Geekbrains

Специальность: Frontend-программист “Цифровые профессии”

создание интернет-магазина позволяющего продавать и покупать одежду рассказ о практической пользе технологий приненяемых в ходе создания проекта тестирование магазина

Абрамов Виталий Николаевич

Камышин 2024год

Дипломный проект : создание интернет-магазина позволяющего продавать и покупать одежду рассказ о практической пользе технологий применяемых в ходе создания проекта тестирование магазина

СОДЕРЖАНИЕ

Введение

суть проекта

практическая польза и преимущества выбранных технологий

план разработки выбор стека технологий

краткое описания специалистов которые понадобятся при разработке в команде

Основы веб приложений

frontend часть

backend часть

цели веб приложений

Frontend веб-приложения, его особенности

технологии на frontend приложения

HTML примеры теги достоинство HTML5

CSS основные принципы

SCSS основные особенности

javascript приемущества при создании интернет магазина

Vue CLI и его возможности

Axios nриемущества даной библиотеки в нашем контексте

figma её приемущества

общие требования к фронтенду приложения

Backend веб-приложения, его особенности.

Особенности backenda

Noda.js выбор этой технологии при создании магазина

Express приемущества фреймворка при использовании на бекенде

особенности Express

принципы работы серверной части

об API

Програграммные продукты и инструменты веб разработки

об IDE

Зачем нужна IDE

Примеры паопулярных IDE

Visual Studio

Eclipse

IntelliJ IDEA

Android Studio

VS Code её особености и приемущества

установка библиотек в js

приемуства библиотек в js

установка подключение

об npm плюсы для данного проекта

Об Vue-router настройка подлючени**е**

Vuex настройка подключение

коментарии к некоторым кодам проекта

тестирование проекта

Про test utils
О unit тестировании в контексте интернет-магазина

заключение к дипломному проекту интернет магазина на vue CLI

Список материалов используемый при написании работы

Введение

Суть дипломного проекта по созданию интернет магазин одежды с каталогом корзиной ценами возможностью добавлять товар в корзину.

практическая польза интерактивного интернет магазина одежды на Vue cli

1. Удобство для пользователей: Интерактивный интернет-магазин на Vue CLI может предложить пользователям удобный и интуитивно понятный интерфейс, который улучшит их опыт покупок. Это может включать в себя быструю загрузку страниц, динамические фильтры для поиска товаров, интерактивные элементы, такие как слайдеры и анимации, а также удобную корзину покупок и оформление заказа.

2. Адаптивность и мобильная оптимизация: Использование Vue CLI позволяет создать адаптивный интернет-магазин, который будет хорошо отображаться на различных устройствах, включая мобильные телефоны и планшеты. Это обеспечит удобство использования магазина для пользователей, которые предпочитают покупки через мобильные устройства.

3. Управление контентом: Vue CLI позволяет легко управлять контентом интернет-магазина, включая добавление новых товаров, управление описаниями, изображениями и ценами. Это позволяет владельцам магазина быстро обновлять ассортимент и информацию о товарах.

4. Аналитика и отслеживание: С помощью Vue CLI можно легко интегрировать аналитические инструменты для отслеживания поведения пользователей, конверсий и других метрик. Это поможет владельцам магазина понять, какие товары пользуются наибольшим спросом, какие страницы привлекают больше внимания пользователей и как улучшить процесс покупок.

5. Быстрая разработка и масштабируемость: Использование Vue CLI обеспечивает быструю разработку интерфейса магазина благодаря его гибкой архитектуре и большому сообществу разработчиков. Это также делает магазин более масштабируемым, что позволяет легко добавлять новые функции и расширять его возможности по мере роста бизнеса.

Интерактивный интернет-магазин одежды на Vue CLI может значительно улучшить опыт пользователей, обеспечить удобное управление контентом для владельцев магазина и обеспечить возможность отслеживания и анализа данных для улучшения бизнес-п

моя цель научиться создавать интернет магазины и интернет сайты веб приложения с интерактивм и динамикой

План разработки веб-приложения с использованием HTML, CSS, JS, Vue.js, Axios, Node.js, Express, figma

1. Разработку пользовательского интерфейса (UI) и взаимодействие с пользователем (UX) приложения с помощью HTML, CSS. Пользовательский интерфейс должен быть визуально привлекательным и простым в навигации.

2. Использование Vue.js для добавления интерактивности и динамического контента в пользовательский интерфейс. Vue.js также можно использовать для создания компонентов и обработки пользовательского ввода, что сделает приложение более отзывчивым и удобным для пользователя;
3 Использование Node.js и Express для создания сервера, который обрабатывает HTTP-запросы и предоставляет клиенту файлы HTML, CSS и JS

Проект будет разрабатываться мной в одиночку. Однако, с точки зрения создания и запуска веб-приложения, в подобных проектах могут участвовать следующие разработчики:

1. Веб-разработчики, обладающие знаниями в области HTML, CSS и JavaScript, могут помочь создать внешний пользовательский интерфейс и добавить интерактивность в приложение с помощью таких фреймворков, как Vue.js;
2. Backend разработчики с опытом работы с Node.js, Express и управлением базами данных могут помочь создать инфраструктуру на стороне сервера, конечные точки API и подключения к базе данных;
3. Инженеры данных могут помочь со сбором, очисткой и хранением данных. Они могут помочь разработать и внедрить конвейеры данных для сбора информации из различных источников, таких как книжные интернет-магазины;
4. QA Инженеры могут помочь с тестированием приложения на наличие ошибок и проблем. Они могут создавать тестовые случаи и сценарии, чтобы убедиться, что приложение работает должным образом;
5. UX-дизайнеры могут помочь с разработкой удобного и визуально привлекательного интерфейса для приложения. Они также могут предоставить рекомендации по передовым методам использования и доступности;
6. Инженеры DevOps могут помочь с развертыванием приложения в производственной среде.

   При разработке веб приложения важно чтобы группа разработчиков обладала разным набором навыков и опыта Сотрудничество и общение между членами команды имеют решающее значение для обеспечения того, чтобы проект соответствовал требованиям и был выполнен вовремя.

Создание интернет магазина потребует совокупности навыкоков программирования , знаний html-css js фреймворка vue библиотек axion и Express,figma

<u>основы веб приложения</u>

Веб-приложение - это программное обеспечение, которое работает веб-браузере и предоставляет пользователю доступ к различным функциям и сервисам через интернет. Веб-приложения могут быть разработаны для широкого спектра целей, включая социальные сети, онлайн-магазины, банковские системы, образовательные платформы, игры и многое другое. Основные компоненты веб-приложения включают:

1. Клиентская часть (Frontend): Клиентская часть веб-приложения выполняется в браузере пользователя и обычно написана на языках HTML, CSS и JavaScript. Она отвечает за отображение пользовательского интерфейса, взаимодействие с пользователем и отправку запросов на сервер.

2. Серверная часть (Backend): Серверная часть веб-приложения выполняется на сервере и обрабатывает запросы от клиентской части. Она может быть написана на различных языках программирования, таких как JavaScript (Node.js), Python, Ruby, Java, PHP и других. Серверная часть обычно отвечает за обработку данных, выполнение бизнес-логики, работу с базами данных и отправку данных клиентской част

3. База данных: Многие веб-приложения хранят данные в базе данных, которая может быть расположена на сервере или в облаке. База данных используется для хранения информации о пользователях, продуктах, заказах, сообщениях и других данных, необходимых для работы приложения.

4. API (Application Programming Interface): API предоставляет интерфейс для взаимодействия между клиентской и серверной частями приложения. Он определяет формат запросов и ответов, а также доступные операции.

Примеры популярных технологий для разработки веб-приложений включают HTML, CSS, JavaScript, React, Angular, Vue.js для клиентской части, а также Node.js, Express.js, Django, Flask для серверной части.

Разработка веб-приложений требует понимания основ веб-технологий, архитектуры приложений, безопасности, производительности и удобства использования для пользователей.

<u>Цели веб приложений</u>

Цели веб-приложений могут быть разнообразными и зависят от конкретного приложения, его назначения и целевой аудитории. Однако в общем виде можно выделить несколько основных целей, которые преследуют веб-приложения:

1. Улучшение доступности: Веб-приложения позволяют пользователям получать доступ к сервисам и функциям из любого места, где есть интернет. Это повышает удобство использования и улучшает доступность для широкой аудитории.

2. Увеличение эффективности: Веб-приложения могут автоматизировать процессы, улучшить совместную работу и упростить выполнение задач. Например, онлайн-системы управления проектами или CRM-системы позволяют организовать работу команды и управлять клиентскими данными более эффективно.

3. Расширение рынка: Веб-приложения позволяют компаниям расширить свою аудиторию и проникнуть на новые рынки, предоставляя свои продукты и услуги онлайн.

4. Увеличение уровня сервиса: Веб-приложения могут улучшить уровень обслуживания клиентов, предоставляя им возможность получать информацию, делать заказы, общаться с поддержкой и т.д. через интернет.

5. Сбор и анализ данных: Веб-приложения позволяют собирать данные о поведении пользователей, их предпочтениях, покупках и т.д., что может быть использовано для анализа и улучшения продуктов и услуг.

