.

Angular – фронтэнд-фреймворк от компании Google для создания клиентских веб-приложений [10]. Прежде всего он нацелен на разработку SPA-решений (Single Page Application), то есть одностраничных приложений. В этом плане Angular является наследником другого фреймворка AngularJS. В то же время Angular это не новая версия AngularJS, а принципиально новый фреймворк.

Angular предоставляет такую функциональность, как двустороннее связывание, позволяющее динамически изменять данные в одном месте интерфейса при изменении данных модели в другом, шаблоны, маршрутизация и так далее.

Одной из ключевых особенностей Angular является то, что он использует в качестве языка программирования TypeScript.

Vue – это прогрессивный фреймворк для создания пользовательских интерфейсов [11]. В отличие от фреймворков-монолитов, Vue создан пригодным для постепенного внедрения. Его ядро в первую очередь решает задачи уровня представления (view), что упрощает интеграцию с другими библиотеками и существующими проектами. С другой стороны, Vue полностью подходит и для создания сложных одностраничных приложений (SPA, Single-Page Applications), если использовать его совместно с современными инструментами и дополнительными библиотеками.

Решение, предоставляемое Microsoft, было выбрано из-за его высокой производительности, кроссплатформенных возможностей и встроенных функций безопасности. Оно также легко интегрируется с современными фронтенд-фреймворками и поддерживает различные форматы данных, что упрощает разработку и взаимодействие с другими системами.

С точки зрения используемой базы данных, существует множество доступных систем управления базами данных (СУБД), каждая из которых имеет свои особенности, преимущества и области применения. Например, MySQL – свободная реляционная система управления базами данных (СУБД). Под словом «свободная» подразумевается ее бесплатность, под

«реляционная» – работа с базами данных, основанных на двумерных таблицах. Система выпущена в 1995 году, ее разработка активно продолжается.

У MySQL есть ряд преимуществ:

1) высокая скорость работы;
2) поддержка практически всех CMS;
3) бесплатная лицензия;
4) надежная и простая система безопасности;
5) поддержка нескольких типов таблиц: MyISAM, InnoDB;
6) плагины, позволяющие упростить и настроить работу под себя;
7) в одной таблице может содержаться несколько миллионов записей [12].

Microsoft SQL Server – это реляционная система управления базами данных [13]. Данная СУБД – платная, пользоваться ей легально можно либо при использовании в качестве разработчика, либо приобретя подписку на каждый сервер с развёрнутой базой данных. Поэтому выбор данной СУБД может отрицательно сказаться на масштабируемости проекта [14].

Oracle Database – коммерческая объектно-реляционная СУБД. Термин «объектно-реляционная» означает, что СУБД позволяет реализовывать одновременно два подхода: и реляционный, и объектный [15].

В текущей работе была выбрана PostgreSQL по нескольким причинам. Во-первых, она обеспечивает высокую надежность и производительность, что важно для наших задач. Во-вторых, PostgreSQL поддерживает широкий спектр типов данных и сложные запросы, что позволяет гибко работать с данными. Наконец, ее открытый исходный код и активное сообщество разработчиков обеспечивают постоянное улучшение и поддержку системы, что делает ее идеальным выбором для нашего проекта [16].

1.2. Веб-сервисы по подбору товаров

В интернете существует немало качественных сервисов по подбору товаров, поскольку они создают удобные условия для покупателя и стимулируют его при следующей покупке рассмотреть магазин, имеющий удобный сервис подбора товаров. Ниже приведены некоторые из них.

«Citilink» – интернет-магазин техники, электроники, товаров для дома и ремонта. Для помощи клиенту с выбором товаров на сайте магазина существует система поиска и подбора товаров.

При посещении главной страницы сайта, пользователя встречает кнопка «Каталог товаров» (данная кнопка относится к кнопкам типа «Call to Action» – призыв к действию [17]), где можно выбрать интересующую покупателя категорию товаров. В центре расположены карточки с наиболее популярными категориями товаров (рисунок 1).

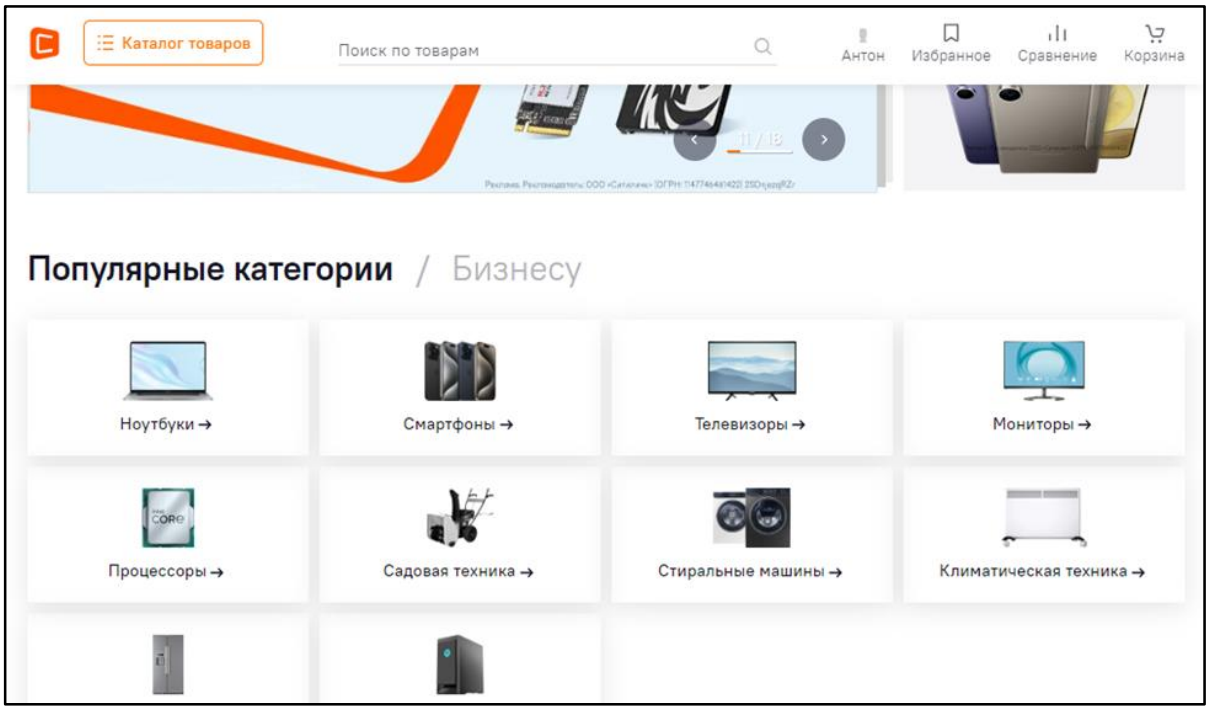

Рисунок 1 – Содержимое главной страницы сайта «Citilink»

При переходе в одну из категорий пользователю предоставляется возможность отсортировать список товаров по различным критериям (рисунок 2), а также – с помощью фильтров отображать лишь те товары, которые удовлетворяют нужным пользователю критериям (рисунок 3).

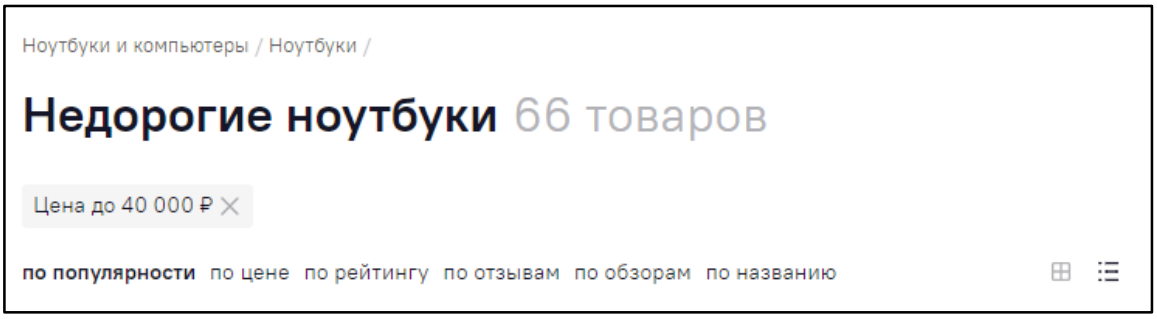

Рисунок 2 – Сортировка списка товаров «Citilink»

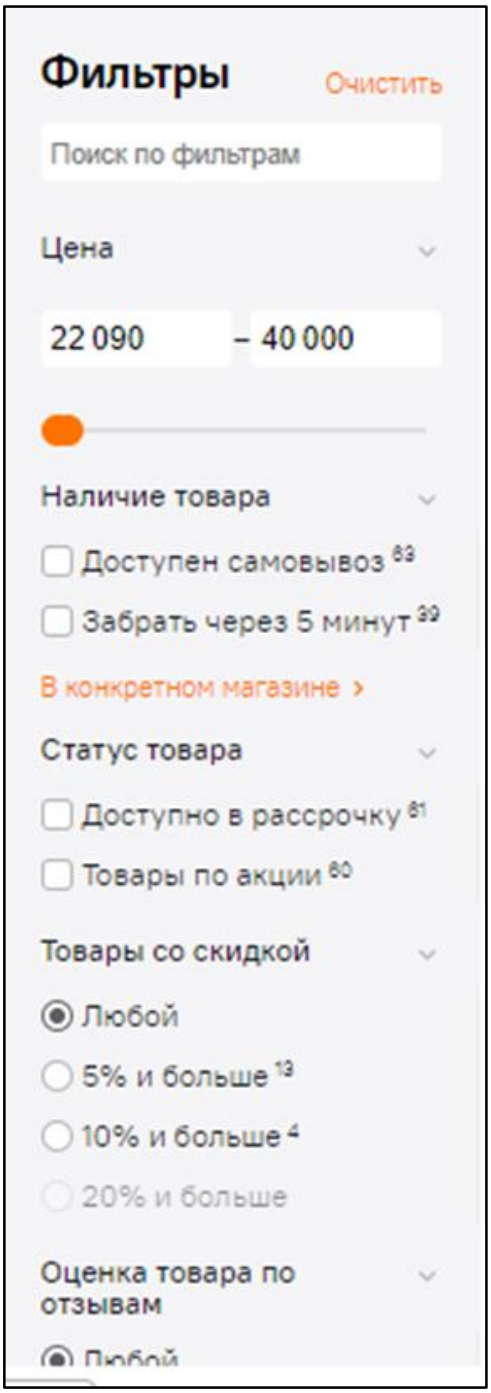

Рисунок 3 – Фильтры товаров «Citilink»

Данный сервис подбора товаров имеет минималистичный дизайн, при этом предоставляя пользователю практически всю нужную информацию и функционал для подбора товара.

«Onlinetrade» – интернет-магазин по продаже бытовой техники, электроники, товаров для дома и дачи, детских товаров, автотоваров, зоотоваров.

При посещении главной страницы сайта пользователя встречает уже не одна, а две кнопки, привлекающие внимание. Из-за наличия двух кнопок внимание пользователя рассредотачивается, и он не сразу может понять, куда ему следует перейти. Интерфейс сайта, таким образом, здесь проигрывает в интуитивности. Цветовая палитра сайта выполнена в трех цветах: синий, оранжевый и белый, что уже не делает дизайн минималистичным. Впрочем, синий и оранжевый цвета – дополнительные друг к другу, что делает сочетание этих цветов выразительным.

При нажатии на кнопку «Каталог», как видно на рисунке 4, пользователь попадает в многоуровневое меню с предложением выбрать нужную ему категорию товаров.

