**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ**

**Кафедра веб-технологий и компьютерного моделирования**

ДЫЛЕВСКИЙ

Максим Сергеевич

**РАЗРАБОТКА СЕРВИСА ПО ЗАКАЗУ АВТОЗАПЧАСТЕЙ НА JAVASCRIPT И REACT**

Курсовая работа

**Научный руководитель**

кандидат физико-математических наук

В. С. Романчик

"Допущен к защите"

"\_\_ " \_\_\_\_\_\_\_\_\_\_\_ 2025 г.

Зав. кафедрой веб-технологий

и компьютерного моделирования,

кандидат физико-математических наук

М. В. Игнатенко

Минск, 2025

**ОГЛАВЛЕНИЕ**

[**ВВЕДЕНИЕ**](#_heading=h.ghjq6omc3ct0) **4**

**ГЛАВА 1. ОБЗОР НЕОБХОДИМОГО ФУНКЦИОНАЛА СЕРВИСА. ТЕХНОЛОГИИ РАЗРАБОТКИ 5**

1.1. Необходимый функционал сервиса 5

1.2. Используемые технологии 7

1.2.1 Клиентская часть. 7

1.2.2. Серверная часть приложения. 8

1.2.3. База данных. 10

**ГЛАВА 2. РАЗРАБОТКА СЕРВИСА 11**

2.1. Разработка клиентской части приложения. 11

2.2. Разработка серверной части приложения. 15

2.3. Проектирование и разработка базы данных. 21

[**ЗАКЛЮЧЕНИЕ**](#_heading=h.8klusbm8vb5f) **25**

[**СПИСОК ИСТОЧНИКОВ**](#_heading=h.ck4ojofo2w8) **26**

**[ПРИЛОЖЕНИЕ “КОД СЕРВИСА ПО ЗАКАЗУ АВТОЗАПЧАСТЕЙ”](#_heading=h.z0votw17acxn) 27**

Белорусский государственный университет

Механико-математический факультет

Кафедра веб-технологий и компьютерного моделирования

**ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

Студент: Дылевский Максим Сергеевич

1. Тема: Разработка сервиса по заказу автозапчастей на JavaScript и React.

2. Срок представления курсового проекта к защите: 26 мая 2025 года

3. Исходные данные для научного исследования (проектирования)

* **Документация по JavaScript**. [Электронный ресурс] – Режим доступа: <https://developer.mozilla.org/ru/docs/Web/JavaScript>
* **Основы работы с React**. [Электронный ресурс] – Режим доступа: <https://reactjs.org/docs/getting-started.html>
* **Документация по Node.js**. [Электронный ресурс] – Режим доступа: <https://nodejs.org/ru/docs/>
* **Документация по MySQL**. [Электронный ресурс] – Режим доступа: <https://www.mysqltutorial.org/>

4. Содержание курсового проекта:

4.1 Спроектировать архитектуру интернет-магазина со следующей структурой:

Ядро – содержит классы для управления данными о товарах, пользователях и заказах, реализующие основной функционал веб-приложения.

Компоненты – классы для визуализации интерфейса, такие как ProductCard, Cart, UserProfile и OrderSummary, которые представляют все типы объектов на страницах.

4.2 Реализовать механику приложения: логику взаимодействия между компонентами, такие как добавление товаров в корзину, оформление заказа и управление пользователями.

4.3 Создание конфигурации и параметры объектов в формате JSON для хранения информации о товарах, пользователях и заказах.

Руководитель курсового проекта \_\_\_\_\_\_\_\_\_\_\_\_\_

*подпись, дата инициалы, фамилия*

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*подпись, дата*

ВВЕДЕНИЕ

В современном мире интернет становится неотъемлемой частью нашей жизни, и онлайн-торговля занимает все более значительную долю рынка. Разработка интернет-магазина автозапчастей является актуальной задачей, так как обеспечивает удобный доступ к широкому ассортименту товаров для автомобилистов.

С учетом увеличения числа автомобилей и потребности в их обслуживании, рынок автозапчастей продолжает расти. Покупка запчастей в интернете позволяет экономить время и средства, предоставляя возможность сравнивать цены и искать необходимые детали без необходимости посещения физических магазинов.

Цель данного курсового проекта — создать функциональный и удобный интернет-магазин автозапчастей, который будет включать в себя каталог товаров, систему поиска, возможность оформления заказов и интеграцию с платежными системами. В результате разработки планируется обеспечить не только эффективную работу магазина, но и высокий уровень пользовательского опыта.

В рамках проекта будут рассмотрены основные аспекты, такие как выбор технологий для реализации, проектирование пользовательского интерфейса и обеспечение безопасности данных потребителей. Ожидается, что созданный интернет-магазин станет успешным инструментом для удовлетворения потребностей клиентов в качественных автозапчастях.

Для достижения поставленной цели необходимо решить следующие задачи:

* Определить необходимый функционал приложения.
* Определить технологии, необходимые для игрового приложения.
* Разработать структуру базы данных и хранения информации.
* Реализовать серверное приложение, предоставляющее API для доступа с клиентской стороны.
* Реализовать клиентское приложение для взаимодействия пользователя.

ГЛАВА 1. ОБЗОР НЕОБХОДИМОГО ФУНКЦИОНАЛА СЕРВИСА. ТЕХНОЛОГИИ РАЗРАБОТКИ

1.1. Необходимый функционал сервиса

Для успешной работы интернет-магазина автозапчастей необходимо реализовать ряд ключевых функций, которые обеспечивают не только удобство, но и безопасность для пользователей. Эти функции включают авторизацию, поиск товаров по фильтрам, управление корзиной и возможность смены пароля. Каждая из этих функций играет важную роль в создании эффективного и привлекательного интерфейса для клиентов.

Авторизация представляет собой процесс идентификации пользователя в приложении и является одним из самых важных элементов интернет-магазина. Она позволяет пользователям создавать личные кабинеты, что, в свою очередь, обеспечивает индивидуальный подход к каждому клиенту. Например, на основе истории покупок и предпочтений система может рекомендовать товары, которые могут заинтересовать конкретного пользователя. С помощью авторизации клиенты могут сохранять свои данные, изучать товары, просматривать активные и завершенные заказы, а также получать персонализированные рекомендации по новым поступлениям или специальным предложениям. Кроме того, авторизация существенно повышает безопасность. Доступ к учетной записи защищен паролем, что предотвращает несанкционированный доступ к личной информации пользователя. Важно отметить, что авторизация осуществляется с помощью JWT (JSON Web Token), что позволяет передавать на клиент зашифрованные токены. Это обеспечивает безопасный доступ к информации на сервере, так как токены могут быть проверены и валидированы на стороне сервера, гарантируя, что только авторизованные пользователи могут получить доступ к своим данным и функционалу.

Функция поиска товаров по фильтрам является критически важной для интернет-магазина, особенно в сфере автозапчастей, где ассортимент может быть весьма обширным. Пользователи должны иметь возможность быстро находить нужные детали по различным критериям, таким как марка автомобиля, модель, год выпуска и тип запчасти. Возможность фильтрации товаров позволяет значительно упростить процесс поиска, что в свою очередь улучшает пользовательский опыт, так как экономит время клиентов и делает покупки более комфортными. Кроме того, возможность фильтрации позволяет пользователю сосредоточиться на тех товарах, которые соответствуют его требованиям, что снижает вероятность разочарования при просмотре неподходящих вариантов. Это также может повысить вероятность покупки, так как клиенты будут быстрее находить именно то, что им нужно

Управление корзиной — это ключевая функция, которая позволяет пользователям добавлять, удалять и изменять количество товаров перед оформлением заказа. Интуитивно понятный интерфейс корзины помогает клиентам легко контролировать свои покупки и видеть общую стоимость заказа. Важно, чтобы из корзины была реализована возможность прибавления и удаления товаров поштучно, а также полное удаление товара из корзины. Эта функция не только облегчает процесс оформления заказа, но и позволяет пользователям сохранять товары для последующей покупки, что значительно способствует повышению вероятности завершения транзакции. Когда клиенты могут легко управлять своими выборами, они более склонны завершить покупку, не испытывая неудобств.

Обеспечение возможности смены пароля является важным аспектом безопасности учетной записи пользователя. Пользователи должны иметь возможность регулярно обновлять свои пароли, чтобы предотвратить несанкционированный доступ к личной информации. Кроме того, эта функция важна для восстановления доступа к аккаунту в случае утери пароля. Процесс смены пароля должен быть простым и безопасным, включая подтверждение через электронную почту или SMS, что добавляет дополнительный уровень защиты. Пользователи должны чувствовать, что их данные находятся в безопасности, и возможность легко изменить пароль — это важный шаг к достижению этой цели.