6. Монетизация: Веб-приложения могут быть использованы для создания новых источников дохода, например, через онлайн-продажи, рекламу, подписки и т.д.

Frontend веб-приложения, его особенности

Фронтенд веб-приложения - это часть приложения, с которой пользователь взаимодействует непосредственно. Основные особенности фронтенда веб-приложения включают:

1. Интерфейс пользователя: Фронтенд веб-приложения отвечает за создание интерфейса, с которым пользователь взаимодействует. Это включает в себя разметку, стилизацию и поведение элементов интерфейса.

2. Визуальный дизайн: Фронтенд отвечает за визуальное оформление приложения, включая выбор цветовой палитры, шрифтов, изображений и других визуальных элементов.

3. Взаимодействие с бэкендом: Фронтенд обеспечивает взаимодействие с бэкендом приложения, отправляя запросы на сервер и обрабатывая полученные данные.

4. Адаптивность: Фронтенд должен быть адаптирован для работы на различных устройствах, таких как компьютеры, планшеты и мобильные телефоны.

5. Оптимизация производительности: Фронтенд должен быть оптимизирован для быстрой загрузки и отзывчивости, чтобы обеспечить плавное пользовательское взаимодействие.

6. Кроссбраузерная совместимость: Фронтенд должен быть совместим с различными браузерами, чтобы обеспечить одинаковое отображение и работоспособность на всех платформах.

7. Использование фреймворков и библиотек: Для упрощения разработки фронтенда часто используются специализированные фреймворки и библиотеки, такие как React, Angular, Vue.js и другие.

Эти особенности фронтенда веб-приложения помогают создать удобный и привлекательный пользовательский интерфейс, который обеспечивает эффективное взаимодействие пользователя с приложением.

В нашем случае фронтендприложения будет представлен технологиями HTML CSS предпроцессорами scss динамика и интерактивность будет обеспечиваться языком программирования javascript его фреймворком Vue

1.HTML (HyperText Markup Language) - это основной язык разметки веб-страниц. Он используется для создания структуры и содержания веб-страницы, определяя различные элементы, такие как заголовки, параграфы, списки, таблицы, изображения, ссылки и многое другое.

HTML состоит из тегов, которые определяют различные элементы страницы. Каждый тег начинается с угловых скобок <>, после которых следует название тега, и заканчивается закрывающей скобкой </>. Например, тег для создания абзаца выглядит так: Текст абзаца .

Пример простой HTML-страницы:

<!DOCTYPE html>

<html>

<head>

```
    <title>Пример HTML-страницы</title>
</head>
<body>
    <h1>Заголовок страницы</h1>
    <p>Это пример абзаца текста.</p>
    <img src="image.jpg" alt="Описание изображения">
    <a href="https://www.example.com">Ссылка на пример</a>
</body>
</html>
```

Здесь:

- <!DOCTYPE html> - объявляет тип документа как HTML5.

- <html> - определяет начало HTML-документа.

- <head> - содержит метаданные страницы, такие как заголовок и ссылки на стили и скрипты.

- <title> - устанавливает заголовок веб-страницы, который отображается в заголовке браузера.

- <body> - содержит основное содержимое страницы.

- <h1> - создает заголовок первого уровня.

- <p> - создает абзац текста.

- <img> - вставляет изображение на страницу.

- <a> - создает гиперссылку.

HTML позволяет создавать структурированный контент для веб-страниц и является основой для большинства веб-сайтов.

HTML5 - это пятая версия языка разметки HTML, который используется для

создания веб-страниц и веб-приложений. HTML5 был разработан с целью улучшения возможностей веб-разработки и поддержки новых технологий. Вот некоторые ключевые особенности HTML5:

1. **Новые элементы**: HTML5 включает множество новых семантических элементов, таких как <header>, <nav>, <section>, <article>, <aside>, <footer>, которые помогают разработчикам более точно определять структуру веб-страницы.

2. **Мультимедийные возможности**: HTML5 предоставляет встроенную поддержку для встраивания аудио и видео без необходимости использования сторонних плагинов, таких как Flash. Это достигается с помощью новых тегов <audio> и <video>.

3. **Графика и анимация**: HTML5 включает элемент <canvas>, который позволяет разработчикам создавать графику, анимацию и другие интерактивные элементы на веб-страницах без использования сторонних плагинов.

4. **Улучшенные формы**: HTML5 предоставляет новые типы полей для форм, такие как email, url, date, которые упрощают работу с формами на веб-страницах.

5. **Локальное хранилище**: HTML5 предоставляет возможность использовать локальное хранилище (localStorage) для хранения данных на стороне клиента, что улучшает производительность и удобство использования веб-приложений.

6. **Поддержка геолокации**: HTML5 предоставляет API для определения местоположения пользователя, что позволяет создавать приложения, использующие геолокацию.

7. **Улучшенная поддержка семантики и доступности**: HTML5 включает новые элементы и атрибуты, которые улучшают семантику веб-страниц и помогают создавать более доступные сайты для пользователей с ограниченными возможностями.

Это лишь некоторые из основных особенностей HTML5. Он представляет собой значительное улучшение по сравнению с предыдущими версиями HTML и открывает новые возможности для создания современных веб-приложений и сайтов

2 CSS (Cascading Style Sheets) - это язык таблиц стилей, который используется для описания внешнего вида документа, написанного на HTML или XML. CSS определяет, как элементы должны отображаться на экране, на бумаге, в речи или на других носителях. Он позволяет веб-разработчикам отделить содержимое и структуру веб-сайта от его визуального представления, что облегчает его поддержку и обновление. CSS можно использовать для управления макетом, цветами, шрифтами и другими визуальными аспектами веб-страницы.

CSS был разработан как ответ на необходимость разделения содержимого и представления веб-страниц. Ранее, стили для HTML-документов определялись непосредственно внутри тегов HTML, что делало код менее читаемым и поддерживаемым. Введение CSS позволило создавать более гибкие и эффективные стили для веб-страниц, что способствовало развитию веб-дизайна.

Основные принципы CSS

- 1. каскадность, которая определяет приоритетность стилей. Это означает, что если один и тот же элемент имеет несколько стилей, то браузер будет использовать стиль с наивысшим приоритетом. Приоритетность определяется с помощью различных факторов, таких как тип селектора, инлайновые стили, классы и идентификаторы.
2. Наследование: Некоторые CSS свойства могут быть унаследованы потомками от их родительских элементов. Например, если вы установите шрифт или цвет текста для родительского элемента, это может быть унаследовано его потомками. Однако не все свойства наследуются, и это зависит от конкретного свойства.

3. Приоритетность: Если один и тот же CSS-свойство определено в разных местах (например, в разных стилях, встроенных стилях и атрибутах style), браузер должен определить, какой стиль применить. Это происходит на основе приоритетности правил:

- Встроенные стили (inline styles) имеют высшую приоритет. Это означает, что если вы устанавливаете стиль непосредственно в атрибуте style элемента, он будет иметь больший приоритет перед другими стилями.

- Селекторы с более высокой специфичностью имеют больший приоритет. Например, селекторы с использованием ID имеют более высокую приоритет,

чем селекторы с использованием классов или элементов.
Понимание этих принципов поможет вам создавать более предсказуемые и эффективные стили для вашего веб-сайта.

- Последнее правило имеет больший приоритет в случае конфликта. Если два правила имеют одинаковую специфичность, то последнее определенное правило будет применено.
CSS позволяет разработчикам создавать адаптивные и отзывчивые дизайны, которые могут адаптироваться к различным размерам экранов и устройств. Это достигается с помощью медиа-запросов, которые позволяют применять различные стили в зависимости от характеристик устройства, таких как ширина экрана или ориентация.

В заключение, CSS является мощным инструментом для создания привлекательных и функциональных веб-сайтов. Он обеспечивает гибкость и контроль над визуальным представлением веб-страницы, что делает его неотъемлемой частью веб-разработк

3 SCSS (Sassy CSS) - это предварительный процессор CSS, который добавляет множество дополнительных функций и возможностей к обычному CSS. SCSS является расширением языка CSS и предоставляет более гибкий и удобный способ написания стилей для веб-сайтов.

Основные особенности SCSS:

1. Переменные: SCSS позволяет определять переменные для цветов, размеров шрифтов, отступов и других значений, что делает код более читаемым и облегчает его поддержку.

2. Вложенность: SCSS позволяет использовать вложенные структуры для организации стилей, что упрощает чтение и понимание кода.

3. Миксины: SCSS позволяет создавать миксины - наборы стилей, которые можно повторно использовать в различных частях кода.

4. Вложенные селекторы: SCSS позволяет использовать вложенные селекторы, что делает код более структурированным и удобным для работы.

5. Импорт файлов: SCSS позволяет импортировать один SCSS-файл в другой, что упрощает организацию и управление большими наборами стилей.

6. Математические операции: SCSS позволяет выполнять математические операции непосредственно внутри стилей, что облегчает расчеты размеров и отступов.