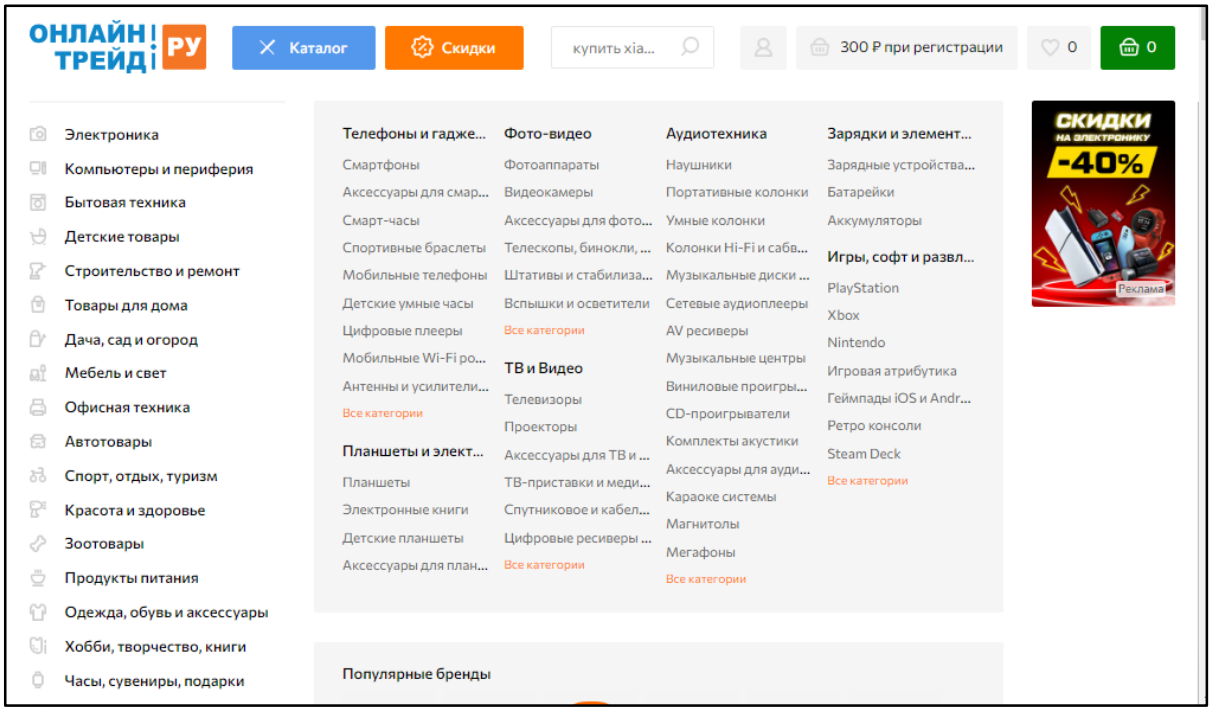

Рисунок 4 – Содержимое главной страницы после нажатия «Каталог»

При выборе категории пользователю отображается список из карточек с товарами. Список можно сортировать и фильтровать по нужным критериям. При наведении курсора мыши на товар начинают прокручиваться картинки, относящиеся к товару, но это выглядит лишним, т.к. усложняет интерфейс. Вид списка товаров представлен на рисунке 5.

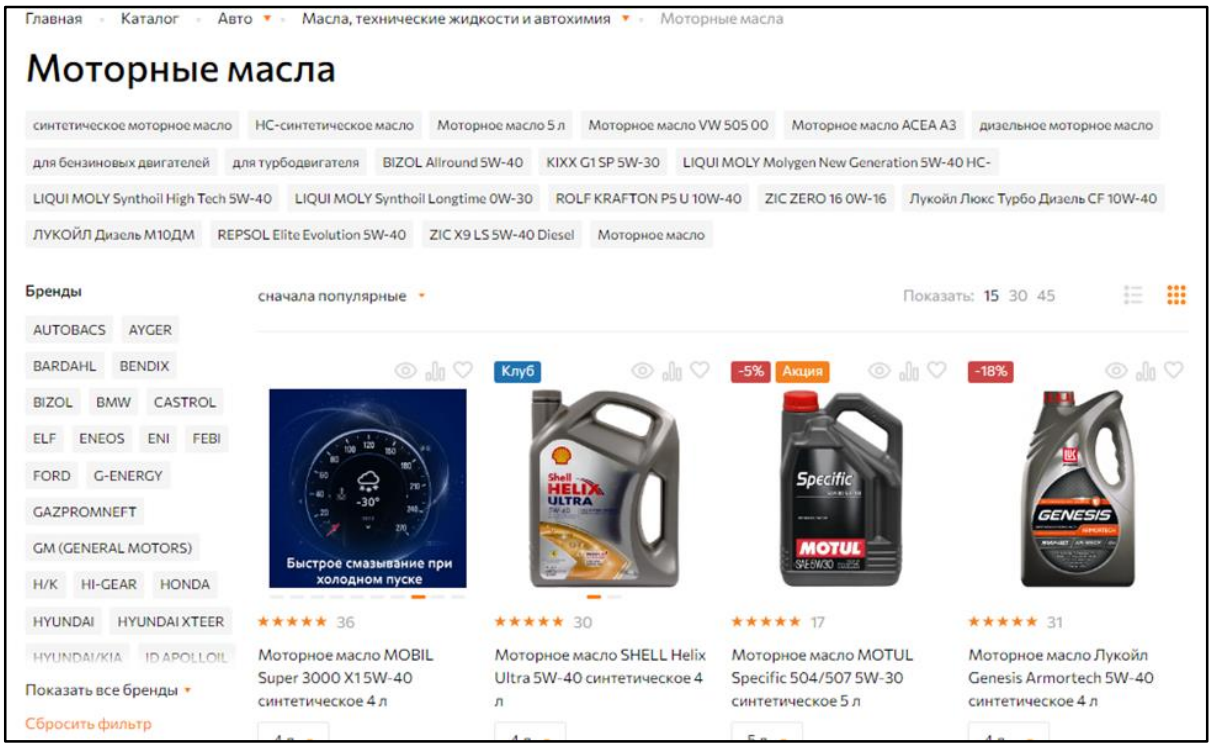

Рисунок 5 – Содержимое страницы просмотра списка товаров «Onlinetrade»

«Масломаркет» – компания, владеющая интернет-магазином с подбором товаров. На веб-сайте используется цветовая гамма с голубым и белыми цветами, кнопка для перехода к выбору категории товаров для поиска по ним выделяется на фоне других, тем не менее – сливается с другими кнопками панели. На рисунке 6 представлена основная панель и содержимое окна для выбора товаров.

Дизайн выбора и поиска товаров минималистичен, при этом форма обладает достаточной функциональностью: есть возможность сортировать и фильтровать список товаров по различным критериям (рисунок 7).

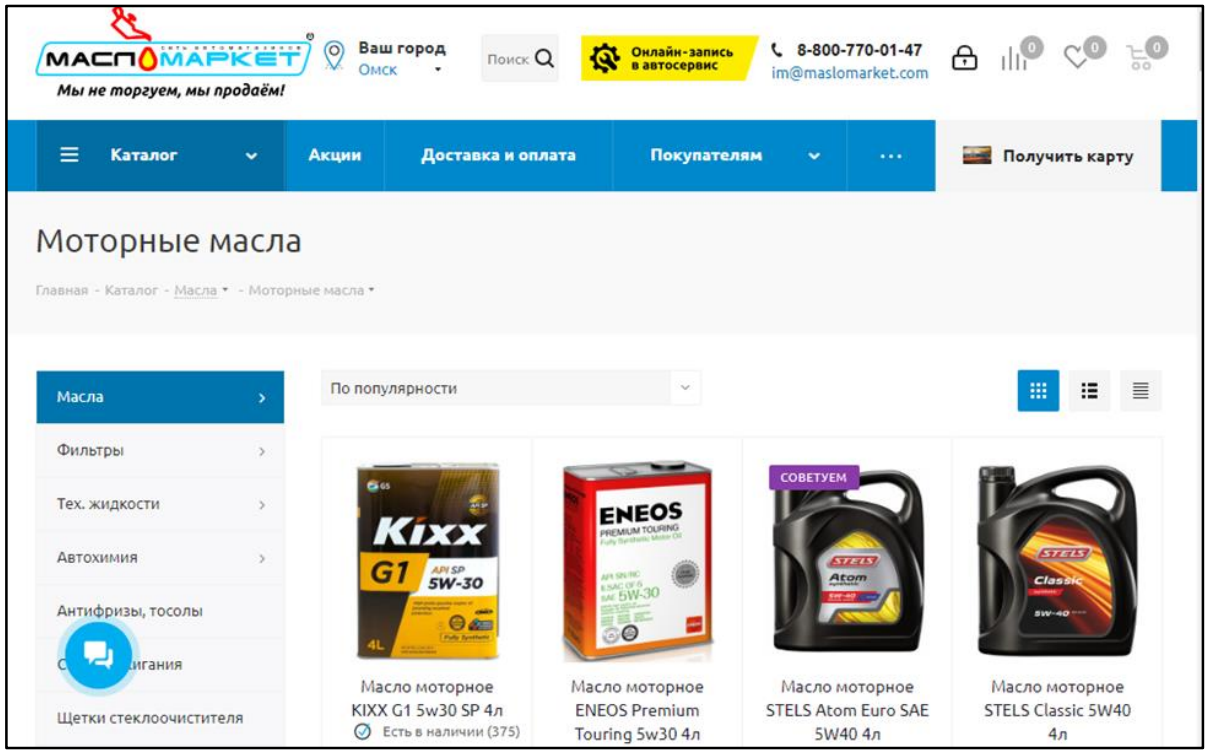

Рисунок 6 – Содержимое страницы списка просмотра товаров «Масломаркет»

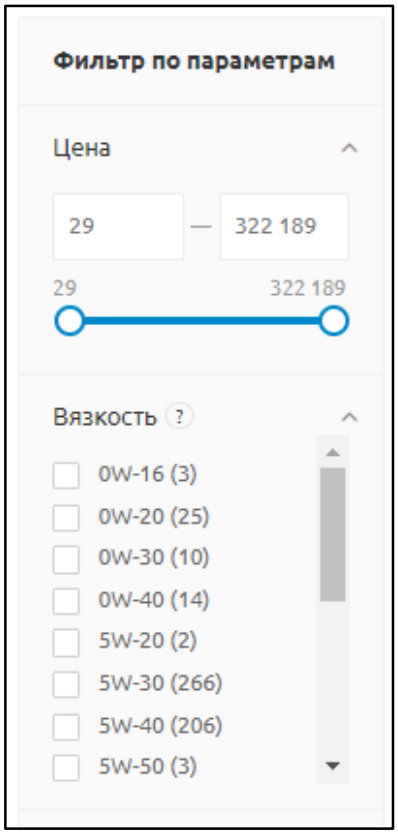

Рисунок 7 – Фильтры сайта «Масломаркет»

«Ozon» – российский маркетплейс (интернет-магазин, предоставляющий информацию о продукте или услуге третьих лиц). На главной странице сайта содержится кнопка CTA (рисунок 8), призывающая пользователя выбрать нужную ему категорию товара из каталога и начать поиск по ней. При изменении масштаба страницы содержимое раздела карточек с товарами не подстраивается под пользователя, пользователь вынужден для просмотра товаров использовать не только вертикальное, но и горизонтальное прокручивание, что создает ему неудобства.

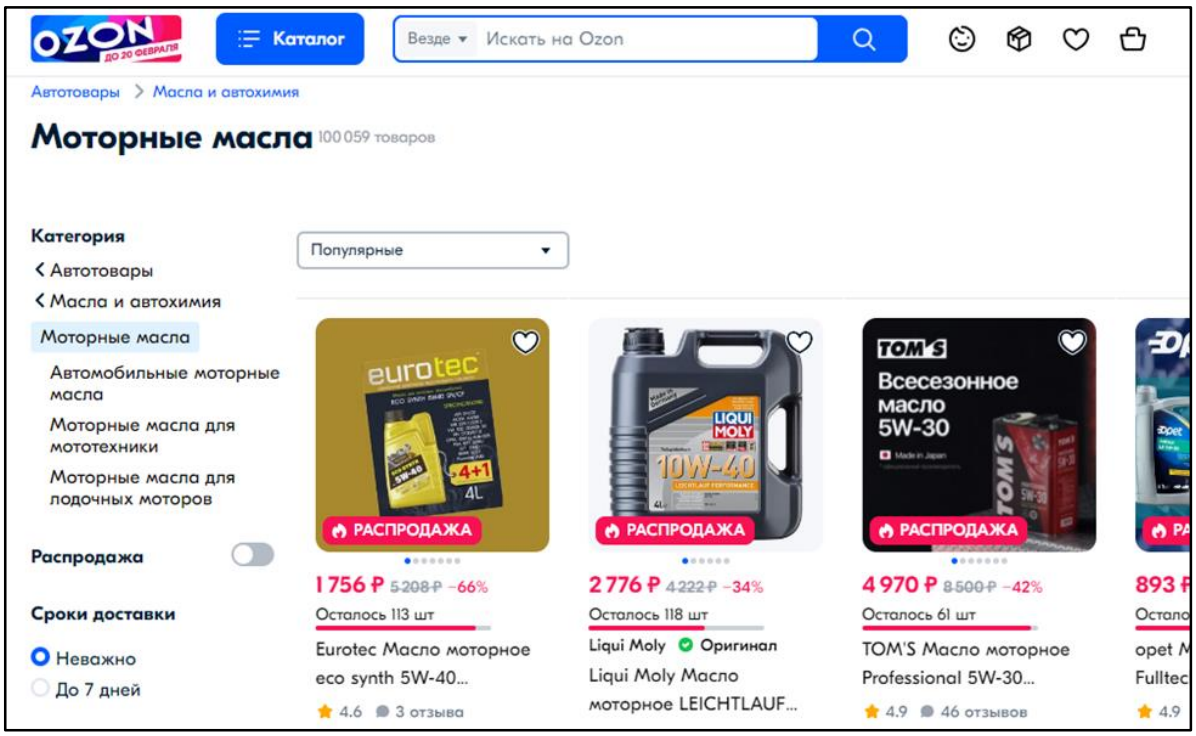

Рисунок 8 – Страница просмотра списка товаров «Ozon»

При подборе товара также есть возможность отфильтровать их и оставить только нужные, отсортировать товары по определенным критериям.