Таким образом, реализация перечисленных функций не только существенно улучшит пользовательский опыт, но и повысит уровень безопасности и удобства работы с интернет-магазином автозапчастей. Эти аспекты являются основой для создания успешного и привлекательного онлайн-проекта, который сможет удовлетворить потребности клиентов и обеспечить их лояльность.

1.2. Используемые технологии

Для разработки интернет-магазина автозапчастей необходимо использовать различные технологии, которые обеспечат эффективное взаимодействие между клиентом, сервером и базой данных. В данном проекте были выбраны следующие технологии:

1.2.1 Клиентская часть.

Для реализации логики отображения страниц и навигации по приложению использован React — это популярная библиотека для создания пользовательских интерфейсов, которая позволяет разрабатывать динамичные и отзывчивые приложения.

Преимущества:

* Компонентный подход: позволяет разбивать интерфейс на независимые компоненты, что упрощает процесс разработки и повторного использования кода.
* Высокая производительность: использует виртуальный DOM для оптимизации обновлений интерфейса, что обеспечивает быстрое реагирование приложения на действия пользователя.

Для реализации логики отображения страниц и навигации по приложению используется библиотека React Router DOM. Эта библиотека является важным инструментом для управления маршрутизацией в одностраничных приложениях (SPA) и обеспечивает удобный и интуитивно понятный интерфейс для пользователей.

Преимущества использования React Router DOM:

* Динамические маршруты: позволяет настраивать маршруты в зависимости от состояния приложения. Например, пользователь может быть перенаправлен на страницу профиля после авторизации, а при попытке доступа к защищенному контенту — на страницу входа.
* Вложенные маршруты: упрощают создание многоуровневых интерфейсов, где одна страница может содержать другие компоненты. Это делает организацию кода более читаемой и облегчает поддержку приложения.
* Управление историей навигации: пользователи могут использовать кнопки "Назад" и "Вперед" в браузере для перемещения по страницам, что создает привычный опыт работы с веб-приложениями.
* Простые компоненты для навигации: компоненты, такие как Link и NavLink, упрощают создание навигационных меню и кнопок, позволяя разработчикам легко реализовывать навигацию без ручного управления состоянием маршрутов.

Таким образом, использование React Router DOM значительно улучшает пользовательский опыт и упрощает процесс разработки, делая навигацию по интернет-магазину более логичной и удобной.

Для сборки клиентского приложения использован Vite — это современный инструмент сборки, который значительно ускоряет процесс разработки.

Преимущества:

* Мгновенная загрузка: обеспечивает быструю сборку и перезагрузку приложения, что делает разработку более эффективной.
* Поддержка современных стандартов: легко интегрируется с современными инструментами и библиотеками, что упрощает настройку проекта.

1.2.2. Серверная часть приложения.

В качестве основы взята среда исполнения Node.js — это JavaScript-окружение, позволяющее запускать серверный код на JavaScript.

Преимущества:

* Асинхронная обработка: обеспечивает высокую производительность при работе с большим количеством параллельных запросов.
* Единый язык: позволяет использовать JavaScript как на клиенте, так и на сервере, что упрощает разработку.

Для быстрого создания удобного API была использована библиотека Express — это минималистичный фреймворк для Node.js, который упрощает создание серверных приложений.

Преимущества:

* Простота и гибкость: позволяет быстро настраивать маршруты и обрабатывать запросы, поддерживая различные пути маршрутизации и методы запросов.
* Многочисленные middleware: имеет большое количество встроенных и сторонних библиотек, такие как Express-Validator, что расширяет функциональность.

Для проведения JWT авторизации на сервере была использована библиотека JsonWebToken — это библиотека, которая берёт на себя ответственность за создание и чтение токенов авторизации и обновления, а также за управление их циклом жизни.

Преимущества:

* Безопасность: обеспечивает защиту данных и позволяет реализовать авторизацию пользователей.
* Простота использования: легко интегрируется с Express и другими библиотеками.

Для обеспечения большей безопасности пароли хранятся в базе данных и сравниваются на сервере в зашифрованном виде. С шифрованием нам поможет библиотека Bcrypt — это библиотека для хеширования паролей, обеспечивающая безопасность хранения пользовательских данных.

Преимущества:

* Защита паролей: обеспечивает надежное хеширование, что делает пароли менее уязвимыми для атак.
* Простота интеграции: легко используется вместе с Node.js и Express.

Также нам понадобится библиотека для связи нашего сервера с базой данных и получения оттуда информации. Используемый в приложении MySQL2 — это клиент для работы с базой данных MySQL.

Преимущества:

* Высокая производительность: обеспечивает быструю работу с базами данных и поддерживает асинхронные операции.
* Поддержка современных возможностей: включает поддержку промисов и других современных JavaScript-функций.

1.2.3. База данных.

Для хранения информации о пользователях, их предпочтениях и покупках была выбрана СУБД MySQL — это реляционная система управления базами данных, широко используемая в веб-приложениях. Она позволяет удобно записывать, извлекать данные, а также в случае чего легко масштабировать схему хранения данных.

Преимущества:

* Надежность: обеспечивает стабильную работу и защиту данных.
* Масштабируемость: легко справляется с увеличением объема данных и пользовательской нагрузки.
* Богатый функционал: поддерживает сложные запросы, индексацию и транзакции, что делает ее идеальной для интернет-магазинов.

Таким образом, выбранные технологии обеспечивают надежную и эффективную работу интернет-магазина автозапчастей, сочетая в себе производительность, безопасность и удобство разработки.

ГЛАВА 2. РАЗРАБОТКА СЕРВИСА

Во второй главе курсовой работы будет подробно рассмотрен процесс разработки интернет-магазина автозапчастей. Основное внимание будет уделено организации связи между клиентом, сервером и базой данных, что является критически важным для создания эффективного и функционального веб-приложения.

В условиях современного рынка пользователи ожидают максимально удобного и интуитивно понятного интерфейса, который обеспечивает быстрый доступ к необходимым товарам и услугам. Для достижения этой цели необходимо реализовать надежную архитектуру, которая включает в себя серверную часть, отвечающую за обработку запросов и управление данными, клиентскую часть, обеспечивающую взаимодействие с пользователем, и базу данных, где хранится вся необходимая информация о пользователях, товарах и заказах.

Эффективная связь между этими компонентами обеспечит быстрое реагирование на действия пользователей и позволит реализовать такие функции, как авторизация, поиск товаров и управление корзиной. Важно обеспечить высокую производительность и безопасность приложения, что также будет достигнуто через правильную архитектурную организацию и использование современных технологий.

Таким образом, в данной главе мы рассмотрим ключевые аспекты разработки интернет-магазина, включая реализацию авторизации с помощью JWT, взаимодействие с базой данных через MySQL и создание динамичного пользовательского интерфейса на основе React. Все эти элементы в совокупности направлены на создание качественного пользовательского опыта и удовлетворение потребностей клиентов интернет-магазина.

2.1. Разработка клиентской части приложения.

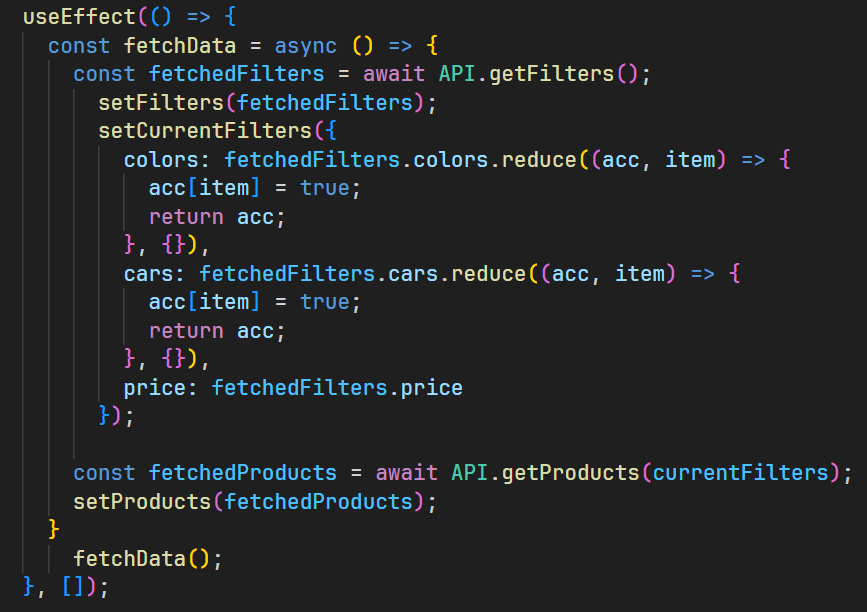
Клиентская часть приложения призвана обеспечить удобную навигацию по сайту и взаимодействие с данными, а также уберечь пользователя от посещения страниц, которые он в данный момент видеть не должен. Например, будучи авторизованным пользователь не должен попадать на страницы логина и регистрации, а будучи не авторизованным, не должен попадать в сам магазин и взаимодействовать с корзиной, аккаунтом и другим функционалом.