Для использования SCSS необходимо скомпилировать его в обычный CSS с помощью специальных инструментов, таких как Sass или Less. После компиляции SCSS-код превращается в обычный CSS, который может быть использован на веб-сайте.
4 *JavaScript*
JavaScript является одним из самых популярных языков программирования, используемых в веб-разработке. Его гибкость и возможность расширения делают его идеальным выбором для создания интерактивных и динамических веб-страниц. Благодаря использованию библиотек JavaScript, разработчики могут значительно ускорить процесс разработки и улучшить функциональность своих проектов

JavaScript имеет множество преимуществ при создании интернет-магазина на **5 5** Vue CLI. Вот некоторые из них:

1. Интерактивность: JavaScript позволяет создавать интерактивные элементы на веб-страницах, такие как корзина покупок, фильтры, динамические списки товаров и многое другое. Это делает пользовательский опыт более удобным и привлекательным для посетителей интернет-магазина.

2. Асинхронные запросы: JavaScript позволяет делать асинхронные запросы к серверу, что особенно полезно при загрузке данных о товарах, оформлении заказов и обновлении информации о наличии товаров без перезагрузки страницы.

3. Vue CLI и Vue.js: Vue CLI предоставляет удобное окружение для разработки веб-приложений на Vue.js, который является одним из самых популярных JavaScript-фреймворков. Vue.js обеспечивает удобное управление состоянием приложения, создание компонентов и маршрутизацию, что делает процесс создания интернет-магазина более эффективным.

4. Модульность и компонентный подход: JavaScript позволяет использовать модули и компоненты для организации кода интернет-магазина. Это упрощает поддержку и масштабирование приложения, поскольку каждый компонент может быть разработан, протестирован и поддерживаться отдельно.

5. Экосистема инструментов: JavaScript имеет обширную экосистему инструментов и библиотек, которые могут быть использованы при создании интернет-магазина. Например, с помощью библиотеки Axios можно удобно выполнять HTTP-запросы, а библиотека Vuex обеспечивает удобное управление состоянием приложения.

6. Отзывчивость и скорость: JavaScript позволяет создавать отзывчивые интерфейсы и оптимизировать производительность интернет-магазина, что особенно важно для пользователей, ожидающих быструю загрузку страниц и мгновенные реакции на действия.

В целом, JavaScript является мощным инструментом для создания интернет-магазина на Vue CLI, обеспечивая высокую интерактивность, гибкость и производительность веб-приложения.

5.Vue CLI (Command Line Interface) - это официальный инструмент командной строки для быстрой разработки Vue.js приложений. Он предоставляет разработчикам набор инструментов и конфигураций для создания проектов на Vue.js с использованием современных практик разработки.

Вот некоторые ключевые возможности Vue CLI:

1. **Генерация проектов**: Vue CLI позволяет быстро создавать новые проекты на Vue.js с помощью предварительно сконфигурированных шаблонов. Это упрощает начало работы с новым проектом и позволяет разработчикам сосредоточиться на кодировании, а не на настройке проекта.

2. **Локальная разработка**: Vue CLI включает в себя встроенный сервер разработки, который позволяет разработчикам легко запускать и тестировать свои приложения локально перед развертыванием.

3. **Сборка и оптимизация**: Vue CLI предоставляет инструменты для сборки и оптимизации проектов перед развертыванием. Это включает в себя минификацию кода, объединение файлов, оптимизацию изображений и другие методы улучшения производительности.

4. **Плагины и конфигурации**: Vue CLI поддерживает плагины, которые могут расширять функциональность инструмента, а также конфигурацию проекта через файлы vue.config.js для более гибкой настройки процесса сборки.

5. **Расширенные возможности**: Vue CLI также предоставляет

дополнительные возможности, такие как поддержка TypeScript, автоматическое тестирование с помощью Jest или Mocha, интеграция с системами контроля версий и многое другое.

Vue CLI является мощным инструментом для разработки приложений на Vue.js и облегчает процесс создания, тестирования и развертывания проекта или веб приложения

5.Vue.js - это прогрессивный фреймворк JavaScript для создания пользовательских интерфейсов и приложений. Он разработан для упрощения разработки веб-приложений, обеспечивая эффективную работу с пользовательским интерфейсом и управление состоянием приложения.

Vue.js имеет несколько преимуществ при создании интернет-магазина:

1. Простота в изучении и использовании: Vue.js обладает простым и интуитивно понятным синтаксисом, что делает его легким для изучения и использования даже для новичков. Это позволяет быстро разрабатывать новые функции и компоненты интернет-магазина.

2. Реактивность: Vue.js предоставляет реактивные компоненты, что означает, что изменения данных автоматически обновляют отображение в пользовательском интерфейсе. Это особенно полезно при работе с корзиной покупок, фильтрами товаров и другими динамическими элементами интернет-магазина.

3. Однофайловые компоненты: Vue.js позволяет создавать компоненты в одном файле, объединяя HTML, CSS и JavaScript. Это упрощает организацию кода и поддержку компонентов интернет-магазина.

4. Удобная маршрутизация: Vue Router обеспечивает удобное управление маршрутами в интернет-магазине, что позволяет легко переключаться между страницами и создавать динамические маршруты для различных категорий товаров.

5. Управление состоянием с помощью Vuex: Vuex предоставляет удобный способ управления состоянием приложения, таким как данные о товарах, корзина покупок и информация о пользователе. Это делает код более чистым, структурированным и легким для поддержки.

6. Экосистема плагинов и инструментов: Vue.js имеет обширную экосистему

плагинов и инструментов, которые могут быть использованы для добавления различных функций в интернет-магазин. Например, плагины для управления состоянием, анимаций, форм и многое другое.

В целом, Vue.js обеспечивает удобную разработку интернет-магазина благодаря своей простоте, гибкости и широкому набору инструментов для создания отзывчивых и функциональных пользов

6. axios
Axios - это библиотека JavaScript для выполнения HTTP-запросов из браузера или Node.js. Она имеет несколько преимуществ при создании интернет-магазина на Vue CLI:

1. Простота использования: Axios предоставляет простой и интуитивно понятный API для выполнения HTTP-запросов, что делает его легким для изучения и использования.

2. Удобство работы с REST API: Многие интернет-магазины используют RESTful API для взаимодействия с сервером. Axios предоставляет удобные методы для выполнения запросов к REST API, такие как GET, POST, PUT, DELETE и другие.

3. Поддержка обещаний (Promises): Axios основан на обещаниях, что делает его удобным для работы с асинхронными запросами. Это позволяет легко управлять последовательностью запросов и обработкой ошибок.

4. Интерсепторы запросов и ответов: Axios позволяет добавлять интерсепторы для запросов и ответов, что обеспечивает возможность манипулировать данными до отправки или после получения ответа. Например, можно добавить заголовки авторизации к каждому запросу или глобальную обработку ошибок.

5. Поддержка отмены запросов: Axios предоставляет возможность отменять запросы, что полезно при работе с динамическими элементами интернет-магазина, такими как поиск или фильтры товаров.

6. Интеграция с Vue.js: Axios легко интегрируется с Vue.js и может использоваться вместе с Vue CLI для выполнения HTTP-запросов из компонентов приложения.

В целом, использование Axios при создании интернет-магазина на Vue CLI обеспечивает удобную работу с сервером, выполнение HTTP-запросов и

управление данными, что делает его отличным выбором для взаимодействия с внешними ресурсами.

Figma

Figma - это мощный инструмент для дизайна интерфейсов, который может быть очень полезен при frontend-разработке интернет-магазина на Vue CLI. Вот несколько преимуществ использования Figma для этой цели:

1. Дизайн-ориентированный подход: Figma предоставляет широкие возможности для создания дизайна пользовательского интерфейса, включая создание макетов, прототипов, анимаций и многого другого. Это позволяет frontend-разработчикам получить четкое представление о том, как должен выглядеть и вести себя интернет-магазин, что упрощает процесс реализации дизайна.

2. Коллаборация и командная работа: Figma обеспечивает возможность работы над проектом в режиме реального времени, что позволяет дизайнерам и разработчикам совместно работать над интерфейсом интернет-магазина. Это улучшает коммуникацию и согласование между участниками проекта.

3. Экспорт ресурсов: Figma позволяет легко экспортировать графические ресурсы, такие как иконки, изображения и другие элементы дизайна, в форматах, удобных для использования в коде frontend-разработки. Это упрощает процесс интеграции дизайна в код приложения.

4. Поддержка компонентов и стилей: Figma позволяет создавать и использовать компоненты и стили, что облегчает поддержку единого стиля и повторное использование элементов интерфейса в различных частях интернет-магазина.

5. Прототипирование: Figma предоставляет возможность создания интерактивных прототипов, которые могут быть использованы для тестирования пользовательского опыта перед фактической разработкой. Это помогает выявить потенциальные проблемы и улучшить пользовательский интерфейс до начала разработки.

Использование Figma при frontend-разработке интернет-магазина на Vue CLI поможет создать качественный и современный пользовательский интерфейс, упростить командную работу и улучшить процесс интеграции дизайна в код приложения

Frontend должен иметь понятный и простой в использовании интерфейс, который позволяет пользователям искать книги и сравнивать цены в нескольких книжных онлайн-магазинах. Пользовательский интерфейс должен быть разработан таким образом, чтобы он реагировал и адаптировался к различным размерам экрана и устройствам.

Интерфейс должен отображать обновления в реальном времени для пользователей, когда данные из бэкэнда изменяются, например, когда добавляется информация о новом книге или когда меняются цены на товар.

