Geekbrains

Создание одностраничного сайта питомника собак с блогом

IT-специалист:
Frontend-программист
Пшеничная Н. В.

Ростов-на-Дону
2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире с развитием интернета и цифровых технологий потребность в создании собственного присутствия в сети стала особенно актуальной для предпринимателей. Частные питомники и заводчики, занимающиеся разведением и продажей собак, не являются исключением. Однако, многие из них пока не располагают полноценными сайтами, которые могли бы представить их питомцев и привлечь новых клиентов.

Целью данного дипломного проекта является создание одностраничного сайта питомника собак с блогом. Одностраничный сайт является популярным инструментом среди веб-разработчиков в последние годы, так как он предоставляет пользователю компактное и удобное пространство для представления всей необходимой информации о питомнике и его услугах.

Проектирование и разработка подобного сайта представляет большой интерес, так как позволяет решить несколько задач одновременно. Во-первых, он представляет визуализацию питомника, демонстрируя его особенности и конкурентные преимущества. Во-вторых, создание онлайн-блога позволит питомнику установить отношения с клиентами, предоставлять им полезную информацию, в том числе о содержании и уходе за питомцами, выставках и других мероприятиях и оставаться в курсе последних новостей питомника.

В-третьих, наличие качественного и интуитивно понятного веб-интерфейса способствует созданию доверия клиентов и повышению привлекательности организации.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Изучение теоретических аспектов веб-разработки и дизайна, включая основы HTML, CSS, JavaScript, Vue.js.

2. Анализ существующих одностраничных сайтов и их структуры для определения ключевых элементов, которые должны присутствовать на сайте питомника собак.
3. Разработка дизайн-макета сайта на основе полученных знаний.
4. Создание прототипа сайта на предмет удобства использования и эффективности.
5. Написание и публикация статей в блоге, которые будут содержать информацию о питомнике, породе собак, их воспитании и уходе.

На основе выполненных задач исследования будет оценена эффективность создания одностраничного сайта питомника собак с блогом, а также его влияние на привлечение клиентов и улучшение коммуникации с ними.

Представленный дипломный проект имеет важное значение для дальнейшего развития предпринимательства в сфере разведения и продажи собак. Создание современного и инновационного онлайн-пространства для питомника позволит ему оставаться конкурентоспособным на рынке животных, а также будет способствовать распространению достоверной информации, основанной на опыте содержания и воспитания собак.

План разработки сайта с использованием HTML, CSS, JavaScript и Vue.js состоит из следующих пунктов:

1. Разработка пользовательского интерфейса и взаимодействия с пользователем UI/UX на сайте Figma.com. Простота и интуитивность в приоритете.
2. Использование Vue.js для создания сайта.

Проектирование и разработка будет производиться собственными силами. Но дополнительно при создании подобных одностраничных сайтов очень востребованными могли бы оказаться UX-дизайнеры, т.к. очень важно, чтобы сайт выглядел привлекательно и делались нужные акценты в ограниченном пространстве страницы. Так же в дальнейшем планируется использовать бэкенд, чтоб обеспечить безопасное использование сайта с блогом.

1. ОДНОСТРАНИЧНЫЕ САЙТЫ. ИХ ИСПОЛЬЗОВАНИЕ И ОСОБЕННОСТИ

1.1 Одностраничные сайты: основы

Одностраничные сайты (также известные как одностраничники или лендинги) – это веб-сайты, которые содержат всю информацию на одной странице и не требуют перехода на другие внутренние страницы для получения дополнительной информации. Они довольно популярны среди компаний и фрилансеров, так как они обладают рядом преимуществ и целей разработки.

Основной целью разработки одностраничных сайтов является сосредоточение на одном конкретном продукте, услуге или сообщении, избегая перенасыщения информацией. Они часто используются для маркетинговых кампаний, запусков новых продуктов или представления сообщений, которые не требуют множества страниц. Одностраничные сайты также могут использоваться для создания портфолио, личных блогов или визиток, где главная цель - представить информацию компактно и наглядно.

Основные принципы разработки одностраничных сайтов включают четкость, простоту и хорошую навигацию. Дизайн должен быть минималистичным и лаконичным, с использованием ярких и привлекательных элементов, чтобы удержать внимание посетителей. Эффективный одностраничник должен вызывать интерес с помощью насыщенных изображений и подробной информации с минимальным количеством текста.

Особенностью одностраничных сайтов является скроллинг, поскольку вся информация находится на одной странице. Пользователь может прокручивать сайт вниз, чтобы увидеть весь контент. Это делает навигацию более простой и интуитивной, поскольку нет необходимости переходить на другие страницы или искать нужные разделы.

Процесс разработки одностраничных сайтов имеет свои особенности. Команда разработчиков должна тщательно планировать и упорядочивать контент

на одной странице, чтобы представить информацию логично и последовательно. Также необходимо обратить внимание на оптимизацию сайта под мобильные устройства, так как большинство пользователей используют смартфоны для поиска информации в интернете. Кроме того, скорость загрузки страницы играет важную роль, поэтому необходимо оптимизировать изображения и код, чтобы ускорить переходы и улучшить пользовательский опыт.

Одностраничные сайты могут быть мощным инструментом для представления информации компактно и позволяют компаниям и фрилансерам привлечь клиентов и повысить конверсию. Они предлагают удобный способ презентации продукта или услуги, подчеркивая его ключевые особенности, а также способствуют созданию лаконичного и запоминающегося пользовательского опыта.

1.2 Особенности и преимущества блога на сайте

Собственный блог на сайте одностраничнике для питомника собак имеет несколько преимуществ.

1. Удобство и простота использования: Одностраничный сайт позволяет создать простую и легко воспринимаемую структуру для блога. Это делает навигацию на сайте более интуитивной и удобной для пользователей.
2. Возможность собрать разрозненную информацию в одном месте и в дальнейшем делиться ссылкой на свой блог, где эти информация представлена.
3. Повышение посещаемости и трафика: Блог, размещенный на одностраничном сайте, имеет больше шансов быть замеченным и прочитанным, что привлекает больше посетителей на сайт.
4. Пользователи, привлеченные блогом, могут стать потенциальными клиентами для питомника собак. Блог предоставляет возможность поделиться информацией о породах собак, особенностях их содержания и заботы, чем можно заинтересовать и привлечь новых владельцев животных.

5. Связь с аудиторией: Блог позволяет владельцу питомника установить более тесную связь с любителями собак и потенциальными владельцами собак.
6. Увеличение авторитетности: Качественный блог, полезный для посетителей, помогает установить питомнику собак авторитет и доверие. Регулярные и информативные публикации могут улучшить репутацию и привлечь новых клиентов.
7. Продвижение питомника: С помощью блога можно продвигать определенные породы собак, информировать о предстоящих выставках и мероприятиях, привлекать внимание к питомнику и его услугам. Корректно и показательно написанный контент поможет привлечь качественных потенциальных клиентов.
8. SEO-преимущества: Регулярные публикации информации на своем блоге позволяют увеличить присутствие вашего питомника собак в поисковых системах.

В целом, собственный блог на сайте одностраничнике для питомника собак имеет множество преимуществ, которые помогут продвигать бизнес, привлекать клиентов и укреплять репутацию.

В данной работе блог создается без возможности оставлять комментарии и оценки. Он призван помочь владельцу питомника делиться своими мыслями, идеями и опытом с аудиторией.

Одним из главных преимуществ блога без комментариев и оценок является сохранение авторской самостоятельности и отсутствие необходимости модерации.

В целом, блоги на собственном сайте без функций комментирования и оценивания предоставляют автору большую свободу выражения, возможность сосредоточиться на содержании, избежать конфликтов и сохранить конфиденциальность. Этот подход может быть особенно привлекательным для тех, кто ищет способ делиться информацией без дополнительных забот и ограничений.