1.3. Веб-сервисы по подбору моторного масла по типу автомобиля

В интернете существуют сервисы по подбору масла по типу автомобиля, один из них представлен на рисунке 9.

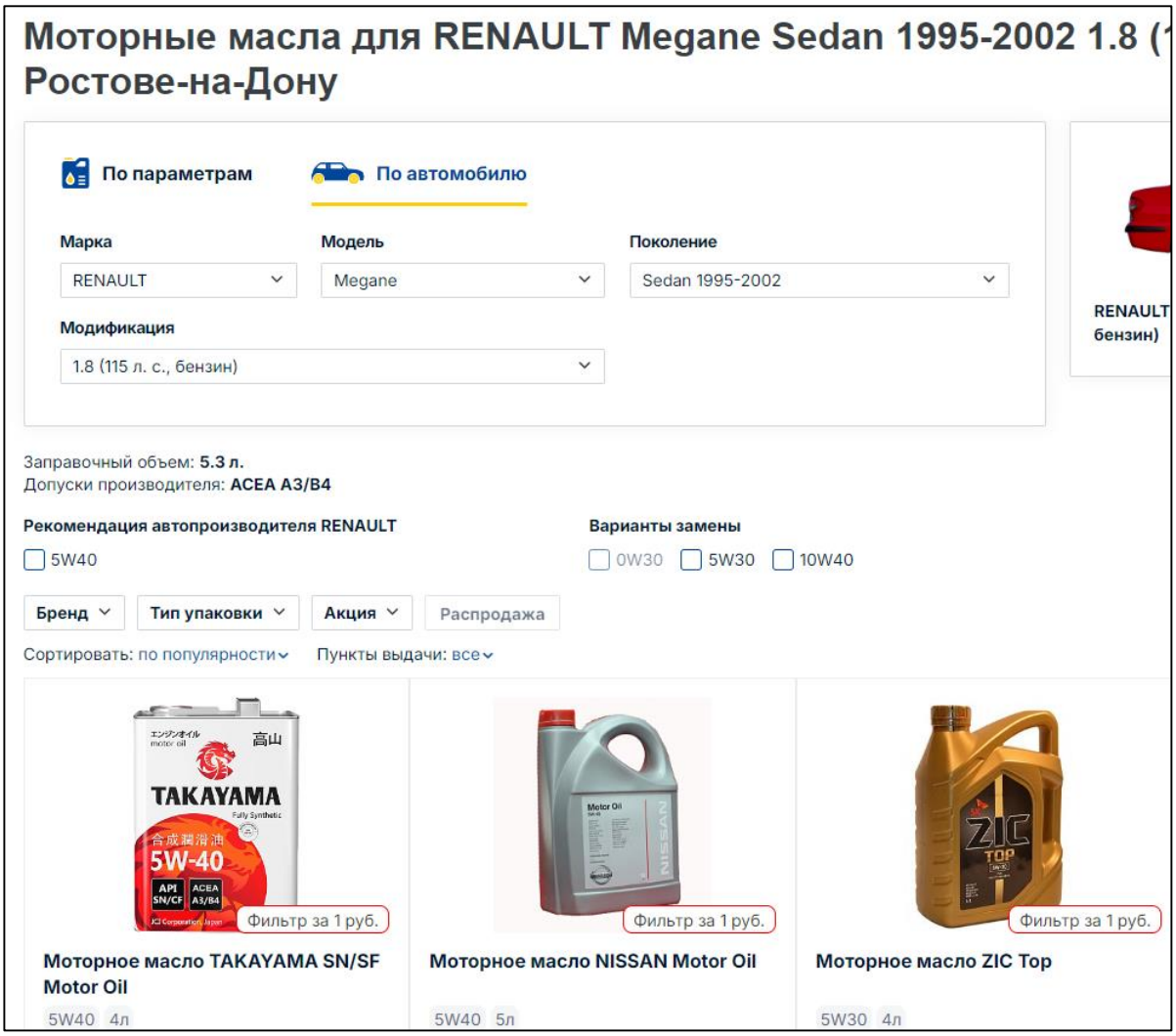

Рисунок 9 – Сервис по подбору моторного масла по типу автомобиля на сайте «ВИРБАКАвто»

Как видно на рисунке, сервис предлагает выбрать нужную модель автомобиля и нужный двигатель в несколько этапов, а затем – выводит список подходящих масел. При этом он также позволяет выбрать вязкость масла.

Сервис компании «LiquiMoly» также позволяет определить тип двигателя, для которого подбирается масло, в несколько этапов. Интерфейс сервиса этой компании по выбору автомобиля для подбора масла представлен на рисунке 10.

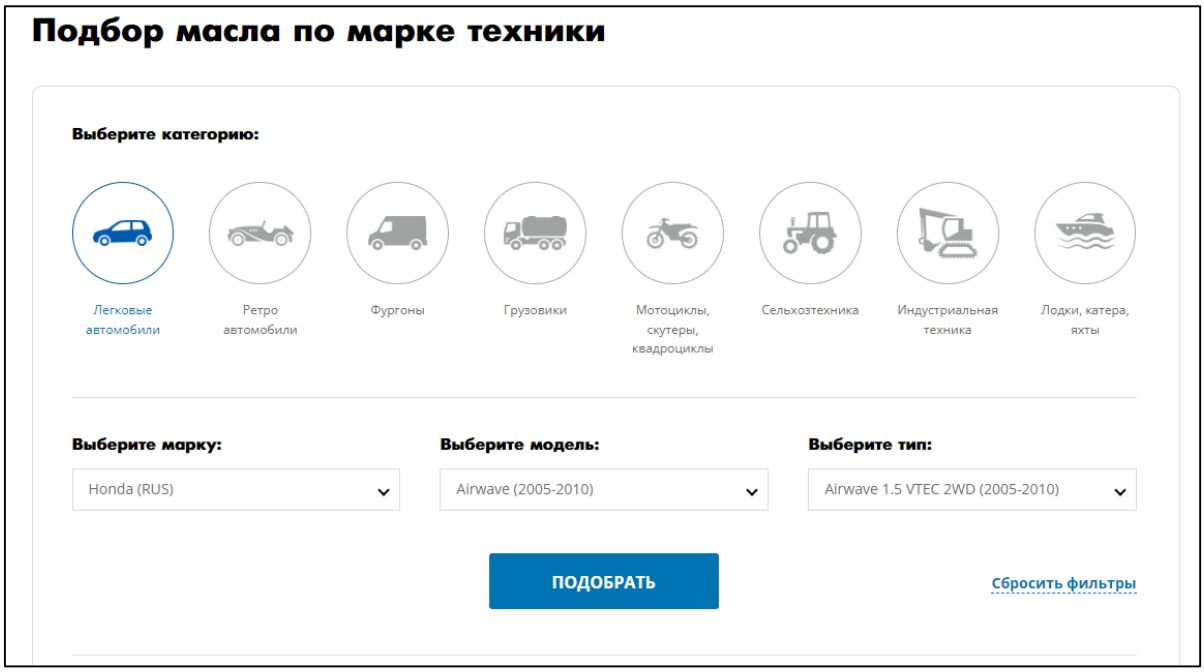

Рисунок 10 – Сервис по подбору моторного масла по типу автомобиля на сайте компании «LiquiMoly»

Как видно на рисунке, данный сервис также предлагает указать имеющийся у автовладельца автомобиль в несколько этапов. При этом у данного сервиса отсутствуют фильтры характеристик моторного масла, фильтры применяются только по типу указанного автомобиля.

Для обеих систем подбора моторного масла по типу автомобиля прослеживается закономерность: выбор типа автомобиля осуществляется в верхнем меню, список масел выводится внизу, после указания типа автомобиля.

2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

2.1. Требования к системе

Функциональные требования:

1) система должна подбирать масла под некоторые модели машин;

2) система должна подбирать масла, соответствующие заданным пользователем вязкости, классу качества;

3) система должна обеспечивать доступ администраторам для внесения изменений в базу данных;

4) система должна обеспечивать доступ владельцам компаний для внесения изменений в базу данных.

Нефункциональные требования:

1) производительность: приложение должно быть отзывчивым для выполнения функциональных требований без значительного ухудшения производительности;

2) безопасность: приложение должно внедрять надежные меры безопасности для защиты данных, включая безопасную аутентификацию и сохранение данных под контролем администратора и продавца масла, каждый из которых несет свои обязанности;

3) простота: приложение должно придерживаться принципа простоты в дизайне, минимизируя количество кликов, необходимых для поиска моторного масла;

4) масштабируемость: архитектура должна поддерживаться мощной системой баз данных для будущего роста и расширения, что позволяет добавлять новые функции и увеличивать количество пользователей.

2.2. Варианты использования системы

Диаграмма вариантов использования (use case diagram) – UML-диаграмма, представляющая собой схематичное изображение вариантов использования и взаимодействия между собой ролей с точки зрения ролей [18].

На основе функциональных требований к системе была составлена диаграмма вариантов использования. Рынок моторных масел быстро меняется, поэтому целесообразно создать систему поддержки и обновления базы данных. Поскольку один и тот же товар может продаваться в разных магазинах и торговых сетях, для отсутствия путаницы и дублирования товаров можно сформировать единую базу типов масел. В связи с этим на диаграмме присутствуют два варианта использования («Внести масло в базу данных», «Внести товар в систему»), связанные с одним и тем же действием: с добавлением масла.

На рисунке 11 представлена диаграмма вариантов использования.

Рисунок 11 – Диаграмма вариантов использования

На диаграмме представлены 3 актера.

1. Автовладелец – человек, желающий приобрести моторное масло для автомобиля.

2. Продавец – лицо, заинтересованное продать моторное масло автовладельцу. Продавец ответственен за формирование данных о своих магазинах, маслах, совместимости масел с автомобилями и предложений о продаже.

3. Администратор – лицо, следящее за добросовестным использованием сервиса продавцами. Поскольку продавец способен неверно или недобросовестно внести данные о масле и о совместимости масла с автомобилями, для исправного функционирования системы требуется роль, созданная для контроля за порядком в ней.

Также на диаграмме представлены следующие варианты использования.

1. «Подобрать масло по характеристикам» – автовладелец может подобрать масло, указав значения характеристик, которыми должно обладать масло.

2. «Подобрать масло по типу автомобиля» – автовладелец может подобрать масло, подходящее для конкретного автомобиля, указав его тип.

3. «Соотнести масла с автомобилями» – администратор может задать характеристики, которыми должны обладать масла для того, чтобы они подходили для автомобиля. Администратор может также вручную добавлять или удалять масла из списка подходящих к данному автомобилю.

4. «Управлять данными системы» – администратор может управлять информацией о маслах, предложениях о продаже, о типах машин и их соответствии маслам, о компаниях и их точках продаж, а также – частично о пользователях. Он может менять их никнейм, логин, пароль и права доступа.

5. «Выдать права» – администратор может выдать пользователям права для доступа к системе.

6. «Удалить добавленную информацию» – продавец может удалять добавленную им информацию о маслах, предложениях о продаже, магазинах.

7. «Редактировать добавленную информацию» – продавец может изменять добавленную им информацию.

8. «Внести масло в базу данных» – продавец вносит масло в базу данных.

9. «Указать адреса своих магазинов» – продавец указывает адреса своих торговых точек.

10. «Добавить предложение о продаже» – на основании внесенного продавцом типа масла он создает предложение о продаже масла с указанием его цены и торговой точки, в котором продается масло.

2.3. Выбор архитектуры и технологий для реализации системы

При реализации использовался шаблон проектирования «MVC». «MVC (model-view-controller)» – это шаблон проектирования, предполагающий разделение кода на модели, представления и контроллеры. Модели – это классы, использующиеся в приложении. Представления – это описания графического интерфейса пользователя. Контроллеры – это связующие звенья между моделями и представлениями, отвечающие за отправку нужных ответов – комбинаций представлений и моделей, на запросы пользователя. Применение шаблона «MVC» вносит порядок в код, упрощая его понимание и облегчая процесс разработки приложения, а также – позволяет грамотно разделить логику приложения.