Пользовательский интерфейс должен быть разработан с учетом высокой производительности и быстрой загрузкой страниц, а также минимальной задержкой.

Внешний интерфейс должен быть совместим с различными веб-браузерами и операционными системами, плюс предназначен для работы на мобильных устройствах, а также на настольных компьютерах.

Пользовательский интерфейс должен быть разработан с учетом безопасности, чтобы защитить личную информацию пользователей за счет использования безопасных протоколов, таких как HTTPS.

Backend веб-приложения, его особенности.

Backend интернет-магазина одежды на Vue может включать в себя следующие особенности:

1. Управление базой данных: Создание и управление базой данных для хранения информации о продуктах, заказах, пользователях и других необходимых данных.

2. API разработка: Разработка API для обмена данными между фронтендом и бэкендом, что позволит фронтенду получать необходимую информацию и отправлять данные обратно на сервер.

3. Аутентификация и безопасность: Реализация системы аутентификации и авторизации пользователей, защита данных и обеспечение безопасности транзакций, включая защиту от CSRF-атак и XSS.

4. Управление заказами и платежами: Разработка функциональности для управления заказами, обработки платежей и интеграции с платежными системами.

5. Административная панель: Создание административной панели для управления товарами, заказами, пользователями и другими аспектами интернет-магазина.

6. Масштабируемость и производительность: Оптимизация бэкенда для обеспечения высокой производительности и масштабируемости, чтобы обрабатывать большие объемы данных и пользовательских запросов.

Интернет-магазин одежды на Vue и backend в целом должен быть разработан с упором на безопасность, производительность и удобство использования, чтобы обеспечить хороший опыт покупателя и эффективное управление продажами.В нашем случае интернет-магазин серверная часть будет состоять из следующих компонентов:

Node.js — это среда выполнения JavaScript, которая позволяет запускать код JavaScript на стороне сервера веб-приложения. Он будет использоваться для обработки логики на стороне сервера и предоставления платформы для запуска кода на стороне сервера.

Node.js предоставляет ряд функций которые делают его популярным выбором для создания масштабируемых высокопроизводительных веб-приложений, в том числе:

1.Неблокирующая модель ввода-вывода (Non-blocking I/O model): Node.js использует управляемую событиями неблокирующую модель ввода-вывода, которая позволяет ему обрабатывать большое количество подключений с небольшим количеством потоков.

2. Единая языковая среда: Использование Node.js позволяет разработчикам использовать JavaScript как на клиентской, так и на серверной стороне. Это обеспечивает единый язык программирования для разработки как фронтенда (Vue.js), так и бэкенда (Node.js).

3. Высокая производительность: Node.js обеспечивает высокую производительность благодаря асинхронной обработке запросов. Это позволяет эффективно обрабатывать большое количество одновременных запросов, что особенно важно для интернет-магазина с большим количеством пользователей.

4. Масштабируемость: Node.js обладает возможностью горизонтального масштабирования, что означает, что при увеличении нагрузки можно легко добавлять новые серверы для распределения нагрузки.

5. Экосистема пакетов: Node.js имеет огромную экосистему пакетов (npm), которая предлагает множество готовых решений для различных задач, таких как обработка платежей, управление сессиями пользователей, работа с базами данных и другие.

6. Разработка API: Node.js хорошо подходит для создания API, которые могут взаимодействовать с фронтендом на Vue.js. Это позволяет создавать мощные и гибкие API для взаимодействия с клиентской частью магазина.

Использование Node.js в сочетании с Vue.js может значительно упростить разработку интернет-магазина, обеспечивая единое окружение выполнения JavaScript как на клиентской, так и на серверной стороне, а также высокую производительность и масштабируемость.

Express

Express - это фреймворк для Node.js, который облегчает создание мощных веб-приложений. Когда он используется в качестве бэкенд-фреймворка интернет-магазина, с фронтендом на Vue, есть несколько особенностей и преимуществ.

Express является одним из самых популярных и широко используемых фреймворков для создания веб-приложений на Node.js. Он предоставляет простой и эффективный способ создания серверных приложений и API, обеспечивая набор функций и инструментов для упрощения разработки.

Вот несколько ключевых особенностей Express:

1. Маршрутизация: Express предоставляет мощный механизм маршрутизации, который позволяет определять обработчики запросов для различных URL-адресов и HTTP-методов. Это делает управление маршрутами и обработкой запросов очень удобным.

2. Middleware: Express использует концепцию middleware, которая позволяет добавлять промежуточные обработчики к запросам. Это может быть полезно для реализации функциональности, такой как аутентификация, логирование, обработка ошибок и другие.

3. Шаблонизация: Express имеет поддержку различных движков шаблонов, таких как EJS, Pug и Handlebars, что позволяет удобно генерировать HTML на сервере и отправлять его клиенту.

4. Статические файлы: С помощью Express легко настроить обслуживание статических файлов, таких как изображения, CSS и JavaScript, что упрощает создание веб-приложений с клиентской частью.

5. Расширяемость: Express предоставляет возможность расширять функциональность с помощью сторонних модулей и middleware, что делает его гибким и мощным инструментом для разработки.

Использование Express в сочетании с Node.js позволяет создавать мощные и эффективные веб-приложения и API, обеспечивая удобство разработки, гибкость и высокую производительность.

Серверная часть интернет-магазина одежды на Vue с бэкендом Node.js и Express будет работать следующим образом:

1. Фронтенд на Vue будет отправлять запросы на серверную часть, например, для получения списка товаров, добавления товара в корзину или оформления заказа.

2. Серверная часть на Node.js с использованием Express будет принимать эти запросы, обрабатывать их и возвращать соответствующие данные. при получении запроса на получение списка товаров, сервер будет обращаться к базе данных, извлекать необходимую информацию и отправлять её обратно на клиент.

3. Для взаимодействия с базой данных (например, для хранения информации о товарах, пользователях и заказах)

4. Также серверная часть будет отвечать за аутентификацию и авторизацию пользователей, обработку платежей и другие серверные операции, связанные с функционалом интернет-магазина.

Таким образом, серверная часть на Node.js с Express будет обеспечивать взаимодействие фронтенда с базой данных и другими внешними сервисами, обеспечивая полноценное функционирование интернет-магазина.

3. Удобство разработки: Express предлагает интуитивно понятный синтаксис и хорошую документацию, что упрощает процесс разработки бэкенд-части интернет-магазина.

4. Контроль безопасности: С помощью Express легко настроить механизмы безопасности, такие как аутентификация и авторизация, что важно для защиты данных интернет-магазина.

Использование Express в качестве бэкенд-фреймворка для интернет-магазина на Vue обеспечивает надежность, производительность и удобство разработки, что делает его привлекательным выбором для создания мощного и эффективного магазина в сети

Подключение к API в контексте интернет-магазина может быть использовано для получения данных о товарах, обработки платежей,

отправки уведомлений и многих других операций. Вот как это может выглядеть в контексте серверной части интернет-магазина на Node.js с Express:

1. Получение данных о товарах: Если ваш интернет-магазин использует сторонний поставщик товаров, вы можете подключиться к его API, чтобы получать информацию о наличии, ценах и других характеристиках товаров. Для этого вы можете использовать библиотеку axios или для отправки HTTP-запросов к API поставщика и обработки полученных данных.

2. Обработка платежей: Если ваш интернет-магазин принимает онлайн-платежи, вы можете подключиться к API платежного шлюза (например, Stripe, PayPal или другого провайдера) для проведения транзакций. Для этого вам потребуется использовать соответствующие библиотеки или SDK, предоставленные провайдером платежей, и реализовать соответствующие методы для выполнения платежей через API.

3. Отправка уведомлений: Вы также можете использовать API сторонних сервисов для отправки уведомлений о статусе заказа, доставке или других событиях в интернет-магазине. Например, вы можете подключиться к сервису отправки SMS-уведомлений или электронной почты через их API, чтобы автоматически уведомлять покупателей о статусе их заказов.

Для работы с API в Node.js вы можете использовать библиотеки для отправки HTTP-запросов (например, axios, node-fetch), а также соответствующие SDK или библиотеки, предоставленные провайдерами API (например, Stripe SDK для работы с API Stripe).

Таким образом, подключение к API позволяет расширить функциональность вашего интернет-магазина, интегрируя его с внешними сервисами и получая доступ к дополнительным данным и возможностям.

Программные продукты и инструменты web-разработки

Интегрированная среда разработки — один из основных инструментов для разработчика. В статье рассказываем, что это такое и зачем его используют при написании кода.

IDE, или Integrated Development Environment, переводится как "интегрированная среда разработки". Это набор ПО для создания кода. В него входят специальный редактор для кодинга и инструменты, которые помогают запускать, тестировать и отлаживать код.

Новички часто путают среды разработки и редакторы кода. IDE включает в себя текстовый редактор, но предоставляет больше возможностей для подключения языков, запуска и отладки кода, компиляции, сборки проекта и т.д.

Интегрированная среда разработки создает окружение, внутри которого разработчик может заниматься кодингом, не отвлекаясь на технические вопросы.

Среды разработки бывают бесплатными и платными, с открытым и закрытым исходным кодом, под разные языки программирования, технологии и задачи.