2. ИНСТРУМЕНТЫ И ПРОГРАММНЫЕ ПРОДУКТЫ РАЗРАБОТКИ

2.1 Язык программирования

Для разработки сайта с использованием Vue.js необходимо иметь знания и опыт работы с языками программирования JavaScript, CSS и HTML.

JavaScript - универсальный язык программирования, используемый для разработки интерактивных сайтов и приложений. Он позволяет добавлять динамическое поведение и функциональность на веб-страницы, такие как обновление контента без перезагрузки страницы, взаимодействие с пользователем и многое другое.

HTML (HyperText Markup Language) - язык разметки, используемый для создания структуры и содержимого веб-страниц. Он определяет, как должны быть организованы элементы на странице, такие как заголовки, абзацы, ссылки, изображения и многое другое. HTML является основой веб-разработки и работает в паре с CSS и JavaScript для создания полноценных интерактивных веб-приложений.

CSS (Cascading Style Sheets) - язык описания стилей, используемый для определения внешнего вида и форматирования веб-страниц и элементов. С помощью CSS можно задавать различные свойства и атрибуты, такие как цвета, шрифты, расположение элементов и другие визуальные аспекты.

2.2 Visual Studio Code

Разработка будет вестись в Visual Studio Code (VS Code) – это бесплатный редактор кода, разработанный Microsoft. Он предоставляет ряд преимуществ, а именно:

1. Мощный и гибкий: VS Code обладает большой функциональностью, которая позволяет разработчикам настраивать его под свои потребности. Он

поддерживает множество языков программирования и позволяет расширять функциональность с помощью плагинов.

2. Интеграция с Git: VS Code предоставляет удобный интерфейс для работы с системой контроля версий Git. Разработчики могут выполнять все основные операции, такие как коммиты, пуши, пуллы и слияния, прямо из редактора кода.
3. Отладка: VS Code обладает встроенным инструментом отладки, который позволяет разработчикам упростить процесс исправления ошибок в их коде. Он поддерживает отладку различных языков программирования, в том числе JavaScript.
4. Автодополнение и подсветка синтаксиса: VS Code предоставляет возможность автоматического завершения кода и подсветки синтаксиса для разных языков программирования. Это позволяет увеличить производительность разработчиков и снизить количество синтаксических ошибок.
5. Интеграция с различными сервисами: VS Code предоставляет встроенную интеграцию с различными сервисами, такими как Azure, Docker, Kubernetes и другие. Это позволяет разработчикам легко взаимодействовать с этими сервисами и управлять ими прямо из редактора кода.

Также необходимо установить Node.js, который является кроссплатформенной средой выполнения JavaScript и позволяет код, написанный на JavaScript, запускать вне веб-браузера.

2.3 Vue.js

Для разработки сайта будет использоваться фреймворк Vue.js, который является своего рода готовой моделью для быстрой разработки. Он используется для разработки пользовательских интерфейсов, а также одностраничных приложений, обладает достаточно высокой производительностью и скоростью.

Из очевидных плюсов фреймворка Vue.js можно перечислить:

1. Легкость изучения. Vue.js имеет простой и понятный синтаксис, который делает его легко изучаемым для начинающих разработчиков. Его документация является исчерпывающей и легко доступной для ознакомления.
2. Масштабируемость: Vue.js предлагает множество возможностей для разработки масштабируемых приложений. Он обеспечивает модульность, позволяющую разрабатывать компоненты приложения отдельно и повторно использовать их по мере необходимости.
3. Быстрая разработка: Благодаря системе компонентов и простому синтаксису Vue.js можно быстро разрабатывать интерактивные интерфейсы. Его элементы UI организованы в так называемые однофайловые компоненты, что позволяет концентрироваться на разработке, вместо того чтобы беспокоиться о разделении логики от представления.
4. Эффективная отрисовка: Vue.js имеет встроенные механизмы для эффективного отслеживания изменений в данных и минимизации перерисовки лишних элементов. Это повышает производительность приложения и улучшает пользовательский опыт.
5. Широкая поддержка и активное сообщество: Vue.js имеет очень активное сообщество разработчиков, что делает его привлекательным выбором при разработке веб-приложений. Сообщество предоставляет множество ресурсов, библиотек и плагинов, которые облегчают разработку и расширение функциональности.
6. Гибкость: Vue.js можно использовать как в небольших проектах, так и в более крупных приложениях. Он позволяет разработчикам выбрать только нужные им фрагменты фреймворка, не навязывая жестких правил и структур.

7. Оптимизированный рендеринг на стороне сервера: Vue.js предоставляет возможность рендеринга на стороне сервера. Это позволяет оптимизировать производительность и улучшить индексацию поисковыми системами.
8. Отличная интеграция: Vue.js можно легко интегрировать с другими JavaScript библиотеками или существующими проектами, что делает его идеальным выбором для разработки как новых, так и существующих приложений.

Еще одним существенным плюсом Vue.js является реактивность, что обеспечивает удобное обновление и отображение данных на веб-странице. Реактивность в Vue.js осуществляется с помощью системы обнаружения изменений.

Когда создаются данные в приложении Vue.js, каждое свойство их объекта становится реактивным. Это означает, что при изменении значения свойства, все зависимые значения и DOM-элементы автоматически обновляются. Изменение значения свойства может быть вызвано как внешним воздействием (например, пользовательским вводом), так и внутренними изменениями внутри приложения.

Vue.js использует систему обнаружения изменений для отслеживания зависимостей между свойствами данных. Когда одно из свойств изменяется, Vue.js обновляет только необходимые компоненты и элементы DOM, вместо полного обновления всего приложения. Это делает приложение более эффективным и производительным.

Vue.js также предоставляет функциональность для создания вычисляемых свойств и наблюдаемых свойств. Вычисляемые свойства позволяют создавать новые значения на основе существующих данных, а наблюдаемые свойства позволяют определить дополнительную логику, которая должна быть выполнена при изменении значения свойства.

В целом, реактивность в Vue.js делает разработку веб-приложений более простой и удобной, позволяя разработчикам легко отслеживать и обновлять данные и отображение на странице.

Как выше упоминалось, Vue.js позволяет разрабатывать одностраничное приложение (SPA, англ. Single Page Application) — это архитектурный подход, при котором весь контент загружается на одной HTML странице, а затем динамически обновляется с помощью JavaScript без перезагрузки страницы.

В SPA во Vue.js, весь контент и функциональность приложения организованы вокруг одного основного HTML-документа, который состоит из различных компонентов Vue. Когда пользователь взаимодействует с приложением, происходят навигационные изменения, но загрузка нового содержимого не происходит. Вместо этого, Vue.js обрабатывает эти изменения и динамически обновляет содержимое на текущей странице, основываясь на состоянии приложения и пользовательских действиях.

Преимущества SPA в Vue.js включают:

1. Быстрая навигация и отзывчивость: поскольку серверу не нужно отправлять и загружать новые страницы, пользовательский опыт становится более быстрым и отзывчивым.
2. Улучшенная маршрутизация: SPA в Vue.js позволяет легко управлять маршрутами и создавать навигацию без необходимости перезагрузки всей страницы. Это делает приложение более гибким и удобным в использовании.
3. Улучшенная пользовательская интерактивность: с помощью Vue.js можно создавать динамические и интерактивные пользовательские интерфейсы, которые реагируют на действия пользователя без перезагрузки всей страницы.
4. Упрощенная разработка и обслуживание: разделение приложения на компоненты в Vue.js делает код более организованным и удобным для чтения и поддержки. Кроме того, добавление новых функциональностей

проще, так как изменения вносятся только в соответствующие компоненты, а не повсюду в коде приложения.

В целом, SPA в Vue.js представляет собой эффективный и удобный подход для разработки веб-приложений, который повышает скорость работы и улучшает пользовательский опыт.