Контроллер «HomeController» будет предназначен исключительно для чтения данных, обеспечения подбора масел. Контроллер «EditController» предназначен для внесения администраторами и владельцами компаний изменений в базу данных системы.

Для реализации приложения было принято решение использовать платформу «ASP.NET Core». Для написания функций, которые необходимо выполнить на стороне клиента, был выбран язык «JavaScript».

2.4. Планирование моделей приложения

Поскольку приложение создано для подбора моторных масел, центральная сущность приложения – моторное масло (рисунок 12). Моторное масло (а именно канистры масла, различаемые по артикулу) имеет название, компанию-производителя, характеризуется классом качества, объемом канистры, а также – вязкостью на горячем и холодном двигателе.

```
public class MotorOil : BaseEntity
{
    public string? Name { get; set; }
    public string? Producer { get; set; }
    //класс качества, по классификации API.
    public APIQualityClass? APIQualityClass { get; set; }
    public int APIQualityClassId { get; set; }
    public SAEViscosity? SAEViscosity { get; set; } //вязкость
    public decimal Volume { get; set; } //объем масла
    static private string ImgPath = "DataStorage/Images/OilPictures/";
    static private string ImgPrefix = "Img";
    static private string ImgExtension = ".jpeg";
    public string GetImgNamePath() { return "~/" + ImgPath + ImgPrefix +
id + ImgExtension; }
}
```

Рисунок 12 – Класс моторного масла

Поскольку для каждого типа масла может существовать несколько товаров, есть смысл создать модель, являющуюся маслом-товаром (рисунок 13). В модели содержится информация о том, какое именно масло продается, магазин, в котором оно продается, цена продажи и количество оставшихся в магазине единиц.

```
public class MotorOilMerch : BaseEntity
{
    public MotorOil? MotorOil { get; set; }
    public int MotorOilId { get; set; }
    public Store? Store { get; set; }
    public int StoreId { get; set; }
    public double Price { get; set; }
    public double StockCount { get; set; }
}
```

Рисунок 13 – Класс товара моторного масла

Каждый магазин создается компанией-продавцом и к ней относится. Поэтому следует ввести еще две модели – магазин и компанию-продавца (рисунки 14 и 15).

```
public class Store : BaseEntity
{
    public Company? Company { get; set; }
    public int CompanyId { get; set; }
    public string? Adress { get; set; }
    public List<MotorOilMerch>? MotorOilMerches { get; set; }
}
```

Рисунок 14 – Класс, описывающий магазин

```
public class Company : BaseEntity
{
    public string? Name { get; set; }
    public List<Store>? Stores { get; set; }
    public User? Owner { get; set; }
}
```

Рисунок 15 – Класс, описывающий компанию

Поскольку система должна подбирать масла, в ней должны присутствовать некоторые условия, которым должно удовлетворять масло для того, чтобы считаться подходящим для автомобиля. Для условий созданы сущности, описанные на рисунках 16–20.

```
public class SAEViscosityCondition: BaseCondition
{
    public int minCold { get; set; }
    public int maxCold { get; set; }
    public int minHot { get; set; }
    public int maxHot { get; set; }
    public override bool OilSuitsCondition(MotorOil oil)
    {
        if (oil.SAEViscosity.OnCold >= minCold && oil.SAEViscosity.On-
Cold <= maxCold
            && oil.SAEViscosity.OnHot >= minHot && oil.SAEViscosity.On-
Hot <= maxHot)
            return true;
        else
            return false;
    }
}
```

Рисунок 16 – Класс, описывающий критерий подбора по вязкости

На рисунке 16 изображен класс, описывающий условие подбора масла по заданной вязкости. Вязкость на холодном и горячем двигателе должна находиться в диапазоне, заданном полями вышеприведенного класса.

```
public class ConditionsSet: BaseEntity
{
    public List<APIQualityCondition> APIQualityConditions { get; set; }
    public List<OilTypeCondition> OilTypeConditions { get; set; }
    public List<SAEViscosityCondition> SAEViscosityConditions { get; set;
}
    public List<CarType> carTypes { get; set; }

    //Это не свойство, поэтому в БД оно не попадет.
    public List<BaseCondition> BaseConditions;
    public void PrepareConditionList()
    {
        BaseConditions = new List<BaseCondition>();
        BaseConditions.Concat(SAEViscosityConditions);
        BaseConditions.Concat(OilTypeConditions);
        BaseConditions.Concat(APIQualityConditions);
        BaseConditions.Sort();
    }
    public bool OilAcceptable(MotorOil oil)
    {
        return OilSuitsSpecificConds(oil, APIQualityConditions.ToArray())
            && OilSuitsSpecificConds(oil, OilTypeConditions.ToArray())
            && OilSuitsSpecificConds(oil, SAEViscosityConditions.
ToArray());
    }
    bool OilSuitsSpecificConds(MotorOil oil, BaseCondition[] condsList)
    {
        if (condsList == null || condsList.Length==0)
            return true;
        //else...
        condsList.Order();
        bool isFirst = true, suits = new bool();
        foreach (BaseCondition cond in condsList)
        {
            if (isFirst)
            {
                suits = !cond.isAllowing;
                isFirst = false;
            }

            if (cond.isAllowing && cond.OilSuitsCondition(oil))
                suits = true;
            else if (!cond.isAllowing && cond.OilSuitsCondition(oil))
                suits = false;
        }
        return suits;
    }
}
```

Рисунок 17 – Класс, описывающий набор условий подбора масла

Класс `ConditionsSet` с рисунка 17 связан с условиями и с типами автомобилей. Для проверки совместимости масел с типами автомобилей в классе предусмотрен метод `OilAcceptable`, проверяющий совместимость по всем условиям, привязанным к данному классу. Метод

`OilSuitsSpecificConds` проверяет, по какому принципу организовано множество условий: «все, что не разрешено – запрещено», или «все, что не запрещено – разрешено», после – проверяет совместимость масла с условиями и возвращает результат проверки.

```
public abstract class BaseCondition: BaseEntity, IComparable
{
    public ConditionsSet ConditionsSet { get; set; }
    public int ConditionsSetId {  get; set; }
    public int priority {  get; set; }
//Allow or Forbid? Какой тип условия: разрешительное или запретительное?
//Разрешительное(Allow): все, попадающее под условие, можно использо-
вать. иначе - запретить.
    public bool isAllowing { get; set; }

    public abstract bool OilSuitsCondition(MotorOil oil);

}
```

Рисунок 18 – Класс, описывающий базовое условие

На рисунке 18 изображен код базового условия. Так как каждое условие создано для того, чтобы проверять, подходит ли по нему масло, в базовом условии определен абстрактный метод OilSuitsCondition. Для удобства задания множества подходящих для автомобиля масел следует добавить возможность регулировать множество подходящих масел так, чтобы одно множество перекрывало другое. Но тогда условия могут противоречить друг другу, поэтому есть смысл добавить приоритет условия: чем выше приоритет – тем приоритетнее условие. Масла из множества, подходящего по более приоритетному условию, перекрывают масла из множества с менее приоритетным условием. Также в данном классе присутствует поле isAllowing, указывающее, является ли условие разрешающим, или запрещающим. Масло, удовлетворяющее разрешающему условию, подходит, удовлетворяющее запрещающему – не подходит.

На рисунке 19 показан класс, метод которого проверяет, имеется ли у масла минимально допустимый класс качества.

```
public class APIQualityCondition: BaseCondition
{
    //минимально допустимый класс качества
    public APIQualityClass MinAPIQualityClass { get; set; }
    public override bool OilSuitsCondition(MotorOil oil)
    {
        if (oil.APIQualityClass >= MinAPIQualityClass)
            return true;
        else
            return false;
    }
}
```

Рисунок 19 – Класс, описывающий критерий подбора по классу качества

```
public class OilTypeCondition: BaseCondition
{
    public MotorOil MotorOil { get; set; }

    public override bool OilSuitsCondition(MotorOil oil)
    {
        return oil == MotorOil ? true : false;
    }
}
```

Рисунок 20 – Класс, сопоставляющий конкретное масло конкретному условию

На рисунке 20 показан класс, для которого, чтобы масло подходило по условию, оно должно быть указано напрямую.

Все модели наследуются от `BaseEntity`, т.к. `BaseEntity` содержит стандартную информацию, свойственную всем моделям приложения – их первичный ключ. Все условия – наследуются от `BaseCondition`. Обозначенные условия позволяют задать диапазон допустимой вязкости масла на холодном и горячем двигателе, минимальный класс качества и список масел, которые для автомобиля точно не подойдут.

2.5. База данных веб-приложения

На основании моделей, описанных в разделе 2.4, была составлена упрощенная схема базы данных, представленная на рисунке 21.

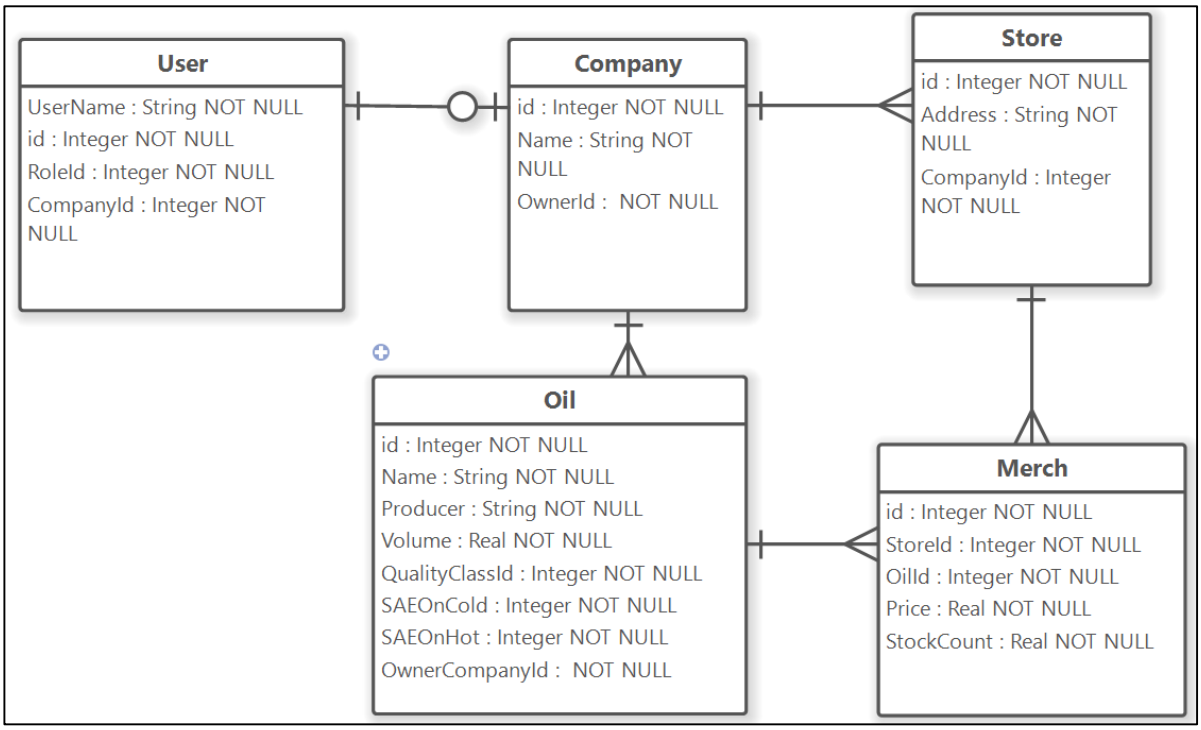

Рисунок 21 – Отношения в базе данных между основными сущностями

Помимо сущностей, описанных выше, на схеме присутствует такая сущность, как «User» (пользователь). На рисунке 22 представлены отношения сущностей, помогающих подобрать моторное масло основываясь на модели автомобиля.