Кто и зачем пользуется IDE?

IDE пользуется большинство разработчиков на разных языках в больших и маленьких задачах, когда важно, чтобы все возможности были под рукой.

Среда дает возможность:

•писать, просматривать, запускать и отлаживать код внутри одного окна;

•редактировать код и частично автоматизировать его рефакторинг;

•компилировать код, собирать проект;

•тестировать написанное;

•быстро искать, устанавливать и подключать новые компоненты;

•управлять проектами, создавать их и редактировать;

•работать с системами контроля версий;

•создавать визуальный интерфейс — если в IDE входит визуальный редактор, как в Visual Studio;

•выполнять другие задачи — в зависимости от назначения среды и языка.
IDE делает разработку удобнее и быстрее.

Приведем примеры нескольких популярных сред разработки.

Примеры популярных IDE

Visual Studio — большая и известная мультиязычная IDE. Поддерживает около десятка языков программирования. Этот продукт разработали в Microsoft, поэтому его используют при создании ПО под Windows. Но в Visual Studio можно писать код и под другие ОС. Visual Studio включает в себя редакторы кода и графического интерфейса.

Eclipse — популярная IDE для разработки под Java. Работает на базе виртуальной машины Java и существует для всех распространенных ОС: Windows, Linux, macOS и Solaris.

Это бесплатная, с открытым исходным кодом среда, которую поддерживает сообщество разработчиков.

IntelliJ IDEA — используется для разработки на Java. Предоставляет множество инструментов для автоматизированного рефакторинга. Среда существует в платной и бесплатной версиях.

Android Studio — специализированная среда для разработки под Android. Предлагает возможности для мобильной разработки. Автор Android Studio — компания Google.

Среда разработки — один из основных инструментов при написании кода. Выбирают ее под конкретные задачи. В первую очередь определяют, для какого языка она нужна. текущую ОС, с которой вы работаете.

Visual Studio Code (VS Code) - это популярная бесплатная интегрированная среда разработки (IDE) от Microsoft, которая предоставляет широкий набор функций для разработки различных типов приложений. Вот ключевые особенности Visual Studio Code:

1. Многоязычная поддержка: VS Code поддерживает большое количество языков программирования, включая JavaScript, TypeScript, Python, C++, Java, HTML, CSS и многие другие. Он также предлагает подсветку синтаксиса, автодополнение кода, отладку и другие инструменты для каждого из этих языков.

2. Расширяемость: VS Code имеет огромное сообщество разработчиков, которые создают расширения для практически любых нужд. Вы можете установить расширения для поддержки конкретных языков, интеграции с различными фреймворками и библиотеками, а также для улучшения рабочего процесса разработки.

3. Интеграция с Git: VS Code предоставляет интеграцию с системой контроля версий Git, что позволяет вам управлять репозиториями, выполнять коммиты, отслеживать изменения и управлять ветками прямо из среды разработки.

4. Отладка: Встроенные инструменты отладки в VS Code позволяют вам запускать и отлаживать код прямо из IDE. Вы можете устанавливать точки останова, отслеживать значения переменных и выполнять шаги выполнения кода.

5. Интеграция с терминалом: VS Code включает в себя встроенный терминал, который позволяет выполнять команды командной строки прямо из среды разработки.

6. Различные темы и настройки: Вы можете настроить внешний вид и поведение VS Code с помощью различных тем оформления и настроек.

7. Live Share: Функция Live Share позволяет вам совместно работать над кодом с другими разработчиками, обеспечивая возможность совместной отладки и редактирования кода в реальном времени.

8. Расширенные возможности работы с Docker: VS Code предоставляет интеграцию с Docker, что облегчает работу с контейнеризацией приложений.

Установка библиотек в js

JavaScript является одним из самых популярных языков программирования, используемых в веб-разработке. Его гибкость и возможность расширения делают его идеальным выбором для создания интерактивных и динамических веб-страниц. Благодаря использованию библиотек JavaScript, разработчики могут значительно ускорить процесс разработки и улучшить функциональность своих проектов.

Установка и использование библиотек в JavaScript несложны, но требуют некоторых знаний и навыков.
Перед тем, как начать использовать библиотеку JavaScript, необходимо ее установить. Обычно, это делается с помощью менеджера пакетов, такого как npm (Node Package Manager). Для установки библиотеки вам понадобится установленный Node.js на вашем компьютере. После этого вы можете открыть командную строку и выполнить команду установки библиотеки.

Библиотеки JavaScript представляют собой наборы готовых компонентов и функций, которые разработчики могут использовать в своих проектах. Они позволяют значительно упростить и ускорить процесс разработки, так как не требуют написания кода с нуля.

Для того чтобы использовать библиотеки JavaScript в своем проекте,

первым шагом является их установка. Это может быть произведено с помощью пакетного менеджера NPM (Node Package Manager) или скачиванием и подключением файлов библиотеки в проект вручную Источник: https://idea-tc.ru/kak-pravilno-ustanovit-i-effektivno-ispolzovat-biblioteki-v-javascript

После установки библиотеки в проект, необходимо подключить ее к HTML-странице с помощью тега <script>. Размещение тега <script> может быть выполнено либо внутри <head>, либо перед закрывающим тегом </body>. В зависимости от библиотеки, может потребоваться также указать путь к файлу библиотеки.

После успешного подключения библиотеки, ее функциональность становится доступной в JavaScript-коде проекта. Многие библиотеки предоставляют API с набором методов, с помощью которых разработчики могут взаимодействовать с функциональностью библиотеки. Для использования этих методов необходимо следовать документации по конкретной библиотеке.

Использование библиотек JavaScript может значительно упростить и ускорить разработку веб-приложений. Благодаря готовым компонентам и функциям, разработчики могут сосредоточиться на реализации бизнес-логики и не тратить время на написание базового функционала с нуля. Одна из ключевых особенностей библиотек JavaScript — их модульность и возможность переиспользования кода. Разработчик может выбирать и подключать только необходимые модули, что позволяет уменьшить объем загружаемого кода и повысить производительность приложения Одним из основных преимуществ использования библиотек является экономия времени и усилий разработчика. Вместо того чтобы писать код

с нуля, можно воспользоваться уже готовыми решениями, что позволяет значительно сократить время разработки и повысить производительность

Еще одним важным преимуществом использования библиотек является поддержка и обновление. Популярные библиотеки обычно активно поддерживаются сообществом разработчиков и регулярно обновляются. Это значит, что вам не придется самостоятельно поддерживать код и следить за изменениями в языке. Вместо этого, вы можете регулярно обновлять библиотеку и получить новые функции и исправления без необходимости внесения значительных изменений в свой код Наиболее популярными пакетными менеджерами в JavaScript являются npm (Node Package Manager) и Yarn. Установка библиотеки с помощью npm или Yarn позволяет легко добавить её в проект и начать её использовать без необходимости скачивать и устанавливать её вручную

Для установки библиотеки с помощью npm необходимо открыть командную строку или терминал в корневой папке проекта и выполнить команду:

```
npm install имя_библиотеки
```

Об npm

npm (Node Package Manager) является инструментом, который используется для управления зависимостями в проектах на Vue CLI. Когда вы создаете проект на Vue CLI, он создает файл package.json, в котором перечислены все зависимости вашего проекта. Вы можете использовать npm для установки новых пакетов, обновления существующих и управления версиями зависимостей.

В контексте интернет-магазина на Vue CLI, вы можете использовать npm для установки пакетов, которые помогут вам создать функциональность вашего магазина, такие как пакеты для работы с API, управления состоянием приложения, роутинга и другие. Кроме того, вы можете использовать npm для управления стилями, например, установки пакетов для работы с CSS или препроцессорами.

Например, если вы хотите добавить функциональность корзины покупок в ваш интернет-магазин на Vue CLI, вы можете использовать npm для установки пакетов, которые помогут

вам в этом, такие как vuex для управления состоянием приложения или axios для работы с API вашего магазина.

В целом, npm является важным инструментом для управления зависимостями и разработки приложений на Vue CLI, включая интернет-магазины.

В контексте интернет-магазина на Vue CLI, вы можете использовать npm для установки пакетов, которые помогут вам создать функциональность вашего магазина, такие как пакеты для работы с API, управления состоянием приложения, роутинга и другие. Кроме того, вы можете использовать npm для управления стилями, например, установки пакетов для работы с CSS или препроцессорами.

Например, если вы хотите добавить функциональность корзины покупок в ваш интернет-магазин на Vue CLI, вы можете использовать npm для установки пакетов, которые помогут вам в этом, такие как vuex для управления состоянием приложения или axios для работы с API вашего магазина.

В целом, npm является важным инструментом для управления зависимостями и разработки приложений на Vue CLI, включая интернет-магазины.

Об Vue-router настройка подлючение

Vue-router - это официальнAndroid Studioый маршрутизатор для Vue.js, который позволяет управлять навигацией в вашем приложении. В контексте интернет-магазина на Vue CLI, вы можете использовать Vue-router для создания маршрутов для различных страниц вашего магазина, таких как главная страница, страница каталога, страница товара, корзина и т.д.