Необходимо немного остановиться на компонентах. Во Vue.js компоненты представляют собой отдельные части пользовательского интерфейса и являются основными строительными блоками в разработке пользовательского интерфейса. Они позволяют разбивать сложные интерфейсы на более мелкие и переиспользуемые части, что упрощает управление кодом, повышает его читабельность и облегчает масштабирование проекта. Они могут быть повторно использованы и настраиваемы. Они могут содержать HTML код, стили и логику JavaScript.

Каждый компонент в Vue.js имеет расширение .vue и состоит из трех основных частей:

Шаблон (Template): Шаблон определяет внешний вид компонента, используя синтаксис HTML. В нем можно использовать данные компонента и директивы Vue.js для динамического изменения отображения.

Скрипт (Script): Скрипт содержит JavaScript-логику компонента, такую как методы, вычисляемые свойства, обработчики событий и импорт необходимых зависимостей. Он также определяет данные, которые могут быть использованы в шаблоне и других частях компонента.

В <script> входят следующие основные элементы:

1. data. Это свойство определяет данные, которые компонент будет использовать. Эти данные могут быть привязаны к элементам пользовательского интерфейса, таким образом, при изменении данных, интерфейс автоматически обновляется.
2. methods. Здесь определяются методы, которые могут быть вызваны внутри компонента. Они могут использоваться для обработки событий, выполнения

асинхронных операций и выполнения других задач связанных с логикой компонента.

3. computed. Это свойство используется для определения вычисляемых свойств, которые зависят от других свойств компонента. Значения этих свойств будут автоматически пересчитываться при изменении зависимых свойств.
4. watch. В данной секции определяются наблюдатели, которые следят за изменением свойств компонента и выполняют дополнительные действия при их изменении. Они могут использоваться для выполнения сетевых запросов, обновления данных или выполнения других действий в ответ на изменение определенного свойства.
5. props. Здесь определяются свойства, которые могут быть переданы в компонент из родительского компонента. Это позволяет передавать данные между компонентами и создавать более гибкий и модульный код.

Скрипт в компонентах Vue.js также может содержать другие необязательные свойства и методы, такие как mounted, created, updated и destroyed, которые позволяют осуществлять дополнительные действия при создании, обновлении или удалении компонента.

В целом, скрипт позволяет описывать все необходимые свойства, методы и состояние компонента, что делает его основной и наиболее важной частью в разработке приложений на Vue.js.

Стили (Styles): Стили могут быть определены в отдельном блоке внутри компонента или импортироваться из внешних файлов. Они могут быть написаны с помощью CSS, Sass, LESS и т.д. В данном проекте будет использоваться Sass.

Компоненты в Vue.js могут быть объявлены как глобальные (доступные во всем приложении) или локальные (доступные только внутри другого компонента). С помощью свойства props можно передавать данные от родительского компонента дочерним.

Компоненты также могут иметь жизненный цикл, состоящий из различных методов, которые автоматически вызываются в определенные моменты времени. Например, метод created вызывается после создания компонента, а метод mounted - после того, как компонент был добавлен в DOM.

В целом, компоненты в Vue.js позволяют разделить интерфейс на маленькие и переиспользуемые части, что упрощает поддержку и разработку приложения.

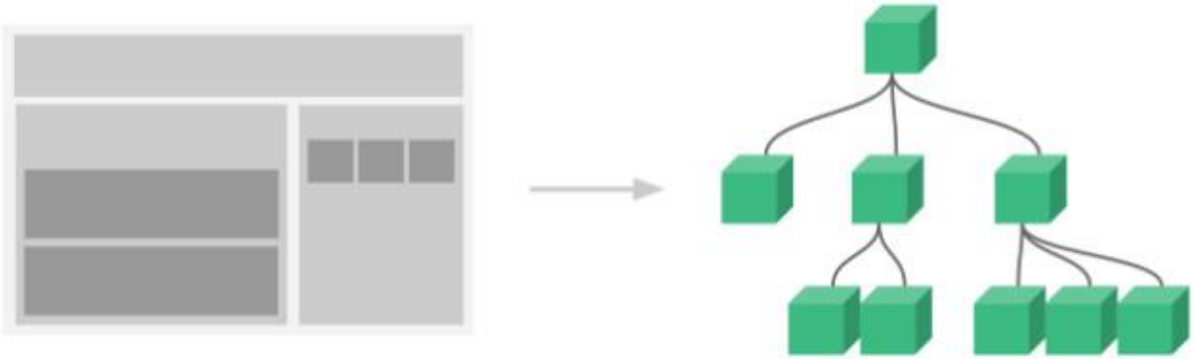

Рис. 1. Дерево вложенных компонентов (https://ru.vuejs.org)

При компонентном подходе корневым родительским элементом у дерева компонентов является App.vue, в который помещаются другие компоненты (рис.1).

2.4 Vue.js devtools

Vue.js Devtools — это расширение для браузера, предоставляющее удобный интерфейс для отладки и разработки веб-приложений Vue.js. Он позволяет разработчику проверять и изменять состояние компонентов, исследовать дерево компонентов, а также анализировать, как данные и свойства передаются по компонентам.

Для использования Vue.js Devtools требуется установить его расширение для браузера. Оно доступно для Google Chrome и Mozilla Firefox. После установки расширения, оно будет автоматически подключаться к вашему приложению на Vue.js.

Когда приложение запущено, в Devtools доступны несколько вкладок:

1. "Components" (Компоненты) - отображает дерево компонентов вашего приложения. Вы можете проверить состояние и свойства каждого компонента, а также изменять их в реальном времени.
2. "Events" (События) - позволяет отслеживать и анализировать события, которые генерируются в вашем приложении. Вы можете видеть, какие события срабатывают и из каких компонентов они вызываются.
3. "Vuex" - если вы используете Vuex для управления состоянием приложения, эта вкладка позволяет вам проверять состояние хранилища, выполнять мутации и откатывать изменения состояния.
4. "Performance" (Производительность) – это вкладка, которая помогает визуализировать производительность вашего приложения. Вы можете анализировать, как компоненты рендерятся и обновляются, и искать возможные узкие места в вашем коде.

Vue.js Devtools является полезным инструментом при разработке, так как он облегчает отладку и анализ кода, позволяет в реальном времени менять состояние и свойства компонентов, а также предоставляет мощные инструменты для анализа и улучшения производительности приложения.

2.5 Axios

Axios — это библиотека для выполнения HTTP-запросов в браузере или на сервере, основанная на промисах. Она широко используется во Vue.js приложениях для отправки AJAX-запросов на сервер и получения данных.

Axios предоставляет простой и удобный интерфейс для отправки и получения данных. Она поддерживает все основные методы запросов, такие как

GET, POST, PUT, DELETE и т.д., а также позволяет устанавливать и обрабатывать заголовки запроса.

Для использования библиотеки Axios в приложении Vue.js, сначала нужно установить ее с помощью менеджера пакетов, такого как npm или yarn. В каталоге проекта следует запустить команду:

```
npm install axios
```

После установки библиотеки, ее можно импортировать и использовать в компонентах Vue.js. Axios также предоставляет дополнительные возможности, такие как установка заголовков запроса, обработка ошибок запроса, отправка данных форм, загрузка файлов и другие. Она также интегрируется хорошо с другими популярными библиотеками Vue.js, такими как Vuex и Vue Router.

В целом, Axios — это удобная и легкая в использовании библиотека для выполнения HTTP-запросов в приложениях Vue.js. Она предоставляет множество полезных функций для работы с сервером и обрабатывает промисы, что делает ее идеальной для асинхронных операций.

3 ПРАКТИЧЕСКАЯ РАЗРАБОТКА САЙТА

3.1 Установка Vue CLI. Начало работы

Vue CLI (Command Line Interface) — это интерфейс командной строки для разработки Vue.js приложений. Он предоставляет разработчикам мощный инструментарий для создания, сборки и развертывания Vue.js проектов.