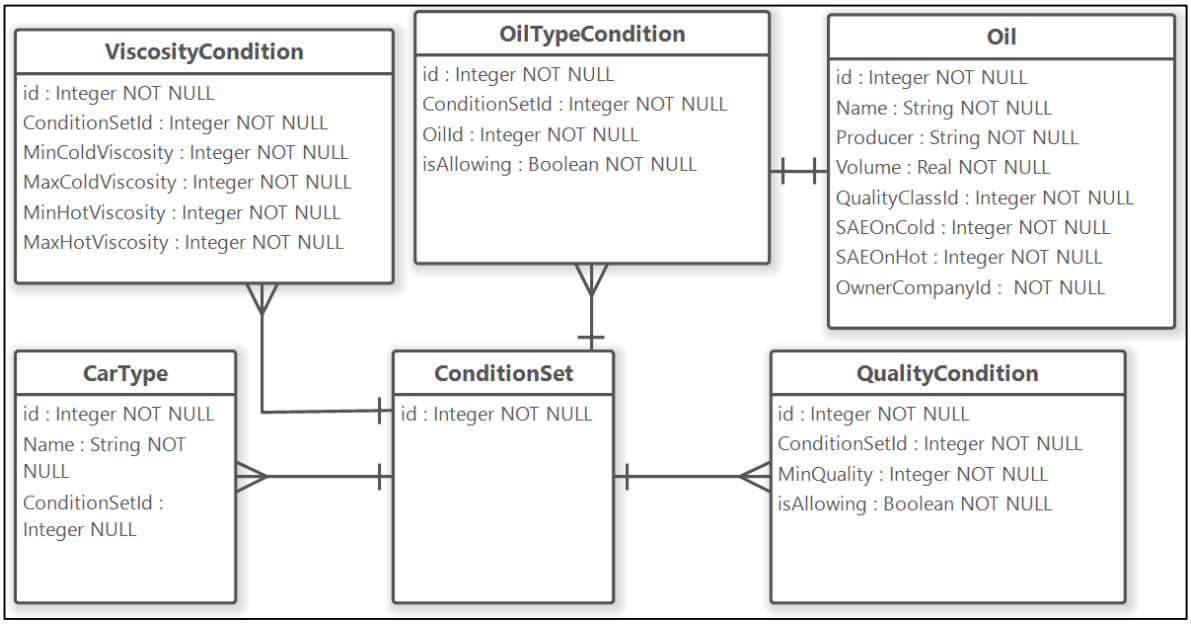

Рисунок 22 – Отношения между сущностями условий выбора масла по типу автомобиля

Не были показаны такие сущности, как «Role» и «Permission», в приложении играющая роль матрицы доступа, модуля, контролирующего доступ пользователя к различным функциям приложения.

Для каждого автомобиля задается набор критериев, по которым определяется допустимость использования масла для заданного автомобиля.

2.6. Рабочие процессы системы

Диаграммы рабочих процессов – это полезные визуальные инструменты, которые компании используют для упрощения и отображения взаимосвязей рабочих процессов. Они похожи на блок-схемы своей визуальной концепцией и служат для улучшения понимания процессов, важных для бизнеса [19].

Диаграмма рабочих процессов, связанных с обеспечением доступа администраторов и продавцов к изменению данных системы, представлена на рисунке 23.

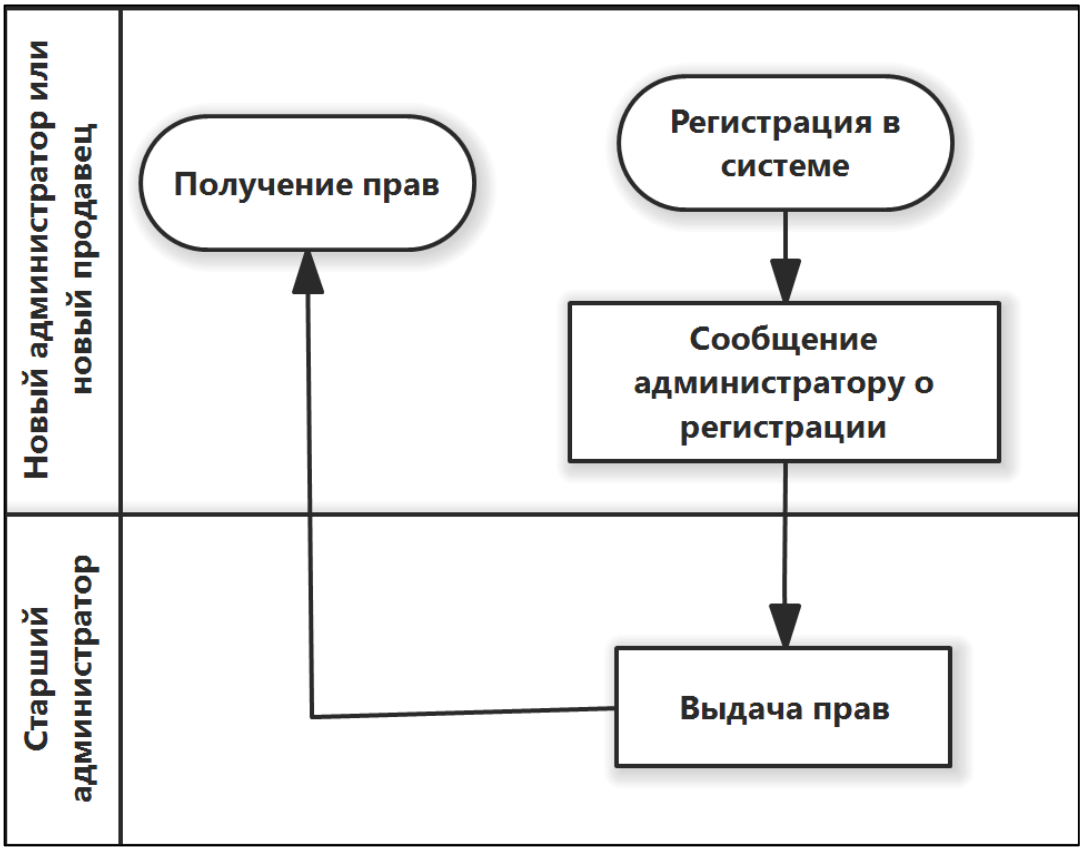

Рисунок 23 – Диаграмма процессов контроля за доступом к системе

Для получения роли, администратор или продавец регистрируются в системе и просят администратора о выдаче им роли. Роль открывает пользователю (продавцу) доступ к изменению данных системы. Администраторы следят за добросовестным использованием системы продавцами, и если они обнаружат недобросовестное использование продавцами системы, они смогут лишить продавца прав на изменение и доступ к данным.

На рисунке 24 представлена диаграмма основных процессов работы продавца и администратора с данными.

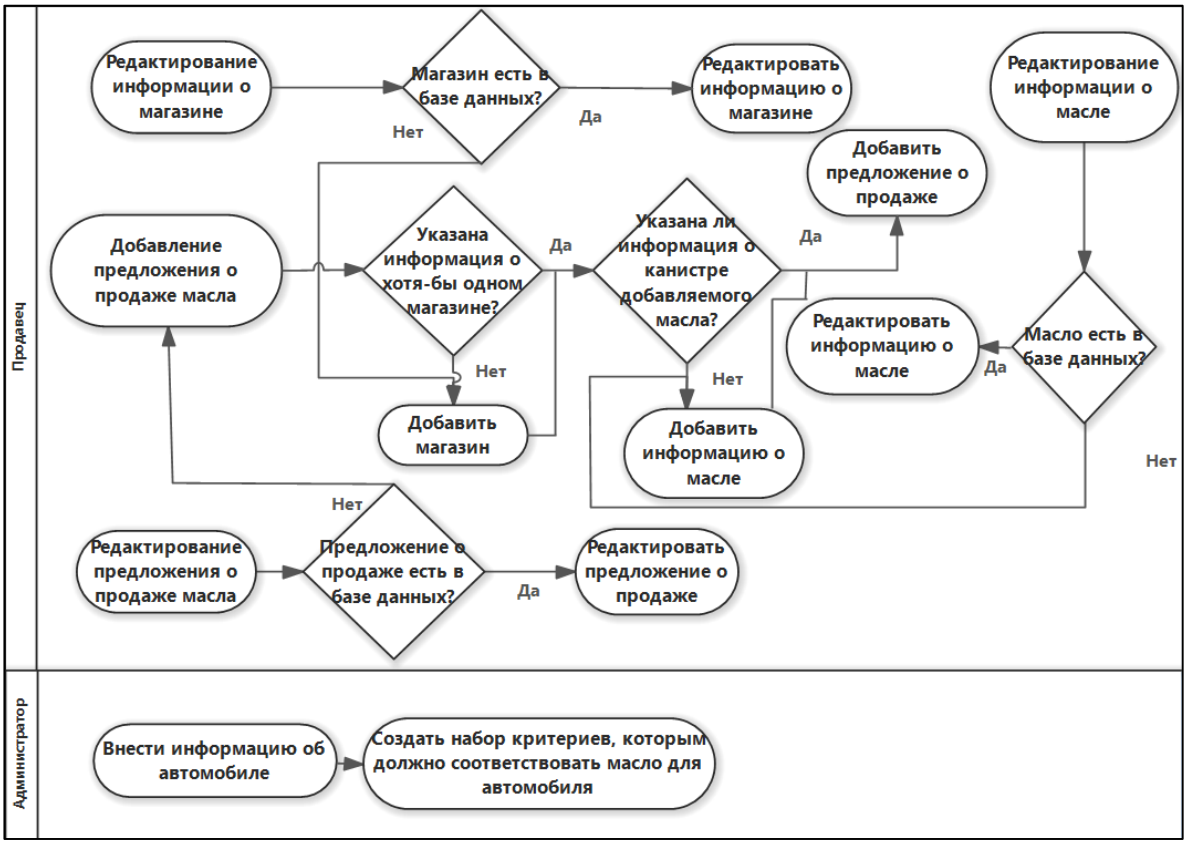

Рисунок 24 – Диаграмма процессов работы с данными

Продавцы вносят в систему информацию о своих маслах и о предложениях о продаже масел в своих торговых точках. Следует отметить, что добавление масла в систему происходит в 2 этапа: сначала добавляется информация о канистре масла, затем – предложение о его продаже. Предложения о продаже отображаются на главной странице пользователя.

Администратор вносит информацию об автомобилях, о совместимости масел с автомобилями. Администратор также может изменять всю информацию, доступную продавцу, может изменять данные пользователей, данные компаний. Однако процессы работы с этими данными не отображены на диаграмме, так как они неосновные и будут использоваться лишь в исключительных случаях.

На рисунке 25 продемонстрирован процесс подбора моторного масла автовладельцем.

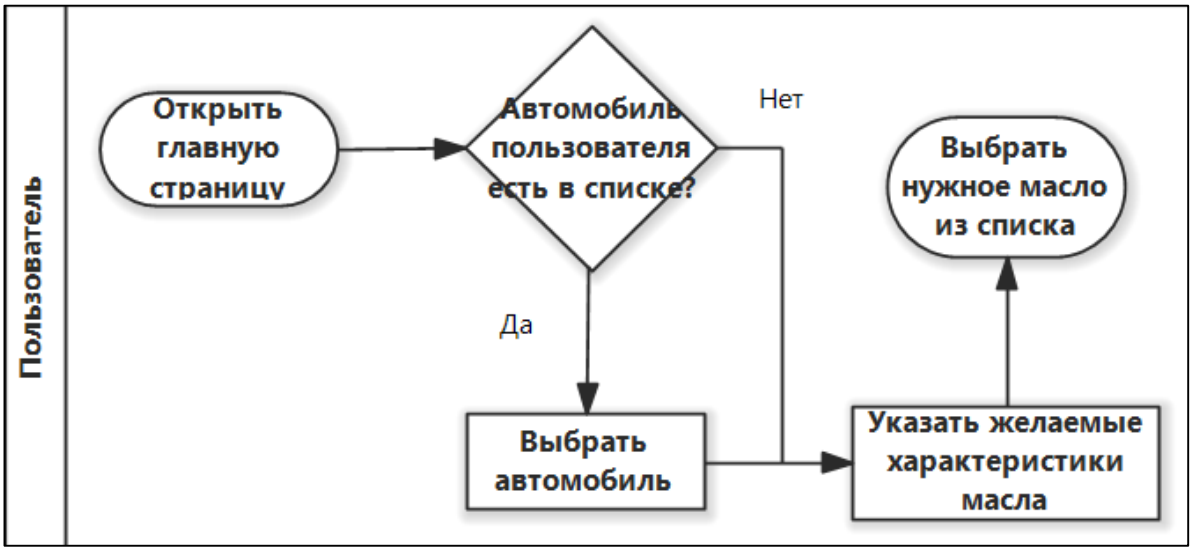

Рисунок 25 – Процесс подбора моторного масла пользователем

Для выбора масла, пользователь сперва открывает главную страницу, затем – пытается найти модель своего автомобиля в списке. Если нужного пользователю автомобиля нет – он подбирает масло вручную, используя фильтры характеристик масла. Если он есть – пользователь выбирает автомобиль, и затем указывает в фильтрах предпочтительные характеристики масла.