Для этого на клиентской стороне реализован специальный layout-компонент AuthOutlet, который отвечает за отображение защищенных страниц. При попытке доступа к таким страницам, если пользователь не авторизован, его перенаправляют на страницу входа. Это обеспечивает безопасность и предотвращает несанкционированный доступ к ресурсам.



*(рис. 1) Код компонента AuthOutlet.*

Также на клиенте стоит отображать только те фильтры, по которым существуют товары. Для этого следует отдельно запрашивать фильтры с сервера. Был реализован запрос на получение всех возможных цветов товаров, автомобилей и диапазона цен. С каждой загрузкой страницы товаров осуществляется этот самый запрос, после чего в зависимости от полученных данных фильтры отображаются на экране. Это позволяет пользователю быстро находить нужные товары, улучшая пользовательский опыт. Фильтры загружаются динамически, что минимизирует время ожидания и снижает нагрузку на сервер.



*(рис. 2) Код получения фильтров с сервера.*

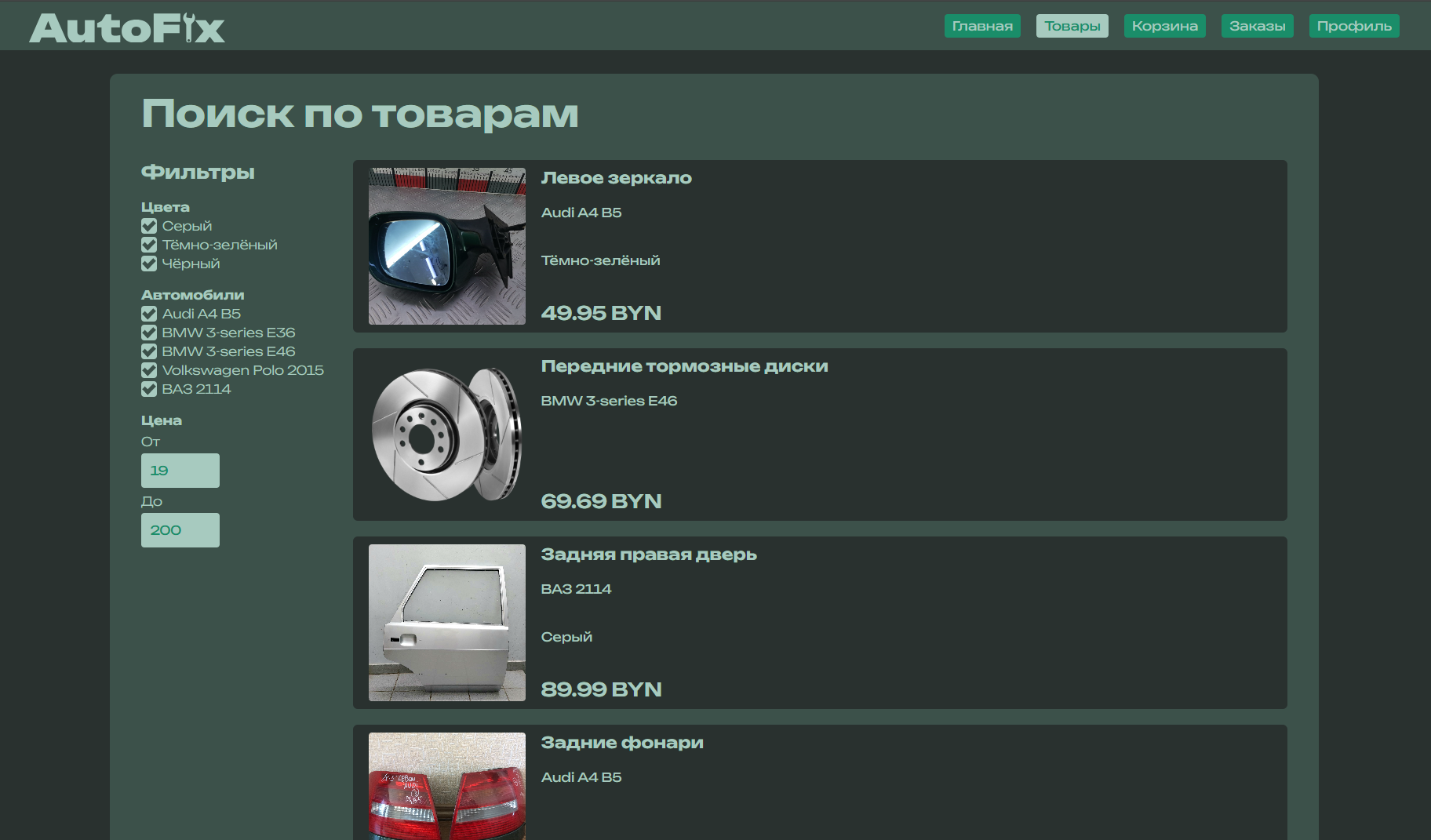


*(рис. 3) Отрисовка полученных фильтров*

Визуальная часть приложения реализована на разметке HTML и стилях CSS. Были реализованы страницы с формами (такие как страница входа, регистрации и смены пароля) и страницы с контентом (такие как страница товаров, корзины и заказов). Интерфейс был адаптирован под разные размеры экранов мониторов с помощью медиазапросов и относительных величин ширины и высоты.



*(рис. 4) Внешний вид страницы входа.*



*(рис. 5) Внешний вид страницы товаров.*

2.2. Разработка серверной части приложения.

Основной задачей сервера является предоставление удобного и безопасного API для клиента для удобства разработки и обеспечения безопасности данных пользователей. Для этого на сервере дополнительно проводится валидация данных и идентификация пользователя, отправившего запрос.

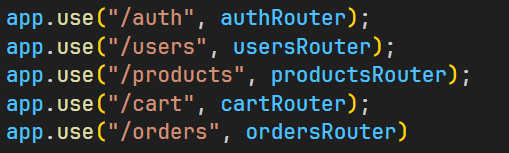
Серверная часть приложения состоит из пяти основных роутеров, каждый из которых отвечает за свою область функциональности. Первым из них является авторизационным, который управляет процессом авторизации и аутентификации пользователей. Он обрабатывает такие запросы, как регистрация новых пользователей, вход в систему, а также обновление и проверка токенов, что обеспечивает безопасность и возможность доступа к защищенным ресурсам.

Вторым является роутер пользователей, который отвечает за управление данными пользователей. Этот роутер предоставляет функционал для получения информации о пользователях, редактирования их профилей и управления настройками аккаунтов.

Третий роутер, роутер товаров, отвечает за работу с товарами, включая добавление, редактирование и удаление продуктов из базы данных. Он также обрабатывает запросы на получение информации о товарах, что позволяет клиенту видеть актуальный ассортимент интернет-магазина, а также осуществляет фильтрацию по запросам пользователя.