Достоинства Vue CLI:

1. Простота использования: Vue CLI предоставляет простой и интуитивно понятный интерфейс командной строки, который упрощает создание и управление проектами.
2. Гибкость настроек: Vue CLI позволяет разработчикам настроить различные аспекты проекта, такие как настройки сборки, конфигурация модулей и плагинов, а также определение переменных среды.
3. Мощный набор инструментов: Vue CLI включает в себя набор инструментов, таких как Vue Router, Vuex, Babel, PostCSS, ESLint и многие другие, которые помогают упростить и ускорить процесс разработки.

Установка Vue CLI:

1. Устанавливаем Node.js: Vue CLI требует наличие Node.js версии 8.9 или выше. Скачать и установить Node.js можно с официального сайта https://nodejs.org.
2. Устанавливаем Vue CLI: необходимо открыть терминал или командную строку и выполнить следующую команду, чтобы установить Vue CLI глобально:

```
npm install -g @vue/cli
```

Эта команда установит Vue CLI глобально в систему, чтобы можно было использовать его в любом проекте.

Разработка с использованием Vue CLI:

1. Создание нового проекта Vue с помощью команды:

```
vue create project-name
```

Эта команда создаст новую директорию с указанным именем проекта (если нужна установка проекта в текущую директорию, вместо имени следует нажать точку) и установит необходимые зависимости. При создании данного проекта подключались Babel, Router, Vuex, Sass.

2. Локальная разработка: После создания проекта, необходимо перейти в его директорию и запустить локальный сервер разработки с помощью команды:

```
npm run serve
```

Это запустит dev-сервер, который будет автоматически обновлять приложение при изменении файлов.

3. Сборка проекта: по окончанию разработки, собрать проект в продакшн можно с помощью команды:

```
npm run build
```

Это создаст оптимизированные и минифицированные файлы в директории "dist", которые готовы для развертывания на сервере.

Vue CLI предоставляет большие возможности для разработки Vue.js приложений. Он упрощает создание и управление проектами, а также предоставляет мощный набор инструментов для ускорения разработки.

3.2 Создание макета сайта в Figma

Для создания макета использовался figma.com — это онлайн-инструмент для дизайна и прототипирования, который позволяет создавать интерфейсы, макеты и анимации для веб-сайтов и мобильных приложений. Этот сайт позволяет работать над проектами совместно с другими членами команды, делиться макетами и собирать обратную связь от заказчиков или коллег. Figma.com имеет множество функций, включая возможность редактирования в режиме реального времени, работу над макетами на разных устройствах, а также интеграцию с другими приложениями и инструментами для дизайна. Этот сайт широко используется в индустрии дизайна для создания высококачественных пользовательских интерфейсов.

Макет сайта в Figma делается для визуализации и представления дизайна интерфейса веб-сайта или веб-приложения.

Главная цель создания макета в Figma заключается в том, чтобы:

- Определить компоновку и структуру страниц сайта, расположение элементов интерфейса и их взаимные связи.
- Определить цветовую схему, шрифты и стили элементов интерфейса.
- Создать взаимодействие и анимацию между элементами интерфейса.
- Протестировать и проверить интерактивность элементов интерфейса.
- Сделать демонстрацию и презентацию дизайна заказчику или команде разработчиков.
- Совместно работать с другими участниками команды на проектировании интерфейса.

Таким образом, макет в Figma является важным инструментом для создания и коммуникации дизайна интерфейса сайта или приложения.

Макеты страниц сайта представлены на рис. 2 – 5.