Поскольку в разрабатываемом приложении могут регистрироваться и добавлять свои масла новые продавцы, создание нетривиальной системы сопоставления масел автомобилю – целесообразно: в случае добавления продавцом нового масла, ему не придется просить администратора указать его

масло в качестве допустимого, масло сразу после добавления в систему будет подходить для автомобиля. При этом доверить самому продавцу возможность подбирать масла для автомобиля – небезопасно, поскольку продавец способен добавить неверные сведения, и обнаружить эти неверные сведения намного труднее, чем обнаружить добавленные в систему масла с неверно заданными характеристиками. В связи с этим, целесообразным выглядит следующее решение: привязать условия подбора масла к типам автомобиля, а не к конкретным маслам, предоставить администраторам возможность добавлять типы автомобиля и наборы условий, при соответствии которым масло будет считаться подходящим для выбранного автомобиля.

Диаграмма деятельности (Activity diagram) – UML-диаграмма, позволяющая детально визуализировать конкретный случай использования. Также она может использоваться для визуализации бизнес-процессов.

На рисунке 26 представлена диаграмма деятельности алгоритма проверки масла на соответствие критериям.

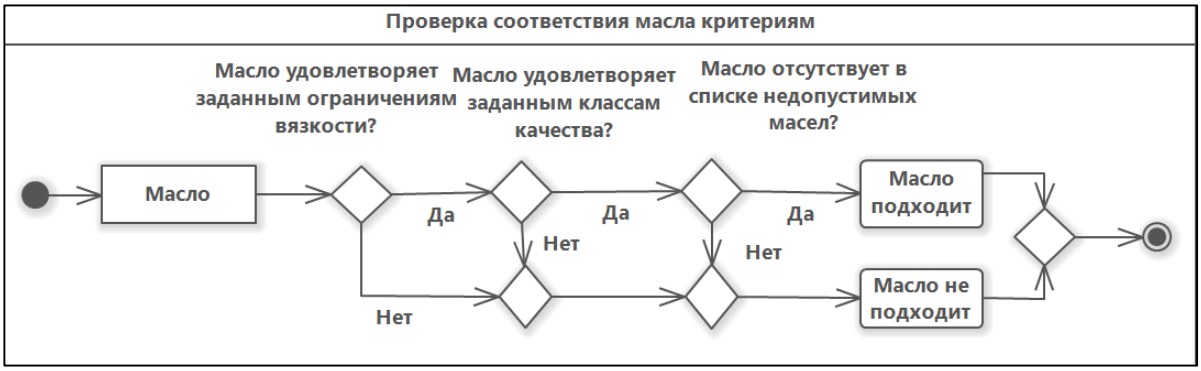

Рисунок 26 – Диаграмма деятельности проверки масла на соответствие условиям

Масло должно удовлетворять трем условиям: подходить по классу качества, иметь заданную вязкость и отсутствовать в списке недопустимых для данного автомобиля масел. Только в случае, если масло будет подходить по данным условиям, оно будет выводиться в списке масел в качестве подходящего для выбранного автомобиля.

3. РЕАЛИЗАЦИЯ СИСТЕМЫ

3.1. Дизайн и верстка окон

Учитывая, что веб-сервис предназначен для выбора моторного масла, есть смысл выполнить дизайн, используя в гамме цвет, близкий к цвету свежего моторного масла. Цветовая гамма должна состоять из минимального числа цветов, чтобы не отвлекать пользователя от его задачи, но при этом быть контрастной.

Дизайн и структура взаимодействия пользователя с приложением должны быть привычными для него в смысле пользовательского опыта, чтобы пользователю было легко ориентироваться в приложении.

Судя по интерфейсу сайтов, показанному на картинках в разделе 1.2 работы, пользователь привычен к сетке с товарами и фильтрами на боковой панели. Разметка приложения соответствует привычному опыту пользователя (рисунок 27).

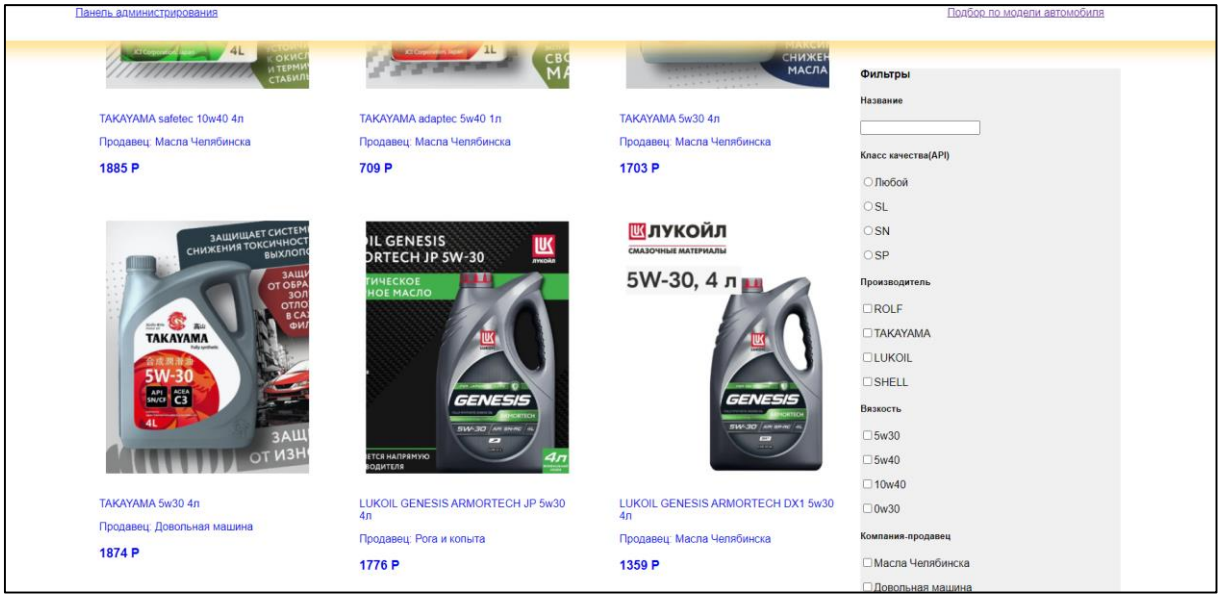

Рисунок 27 – Окно подбора масла

Верхняя панель предназначена для навигации пользователя по сайту.

Окно редактирования данных моторного масла представлено на рисунке 28.

Рисунок 28 – Окно редактирования информации о масле

На рисунке 1 приложения Б изображен стандартный интерфейс редактирования данных с аккаунта администратора.

Для создания данного интерфейса использовались стили CSS, изображенные на рисунке 29.

```
.editUnit {
    box-shadow: 0 0 20px #ffe292;
    padding: 0px;
    border-radius: 2em;
    margin: 15px 0;
}

.subEditUnit {
    padding: 20px;
    margin: 0px;
    border-top: 5px solid #f7ce00;
}

.subsubEditUnit {
    border-top: 2px solid #f7ce00;
    padding: 5px 0;
}

.paddingUnit{
    padding: 20px;
}
```

Рисунок 29 – Стили редактирования данных приложения

Верстка – это описание визуальной части сайта с помощью гипертекстового документа на основе HTML-разметки [20].

Для верстки в данном проекте используются такие технологии, как HTML5, CSS и Razor Pages.

HTML5 – последняя и наиболее мощная редакция языка разметки HTML [21].

HTML – это язык разметки гипертекстовых документов. Он нужен, чтобы отображать в браузере специальным образом отформатированный документ с множеством вложенных элементов: заголовками, абзацами, списками, гиперссылками, медиаисточниками, расположением изображений, видео и аудио [22].

CSS – это язык описания внешнего вида документа, то есть он отвечает за то, как выглядят веб-страницы: цвет фона и декоративных элементов, размер и стиль шрифтов [23].

Razor Pages – это фреймворк для внедрения кода на основе .NET на веб-страницы [24].

В листинге 1 приложения А приведен код Razor и HTML для отрисовки общей шапки и подвала веб-страниц приложения.

Отрисовка – это генерация HTML-кода всей страницы на сервере в ответ на запрос [25].

Шапка и подвал – соответственно верхняя и нижняя части веб-страницы.

Метод `@RenderSection("Styles", required: false)` отрисовывает часть, отвечающую за загрузку css-стилей с сервера, `@RenderBody()` отрисовывает тело. Метод `@RenderSection("Scripts", required: false)` отрисовывает часть приложения, отвечающую за загрузку скриптов JavaScript. Элемент `<div class="header_menu_links" id="header_menu_links">` содержит в себе ссылки для навигации по приложению, которые могут меняться в зависимости от роли пользователя, использующего приложение. Ссылки в вышеуказанный элемент подгружаются с помощью fetch-запроса в JavaScript-скрипте.

В листинге 2 приложения А приведен код Razor и HTML для отрисовки главной страницы приложения – страницы подбора масла.

В качестве модели передачи данных в страницу Razor используется отдельный C# класс. Масла выводятся с помощью цикла `foreach`, сущности, по которым пользователь фильтрует список масел – тоже выводятся с помощью данного цикла. Для того, чтобы пользователю при указании характеристик масла выводились лишь часть масел, прошедшая фильтры, обработка запросов фильтров осуществляется с помощью JavaScript кода из подключенного файла с помощью fetch-запросов (рисунок 30).

В представленном блоке кода в цикле внизу берется список элементов формы фильтров, предназначенных для ввода данных. Для них создается событие, реагирующее на изменение фильтров и вызывающее fetch-запрос.

Функция call_grid_update считывает данные из формы фильтров, отправляет данные формы на сервер, в ответе сервер возвращает html-код списка масел в виде PartialView, который вставляется внутрь контейнера списка масел oilstable.

```
    let elements = document.getElementsByClassName("filterinput");

async function call_grid_update() {
    console.log("emited");
    let formData = new FormData(document.getElementById("FiltersForm"));

    let response = await fetch(fpath,
        {
            method: 'POST',
            body: formData
        });
    let string = await response.text();
    //let t = await json;
    document.getElementById("oilstable").innerHTML = string;

}


for (let i = 0; i < elements.length; i++) {
    console.log("lol");
    console.log(elements[i]);
    elements.item(i).oninput = call_grid_update;
    //document.getElementById("FiltersForm").
}
```

Рисунок 30 – JavaScript-код для обновления fetch-запросов списка моторных масел

3.2. Работа контроллеров приложения

В листинге 3 приложения А приведен код контроллера «HomeController», отвечающего за исправную работу главного окна приложения. Приватное поле _logger служит для логгирования информации, поле db – для доступа из контроллера к базе данных, поле choosebyCar помогает получить информацию о том, какие масла соответствуют заданным критериям.

Поскольку отрисовка списка масел необходим после вызова fetch-запроса на обновление списка, поле choosebyCar используется в методе Filters, созданном специально для работы с этим fetch-запросом.

Для получения подробной информации о конкретном масле используется метод `OilDetails`. Информация о масле, получаемая после вызова метода, приведена на рисунке 31.

Рисунок 31 – Подробная информация о масле

Для получения подробной информации о конкретном масле используется метод `OilDetails`. Информация о масле, получаемая после вызова метода, приведена на рисунке.

Для редактирования информации о маслах и обо всех сущностях системы в целом, в приложении существует контроллер «EditController», код которого приведен также в листинге 4 приложения А.

Метод `EditOil` отрисовывает страницу добавления/изменения конкретного масла.

`EditOilList` и `EdittedAddedOil` позволяют изменять и добавлять масла, принимая запросы от пользователя. В то время как метод `EditOilList` принимает первичный запрос на обработку данных масла из списка масел, `EdittedAddedOil` принимает данные формы, и предназначен для сохранения на изменённых данных о масле, либо данных о добавленном масле.

Метод `OilMerchList` отрисовывает список предложений о продаже масел, методы `CreateMerch`, `PutMerch` и `DeleteMerch` принимают запросы

от клиента на изменение данных о предложениях продажи масел. Метод `OilList` отрисовывает список масел, которые можно удалить, отредактировать и добавить новое масло. При этом вначале этого и многих других методов данного контроллера содержится блок кода, приведенный на рисунке 32.

```
var u = _us.GetUserBySessionId(Request.
        Cookies[Models.Users.User.SessionIdCookieName]);
if (u == null || !u.Role.Permission.CanEditOils)
    return BadRequest();
```

Рисунок 32 – Функция проверки доступа

Контроллеры через этот блок кода для проверки авторизации используют cookie, в которых содержится идентификатор сессии. Система идентифицирует и авторизует пользователя исключительно по идентификатору сессии. `UsersService` – компонент-класс, предназначенный для авторизации, аутентификации и регистрации пользователей. На рисунке 33 экземпляр данного класса именуется как `_us`, у него вызывается метод `GetUserBySessionId`, находящий в базе данных пользователя по идентификатору сессии, извлекаемого в контроллере из cookie-файлов клиента. Далее данные о пользователе, хранящиеся в объекте, именуемом `u`, могут использоваться в методе контроллера для определения, к чему пользователь авторизован, каким образом для него следует отрисовать страницу.