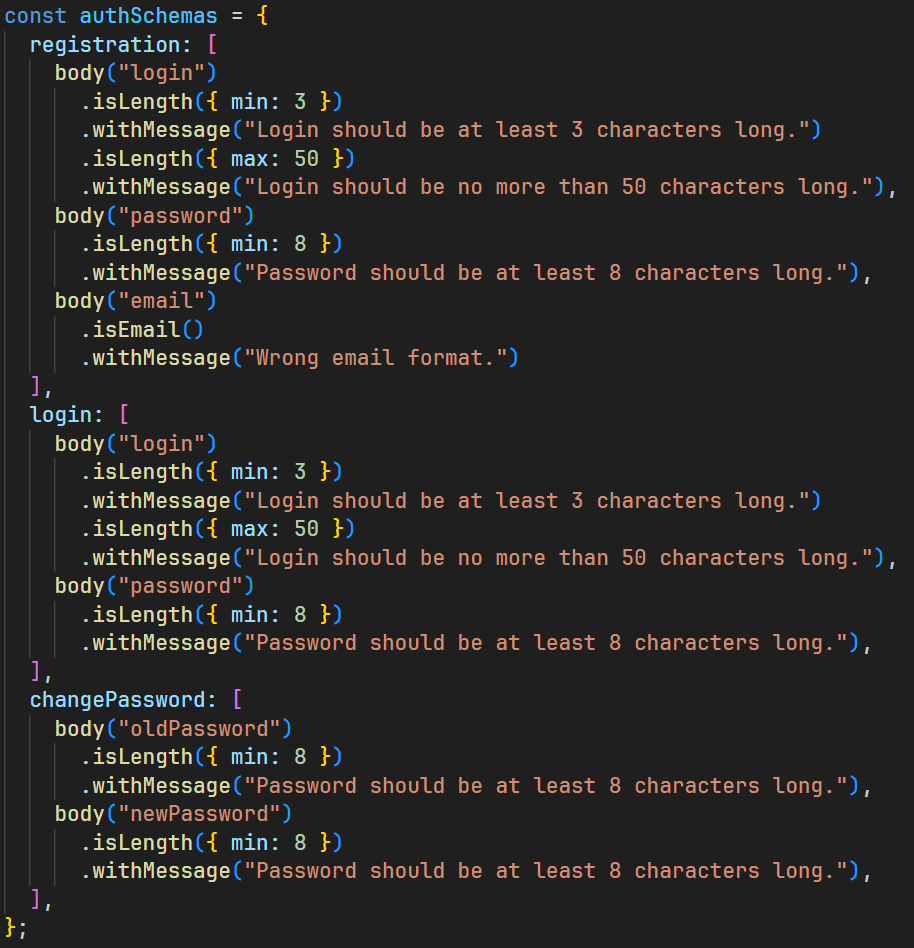
Следующий роутер — роутер корзины, который отвечает за управление корзиной покупок пользователей. Этот роутер обрабатывает запросы, связанные с добавлением товаров в корзину, изменением их количества и удалением. Он также позволяет пользователям получать информацию о текущем содержимом корзины и общей стоимости товаров, что значительно упрощает процесс оформления заказа. Благодаря роутеру корзины клиенты могут легко управлять своими выборами, добавляя или удаляя товары в зависимости от своих потребностей. Кроме того, роутер обеспечивает сохранение состояния корзины между сессиями, что позволяет пользователям вернуться к своим покупкам даже после перезагрузки страницы или выхода из системы. Это повышает удобство и способствует повышению вероятности завершения транзакции, так как клиенты могут без труда вернуться к ранее выбранным товарам. Таким образом, он играет ключевую роль в создании интуитивно понятного и удобного интерфейса для пользователей, что является важным аспектом успешной работы интернет-магазина.

Наконец, роутер заказов управляет процессом оформления заказов, включая создание новых заказов, получение информации о статусах существующих заказов и взаимодействие с товарами в заказах. Этот роутер обеспечивает связь между пользователями и их покупками, что делает процесс покупки более удобным и управляемым.



*(рис. 6) Внедрение роутеров в приложение.*

На серверной стороне для валидирования входящих форм и query-параметров используется библиотека Express-Validator. Эта библиотека позволяет разработчикам проверять, что данные, отправленные пользователем, являются корректными и безопасными, что особенно важно в условиях современных веб-приложений, где безопасность данных занимает первостепенное значение. Используя Express-Validator, мы можем реализовать множество правил валидации, что дает возможность более точно контролировать вводимые данные. Например, при регистрации нового пользователя проверяется, что введенный email имеет правильный формат, соответствующий стандартам, а пароли соответствуют заданным критериям, таким как минимальная длина или наличие специальных символов. Это позволяет избежать распространенных ошибок, таких как ввод некорректного адреса электронной почты, что может привести к путанице при отправке подтверждений или уведомлений. Также важно, что валидация паролей помогает защитить пользователей от использования слабых паролей, что делает учетные записи менее уязвимыми для атак. Кроме того, Express-Validator позволяет нам настраивать пользовательские сообщения об ошибках, что делает взаимодействие с пользователями более понятным и удобным. Если данные не проходят валидацию, пользователю возвращается четкое и информативное сообщение, объясняющее, что именно было введено неправильно и как это исправить. Это способствует повышению качества вводимых данных и улучшает общий пользовательский опыт, так как клиенты получают возможность быстро реагировать на ошибки и исправлять их без лишних задержек. В конечном итоге использование Express-Validator не только упрощает процесс разработки, но и значительно повышает безопасность и надежность нашего приложения, обеспечивая защиту как пользователей, так и самой системы.



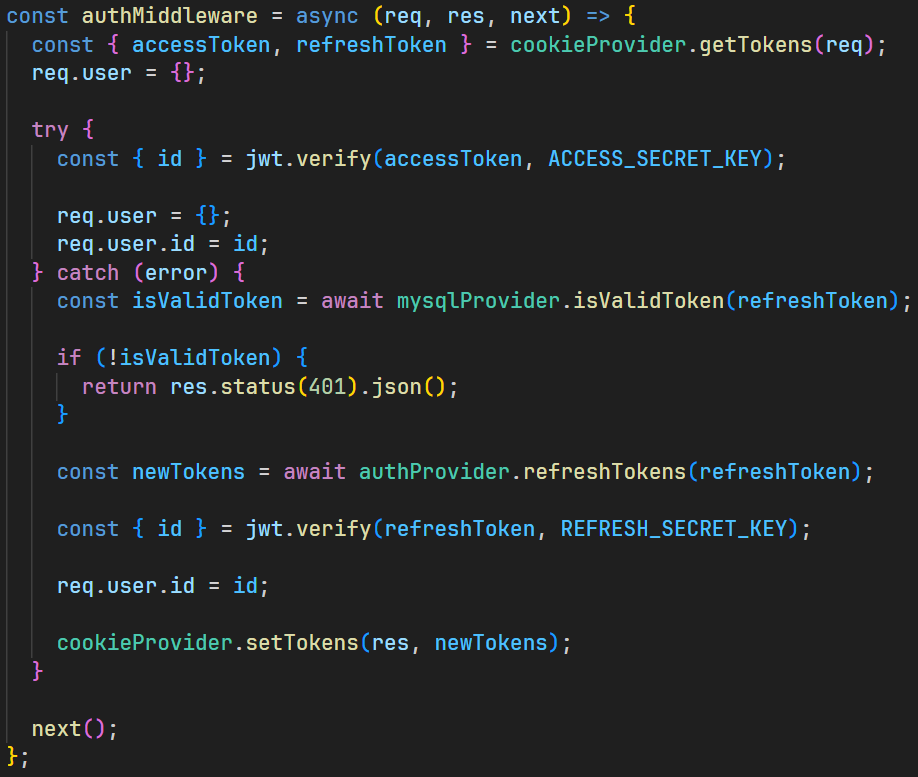
*(рис. 7) Схемы валидации форм регистрации, входа и смены пароля.*

В приложении безопасность и идентификация пользователя играют крайне важную роль, и в нашем проекте эти аспекты реализованы с помощью авторизационного миддлвара. Этот миддлвар выполняет несколько ключевых функций, которые обеспечивают надежную защиту данных пользователей и контроль доступа к различным ресурсам системы. В первую очередь, он отвечает за проверку и обновление токенов авторизации, что является основным механизмом для подтверждения личности пользователя на сервере.

При каждом запросе к защищенным маршрутам, которые требуют наличия авторизации, миддлвар осуществляет проверку на наличие и валидность JSON Web Token (JWT). Этот процесс включает в себя декодирование токена и проверку его сроков действия, а также проверку подписи, чтобы убедиться, что токен не был подменен или скомпрометирован. Если токен действителен, миддлвар разрешает доступ к запрашиваемому ресурсу, что позволяет пользователю продолжать взаимодействовать с приложением без каких-либо преград. В случае, если токен невалиден или отсутствует, миддлвар возвращает четкое и информативное сообщение об ошибке. Это позволяет пользователю понимать, почему доступ к ресурсу был заблокирован, и предоставляет возможность предпринять необходимые действия, такие как повторная авторизация.

Кроме того, в нашем приложении реализован механизм обновления токена, который позволяет поддерживать сессию пользователя активной без необходимости повторной авторизации. Этот процесс обеспечивает дополнительный уровень удобства для пользователей, так как они не будут сталкиваться с необходимостью постоянно вводить свои учетные данные при каждом новом запросе. Механизм обновления токена работает следующим образом: когда срок действия текущего токена истекает, клиент может отправить запрос на получение нового токена, используя специальный механизм, который основан на уже имеющемся действительном токене. Если запрос проходит успешно, клиент получает новый токен, который может быть использован для дальнейших взаимодействий с сервером.