Рис. 2. Макет домашней страницы сайта

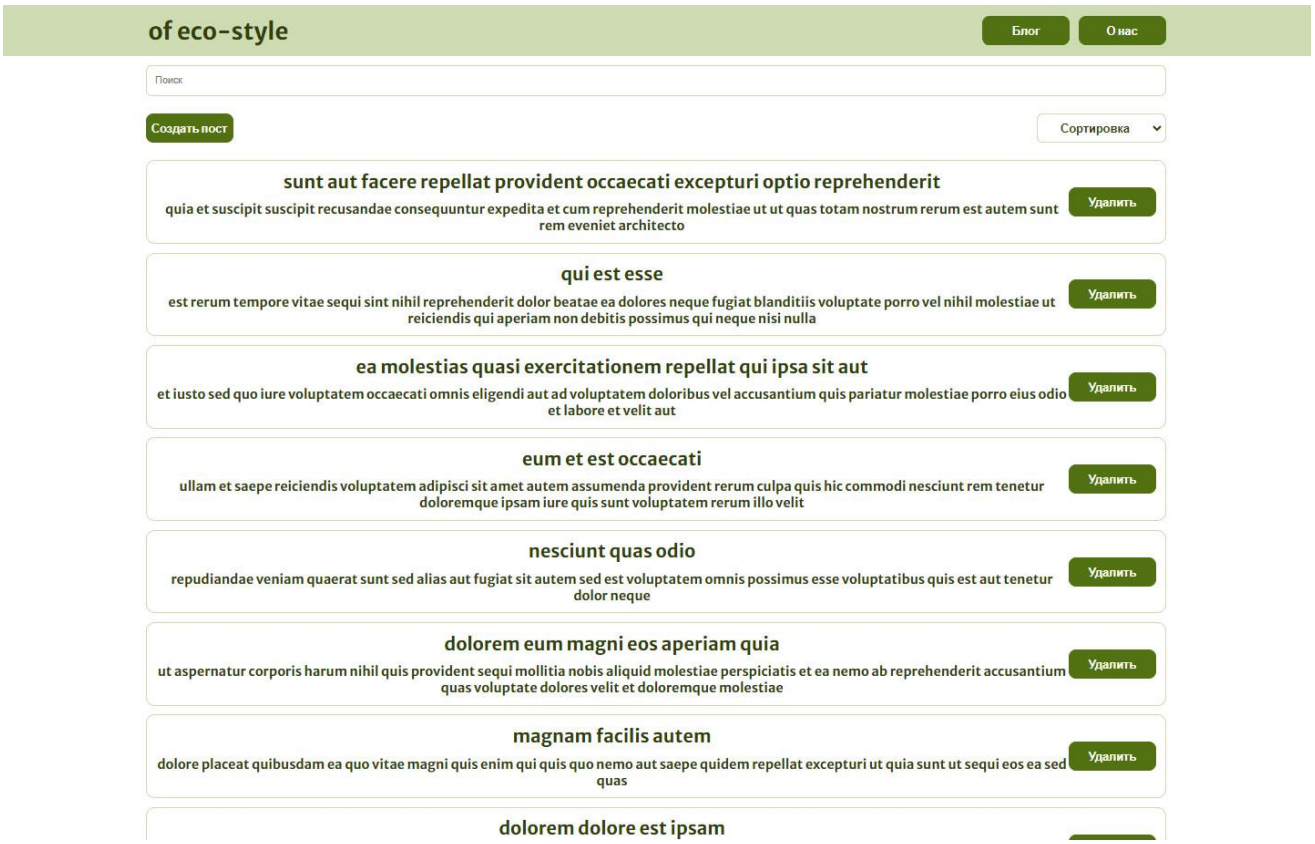

Рис. 3. Макет страницы блога

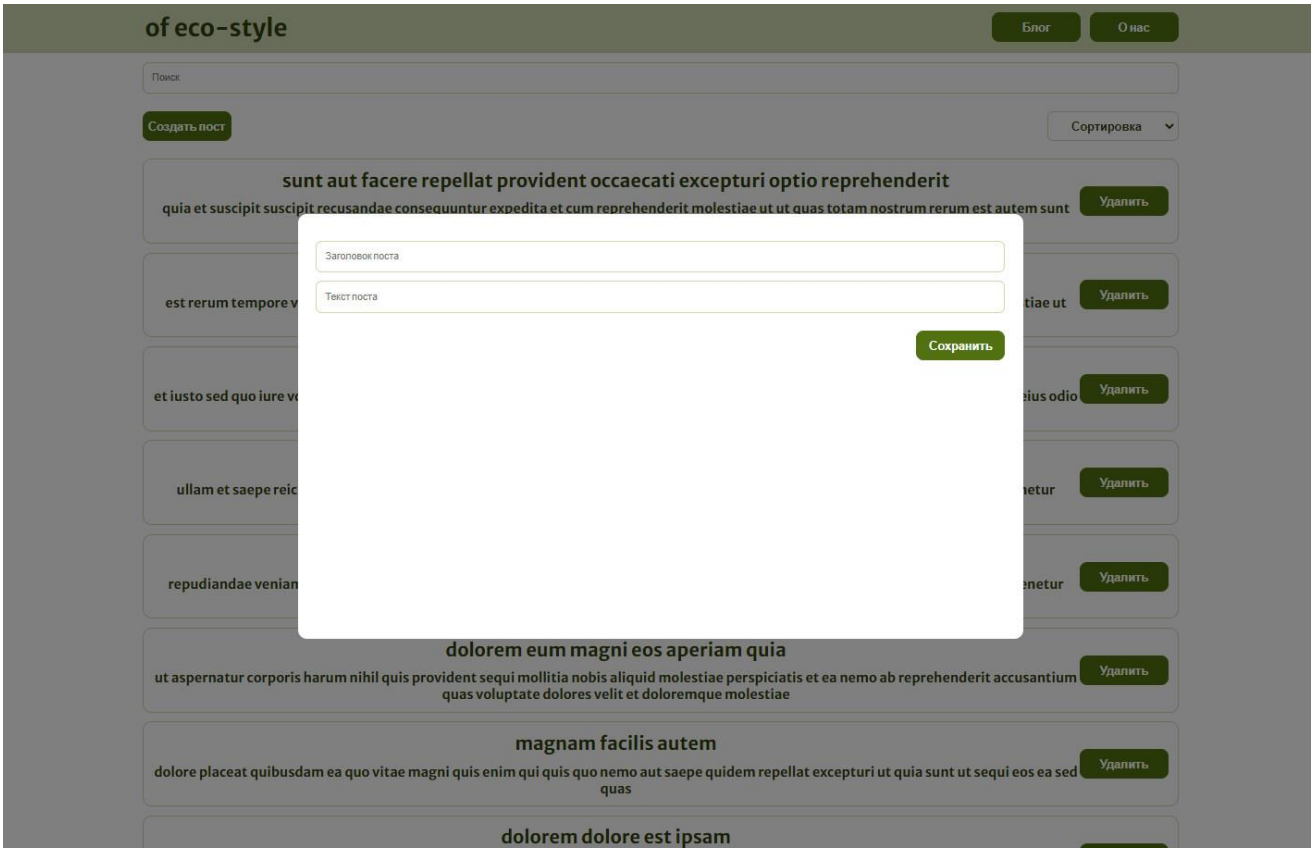

Рис. 4. Макет модального окна

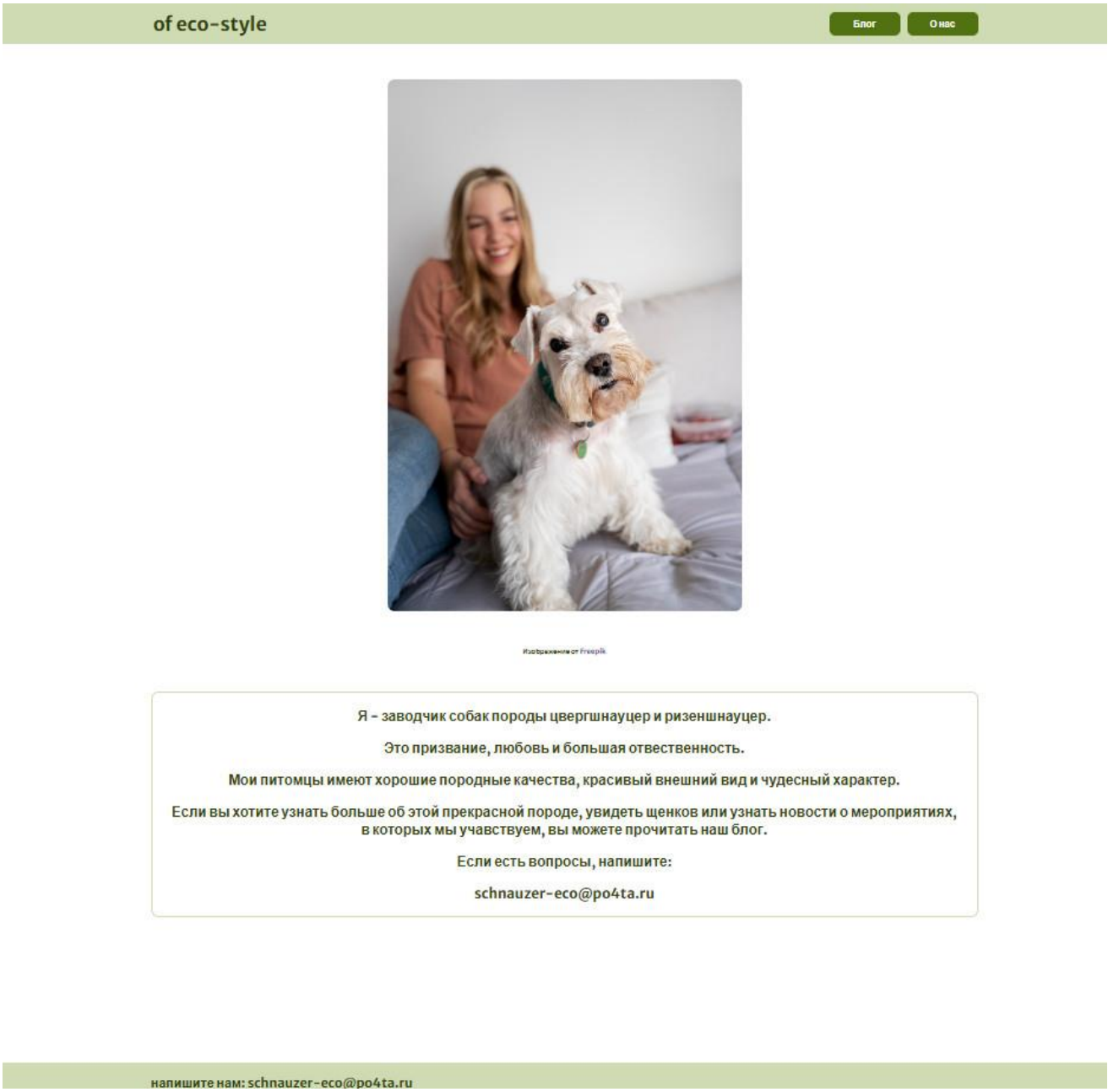

Рис. 5. Макет страницы “о нас”

3.3 Структура проекта

Структура проекта важна для понимания и организации работы над ним. Основные папки и файлы представлены на рис. 6. Мы видим три корневые директории и файлы. Основная работа будет проводиться в директории src и

немного в директории public. Далее в работе будут рассмотрены основные файлы подробнее.