4. ТЕСТИРОВАНИЕ СИСТЕМЫ

Для проверки работоспособности системы было произведено функциональное тестирование, результаты которого приведены в таблице 1.

Таблица 1 – Функциональное тестирование веб-приложения

№	Название теста	Действия	Ожидаемый результат	Тест пройден?
1	Тестирование работы подбора масел по фильтрам характеристик	Пользователь выбирает способ фильтрации списка масел	Выводится список масел, прошедших пользовательские фильтры	Да
2	Тестирование работы подбора масел по типу автомобиля	Пользователь выбирает автомобиль	Выводится список масел, подходящих для данного автомобиля по заданным условиям	Да
3	Тестирование добавления масел	Пользователи с ролями администратор или владелец компании добавляют масло в базу данных	Масло добавляется в базу данных	Да
4	Тестирование добавления предложений о продаже	Пользователи с ролями администратор или владелец компании добавляют предложение о продаже в базу данных	Предложение добавляется в базу данных и выводится на главной странице приложения	Да
5	Тестирование задания условий соответствия масла автомобилю	Администратор добавляет условия о соответствии масел автомобилю	Приложение при подборе масла по типу автомобиля выводит список масел в точном соответствии с заданными условиями	Да
6	Регистрация	Пользователь регистрируется в системе	Система запоминает пользователя	Да
7	Аутентификация	Пользователь сообщает системе свои данные	Система распознает пользователя	Да
8	Авторизация	Пользователь запрашивает у системы доступ, на который имеет права	Система предоставляет пользователю доступ	Да
9	Ложная авторизация	Пользователь запрашивает у системы доступ, на который не имеет прав	Система отказывает пользователю в доступе	Да

ЗАКЛЮЧЕНИЕ

Целью данной выпускной квалификационной работы была разработка веб-сервиса, помогающего подобрать моторное масло по типу автомобиля.

Достижение цели происходило в несколько этапов:

1) проведение обзора аналогичных приложений;
2) проектирование архитектуры приложения и требуемой структуры базы данных;
3) проектирование пользовательского интерфейса;
4) разработка серверной части;
5) тестирование приложения, выявление возможных проблем и их решение.

В рамках данной выпускной квалификационной работы был произведен обзор аналогичных приложений, спроектирована архитектура приложения и требуемая структура базы данных, спроектирован пользовательский интерфейс, разработана серверная часть приложения и протестировано приложение.

Исходный код разработанной системы доступен по URL-адресу: https://github.com/ClassEquivalence/WebMotorOilChooser.

В ходе работы были решены все поставленные задачи и достигнута цель. В дальнейшем система может быть улучшена добавлением адреса магазинов продавцов на карту, проверкой на карте расстояния от покупателя до магазина, добавлением возможности отсортировать список масел нужным пользователю образом. В системе может быть введен выбор нужной модели автомобиля в несколько этапов. В системе может быть улучшена система условий для подбора моторного масла. Систему также можно расширить, добавив в нее отзывы о моторных маслах, в том числе, собирая отзывы с крупных торговых платформ, таких как «Amazon».

ЛИТЕРАТУРА

1. Число легковых автомобилей впервые сократилось более чем в пяти регионах России | Forbes.ru. [Электронный ресурс] URL: https://www.forbes.ru/biznes/485397-cislo-legkovyh-avtomobilej-vpervye-sokratilos-bolee-cem-v-pati-regionah-rossii (дата обращения: 10.06.2024).

2. Как часто нужно менять моторное масло? | Автосервис в Москве. [Электронный ресурс] URL: https://www.dsrv.ru/advices/kak-chasto-nuzhno-menyat-motornoe-maslo/ (дата обращения: 10.06.2024).

3. Как выбрать моторное масло: критерии и рекомендации | OZON. [Электронный ресурс] URL: https://www.ozon.ru/club/article/kak-vybrat-motornoe-maslo-kriterii-i-rekomendatsii-16303639/ (дата обращения: 14.02.2024 г.).

4. Django. [Электронный ресурс] URL: https://www.djangoproject.com/ (дата обращения: 14.02.2024 г.).

5. Django Examples: Top 18 Django Websites | SXT Next. [Электронный ресурс] URL: https://www.stxnext.com/blog/django-websites-examples/ (дата обращения: 14.02.2024 г.).

6. ASP.NET – что это, как работает и зачем нужно, основы | Skillfactory. [Электронный ресурс] URL: https://blog.skillfactory.ru/glossary/asp-net/ (дата обращения: 14.02.2024 г.).

7. Что такое Asp net core: основные характеристики и преимущества | Мониторинг ЖКХ. [Электронный ресурс] URL: https://uchet-jkh.ru/i/cto-takoe-asp-net-core-osnovnye-xarakteristiki-i-preimushhestva/ (дата обращения: 14.02.2024 г.).

8. Top 10 Websites Written Using ASP.NET MVC | DanylkoWeb. [Электронный ресурс] URL: https://www.danylkoweb.com/Blog/top-10-websites-written-using-aspnet-mvc-JK (дата обращения: 14.02.2024 г.).

9. Laravel: что это за PHP-фреймворк и для чего нужен, возможности | Skillfactory. [Электронный ресурс] URL: https://blog.skillfactory.ru/glossary/laravel/ (дата обращения: 14.02.2024 г.).

10. Что такое Angular. Первый проект | METANIT.COM. [Электронный ресурс] URL: https://metanit.com/web/angular2/1.1.php (дата обращения: 14.02.2024 г.).

11. Введение | Vue.js. [Электронный ресурс] URL: https://ru.vuejs.org/v2/guide/index.html (дата обращения: 14.02.2024 г.).

12. Что такое MySQL | nic.ru. [Электронный ресурс] URL: https://www.nic.ru/help/chto-takoe-mysql_8510.html (дата обращения: 10.06.2024).

13. Что такое SQL Server? | Microsoft Learn. [Электронный ресурс] URL: https://learn.microsoft.com/ru-ru/sql/sql-server/what-is-sql-server?view=sql-server-ver16 (дата обращения: 10.06.2024).

14. SQL Server 2022 | Microsoft. [Электронный ресурс] URL: https://www.microsoft.com/ru-ru/sql-server/sql-server-2022-pricing (дата обращения: 10.06.2024).

15. База данных Oracle DB: как она устроена и чем хороша | Skillbox. [Электронный ресурс] URL: https://skillbox.ru/media/code/baza-dannykh-oracle-db-kak-ona-ustroena-i-chem-khorosha/ (дата обращения: 10.06.2024).

16. СУБД PostgreSQL: почему ее стоит выбрать для работы с данными и как установить | Яндекс Практикум. [Электронный ресурс] URL: https://practicum.yandex.ru/blog/chto-takoe-subd-postgresql/ (дата обращения: 10.06.2024).

17. Словарь маркетолога. Что такое CTA, где его размещают и как сформулировать сильный CTA | Skillbox. [Электронный ресурс] URL: https://skillbox.ru/media/marketing/slovar-marketologa-chto-takoe-cta-gde-ego-razmeshchayut-i-kak-sformulirovat-silnyy-cta/ (дата обращения: 14.02.2024 г.).

18. Использование диаграммы вариантов использования UML при проектировании программного обеспечения | Хабр. [Электронный ресурс] URL: https://habr.com/ru/articles/566218/ (дата обращения: 07.06.2024).

19. Что такое диаграмма рабочих процессов? | Miro. [Электронный ресурс] URL: https://miro.com/ru/diagramming/what-is-a-workflow-diagram/ (дата обращения: 07.06.2024).

20. Верстка | Skillfactory. [Электронный ресурс] URL: https://blog.skillfactory.ru/glossary/verstka/ (дата обращения: 07.06.2024).

21. HTML5 | Skillfactory. [Электронный ресурс] URL: https://blog.skillfactory.ru/glossary/html5/ (дата обращения: 07.06.2024).

22. HTML | Skillfactory. [Электронный ресурс] URL: https://blog.skillfactory.ru/glossary/html/ (дата обращения: 07.06.2024).

23. CSS | Skillfactory. [Электронный ресурс] URL: https://blog.skillfactory.ru/glossary/css/ (дата обращения: 07.06.2024).

24. Razor Справочник по синтаксису для ASP.NET Core | Microsoft Learn. [Электронный ресурс] URL: https://learn.microsoft.com/ru-ru/aspnet/core/mvc/views/razor?view=aspnetcore-8.0 (дата обращения: 07.06.2024).

25. Server Side Rendering, SSR | Skillfactory. [Электронный ресурс] URL: https://blog.skillfactory.ru/glossary/ssr/ (дата обращения: 07.06.2024).

ПРИЛОЖЕНИЯ

Приложение А. Листинги

Листинг 1 – Код шапки и подвала страниц приложения

```
@using WebApplication1.Controllers;

@section Styles {
    <link rel="stylesheet" href="/css/Home/_Layout.css" />
}

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Спецификация</title>
    @RenderSection("Styles", required: false)
</head>
<body>

    <div class="everything_but_footer">
        <header>
            <div class="header_menu">
                <div><h1>Добро пожаловать в помощник подбора
масла!</h1></div>
            </div>
        </header>
        <div class="header_menu_links" id="header_menu_links">
            <a asp-controller="Home" asp-action="@nameof(HomeController.Oil-
ChooseCar)">Подбор по модели автомобиля</a>
        </div>

        <main>
            @RenderBody()
        </main>
    </div>

    <footer>
        <p>&copy; 2023 Мой Интернет Магазин</p>
    </footer>
    @RenderSection("Scripts", required: false)
    <script>
        let header_menu_path = "../" + "Users/" + "HeaderHrefs";
        async function makeHeaderHrefs() {
            let response = await fetch(header_menu_path,
                {
                    method: 'GET'
                });
            let string = await response.text();
            document.getElementById("header_menu_links").innerHTML = string
                + document.getElementById("header_menu_links").innerHTML;
        }
        makeHeaderHrefs();

    </script>
</body>
</html>
```

Листинг 2 – Код главной страницы приложения

```
@model WebApplication1.Models.ToRender.ChoosebyCar;
@using WebApplication1.Models;
@using WebApplication1.Models.MotorOilStats;
@using WebApplication1.Models.ChoiceHelpers;
@using WebApplication1.Controllers;


@section Scripts {
    <script>let fpath = "@Url.Action("Filters", "Home")"</script>
    <script          src="~/js/Home/OilChooseCar.js"          asp-append-ver-
sion="true"></script>
}

@section Styles {
    <link rel="stylesheet" href="/css/Home/_Layout.css" />
    <link rel="stylesheet" href="/css/Home/OilChooseCar.css" />
}

<div class="motordiv">
    <h3>Выберите модель автомобиля:</h3>
    <input list="motorchoose" name="Car" form="FiltersForm" class = "filter-
input">
    <datalist id="motorchoose">
        @foreach (CarType item in Model.CarList)
        {
            <option id="motoropt @item.id" value="@item.Name"></option>
        }
    </datalist>
</div>
<table>
    <tr>
        <td>
            <section class="maincontent">
                <div class="oilstable" id="oilstable">
                    @{
                        @foreach        (MotorOilMerch        item        in
Model.MotorOilMerches)
                        {
                            <div>
                                <a          asp-action="@nameof(HomeControl-
ler.OilDetails)" asp-route-id="@item.id">
                                    <img
src="@item.MotorOil.GetImgNamePath()" asp-append-version=true width="85%">
                                    <div>
                                        <p>@item.MotorOil.Name
@item.MotorOil.SAEViscosity <text>@item.MotorOil.Volume</text>л</p>
                                        <p>Продавец:
@item.Store.Company.Name </p>
                                        <h3>@item.Price P</h3>
                                        <h3>Адрес: @item.Store.Adress</h3>
                                    </div>
                                </a>
                            </div>
                        }
                    }
                </div>
            </section>
        </td>
        <td>
            <aside>
```

```
                <div>
                    <h4>Фильтры</h4>
                    <form id="FiltersForm">
                        <h5>Название</h5>
                        <div>
                            <p><input    class="filterinput"    type="text"
name="Name"></p>
                        </div>
                        <h5>Класс качества(API)</h5>
                        <div>
                            <p>
                                <input   class="filterinput"   type="radio"
name="APIClass" value="Any">Любой
                            </p>
                            @foreach (APIQualityClass item in Model.APIQual-
ityClasses)
                            {
                                <p>
                                    <input class="filterinput" type="radio"
name="APIClass" value="@item.id">@item.Name
                                </p>
                            }
                        </div>

                        <h5>Производитель</h5>
                        <div>
                            @foreach (string item in Model.Producers)
                            {
                                <p>
                                    <input class="filterinput" type="check-
box" name="Producer" value="@item">@item
                                </p>
                            }
                        </div>

                        <h5>Вязкость</h5>
                        <div>
                            @foreach (SAEViscosity item in Model.SAEViscos-
ities)
                            {
                                <p>
                                    <input class="filterinput" type="check-
box" name="SAEViscosity" value="@item.ToString()">@item
                                </p>
                            }
                        </div>
                        <h5>Компания-продавец</h5>
                        <div>
                            @foreach (Company item in Model.Companies)
                            {

                                <p>
                                    <input class="filterinput" type="check-
box" name="Company" value="@item.id">@item.Name
                                </p>
                            }
                        </div>
                        <h5>Объем(литров)</h5>
                        <div>
                            <p>От:<input  class="filterinput"  type="number"
name="VolumeFrom"></p>
```