Таким образом, авторизационный миддлвар не только обеспечивает безопасность приложения и защищает данные пользователей, но и значительно улучшает общий пользовательский опыт, позволяя пользователям сосредоточиться на своих действиях в приложении, не беспокоясь о необходимости постоянной повторной авторизации. Это делает наше приложение более удобным и привлекательным для конечного пользователя, а также добавляет важный уровень защиты, который критически важен в условиях современного цифрового мира, где безопасность данных находится на первом месте.



*(рис.8) Авторизационный миддлвар.*

Для обеспечения надежной связи между Express-сервером и клиентом MySQL был реализован специальный класс под названием MySqlProvider. Этот класс представляет собой модуль, который играет ключевую роль в организации и управлении взаимодействием с базой данных MySQL. MySqlProvider инкапсулирует всю логику запросов к базе данных, что позволяет разработчикам сосредоточиться на бизнес-логике приложения, не углубляясь в детали взаимодействия с самой базой данных.

Основная задача MySqlProvider заключается в предоставлении удобного интерфейса для выполнения операций CRUD, что означает создание, чтение, обновление и удаление данных. Благодаря этой инкапсуляции разработчики могут легко выполнять необходимые запросы к базе данных, не беспокоясь о сложностях, связанных с формированием SQL-запросов и обработкой ответов от базы данных. Это значительно упрощает работу с данными и позволяет сократить количество повторяющегося кода, который обычно возникает при прямом взаимодействии с базой данных в различных частях приложения.

Кроме того, использование MySqlProvider облегчает поддержку приложения и его дальнейшую эволюцию. Например, если потребуется изменить структуру базы данных или оптимизировать запросы, это можно сделать в одном месте — в классе MySqlProvider, что автоматически отразится на всех частях приложения, которые используют этот модуль. Это значительно снижает вероятность появления ошибок и повышает общую стабильность системы.

Помимо этого, MySqlProvider включает в себя механизмы обработки ошибок, что позволяет более эффективно управлять ситуациями, когда запросы к базе данных не выполняются успешно. Например, если возникает ошибка подключения или проблема с выполнением SQL-запроса, MySqlProvider может обработать эти исключения и вернуть понятные сообщения об ошибках, что помогает разработчикам и пользователям быстро находить и устранять проблемы.

Таким образом, реализация MySqlProvider значительно улучшает архитектуру приложения, повышая его гибкость и удобство в использовании. Это позволяет разработчикам сосредоточиться на создании функционала и улучшении пользовательского опыта, не отвлекаясь на рутинные задачи, связанные с взаимодействием с базой данных. В конечном итоге, MySqlProvider становится важным инструментом, который способствует созданию высококачественного и надежного программного обеспечения.

2.3. Проектирование и разработка базы данных.

База данных играет ключевую роль в определении структуры хранения данных, а её проектирование обеспечивает удобный и эффективный доступ к необходимой информации. На этапе проектирования базы данных можно выделить несколько основных сущностей, которые будут составлять основу нашей системы.

Первой из этих сущностей является пользователь, который представляет собой любого человека, взаимодействующего с интернет-магазином. Пользователь может иметь различные атрибуты, такие как логин, зашифрованный пароль, адрес электронной почты и дата создания аккаунта. Эти данные необходимы для идентификации и авторизации пользователя, а также для обеспечения безопасности его учетной записи. Зашифрованный пароль гарантирует защиту личной информации, а дата создания аккаунта позволяет отслеживать активность пользователя и анализировать поведение клиентов.

Второй важной сущностью является товар, который представляет собой конкретные автозапчасти, доступные для покупки. Каждый товар будет иметь свои характеристики, такие как цвет, автомобиль, название, описание и ссылка на изображение. Эти параметры помогут пользователям лучше понять, что именно они покупают, и облегчат процесс выбора нужных запчастей. Например, информация о цвете и типе автомобиля позволит пользователям находить наиболее подходящие детали, соответствующие их транспортным средствам.

Третьей сущностью является заказ, который фиксирует информацию о покупках, сделанных пользователями. Заказ будет содержать данные о пользователе, который его оформил, такие как его ID, а также информацию о сумме заказа и адресе доставки. Это позволит эффективно управлять процессом выполнения заказов и гарантировать, что товары будут доставлены по правильному адресу. Дополнительные данные о сумме заказа помогут в аналитике и управлении финансовыми потоками интернет-магазина.

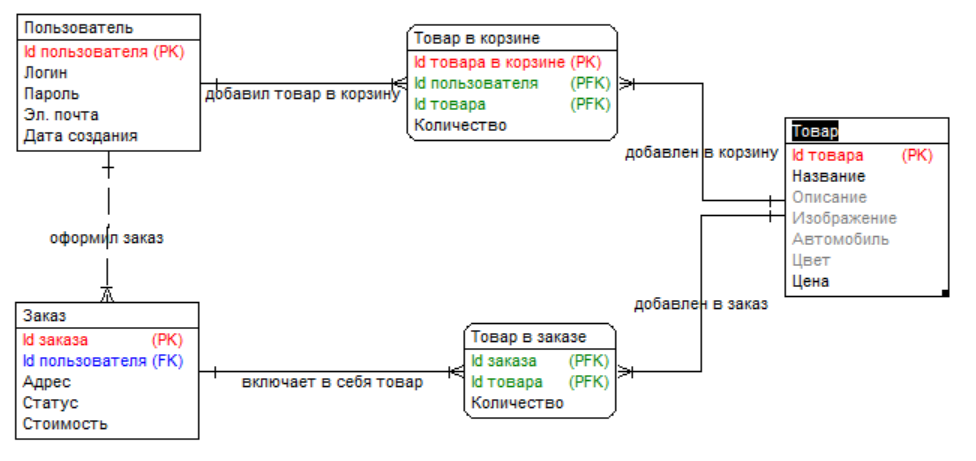


*(рис.9) Сущности и их атрибуты.*

При проектировании базы данных также следует учитывать связи между этими сущностями. В нашем случае можно выделить несколько отношений "многие ко многим", например, между пользователями и товарами, а также между товарами и заказами. Это означает, что один пользователь может иметь несколько товаров в своей корзине, а один товар может быть частью нескольких заказов. Для реализации таких связей мы создадим промежуточные таблицы.

Первая из них — это таблица "Товар в корзине", которая будет хранить информацию о товарах, добавленных пользователями в их корзины. Эта таблица позволит отслеживать, какие именно товары находятся в каждой корзине, а также управлять их количеством.

Вторая промежуточная таблица — "Товар в заказе" — будет использоваться для хранения информации о том, какие товары были включены в каждый конкретный заказ. Это позволит более эффективно управлять процессом выполнения заказов и отслеживать статус каждой единицы товара в рамках заказа.



*(рис.10) Полноатрибутная модель базы данных.*

Кроме того, для обеспечения безопасной авторизации пользователей мы также создадим отдельную таблицу для хранения токенов обновления. Эта таблица будет необходима для управления сессиями пользователей и обеспечит возможность обновления токенов авторизации, что повысит безопасность и удобство использования приложения. В итоге, тщательное проектирование базы данных с учетом всех этих сущностей и связей обеспечит надежное и эффективное функционирование интернет-магазина, позволяя пользователям легко находить и приобретать необходимые товары, а также обеспечивая безопасность их данных.

**ЗАКЛЮЧЕНИЕ**

В результате проделанной работы были успешно выполнены все поставленные задачи. В рамках разработки интернет-магазина автозапчастей было написано значительное количество строк кода и реализовано множество компонентов, обеспечивающих функциональность приложения. Спроектирована и реализована архитектура, которая включает серверную и клиентскую части, взаимодействующие через API.

В приложении была реализована авторизация пользователей с использованием JWT, что обеспечило высокий уровень безопасности и индивидуальный подход к каждому клиенту. Разработаны основные функции, такие как поиск товаров по фильтрам, управление корзиной, а также возможность смены пароля, что значительно улучшает пользовательский опыт.

Создана база данных, в которой хранится информация о пользователях, товарах и заказах. Реализованы связи между сущностями, что позволяет эффективно управлять данными и улучшать взаимодействие с пользователями. Были написаны конфигурационные файлы, обеспечивающие удобный доступ к данным о товарах и пользователях.

В ходе работы был получен ценный опыт работы в команде, что является важным аспектом, так как программирование часто требует совместных усилий. Также был приобретен значительный опыт разработки на современных технологиях, таких как React, Node.js и MySQL.