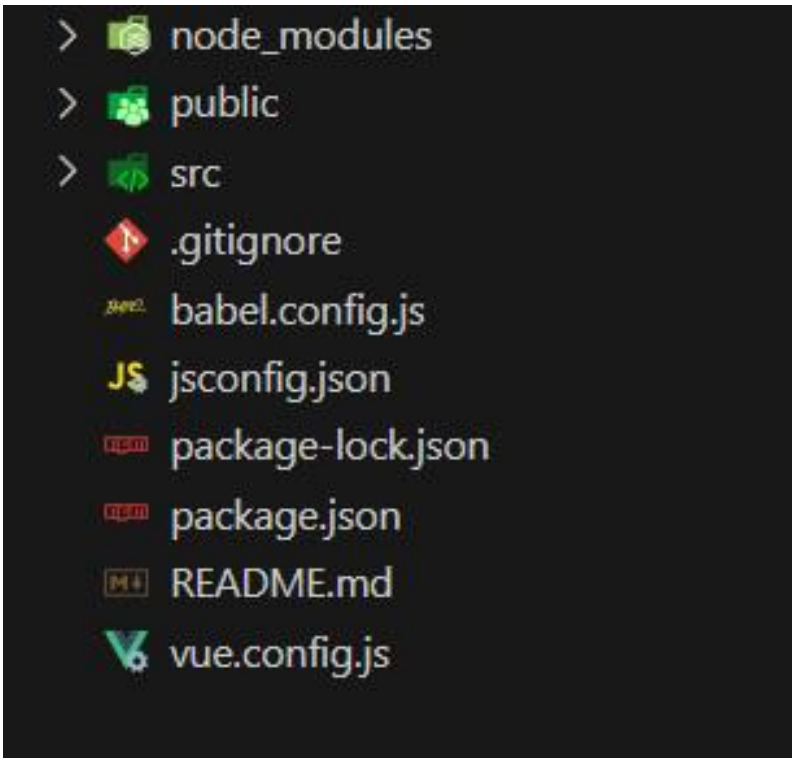

Рис. 6. Структура проекта

3.3.1 package.json

В файле package.json описаны название проекта, версия, основные зависимости и скрипты serve (для запуска в режиме разработки) и build (для сборки).

package.json

```
{
  "name": "diploma_blog",
  "version": "0.1.0",
```

```
  "private": true,
  "scripts": {
    "serve": "vue-cli-service serve",
    "build": "vue-cli-service build"
  },
  "dependencies": {
    "axios": "^1.6.5",
    "core-js": "^3.8.3",
    "vue": "^3.2.13",
    "vue-router": "^4.0.3",
    "vuex": "^4.0.0"
  },
  "devDependencies": {
    "@vue/cli-plugin-babel": "~5.0.0",
    "@vue/cli-plugin-router": "~5.0.0",
    "@vue/cli-plugin-vuex": "~5.0.0",
    "@vue/cli-service": "~5.0.0",
    "sass": "^1.32.7",
    "sass-loader": "^12.0.0"
  },
  "browserslist": [
    "> 1%",
    "last 2 versions",
    "not dead",
    "not ie 11"
  ]
}
```

3.3.2 node_modules/

Директория node_modules в Vue 3 используется для хранения всех зависимостей и модулей, которые устанавливаются в проекте с помощью пакетного менеджера npm (Node Package Manager) или Yarn. В ней автоматически сохраняются все необходимые библиотеки, плагины, компоненты и другие модули, которые устанавливаются для работы с Vue 3 и его экосистемой.

Эти зависимости и модули обеспечивают функциональность, расширяющую возможности Vue 3 и позволяющую разрабатывать приложения на Vue более эффективно и удобно. Каждая зависимость, установленная в проекте, сохраняется в отдельной директории внутри node_modules, и они импортируются и использованы в коде приложения.

Важно не изменять или удалять файлы внутри директории node_modules, так как они генерируются автоматически и управляются пакетными менеджерами. Если вам необходимо добавить или удалить зависимости, вы должны использовать команды установки или удаления пакетов, предоставляемые пакетным менеджером.

3.3.3 public/

В директории public расположены немногочисленные фотографии сайта, которые при увеличении их количества удобнее будет расположить в созданную для этого директорию images и файл index.html.

index.html

index.html в директории public, в который будет встроено приложение, является основным файлом HTML-разметки приложения. Он отображается в браузере и содержит структуру страницы и ссылки на внешние ресурсы, такие как

стили CSS, скрипты JavaScript и другие файлы. Index.html также содержит контейнер , куда Vue.js будет вставлять компоненты и контент приложения:

```
<div id="app"></div>
```

Этот файл обычно используется для настройки глобальных настроек приложения, загрузки стилей и скриптов, а также создания корневого элемента приложения.

3.3.4 src/

Основная рабочая директория в данном проекте – это директория src, структуру которой более детально можно рассмотреть на рис. 7.

В папке src обычно содержится основной исходный код приложения, здесь представлены все наши компоненты, страницы, роутер, корневой компонент приложения App.vue, main.js, который является основной точкой входа в приложение, где происходит инициализация Vue и настройка основных компонентов и плагинов. Компоненты сгруппированы в папку components. Так же здесь находится папка router с файлом, где прописана конфигурация маршрутов.

Здесь же могла бы располагаться папка store, в которой традиционно располагаются файлы конфигурации хранилища, если бы в приложении использовалось глобальное хранилище состояния (Vuex). В данном проекте автором было принято решение не использовать глобальное хранилище. Причины такого решения будут представлены далее в работе.

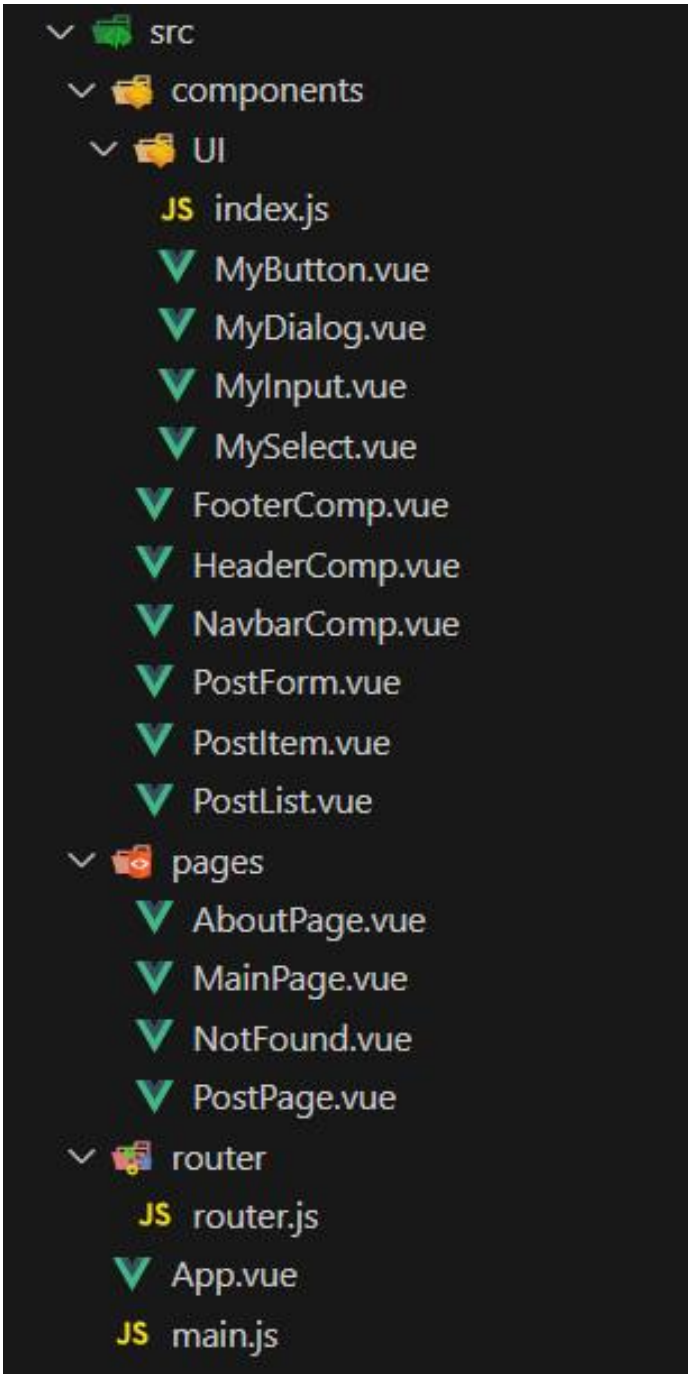

Рис. 7. Структура папки src

main.js

main.js в директории src является точкой входа в приложение. В этом файле происходит инициализация Vue и подключение основных компонентов и модулей, необходимых для работы приложения., Здесь определяется экземпляр Vue, который позволяет управлять состоянием и взаимодействовать с

компонентами и роутером. Подключаются компоненты, созданные внутри приложения или импортированные из внешних модулей. Помимо этого, он отвечает за установку структуры Vue-приложения.