Чтобы начать использовать Vue-router в вашем интернет-магазине на Vue CLI, вам нужно сначала установить его с помощью npm: npm install vue-router

Затем вы можете создать файл router.js, где вы определите все маршруты для вашего приложения. Например:

```
import Vue from 'vue'
import Router from 'vue-router'
import Home from './views/Home.vue'
import Catalog from './views/Catalog.vue'
import Product from './views/Product.vue'

Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '/',
      name: 'home',
      component: Home
    },
```

```
    {
      path: '/catalog',
      name: 'catalog',
      component: Catalog
    },
    {
      path: '/product/:id',
      name: 'product',
      component: Product
    }
  ]
})
```

Затем вы можете подключить ваш роутер к главному компоненту вашего приложения:

```
import Vue from 'vue'
import App from './App.vue'
import router from './router'

new Vue({
  router,
  render: h => h(App)
}).$mount('#app')
```

Теперь у вас есть маршруты для главной страницы, каталога и страницы товара. Вы можете использовать Vue-router для создания динамических маршрутов, передавая параметры в URL (например, id товара) и обрабатывая их в соответствующих компонентах.

Vuex - это официальное состояние управления для Vue.js, которое позволяет управлять состоянием вашего приложения централизованно. В контексте интернет-магазина на Vue CLI, вы можете использовать Vuex для управления состоянием, таким как данные о товарах, информация о пользователе, содержимое корзины и т.д.

Чтобы начать использовать Vuex в вашем интернет-магазине на Vue CLI, вам нужно сначала установить его с помощью npm:

```
npm install vuex
```

Затем вы можете создать файл store.js, где вы определите все модули и состояния для вашего приложения. Например:

```
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex)
```

```
export default new Vuex.Store({
  state: {
    products: [],
    user: null,
    cart: []
  },
  mutations: {
    setProducts(state, products) {
      state.products = products
    },
    setUser(state, user) {
      state.user = user
    },
    addToCart(state, product) {
      state.cart.push(product)
    }
    // другие мутации
  },
  actions: {
    fetchProducts({ commit }) {
      // Здесь можно делать запросы к API и вызывать мутации для обновления состояния
    },
    // другие действия
  },
  getters: {
    // геттеры для получения данных из состояния
  }
})
```

Затем вы можете подключить ваш store к главному компоненту вашего приложения:

```
import Vue from 'vue'
import App from './App.vue'
import store from './store'

new Vue({
  store,
  render: h => h(App)
}).$mount('#app')
```

Теперь у вас есть централизованное состояние для управления данными о товарах, информацией о пользователе, содержимым корзины и т.д. Вы можете использовать Vuex для обновления состояния с помощью мутаций и действий, а также получать данные из состояния с помощью геттеров.

Комментарии к некоторым кодам моего проекта

```
const path = require("path");

const fs = require("fs");
const express = require("express");
const app = express();

const port = 3000;

const catalog_path = path.resolve(__dirname, "./data/products.json");
const cart_path = path.resolve(__dirname, "./data/cart.json");
const static_dir = path.resolve(__dirname, "../dist/");

app.use(express.static(static_dir));
app.use(express.json());

app.get("/api/v1/catalog", (req, res) => {
 fs.readFile(catalog_path, "utf-8", (err, data) => {
  if (!err) {
   res.send(data);
  } else {
   res.status(500).send(err);
  }
 });
});

app.get("/api/v1/product/:id", (req, res) => {
 fs.readFile(catalog_path, "utf-8", (err, data) => {
  if (!err) {
   const products = JSON.parse(data);
   products.forEach((value) => {
    if (value.id == req.params.id) {
     res.send(JSON.stringify(value));
```

```
    }
   });
   //console.log(products);
   //res.send(JSON.stringify(product));
  } else {
   res.status(500).send(err);
  }
 });
});

app.get("/api/v1/cart", (req, res) => {
 fs.readFile(cart_path, "utf-8", (err, data) => {
  if (!err) {
   res.send(data);
  } else {
   res.status(500).send(err);
  }
 });
});

app.post("/api/v1/cart", (req, res) => {
 fs.readFile(cart_path, "utf-8", (err, data) => {
  if (!err) {
   const cart = JSON.parse(data);
   let productCart = cart.find((item) => item.id === req.body.id);
   if (typeof productCart === "undefined") {
    cart.push(req.body);
   }
   fs.writeFile(cart_path, JSON.stringify(cart), "utf-8", () => {
    res.sendStatus(201);
   });
  } else {
   res.status(500).send(err);
  }
 });
});
app.put("/api/v1/cart/:id", (req, res) => {
 fs.readFile(cart_path, "utf-8", (err, data) => {
  if (!err) {
   const cart = JSON.parse(data);

   cart.forEach((product) => {
    if (req.params.id == product.id) {
     product.count = req.body.count;
    }
   });
```

```
    fs.writeFile(cart_path, JSON.stringify(cart), "utf-8", () => {
     res.sendStatus(201);
    });
   } else {
    res.status(500).send(err);
   }
  });
});
app.delete("/api/v1/cart/delete/:id", (req, res) => {
 fs.readFile(cart_path, "utf-8", (err, data) => {
  if (!err) {
   const cart = JSON.parse(data);

   cart.forEach((value, index) => {
    if (req.params.id == value.id) {
     cart.splice(index, 1);
    }
   });
   fs.writeFile(cart_path, JSON.stringify(cart), "utf-8", () => {
    res.send(201);
   });
  } else {
   res.status(500).send(err);
  }
 });
});

app.listen(port, () => {
 console.log(`Example app listening at http://localhost:${port}`);
});
```

<u>комментарий:server index js</u>

1. Импорт необходимых модулей: path, fs, и express.
2. Создание экземпляра приложения Express.
3. Определение порта, на котором будет слушать сервер.
4. Настройка путей для файлов данных каталога и корзины, а также статической директории для обслуживания статических файлов (например, HTML, CSS, JavaScript).
5. Добавление промежуточного ПО для разбора входящих запросов в формате JSON.
6. Определение нескольких конечных точек API с использованием методов HTTP, таких как GET, POST, PUT и DELETE, для обработки запросов, связанных с каталогом и корзиной.

Краткий обзор конечных точек API:
- GET /api/v1/catalog: Возвращает продукты из файла данных каталога.
- GET /api/v1/product/:id: Возвращает конкретный продукт из каталога на основе его ID.
- GET /api/v1/cart: Возвращает товары в корзине.
- POST /api/v1/cart: Добавляет продукт в корзину.
- PUT /api/v1/cart/:id: Обновляет количество продукта в корзине.

- DELETE /api/v1/cart/delete/:id: Удаляет продукт из корзины.

Наконец, запускается сервер Express для прослушивания указанного порта

```
import { createStore } from "vuex";

import Cart from "./cart";
import axios from "axios";

export default createStore({
 state: { products: [], oneProduct: [] },
 mutations: {
  setProducts: (state, products) => {
   state.products = products;
  },
  setOneProduct: (state, product) => {
   state.oneProduct = product;
  },
 },
 actions: {
  loadProducts({ commit }) {
   axios({
    method: "GET",
    url: `/api/v1/catalog`,
    params: {
     //user_key_id: "USER_KEY_ID",
    },
    data: {},
    headers: {
     "Content-type": "application/json; charset=UTF-8",
    },
   })
    .then((response) => {
     commit("setProducts", response.data);
     //console.log(response.data);
    })
    .catch((error) => {
     console.log(error);
    })
    .finally(() => {
     this.loadingDataApi = false;
    });
  },
  loadOneProduct({ commit }, id) {
   axios({
    method: "GET",
```

```
      url: `/api/v1/product/${id}`,
      params: {
        //user_key_id: "USER_KEY_ID",
      },
      data: {},
      headers: {
        "Content-type": "application/json; charset=UTF-8",
      },
    })
      .then((response) => {
        commit("setOneProduct", response.data);
        // console.log(response.data);
      })
      .catch((error) => {
        console.log(error);
      })
      .finally(() => {});
  },
 },
 getters: {
  getProducts: (state) => [...state.products],
  getOneProduct: (state) => state.oneProduct,
 },
 modules: { Cart },
});
```

<u>Комментарий src/store/index.js</u>

Этот код представляет собой Vuex-хранилище для управления состоянием приложения во Vue.js. В этом хранилище определены состояние, мутации, действия и геттеры.

1. state: Определяет начальное состояние приложения, в данном случае - массивы products и oneProduct.

2. mutations: Определяет функции для изменения состояния. В данном случае, setProducts и setOneProduct принимают состояние и данные, и устанавливают соответствующие значения состояния.

3. actions: Содержит асинхронные операции, такие как загрузка данных с сервера. В данном случае, loadProducts и loadOneProduct используют библиотеку Axios для выполнения HTTP-запросов к API и коммитят мутации для обновления состояния при получении ответа.

4. getters: Предоставляет доступ к состоянию приложения через вычисляемые свойства. В данном случае, getProducts возвращает копию массива products, а getOneProduct возвращает объект oneProduct.

5. modules: Позволяет разделить хранилище на модули для более удобного управления

большими приложениями.