В результате была создана полноценная платформа для интернет-магазина автозапчастей, адаптированная для удобного использования на различных устройствах, включая мобильные телефоны и ПК. Этот проект стал важным шагом на пути к созданию качественного веб-приложения, способного удовлетворить потребности пользователей и обеспечить их лояльность.

**СПИСОК ИСТОЧНИКОВ**

1. **Документация к языку программирования JavaScript**. [Электронный ресурс] – Режим доступа: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> – Дата доступа: 16.05.2025.
2. **Основы работы с React**. [Электронный ресурс] – Режим доступа: <https://reactjs.org/docs/getting-started.html> – Дата доступа: 16.05.2025.
3. **Документация по работе с MySQL**. [Электронный ресурс] – Режим доступа: <https://www.mysqltutorial.org/> – Дата доступа: 16.05.2025.
4. **Документация по Node.js**. [Электронный ресурс] – Режим доступа: <https://nodejs.org/en/docs/> – Дата доступа: 16.05.2025.
5. **"JavaScript: The Good Parts"**. [Книга] – Дуглас Крокфорд. – Издательство: O'Reilly Media, 2008.
6. **"React Up & Running"**. [Книга] – Стойан Стефанов. – Издательство: O'Reilly Media, 2016.
7. **"Node.js Design Patterns"**. [Книга] – Марио Кассио. – Издательство: Packt Publishing, 2019.

[ПРИЛОЖЕНИЕ “КОД СЕРВИСА ПО ЗАКАЗУ АВТОЗАПЧАСТЕЙ”](#_heading=h.z0votw17acxn)

Server.js:

import express from "express";

import cors from "cors";

import authRouter from "./routers/auth.router.js";

import usersRouter from "./routers/users.router.js";

import cookieParser from "cookie-parser";

import productsRouter from "./routers/products.router.js";

import cartRouter from "./routers/cart.router.js";

import ordersRouter from "./routers/orders.router.js";

const app = express();

app.use(express.json());

app.use(cookieParser());

app.use(cors({

    origin: 'http://localhost:5173',

    credentials: true

}));

const PORT = 3000;

app.use("/auth", authRouter);

app.use("/users", usersRouter);

app.use("/products", productsRouter);

app.use("/cart", cartRouter);

app.use("/orders", ordersRouter)

app.listen(PORT, () => {});

authRouter.js:

import express from "express";

import schemaCheck from "../validations/schema-check.js";

import authSchemas from "../validations/auth.schemas.js";

import mysqlProvider from "../providers/mysql.provider.js";

import authProvider from "../providers/auth.provider.js";

import cookieProvider from "../providers/cookie.provider.js";

import authMiddleware from "../middlewares/auth.middleware.js";

const authRouter = express.Router();

authRouter.post(

  "/register",

  schemaCheck(authSchemas.registration),

  async (req, res) => {

    try {

      const user = req.body;

      const result = await mysqlProvider.addUser(user);

      const userId = result[0].insertId;

      const tokens = await authProvider.createTokens(userId, 0);

      cookieProvider.setTokens(res, tokens);

      res.json();

    } catch (error) {

      res.status(400).json({ error: error.message });

    }

  },

);

authRouter.post(

  "/login",

  schemaCheck(authSchemas.login),

  async (req, res) => {

    try {

      const { login, password } = req.body;

      const user = await mysqlProvider.authenticateUser(login, password);

      const tokens = await authProvider.createTokens(user.id, user.status);

      cookieProvider.setTokens(res, tokens);

      res.json();

    } catch (error) {

      if (error.message === "Wrong password.") {

        return res.status(400).json({ error: error.message });

      }

      res.status(404).json({ error: error.message });

    }

  },

);

authRouter.get(

  "/check-auth",

  authMiddleware,

  async (req, res) => {

    try {

      res.json({ isAuth: true });

    } catch (error) {

      res.json({ isAuth: false });

    }

  },

);

authRouter.patch(

  "/change-password",

  schemaCheck(authSchemas.changePassword),

  authMiddleware,

  async (req, res) => {

    try {

      const { oldPassword, newPassword } = req.body;

      await mysqlProvider.changePassword(req.user.id, oldPassword, newPassword);

      res.json();

    } catch (error) {

      res.status(400).json({ error: error.message });

    }

  },

);

authRouter.post(

  "/logout",

  authMiddleware,

  async (req, res) => {

    await mysqlProvider.deleteToken(req.cookies.refresh\_token);

    cookieProvider.deleteTokens(res);

    res.json();

  },

);

export default authRouter;

cartRouter.js:

import express from "express";

import authMiddleware from "../middlewares/auth.middleware.js";

import mysqlProvider from "../providers/mysql.provider.js";

import schemaCheck from "../validations/schema-check.js";

import cartSchemas from "../validations/cart.schemas.js";

const cartRouter = express.Router();

cartRouter.get(

  "/",

  authMiddleware,

  async (req, res) => {

    const cart = await mysqlProvider.getCart(req.user.id);

    res.json(cart);

  },

);

cartRouter.delete(

  "/",

  authMiddleware,

  async (req, res) => {

    const userId = req.user.id;

    await mysqlProvider.clearCart(userId);

    res.json();

  },

);

cartRouter.delete(

  "/:id",

  schemaCheck(cartSchemas.delete),

  authMiddleware,

  async (req, res) => {

    const id = req.params.id;

    const userId = req.user.id;

    const deleteCount = req.body.deleteCount | 1;

    try {

        await mysqlProvider.deleteFromCart(userId, id, deleteCount);

        res.status(200).json();

    } catch (error) {

        return res.status(404).json();

    }

  },

);

export default cartRouter

ordersRouter.js:

import express from "express";

import authMiddleware from "../middlewares/auth.middleware.js";

import mysqlProvider from "../providers/mysql.provider.js";

import schemaCheck from "../validations/schema-check.js";

import ordersSchemas from "../validations/orders.schemas.js";

const ordersRouter = express.Router();

ordersRouter.get(

  "/",

  authMiddleware,

  async (req, res) => {

    const orders = await mysqlProvider.getOrders(req.user.id);

    res.json(orders);

  },

);

ordersRouter.get(

  "/:id",

  authMiddleware,

  async (req, res) => {

    const orderId = req.params.id

    try {

      const orders = await mysqlProvider.getOrder(req.user.id, orderId);

      res.json(orders);

    } catch (error) {

      res.status(404).json();

    }

  },

);

ordersRouter.delete(

  "/:id",

  authMiddleware,

  async (req, res) => {

    const orderId = req.params.id

    try {

      await mysqlProvider.deleteOrder(req.user.id, orderId);

      res.json();

    } catch (error) {

      res.status(404).json();

    }

  },

);

ordersRouter.post(

  "/",

  schemaCheck(ordersSchemas.new),

  authMiddleware,

  async (req, res) => {

    try {

      await mysqlProvider.newOrder(req.user.id, req.body.address);

      res.json();

    } catch (error) {

      console.log(error);

      if (error.message === "Order must be non empty.") {

        res.status(400).json({ error: error.message })

      }

      res.status(500).json({ error: error.message })

    }

  },

);

export default ordersRouter;

productRouter.js:

import express from "express";

import authMiddleware from "../middlewares/auth.middleware.js";

import mysqlProvider from "../providers/mysql.provider.js";

const productsRouter = express.Router();

productsRouter.get("/", authMiddleware, async (req, res) => {

  const { cars, colors, price } = req.query;

  const products = await mysqlProvider.getProducts();

  const filteredProducts = products.filter((product) => {

    let isValid = true;

      const carsArray = Array.isArray(cars) ? cars : [cars];

      isValid = isValid && (carsArray.includes(product.car) || product.car === null);

      const colorsArray = Array.isArray(colors) ? colors : [colors];

      isValid = isValid && (colorsArray.includes(product.color) || product.color === null);

      const [minPrice, maxPrice] = price.map(Number);

      isValid =

        isValid && product.price >= minPrice && product.price <= maxPrice;

    return isValid;

  });

  res.json(filteredProducts);

});

productsRouter.get("/filters", authMiddleware, async (req, res) => {

    const filters = await mysqlProvider.getFilters();

    res.json(filters);

});

productsRouter.get("/:id", authMiddleware, async (req, res) => {

  try {

    const id = req.params.id;

    const product = await mysqlProvider.getProduct(id);

    res.json(product);

  } catch (error) {

    if (error.message === "Product with this ID doesn`t exist.") {

      return res.status(404).json();

    }

    res.status(500).json({ error: error.message });