В основном, файл main.js включает следующую структуру:

- импорт необходимых зависимостей:

- создание и монтирование экземпляра Vue

Компонент App.vue используется как корневой компонент приложения.

Опционально можно добавить дополнительные настройки, плагины, расширения и роутеры в экземпляр Vue.

В целом, файл main.js отвечает за инициализацию Vue-приложения и установку его структуры. Он является точкой входа для всего приложения и позволяет определить корневой компонент и любые другие настройки, необходимые для его работы.

Описание некоторых строк файла main.js будут даны далее по ходу работы, однако здесь приводится код целиком.

Main.js

```
import { createApp } from 'vue';
import App from './App.vue';
import components from '@/components/UI';
import router from '@/router/router';
const app = createApp(App)

components.forEach(component => {
    app.component(component.name, component)
});

app.use(router)
app.mount('#app')
```

src/components/

В директории components, как следует из названия, хранятся наши компоненты. Можно выделить классические для всех сайтов компоненты: Footer и Header, в который включен Navbar. Помимо этих компонентов здесь находятся PostForm, PostItem, PostList.

Также здесь расположена папка UI для графических компонентов: MyButton, MyDialog, MyInput, MySelect.

src/components/UI

Для того, чтобы не прописывать постоянно импорт и регистрацию часто используемых UI компонентов, создадим файл index.js, из которого будем по дефолту экспортировать массив UI-компонентов. Для этого импортируем в index.js те компоненты, которые будем добавлять в данный массив. В main.js необходимо импортировать массив UI-компонентов: `import components from '@/components/UI'`. Проходим по массиву компонентов, используя функцию `component` , чтобы не импортировать каждый раз какой-либо из этих компонентов, когда мы захотим их использовать.

```
components.forEach(component => {
    app.component(component.name, component)
});
```

Файл index.js будет выглядеть следующим образом:

Index.js

components/UI

index.js

```
// "библиотека" импортов UI компонентов + main.js
import MyButton from "@/components/UI/MyButton";
import MyInput from "@/components/UI/MyInput";
import MyDialog from "@/components/UI/MyDialog";
import MySelect from "@/components/UI/MySelect";

export default [
    MyButton,
    MyInput,
    MyDialog,
    MySelect
]
```

MyButton.vue

Кнопки во всем приложении делаются однотипными и чтобы не повторяться, создаем компонент кнопки. Для того, чтобы внутренний текст можно было изменять в разных компонентах, используем <slot></slot>

components/UI

MyButton.vue

```
<template>
    <button class="btn">
        <slot></slot>
    </button>
</template>
```

```
<script>
export default {
    name: 'MyButton',
};
</script>
<style lang="scss" scoped>
.btn {
    width: 120px;
    height: 40px;
    border-radius: 10px;
    background: #517113;
    border: 1px solid #517113;
    color: #FFF;
    text-align: center;
    font-size: 16px;
    font-style: normal;
    font-weight: 700;
    line-height: normal;
    transition: 0.2s;
    &:hover {
        background-color: #688e1a;
    }
}
@media (max-width: 1024px) {
    .btn {
        width: 90px;
        font-size: 14px;
    }
}
</style>
```

MyDialog.vue

Модальное окно для создания поста. Пропс будет отвечать за то, будет ли модальное окно видно или нет. Используем тип Boolean, по дефолту значение false. Используем v-if для проверки пропса show. Чтобы модальное окно было динамически изменяемым, помещаем в него `<slot>`. Главный div делаем на весь экран и затеняем, а внутренний уменьшаем в размере.

Для того, чтобы диалоговое окно закрывалось, делаем слушатель события нажатия и функцию hideDialog(): `@click.stop="hideDialog"`. Чтоб предотвратить событие всплытия в JavaScript, используем модификатор .stop. Эмитим событие, названное `'update:show'` из функции и вторым параметром передаем false, т.к. окно будет закрыто. Таким образом делаем двустороннее связывание с родительским компонентом PostPage, где show передадим как атрибут v-model `v-model:show="dialogVisible"`.

components/UI

MyDialog.vue

```
<template>
    <div
         v-if="show"
         class="dialog center"
         @click.stop="hideDialog"
    >
        <div @click.stop class="dialog_content">
            <slot></slot>
        </div>
    </div>
</template>
<script>
export default {
    name: 'MyDialog',
```

```
    props: {
        show: {
            type: Boolean,
            default: false
        }
    },
    methods: {
        hideDialog() {
            this.$emit('update:show', false);
        }
    },
};
</script>
<style lang="scss" scoped>
.dialog {
    top: 0;
    bottom: 0;
    right: 0;
    left: 0;
    background: rgba(0,0,0,0.5);
    position: fixed;
    display: flex;
    &_content {
        margin: auto;
        background: white;
        border-radius: 12px;
        min-height: 50%;
        min-width: 70%;
    }
} </style>
```

MyInput.vue

Компонент MyInput.vue используется для пользовательского ввода текста и представляет собой текстовое поле, он может быть использован в других местах приложения, где потребуется ввод текста. Он будет связан с моделью данных, предоставленной родительским компонентом, и будет генерировать событие при каждом изменении введенного значения.

Отметим, что во Vue3 внесены кардинальные изменения: при использовании на компонентах v-model входной параметр и имя события поменялись:

- входной параметр: value -> modelValue;

- событие: input -> update:modelValue;

Во Vue3 появилась особенность у v-model – он может быть множественным. Его можно использовать как без явного указания атрибута, с которым идет связывание (тогда будет v-model связано с modelValue), так и с явным атрибутом (например v-model:title) тогда в нашем примере будет 'update:modelValue' - с любым атрибутом, в отличии от vue2, где v-model работало только value.

Пропсом принимаем modelValue, который будет либо строкой, либо числом. И к инпуту байндим modelValue `:value="modelValue"`

И событие `@input` работает с изменением значения внутри импута. Для этого создаем метод, который будет обрабатывать данное событие: эмитим данное событие.

components/UI

MyInput.vue

```
<template>
    <input
        :value="modelValue"
        @input="updateInput"
        class="input"
```

```
            type="text"
        >
    </template>

    <script>
    export default {
        name: 'MyInput',
        props: {
            modelValue: [String, Number]
        },
        methods: {
            updateInput(event) {
                        this.$emit('update:modelValue',
event.target.value);
            }
        },
    };
    </script>

    <style lang="scss" scoped>
    .input {
        width: 100%;
        padding: 12px;
        margin-top: 24px;
        border: 1px solid rgba(93, 137, 3, 0.30);;
        border-radius: 8px;
        margin-top: 12px;
    }
    </style>
```

MySelect.vue

В этом компоненте реализован выпадающий список, в котором можно выбрать, по какому полю будет делаться сортировка. В данном случае сортировка будет производиться по заголовку и содержанию. Используется дефолтный тег <select> который используется для создания выпадающего списка (селектора) на веб-странице. Он позволяет пользователю выбрать одну или несколько опций из предложенного списка. Тег < select > обычно используется вместе с тегами <option> для определения доступных вариантов выбора. При выборе опции из списка, выбранное значение сохраняется и может быть отправлено на сервер для дальнейшей обработки. Первая опция по умолчанию будет отключена `disabled`.