Этот код представляет собой основной конфигурационный файл хранилища Vuex и позволяет управлять данными в приложении Vue.js.

s***rc/components/footer.vue***

```
<template>

 <footer class="footer">
  <div class="container footer-wrapper">
   <div class="footer__copyright">© 2021 Brand All Rights Reserved.</div>
   <ul class="footer__social-links">
    <li>
     <a class="footer__social-link" href="#">
      <i class="fab fa-facebook-f" aria-hidden="true"></i
     ></a>
    </li>
    <li>
     <a class="footer__social-link" href="#">
      <i class="fab fa-instagram" aria-hidden="true"></i
     ></a>
    </li>
    <li>
     <a class="footer__social-link" href="#">
      <i class="fab fa-pinterest-p" aria-hidden="true"></i
     ></a>
    </li>
    <li>
     <a class="footer__social-link" href="#">
      <i class="fab fa-twitter" aria-hidden="true"></i
     ></a>
    </li>
   </ul>
  </div>
 </footer>
</template>
<style lang="scss">
.footer {
 background: #222224;
 .footer-wrapper {
  padding: 24px 0;
  display: flex;
  justify-content: space-between;
  align-items: center;
 }
 &__copyright {
  font-size: 16px;
```

```
  line-height: 19px;
  color: #fbfbfb;
 }
 &__social-links {
  display: flex;
 }
 &__social-link {
  display: block;
  width: 32px;
  height: 32px;
  background: #ffffff;
  color: black;
  margin-left: 7px;
  text-align: center;
  padding-top: 6px;
 }
 &__social-link:hover {
  background: #f16d7f;
  i {
   color: white;
  }
 }
}

/*mobile*/
@media (max-width: 767px) {
 .footer-wrapper {
  padding-top: 43px !important;
  flex-direction: column-reverse;
 }
 .footer__social-links {
  padding-bottom: 43px;
 }
}
</style>
```

Шаблон содержит разметку для подвала (footer) веб-страницы, включая ссылки на социальные сети и копирайт.

Стили, указанные в блоке <style>, определяют внешний вид элементов подвала. Например, цвет фона, отступы, шрифты и размеры элементов.

В медиа-запросе @media указаны стили для мобильных устройств с шириной экрана до 767 пикселей. Они изменяют отступы и расположение элементов для лучшей адаптации к маленьким экранам.

Комментарии в коде (//) содержат стили, которые должны быть применены для различных

элементов на планшетах и мобильных устройствах, но они закомментированы и неактивны.

adaptiv scss

```
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

.container {
  padding-right: 0;
  padding-left: 0;
  margin-right: auto;
  margin-left: auto;
  margin-top: 0;
  margin-bottom: 0;
  max-width: 1140px;
}

/* tablet */
@media (max-width: 1180px) {
  .container {
    max-width: calc(768px - 16px * 2);
  }

  //     @include promo-tablet;
  //  @include categories-tablet;
  //  @include features-tablet;
  //  @include feedback-tablet;
  //  @include products-tablet;
  //  @include subheader-tablet;
  //  @include slider-tablet;
  //  @include product-info-tablet;
  //  @include order-tablet;
  //  @include registration-tablet;
  //  @include sort-tablet;
}

/*mobile*/
@media (max-width: 767px) {
  .container {
    max-width: calc(375px - 8px * 2);
  }

  .sm-hidden {
    display: none;
  }
```

```
  // @include promo-mobile;
  // @include categories-mobile;
  // @include features-mobile;
  // @include feedback-mobile;
  // @include products-mobile;
  //@include footer-mobile;
  // @include slider-mobile;
  // @include product-info-mobile;
  // @include order-mobile;
  // @include registration-mobile;
  // @include sort-mobile;
}
```

Этот CSS-код устанавливает отступы и размеры контейнера для различных видов устройств. В основном, он устанавливает максимальную ширину контейнера и обнуляет отступы элементов.

Код также содержит медиа-запросы для различных размеров экрана, которые изменяют максимальную ширину контейнера в зависимости от размера экрана устройства.

В комментариях указаны различные стили, которые должны быть применены для различных элементов на планшетах и мобильных устройствах, но они закомментированы (//) и неактивн***ы.***

Router index.js

```
import { createRouter, createWebHistory } from "vue-router";


const routes = [
  {
    path: "/",
    name: "home",
    component: () => import("../views/Home.vue"),
  },
  {
    path: "/cart",
    came: "cart",
    component: () => import("../views/Cart.vue"),
  },
  {
    path: "/catalog",
    came: "catalog",
    component: () => import("../views/Catalog.vue"),
  },
  {
    path: "/product/:id",
    came: "catalog",
    component: () => import("../views/Product.vue"),
```

```
  },
  {
    path: "/registration",
    came: "catalog",
    component: () => import("../views/Registration.vue"),
  },
  {
    path: "/:pathMatch(.*)*",
    name: "NotFound",
    component: () => import("../views/404.vue"),
  },
];

const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes,
});

export default router;
```

Этот код является частью настройки маршрутизации во Vue.js с использованием библиотеки vue-router.

1. Импорт необходимых функций createRouter и createWebHistory из библиотеки "vue-router".
2. Определение массива routes, содержащего объекты, описывающие маршруты приложения. Каждый объект содержит информацию о пути (path), имени (name) и компоненте, который будет отображаться при переходе по данному маршруту. Компоненты загружаются динамически с помощью функции import.
3. Создание экземпляра маршрутизатора с помощью функции createRouter, передавая в нее объект с параметрами. В данном случае используется HTML5 History API для управления историей браузера с помощью createWebHistory, а также передается массив routes с описанием маршрутов.
4. Экспорт созданного экземпляра маршрутизатора для использования в основном приложении Vue.

Этот код определяет маршруты для различных страниц приложения, а также настраивает маршрутизацию для них.

Store cart.js

```
import axios from "axios";

export default {
  state: {
    cart: [], // Корзина товаров
    sumPriceProductInCart: 0, // Цена всех товаров в корзине
    countProductsInCart: 0, // Количество товара в корзине
    cartStatus: false, // Open / Close
```

```
},
getters: {
  getCart: (state) => [...state.cart],
  getCountProductsInCart: (state) => state.countProductsInCart,
  getSumPriceProductInCart: (state) => state.sumPriceProductInCart,
  getCartStatus: (state) => state.cartStatus,
},
mutations: {
  // Установка корзины, выгрузка товара в STATE корзины
  setCart: (state, cart) => {
    state.cart = cart;
  },
  /**
   * Добавление товара в STATE корзины
   * @param {*} state
   * @param {*} product
   */
  setAddToCart: (state, product) => {
    if (!state.cart.some((item) => item.id === product.id)) {
      state.cart.push(product);
      ++state.countProductsInCart;
      state.sumPriceProductInCart += product.price;
    }
  },
  // Изменение значения товара в STATE корзины,
  setEditCart: (state, product) => {
    let indx = state.cart.findIndex((item) => item.id === product.id);
    state.cart[indx] = product;
  },
  // Увеличение количества товара в корзине
  setCountProductsInCartPlus: (state) => {
    ++state.countProductsInCart;
  },
  // Уменьшение количества товара в корзине
  setCountProductsInCartMinus: (state) => {
    --state.countProductsInCart;
  },
  /**
   * Перерасчёт количества товаров при манипуляции товарами в корзине
   * @param {*} state
   * @param {*} products
   */
  setCountProductsInCart: (state, products) => {
    let count = 0;
    products.forEach((product) => {
      count += product.count;
    });
```

```
    state.countProductsInCart = count;
  },
  // Показывает корзину
  setCartStatusOpen: (state) => {
    state.cartStatus = true;
  },
  // Скрывает корзину
  setCartStatus: (state) => {
    state.cartStatus = false;
  },
  // Открывает - закрывает корзину
  setCartStatusSwith: (state) => {
    state.cartStatus = !state.cartStatus;
  },
  /**
   * Общая стоимость товаров корзины / перерасчёт при манипуляции товарами в
корзине
   * @param {*} state
   */
  setSumPriceProductInCart: (state) => {
    let price = 0;
    state.cart.forEach((product) => {
      price += product.price * product.count;
    });
    state.sumPriceProductInCart = price;
  },
  /**
   * Удаляет из STATE корзины конкретный товар
   * Исключает значения товара (шт и цену) из общего стека
   * @param {*} state
   * @param {*} product
   */
  setCartDeliteProduct: (state, product) => {
    let indx = state.cart.findIndex((p) => p.id == product.id);
    state.countProductsInCart -= product.count;
    state.sumPriceProductInCart -= product.count * product.price;
    state.cart.splice(indx, 1);
  },
},
actions: {
  /**
   * Показывает или скрывает окно корзины с товарами
   * @param {Object} commit
   */
  actionSwithCatrStatus({ commit }) {
    commit("setCartStatusSwith");
  },
```

```
  // Загружает из БД в STATE данные корзины
  loadProductsFromCart({ commit }) {
    axios({
      method: "GET",
      url: `/api/v1/cart`,
      params: {
        //user_key_id: "USER_KEY_ID",
      },
      data: {},
      headers: {
        "Content-type": "application/json; charset=UTF-8",
      },
    })
      .then((response) => {
        commit("setCart", response.data);
        commit("setCountProductsInCart", response.data);
        commit("setSumPriceProductInCart");
      })
      .catch((error) => {
        console.log(error);
      })
      .finally(() => {});
  },

  /**
   * Изменяет товар в корзине, а именно его количество, и итоговую цену
   * @param {*} commit
   * @param {Object} product Товар
   */
  actionUpdateProductFromCart({ commit }, product) {
    axios({
      method: "PUT",
      url: `/api/v1/cart/${product.id}`,
      params: {
        //user_key_id: "USER_KEY_ID",
      },
      data: JSON.stringify(product),
      headers: {
        "Content-type": "application/json; charset=UTF-8",
      },
    })
      .then(() => {
        commit("setCountProductsInCartPlus");
        commit("setSumPriceProductInCart");
      })
      .catch((error) => {
        console.log(error);
```

```
      })
      .finally(() => {});
    commit("setCartStatusOpen");
  },
  /**
   * Добавляет товар в корзину и записывает данные в базу
   * Перед добавлением проверяет есть ли товар в корзине (если есть то не
добавляет),
   * сделано для того, что бы при множественном клике на кнопку добавить в корзину
товар не дублировался
   * на кнопку добавления товара
   * @param {*} commit
   * @param {*} product
   */
  actionAddProductToCart({ commit }, product) {
    if (!this.getters.getCart.some((item) => item.id === product.id)) {
      // Если в корзине есть товар
      product.count = 1;
      axios({
        method: "POST",
        url: `/api/v1/cart`,
        params: {
          //user_key_id: "USER_KEY_ID",
        },
        data: JSON.stringify(product),
        headers: {
          "Content-type": "application/json; charset=UTF-8",
        },
      })
        .then(() => {
          commit("setAddToCart", product);
        })
        .catch((error) => {
          console.log(error);
        })
        .finally(() => {});

      commit("setCartStatusOpen"); // Открывает окно корзины
    }
  },
  /**
   * Экшен удаления товаров из корзины. Удаляет сразу любое количество товара
данного артикула
   * Производит запись данных в базу через API
   * @param {*} commit
   * @param {*} product
   */
```

```
  actionDeleteProductFromCart({ commit }, product) {
    axios({
      method: "DELETE",
      url: `/api/v1/cart/delete/${product.id}`,
      params: {},
      headers: {
        "Content-type": "application/json; charset=UTF-8",
      },
    })
      .then(() => {
        commit("setCartDeliteProduct", product);
      })
      .catch((error) => {
        console.log(error);
      })
      .finally(() => {});
  },
  actionCloseCart({ commit }) {
    commit("setCartStatus");
  },
 },
};
```