Окончание листинга 2 приложения А

```
                            <p>До:<input  class="filterinput"  type="number"
name="VolumeTo"></p>
                        </div>
                        <h5>Цена</h5>
                        <div>
                            <p>От:<input  class="filterinput"  type="number"
name="PriceFrom"></p>
                            <p>До:<input  class="filterinput"  type="number"
name="PriceTo"></p>
                        </div>
                    </form>
                </div>
            </aside>
        </td>
    </tr>
</table>
```

Листинг 3 – Код контроллера HomeController

```
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;
        public ApplicationContext db;
        OilsAndFilters OilsAndFilters;
        ChoosebyCar choosebyCar;
        public  HomeController(ILogger<HomeController>  logger,  Application-
Context context)
        {
            _logger = logger;
            db = context;
            OilsAndFilters = new OilsAndFilters();
            choosebyCar = new ChoosebyCar();
        }

        public IActionResult Index()
        {
            return View();
        }

        public IActionResult Privacy()
        {
            return View();
        }

        public IActionResult Test()
        {
            OilsAndFilters.PrepareData(db);
            OilsAndFilters.User = null;
            return View(OilsAndFilters);
        }

        [HttpPost]
        public async Task<IActionResult> TForm(MotorOil motoroil)
        {
            db.MotorOils.Add(motoroil);
            await db.SaveChangesAsync();
            return RedirectToAction("Test");
        }
        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None,
NoStore = true)]        public IActionResult Error()
```

```
        {
            return View(new ErrorViewModel { RequestId = Activity.Cur-
rent?.Id ?? HttpContext.TraceIdentifier });
        }
        //Ajax запрос
        [HttpPost]
        public IActionResult Filters(Models.FormsData.OilFiltersFormData
FormData)
        {
            if(Request.Form.Keys.Contains("Car"))
            {
                choosebyCar.PrepareData(db, FormData, Request.Form["Car"]);
                return PartialView(choosebyCar);
            }
            else
            {
                OilsAndFilters.PrepareData(db, FormData);
                //_logger.LogInformation(FormData.ToString());
                return PartialView(OilsAndFilters);
            }
        }

        public IActionResult OilDetails(int id)
        {
            SpecificOilMerch renderModel = new SpecificOilMerch();
            if (renderModel.FormData(db, id))
                return View(renderModel);
            else
                return new NotFoundResult();
        }
        public IActionResult OilChooseCar()
        {
            choosebyCar.PrepareData(db);
            return View(choosebyCar);
        }
        public IActionResult initSomeTestDb()
        {
            TestDBMaker tdb = new TestDBMaker();
            tdb.db = db;
            tdb.initTestDb();
            return new NoContentResult();
        }
    }
```

Листинг 4 – Код контроллера EditController

```
    public class EditController : Controller
    {
        private readonly ILogger<HomeController> _logger;
        public ApplicationContext db;
        protected UsersService _us;
        public EditController(ILogger<HomeController> logger, Application-
Context context)
        {
            _logger = logger;
            db = context;
            _us = new(new PasswordHasher(), db);
        }
        public IActionResult OilList()
        {
```

```
            var u = _us.GetUserBySessionId(Request.
                    Cookies[Models.Users.User.SessionIdCookieName]);
            if (u == null || !u.Role.Permission.CanEditOils)
                return BadRequest();

            var OilList = new OilList(db);
            return View(OilList);
        }
        public IActionResult EditOilList(int? delete, int? edit, int? add)
        {
            var u = _us.GetUserBySessionId(Request.
                    Cookies[Models.Users.User.SessionIdCookieName]);
            if (u == null || !u.Role.Permission.CanEditOils)
                return BadRequest();

            _logger.LogInformation("LoggerAtReady! ");
            if (Request.Query.ContainsKey("delete"))
            {
                _logger.LogInformation("delete is...");
                var oil = db.MotorOils.Where(id => id.id == de-
lete).ToList()[0];
                db.MotorOils.Remove(oil);
                db.SaveChanges();
                return RedirectToAction(nameof(OilList));
            }
            else if (Request.Query.ContainsKey("edit"))
            {
                _logger.LogInformation("Edit is " + edit);
                return RedirectToAction(nameof(EditOil), new { id = edit });
            }
            else
            {
                _logger.LogInformation("add is...");
                return RedirectToAction(nameof(EditOil), new { id = -1 });
            }
        }
        public IActionResult EditOil(int id)
        {
            var u = _us.GetUserBySessionId(Request.
                    Cookies[Models.Users.User.SessionIdCookieName]);
            if (u == null || !u.Role.Permission.CanEditOils)
                return BadRequest();
            if (id != -1)
            {
                var oil = db.MotorOils.Include(qc => qc.APIQualityClass).In-
clude(sae=>sae.SAEViscosity).Where(mo => mo.id == id).ToList()[0];
                var dto = new EditOil(db, oil, true);
                return View(dto);
            }
            else
            {
                MotorOil oil = new();
                var dto = new EditOil(db, oil, false);
                return View(dto);
            }
        }
        public IActionResult EditedAddedOil(Models.Edit.ToAccept.MotorOil
motorOil)
        {
            var u = _us.GetUserBySessionId(Request.
                    Cookies[Models.Users.User.SessionIdCookieName]);
```

```
            if (u == null || !u.Role.Permission.CanEditOils)
                return BadRequest();
            IFormFile f = motorOil.oilImgInput;
            var mo = motorOil.toMotorOil(db);
            mo.SaveImg(f);
            return View();
        }
        public IActionResult OilMerchList()
        {
            var u = _us.GetUserBySessionId(Request.
                    Cookies[Models.Users.User.SessionIdCookieName]);
            if (u==null || !u.Role.Permission.CanEditMerch)
                return BadRequest();

            MerchList model = new MerchList(db);
            return View(model);
        }
        [HttpGet]
        public IActionResult CreateMerch()
        {
            var u = _us.GetUserBySessionId(Request.
                    Cookies[Models.Users.User.SessionIdCookieName]);
            if (u == null || !u.Role.Permission.CanEditMerch)
                return BadRequest();
            //post, put, delete
            Models.Edit.ToRender.ListUnits.Merch merch = new(db);
            return PartialView(merch);
        }
        [HttpPut]
        public IActionResult PutMerch(Merch merch)
        {
            var u = _us.GetUserBySessionId(Request.
                    Cookies[Models.Users.User.SessionIdCookieName]);
            if (u == null || !u.Role.Permission.CanEditMerch)
                return BadRequest();
            MotorOilMerch m = db.MotorOilMerches.Where(m => m.id ==
merch.merchId).ToList()[0];
            //post, put, delete
            merch.ToMotorOilMerch(m);
            db.SaveChanges();
            return NoContent();
        }
        [HttpDelete]
        public IActionResult DeleteMerch(int id)
        {
            var u = _us.GetUserBySessionId(Request.
                    Cookies[Models.Users.User.SessionIdCookieName]);
            if (u == null || !u.Role.Permission.CanEditMerch)
                return BadRequest();
            //post, put, delete
            MotorOilMerch m = db.MotorOilMerches.Where(m => m.id ==
id).ToList()[0];
            db.Remove(m);
            db.SaveChanges();
            return NoContent();
        }

        public IActionResult StoreList()
        {
            return View();
        }
```

```
    public IActionResult CompanyList()
    {
        return View();
    }
    public IActionResult UsersList()
    {
        return View();
    }
    public IActionResult CarMotorList()
    {
        return View();
    }
    public IActionResult ConditionsList()
    {
        var u = _us.GetUserBySessionId(Request.
                Cookies[Models.Users.User.SessionIdCookieName]);
        if (u == null || !u.Role.Permission.CanEditOils)
            return BadRequest();

        ConditionsList cl = new(db);
        return View(cl);
    }
}
```

Приложение Б. Интерфейс редактирования условий подбора масла

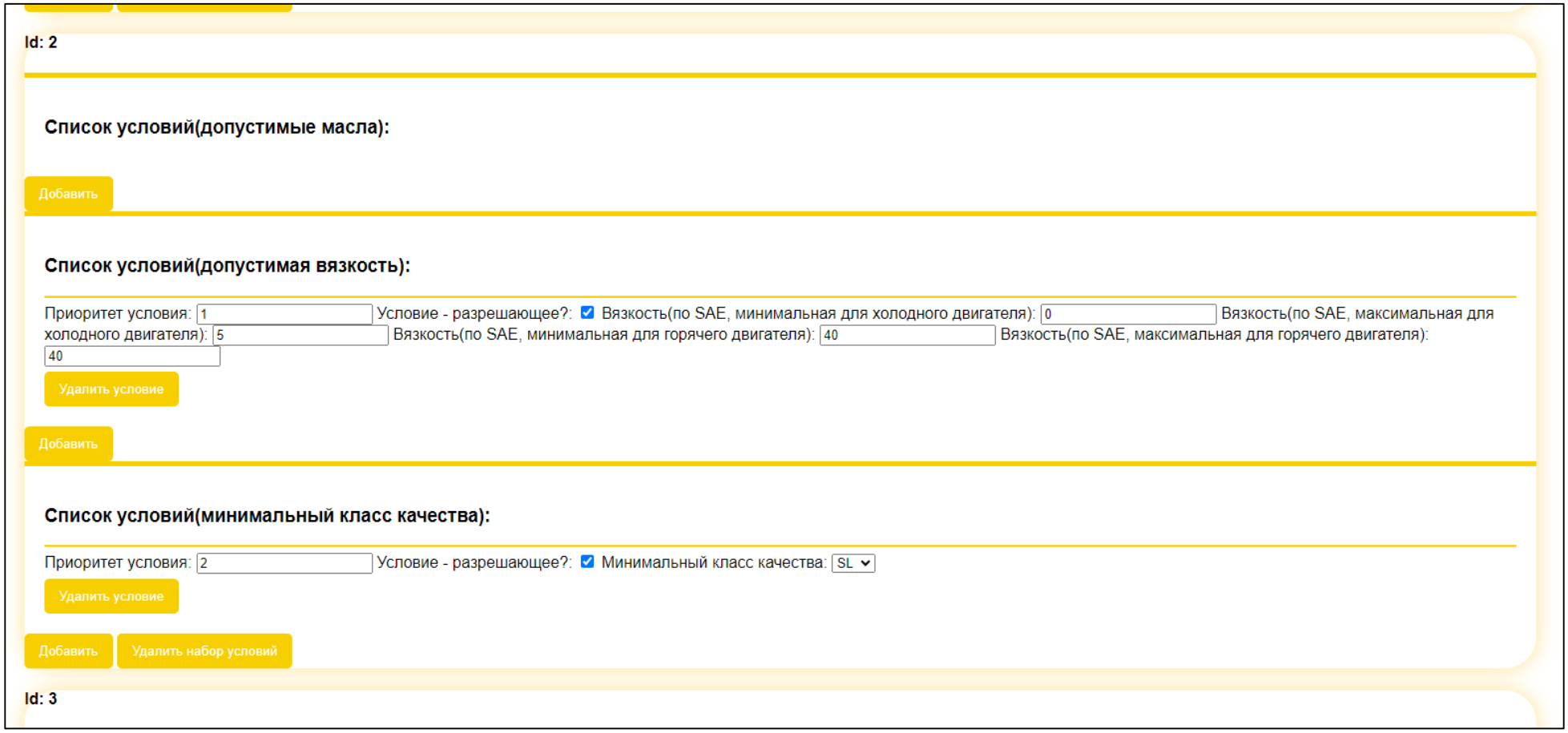

Рисунок 1 – Интерфейс редактирования условий подбора масла