  }

});

productsRouter.post("/:id/to-cart", authMiddleware, async (req, res) => {

  try {

    const productId = req.params.id;

    const userId = req.user.id;

    const id = await mysqlProvider.addToCart(userId, productId);

    res.status(200).json(id);

  } catch (error) {

    if (error.message === "Product with this ID doesn`t exist.") {

      return res.status(404).json();

    }

    res.status(500).json({ error: error.message });

  }

});

export default productsRouter;

usersRouter.js:

import express from "express";

import authMiddleware from "../middlewares/auth.middleware.js";

import mysqlProvider from "../providers/mysql.provider.js";

const usersRouter = express.Router();

usersRouter.get(

  "/me",

  authMiddleware,

  async (req, res) => {

    const user = await mysqlProvider.getFullUserInfo(req.user.id);

    res.json(user);

  },

);

usersRouter.get(

  "/:id",

  authMiddleware,

  async (req, res) => {

    try {

      const id = req.params.id;

      const user = await mysqlProvider.getFullUserInfo(id);

      res.json(user);

    } catch (error) {

      if (error.message === "Account with this ID doesn`t exist.") {

        return res.status(404).json();

      }

      res.status(500).json({ error: error.message });

    }

  },

);

export default usersRouter;

database-create.sql:

DROP DATABASE IF EXISTS AutoFix;

CREATE DATABASE AutoFix default charset 'utf8mb4';

USE AutoFix;

CREATE TABLE Users (

    id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

    login VARCHAR(50) UNIQUE NOT NULL,

    password CHAR(64) NOT NULL,

    email VARCHAR(50) NOT NULL,

    create\_time TIMESTAMP NOT NULL

);

CREATE TABLE Products (

    id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

    name VARCHAR(100) NOT NULL,

    description VARCHAR(200) NULL,

    image\_url VARCHAR(100) NULL,

    car VARCHAR(50) NULL,

    color VARCHAR(50) NULL,

    price DECIMAL(10, 2) NOT NULL

);

CREATE TABLE Products\_in\_cart (

    id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

    product\_id INT UNSIGNED NOT NULL,

    user\_id INT UNSIGNED NOT NULL,

    count INT UNSIGNED NOT NULL DEFAULT 1,

    FOREIGN KEY (user\_id) REFERENCES Users(id)

    ON DELETE CASCADE ON UPDATE CASCADE,

    FOREIGN KEY (product\_id) REFERENCES Products(id)

    ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE Orders (

    id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

    user\_id INT UNSIGNED NOT NULL,

    price DECIMAL(10, 2) NOT NULL,

    status VARCHAR(50) NOT NULL,

    address VARCHAR(100) NOT NULL,

    FOREIGN KEY (user\_id) REFERENCES Users(id)

    ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE Products\_in\_orders (

    id INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

    order\_id INT UNSIGNED NOT NULL,

    product\_id INT UNSIGNED NOT NULL,

    count INT UNSIGNED NOT NULL DEFAULT 1,

    FOREIGN KEY (order\_id) REFERENCES Orders(id)

    ON DELETE CASCADE ON UPDATE CASCADE,

    FOREIGN KEY (product\_id) REFERENCES Products(id)

    ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE Refresh\_tokens (

    token VARCHAR(136) PRIMARY KEY NOT NULL

);

add-products.sql:

USE AutoFix;

set names 'utf8mb4';

INSERT INTO Products (name, description, image\_url, car, color, price) VALUES

('Левое зеркало', 'Зеркало на водительскую дверь, имеется электронная регулировка и подогрев.', 'https://r.autostrong-m.by/p/10000932', 'Audi A4 B5', 'Тёмно-зелёный', 49.95),

('Передние тормозные диски', NULL, 'https://zapzap.by/media/files/adb\_rebuild\_modern/16.png', 'BMW 3-series E46', NULL, 69.69),

('Задняя правая дверь', NULL, 'https://magauto102.ru/wp-content/uploads/2023/10/photo\_07\_10\_2023\_06\_18\_34', 'ВАЗ 2114', 'Серый', 89.99),

('Задние фонари', NULL, 'https://a.d-cd.net/b8dd9ds-1920.jpg', 'Audi A4 B5', NULL, 129.99),

('Передний бампер', NULL, NULL, 'Volkswagen Polo 2015', 'Чёрный', 99.99),

('Задний бампер', NULL, 'https://protuning.com/media/cache/product\_page\_image/uploads/images/97/image-5111323jom-1.jpg', 'BMW 3-series E46', 'Серый', 119.99),

('Передние фары', 'Рестайлинг LED Black', 'https://design-tuning.com/images/img\_item/17079886290.JPG', 'BMW 3-series E46', NULL, 189.99),

('Балка подвески', 'Задняя, рестайлинг', 'https://st.carro.su/gallery/version/118/car-part/32829777/148161040/base.jpg', 'Volkswagen Polo 2015', NULL, 119.99),

('Задний бампер', NULL, 'https://i0774.ru/i/2229/600/60008338-354104b3530135462abfb941d8c87549.jpg', 'Audi A4 B5', 'Чёрный', 109.99),

('Левое зеркало', NULL, NULL, 'ВАЗ 2114', 'Чёрный', 39.99),

('Дворники', 'Щётки стеклоочистителя 2 шт., 600мм, каркасные', 'https://ir-3.ozone.ru/s3/multimedia-r/c1000/6523567503.jpg', NULL, NULL, 24.99),

('Пружины передние', 'Небольшой заводской деффект в виде царапины на одной из пружин', 'https://tuningprosto.ru/upload/iblock/923/923a561c4678ccbb6cb6607dce95aaf3.jpg', 'ВАЗ 2114', NULL, 49.99),

('Редуктор газовый', NULL, 'https://avcdn.av.by/advertmedium/0002/5427/5157.jpg', 'Volkswagen Polo 2015', NULL, 149.99),

('Свеча зажигания', NULL, 'https://auto1.by/ImageService/full/387/43018Z.JPG', 'BMW 3-series E36', NULL, 19.99),

('Амортизаторы', NULL, 'https://auto1.by/ImageService/full/32/0032290985.JPG', 'BMW 3-series E46', NULL, 84.99),

('Передние фары', 'Дорестайлинг LED Black', NULL, 'BMW 3-series E36', NULL, 79.99),

('Резина летняя', 'Комплект летней резины, 17-ый радиус', 'https://megasklad24.by/storage/items/thumb/w310\_h310\_0929bcc7-8e8a-44ee-9d31-15affa90efb8.jpeg', NULL, NULL, 109.99),

('Передний бампер', NULL, 'https://sales24.by/wp-content/uploads/pl-09200.jpg', 'Volkswagen Polo 2015', 'Серый', 99.99),

('Левая передняя дверь', 'Имеется электростеклоподъемник. Небольшой заводской деффект в виде царапины в районе ручки.', 'https://avcdn.av.by/advertmedium/0003/8598/2794.jpg', 'Audi A4 B5', 'Серый', 200.00),

('Глушитель', NULL, 'https://ir-3.ozone.ru/s3/multimedia-k/c1000/6381397412.jpg', 'ВАЗ 2114', NULL, 89.99);

RegisterPage.jsx:

import { useState } from "react";

import { Link, useNavigate } from "react-router-dom";

import "./RegisterPage.css"

import API from "../../api/api";

export default function RegisterPage() {

    const navigate = useNavigate();

    const [formData, setFormData] = useState({

        login: '',

        email: '',

        password: '',

        passwordRepeat: ''

    });

    const [errors, setErrors] = useState({

        login: '',

        email: '',

        password: '',

        passwordRepeat: ''

    })

    const handleChange = (event) => {

        const { name, value } = event.target;

        setFormData({

            ...formData,

            [name]: value

        });

    };

    const handleSubmit = (event) => {

        event.preventDefault();

        const currentErrors = {

            login: formData.login.includes(" ") ? "Логин не должен содержать пробельных символов." : formData.login.length < 3 ? "Логин должен быть длиной минимум 3 символа." : "",

            email: /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(formData.email) ? "" : "Некорректный формат электронной почты.",

            password: formData.password.length < 8 ? "Длина пароля должна быть минимум 8 символов" : "",

            passwordRepeat: formData.password !== formData.passwordRepeat ? "Пароли не совпадают" : ""

        }

        console.log(formData.password, currentErrors);

        setErrors(currentErrors)

        if(Object.values(currentErrors).some(value => value.length !== 0)) {

            return;

        }

        const register = async () => {

            try {

                await API.register(formData);

                window.location.reload()

            } catch (error) {

                if (error.message === "Bad Request") {

                    const error = { ...currentErrors, login: 'Пользователь с таким логином уже существует.' };

                    setErrors(error);