Этот код представляет собой экшены (actions) в Vuex, который является централизованным хранилищем для управления состоянием приложения во Vue.js. Давайте прокомментируем его по-русски:

1. Экшен actionAddProductToCart добавляет товар в корзину и записывает данные в базу. Перед добавлением он проверяет, есть ли уже такой товар в корзине, чтобы избежать дублирования. Если товара нет в корзине, то выполняется запрос к API для добавления товара в базу данных. Затем вызывается мутация setAddToCart, чтобы обновить состояние корзины, и открывается окно корзины с помощью мутации setCartStatusOpen.

2. Экшен actionDeleteProductFromCart удаляет товар из корзины. Он производит запрос к API для удаления товара из базы данных по его идентификатору. Затем вызывается мутация setCartDeliteProduct, чтобы обновить состояние корзины.

3. Экшен actionCloseCart закрывает окно корзины путем вызова мутации setCartStatus.

Эти экшены используются для взаимодействия с сервером (API) для добавления, удаления товаров из корзины и управления состоянием приложения в соответствии с этими действиями.

Тестирование проекта

информацией о пользователе, содержимым корзины и т.д. Вы можете использовать Vuex для обновления состояния с помощью мутаций и действий, а также получать данные из состояния с помощью геттеров.

Про test utils

Когда вы используете @vue/test-utils в контексте интернет-магазина на Vue CLI, вы обычно создаете тесты для компонентов вашего интернет-магазина. @vue/test-utils предоставляет удобные утилиты для создания и манипуляции с Vue компонентами во время тестирования.

Например, если у вас есть компонент <ProductList> отображающий список товаров, вы можете написать тест для этого компонента следующим образом:

```
import { shallowMount } from '@vue/test-utils'
import ProductList from '@/components/ProductList.vue'

describe('ProductList.vue', () => {
  it('renders a list of products', () => {
    const products = [
      { id: 1, name: 'Product 1', price: 100 },
      { id: 2, name: 'Product 2', price: 150 }
    ]
    const wrapper = shallowMount(ProductList, {
      propsData: { products }
    })
    expect(wrapper.findAll('.product').length).toBe(2)
  })
})
```

В этом примере мы создаем тест для компонента ProductList, передаем ему фиктивные данные о товарах и проверяем, что он правильно отображает список товаров.

Таким образом, @vue/test-utils позволяет вам создавать тесты для ваших компонентов, монтировать их, взаимодействовать с ними и проверять их поведение в изолированной среде тестирования.

О unit тестировании в контексте интернет-магазина

При разработке интернет-магазина на Vue CLI, unit-тестирование играет важную роль для обеспечения качества кода и функциональности. Unit-тестирование позволяет проверить отдельные части кода (например, компоненты, утилиты, хелперы) на корректность их работы в изоляции от других частей приложения.

Если вы используете Vue CLI, то вы можете использовать встроенные инструменты для написания и запуска unit-тестов. Например, Vue CLI предоставляет возможность использовать Jest в качестве тестового фреймворка по умолчанию.

Для unit-тестирования интернет-магазина на Vue CLI, вы можете писать тесты для отдельных компонентов, моделей данных, сервисов API и других частей вашего приложения. Например, для компонента отображения товара вы можете написать тесты, которые проверяют его отображение, взаимодействие с пользователем и обработку данных.

Пример теста для компонента отображения товара может выглядеть следующим образом:

```
import { shallowMount } from '@vue/test-utils'
import ProductComponent from '@/components/ProductComponent.vue'

describe('ProductComponent', () => {
  it('correctly displays product information', () => {
    const product = {
      id: 1,
      name: 'Product 1',
      price: 100
    }
    const wrapper = shallowMount(ProductComponent, {
      propsData: { product }
    })
    expect(wrapper.find('.product-name').text()).toBe('Product 1')
    expect(wrapper.find('.product-price').text()).toBe('$100')
  })
})
```

Этот пример демонстрирует тестирование компонента ProductComponent, который отображает информацию о товаре. Мы монтируем компонент с фиктивными данными о товаре и проверяем, что он корректно отображает имя и цену товара.

Таким образом, unit-тестирование в контексте интернет-магазина на Vue CLI позволяет вам убедиться в корректной работе отдельных частей приложения и предотвратить появление ошибок при изменении кода.

Заключение к дипломному проекту интернет магазина на Vue cli

В заключение хочется отметить, что разработка интернет магазина на Vue cli стала значимым шагом в мире электронной коммерции. Полученные в процессе работы навыки и знания позволили глубже понять технологии разработки веб-приложений и их применение в создании современных онлайн платформ. Проект интернет магазина на Vue cli демонстрирует гибкость, масштабируемость и производительность фреймворка Vue, что подтверждает его актуальность в сфере веб-разработки.

В процессе создания данного проекта были преодолены значительные технические и организационные сложности, что позволило усовершенствовать навыки командной работы, анализа требований пользователя, планирования и разработки веб-приложения. Благодаря этому были получены ценные знания о практическом применении актуальных инструментов и технологий разработки, что является важным вкладом в профессиональное развитие.
В ходе выполнения дипломного проекта был разработан интернет-магазин на Vue CLI, который представляет собой современное и инновационное веб-приложение. В процессе работы над проектом были решены множество технических и дизайнерских задач, что позволило создать функциональный и привлекательный продукт.

Одной из основных целей проекта было создание удобного и интуитивно понятного интерфейса для пользователей. Благодаря использованию Vue CLI и современных технологий веб-разработки, удалось достигнуть высокого уровня интерактивности и отзывчивости приложения. Компонентный подход Vue позволил эффективно организовать код и упростить его поддержку и расширение.

Важным аспектом работы над проектом была также оптимизация производительности приложения. С помощью Vue CLI удалось провести оптимизацию загрузки страниц, уменьшить объем передаваемых данных и повысить скорость отклика приложения. Это позволило создать более комфортное пользовательское взаимодействие и улучшить общее впечатление от использования интернет-магазина.

В заключение можно сказать, что разработка интернет-магазина на Vue CLI была увлекательным и познавательным процессом. Полученные знания и опыт позволили не только создать функциональный продукт, но и улучшить навыки веб-разработки. Проект демонстрирует преимущества использования Vue CLI для создания современных веб-приложений и представляет собой успешный результат работы над дипломным проектом.

Список литературы используемый при написании работы

1.лекции и вебинары GB

2 Интегрированная среда разработки (IDE): что такое, примеры, как выбрать (optimalgroup.ru)

3 Как установить и использовать библиотеки в JavaScript (idea-tc.ru)

4 https://www.youtube.com/watch?v=bVh65QWVmfA&list=PL44epM9JNpI_6ZoUYN4-TrenphXo8nNGI&index=2&t=164s

5 https://www.youtube.com/watch?v=U7aTR3qLJxY&list=PL44epM9JNpI_6ZoUYN4-TrenphXo8nNGI&index=4&t=18s