                }

            }

        }

        register();

    };

    return <div className="form-page">

        <div className="form-page\_\_container">

        <h2>Регистрация</h2>

        <form action="/login" onSubmit={handleSubmit} className="form">

            <div className="form\_\_row">

                <label htmlFor="login">Логин:</label>

                <input type="text" id="login" name="login" value={formData.login} onChange={handleChange}/>

                <div className="row-error">{errors.login}</div>

            </div>

            <div className="form\_\_row">

                <label htmlFor="email">Электронная почта:</label>

                <input type="text" id="email" name="email" value={formData.email} onChange={handleChange}/>

                <div className="row-error">{errors.email}</div>

            </div>

            <div className="form\_\_row">

                <label htmlFor="password">Пароль:</label>

                <input type="password" id="password" name="password" value={formData.password} onChange={handleChange}/>

                <div className="row-error">{errors.password}</div>

            </div>

            <div className="form\_\_row">

                <label htmlFor="passwordRepeat">Повторите пароль:</label>

                <input type="password" id="passwordRepeat" name="passwordRepeat" value={formData.passwordRepeat} onChange={handleChange}/>

                <div className="row-error">{errors.passwordRepeat}</div>

            </div>

            <button type="submit">Зарегистрироваться</button>

        </form>

        <div className="form-page\_\_text">Есть аккаунт? <Link to="/login" className="span-link">Вход</Link></div>

        </div>

    </div>

}

ProfilePage.jsx:

import { Link, useNavigate } from "react-router-dom";

import "./ProfilePage.css"

import { useEffect, useState } from "react";

import API from "../../api/api";

export default function ProfilePage() {

    const [user, setUser] = useState({})

    const navigate = useNavigate()

    useEffect(() => {

        const fetchData = async () => {

            const fetchedMe = await API.getMe();

            setUser(fetchedMe)

        }

        fetchData();

    },[])

    const handleLogout = async () => {

        const fetchData = async () => {

            await API.logout();

            window.location.reload()

        }

        fetchData();

    }

    const date = new Date(user.create\_time);

    const options = { year: 'numeric', month: 'long', day: 'numeric', hour: '2-digit', minute: '2-digit', second: '2-digit', timeZoneName: 'short' };

    const formattedDate = date.toLocaleString('ru-RU', options);

    return <div className="profile-page">

        <div className="profile-page\_\_container">

            <img src="/svg/avatar.svg" className="avatar" />

            <div className="side">

            <div className="user-info">

                <div className="user-info\_\_block">

                    <span className="info-block\_\_name">Логин</span>

                    {user.login}

                </div>

                <div className="user-info\_\_block">

                    <span className="info-block\_\_name">Электронная почта</span>

                    {user.email}

                </div>

                <div className="user-info\_\_block">

                    <span className="info-block\_\_name">Аккаунт создан</span>

                    <span className="info-block\_\_sub">{formattedDate}</span>

                </div>

                </div>

                <div className="buttons">

                    <div className="btn"><Link to="/change-password"><button>Сменить пароль</button></Link></div>

                    <div className="btn"><button onClick={handleLogout}>Выйти из аккаунта</button></div>

                </div>

            </div>

        </div>

    </div>

}

ProductsPage.jsx:

import { useEffect, useState } from "react";

import CheckboxFilter from "../../components/filters/ColorFilter";

import NumberFilter from "../../components/filters/NumberFilter";

import Product from "../../components/product/Product";

import "./ProductsPage.css";

import API from "../../api/api";

export default function ProductsPage() {

  const [products, setProducts] = useState([]);

  const [filters, setFilters] = useState({

    colors: [],

    cars: [],

    price: {

      min\_price: 0,

      max\_price: 0,

    },

  });

  const [currentFilters, setCurrentFilters] = useState({

    colors: {},

    cars: {},

    price: {},

  });

  const handleCheckColor = (e) => {

    setCurrentFilters({

      ...currentFilters,

      colors: { ...currentFilters.colors, [e.target.name]: e.target.checked },

    });

  };

  const handleCheckCar = (e) => {

    setCurrentFilters({

      ...currentFilters,

      cars: { ...currentFilters.cars, [e.target.name]: e.target.checked },

    });

  };

  const handleChangePrice = (e) => {

    e.target.value = e.target.value.replace(/[^0-9]/g, "");

    setCurrentFilters({

      ...currentFilters,

      price: { ...currentFilters.price, [e.target.name]: e.target.value },

    });

  };

  useEffect(() => {

    const fetchData = async () => {

      const fetchedFilters = await API.getFilters();

      setFilters(fetchedFilters);

      setCurrentFilters({

        colors: fetchedFilters.colors.reduce((acc, item) => {

          acc[item] = true;

          return acc;

        }, {}),

        cars: fetchedFilters.cars.reduce((acc, item) => {

          acc[item] = true;

          return acc;

        }, {}),

        price: fetchedFilters.price,

      });

      const fetchedProducts = await API.getProducts(currentFilters);

      setProducts(fetchedProducts);

    };

    fetchData();

  }, []);

  useEffect(() => {

    const fetchData = async () => {

      const fetchedProducts = await API.getProducts(currentFilters);

      setProducts(fetchedProducts);

    };

    fetchData();

  }, [currentFilters]);

  return (

    <div className="products-page">

      <div className="products-page\_\_container">

        <h2>Поиск по товарам</h2>

        <div className="main-cont">

          <div className="filters">

            <h3>Фильтры</h3>

            <div className="filters\_\_colors">

              <h4>Цвета</h4>

              {filters.colors.map((filter, i) => (

                <CheckboxFilter

                  value={filter}

                  key={i}

                  onCheck={handleCheckColor}

                  checked={currentFilters.colors[filter]}

                />

              ))}

            </div>

            <div className="filters\_\_cars">

              <h4>Автомобили</h4>

              {filters.cars.map((filter, i) => (

                <CheckboxFilter

                  value={filter}

                  key={i}

                  onCheck={handleCheckCar}

                  checked={currentFilters.cars[filter]}

                />

              ))}

            </div>

            <div className="filters\_\_price">

              <h4>Цена</h4>

              <NumberFilter

                text="От"

                id={"min\_price"}

                onChange={handleChangePrice}

                value={currentFilters.price.min\_price}

              />

              <NumberFilter

                text="До"

                id={"max\_price"}

                onChange={handleChangePrice}

                value={currentFilters.price.max\_price}

              />

            </div>

          </div>

          <div className="products">

            {products.map((product) => (

              <Product product={product} key={product.id} />

            ))}

          </div>

        </div>

      </div>

    </div>

  );

}

CartPage.jsx:

import { Link } from "react-router-dom"

import ProductInCart from "../../components/product-in-cart/ProductInCart"

import "./CartPage.css"

import { useEffect, useState } from "react"

import API from "../../api/api"

export default function CartPage() {

    const [cart, setCart] = useState({

      products: [],

      sum: 0

    })

    useEffect(() => {

      const fetchData = async () => {

        const fetchedCart = await API.getCart();

        setCart(fetchedCart)

      }

      fetchData();

    }, [])

    const handleClearCart = () => {

      const fetchData = async () => {

        await API.clearCart();

        setCart({

      products: [],

      sum: 0

    })

      }

      fetchData();

    }

    const handleAddProduct = (id) => () => {

      const fetchData = async () => {

        await API.toCart(id);

        const fetchedCart = await API.getCart();

        setCart(fetchedCart)

      }

      fetchData();

    }

    const handleDeleteProduct = (id, count) => () => {

      const fetchData = async () => {

        await API.deleteFromCart(id, count);

        const fetchedCart = await API.getCart();

        setCart(fetchedCart)

      }

      fetchData();

    }

    return <div className="cart-page">

        <div className="cart-page\_\_container">

          <div className="top-inf">

            <h2>Корзина</h2>

            <div className="cart\_\_side-menu">

              Итого: {cart.sum} BYN

              <div className="side-menu\_\_buttons">

                <div className="btn"><Link to="/orders/new"><button>Оформить заказ</button></Link></div>

                <div className="btn"><button onClick={handleClearCart}>Очистить корзину</button></div>

              </div>

            </div>

            </div>

            <div className="main-cont">

              <div className="products">

                  {cart.products.length ? cart.products.map(product => <ProductInCart product={product} key={product.id} addFcn={handleAddProduct} delFcn={handleDeleteProduct}/>) : <div className="no-orders">Ваша корзина пуста.</div>}

              </div>

            </div>

        </div>

    </div>

}