Реализовано двустороннее связывание `:value="modelValue"` и modelValue принимаем как propc с типом String, т.е. поле, по которому будет идти сортировка (title или body). Второй пропс – options – массив объектов выпадающего списка, по дефолту будут пустым массивом. Проходим по массиву с помощью директивы v-for. Отслеживание изменения значения в списке `@change="changeOption"` . Для списков используется событие change. В функции `"changeOption"` делаем эмит в PostPage.

components/UI

MySelect.vue

```
<template>
    <select
        :value="modelValue"
        @change="changeOption"
        class="select"
    >
            <option disabled value="">Выберите из списка</option>
        <option
```

```
                v-for="option in options"
                :key="option.value"
                :value="option.value"
            >
                {{ option.name }}
            </option>
        </select>
    </template>

    <script>
    export default {
        name: 'MySelect',
        props: {
            modelValue: {
                type: String
            },
            options: {
                // массив объектов выпадающего списка
                type: Array,
                default: () => []
            }
        },
        methods: {
            changeOption(event) {
                        this.$emit('update:modelValue',
event.target.value);
            }
        },
    };
    </script>
```

```
<style lang="scss" scoped>
.select {
    border: 1px solid rgba(93, 137, 3, 0.30);;
    border-radius: 8px;
    color: #2F4209;
    text-align: center;
    font-style: normal;
    font-weight: 700;
    line-height: normal;
    font-size: 16px;
    padding-left: 12px;
    padding-right: 12px;
}
</style>
```

NavbarComp.vue

Для навигации используется навигационная панель - компонент NavbarComp, в котором сделаны навигационные кнопки и текстовый логотип. В случае, если понадобится расширить приложение, то можно корректировать навигацию достаточно просто. Данный компонент встраивается в компонент HeaderComp.

NavbarComp.vue

```
<template>
    <nav class="navbar center">
        <div
            @click="$router.push('/')"
```

```
            class="navbar_logo"
        >
            <h3>of eco-style</h3>
        </div>
        <div class="navbar_btns">
            <MyButton @click="$router.push('/posts')">
                Блог
            </MyButton>
            <MyButton @click="$router.push('/about')">
                О нас
            </MyButton>
        </div>
    </nav>
</template>

<script>
export default {
    name: 'NavbarComp',
};
</script>

<style lang="scss" scoped>
.navbar {
    width: 100%;
    height: 100%;
    background-color: none;
    display: flex;
    justify-content: space-between;
    align-items: center;
```

```
    &_logo {
        width: 200px;
        color: #2F4209;
        text-align: center;
        font-size: 28px;
        font-style: normal;
        font-weight: 700;
        line-height: normal;
        transition: 0.2s;
        &:hover {
            text-shadow: 0px 5px 5px
                         rgba(63, 68, 51, 0.59);
        }
    }

    &_btns {
        // width: 400px;
        display: flex;
        gap: 12px;
    }
}
@media (max-width: 1024px) {
    .navbar {
        &_logo {
            width: 100px;
            font-size: 14px;
        }
    }
}
</style>
```

HeaderComp.vue

Компонент HeaderComp.vue представляет собой верхнюю часть страницы или заголовок. Он обычно содержит элементы навигации, логотип сайта, ссылки на профиль пользователя и другие важные элементы интерфейса.

В данном случае, header включает в себя navbar и стили. В дальнейшем, если понадобится разработать дополнительный функционал, можно будет дополнить его другими компонентами. При этом он уже подключен ко всем страницам сайта через корневой компонент App.vue для обеспечения единообразного внешнего вида и функциональности.

HeaderComp.vue

```
<template>
    <header class="header">
        <NavbarComp></NavbarComp>
    </header>
</template>

<script>
import NavbarComp from './NavbarComp.vue';

export default {
    name: 'HeaderComp',
    components: { NavbarComp }
};
</script>

<style lang="scss" scoped>
.header {
    width: 100%;
```

```
        min-height: 70px;
        background-color: rgba(93, 137, 3, 0.30);
        display: flex;
        justify-content: space-between;
        align-items: center;
    }
    </style>
```

FooterComp.vue

Компонент FooterComp.vue представляет собой нижнюю часть страницы веб-приложения или сайта. Он содержит информацию, которая отображается на всех страницах и обычно включает в себя ссылки на социальные сети, контактную информацию, полезные ссылки и копирайт.

Когда компонент "Footer.vue" используется веб-приложением или сайтом, он обычно подключается и размещается внизу каждой страницы для обеспечения единообразия дизайна и обеспечения удобства пользователей при поиске информации или связи с владельцем сайта или компанией. Он также подключен ко всем страницам сайта через корневой компонент App.vue для обеспечения единообразного внешнего вида и функциональности.

FooterCop.vue

```
    <template>
        <footer class="footer">
            <p class="footer_text center">напишите нам: schnauzer-eco@po4ta.ru</p>
        </footer>
    </template>
    <script>
```

```
export default {
    name: 'FooterComp',
};
</script>
<style lang="scss" scoped>
.footer {
    min-width: 100%;
    min-height: 70px;
    background-color: rgba(93, 137, 3, 0.30);;
    display: flex;
    justify-content: flex-start;
    align-items: center;
    &_text {
        color: #2F4209;
        text-align: center;
        font-size: 22px;
        font-style: normal;
        font-weight: 700;
        line-height: normal;
    }
}
@media (max-width: 1024px) {
    .footer {
        justify-content: center;
        &_text {
            font-size: 14px;
        }
    }
}
</style>
```

PostForm.vue

Компонент PostForm создан для того, чтоб была возможность использовать выплывающую форму в том случае, если она нам понадобится в дальнейшем в другом месте нашего приложения. Или если будет необходимость добавить редактирование поста.

В данном компоненте импортируем два компонента MyInput и компонент MyButton с генерированным событием `@click="createPost"`, которым мы вызываем `$emit` с названием генерируемого события `'create'` и передаем `post` родительскому компоненту (PostPage). В свою очередь, в родительском компоненте мы подписываемся на это событие и создаем функцию по добавлению поста в массив постов.

Создаем модель post с полями title и body. В поле methods создаем функцию `createPost()`, в которой создаем объект поста. У объекта формируется уникальный id, с помощью функции `Date.now()` получим его из текущей даты, предаем эмит, затем очищаем title и body, чтобы инпуты были пустые.

Вешаем на форму `@submit.prevent` — это атрибут элемента формы HTML, который предотвращает отправку формы при нажатии на кнопку "submit" или клавишу Enter. Значение "prevent" указывает, что при отправке формы необходимо предотвратить стандартное поведение браузера, то есть не перезагружать страницу или не выполнять другие действия, которые обычно связаны с отправкой формы.

Директива v-model для двустороннего связывания позволяет связать вводимые в input данные с конкретной моделью в компоненте, а именно записывает заголовок и содержание поста. Одновременно убираем лишние пробелы в начале и в конце строки, используя строковый метод для удаления пробелов в начале и конце строки - модификатор `.trim`.

PostForm.vue

```
<template>
    <form @submit.prevent class="postForm center">
        <MyInput
            v-model.trim="post.title"
            type="text"
            placeholder="Заголовок поста"
        />
        <MyInput
            v-model.trim="post.body"
            type="text"
            placeholder="Текст поста"
        />
        <div class="btn-form">
            <MyButton
                @click="createPost"
            >
                Сохранить
            </MyButton>
        </div>
    </form>
</template>

<script>
export default {
    name: "PostForm",
    data() {
        return {
            post: {
                title: '',
```

```
                body: ''
            }
        };
    },

    methods: {
        createPost() {
            this.post.id = Date.now();
            this.$emit('create', this.post)
            this.post = {
                title: '',
                body: ''
                }
        },
    }
};
</script>

<style lang="scss" scoped>
.postForm {
    display: flex;
    flex-direction: column;
    padding: 24px;
}
.btn-form {
    display: flex;
    justify-content: flex-end;
    margin-top: 24px;
}
</style>
```

PostItem.vue

Компонент PostItem служит для отрисовки поста. В нем мы расположили кнопку удаления поста, для этого мы импортировали компонент MyButton, а также интерполируем заголовок и содержание поста. Компонент принимает props – пост , тип которого – объект и этот пропс обязательный. Добавляем кнопку для удаления поста Событие @click="$emit('remove', post)" передаем выше в компонент PostList, в котором список постов принимаем в качестве пропсов.

PostItem.vue

```
<template>
    <div class="post center">
        <div>
           <h3 class="post_title">{{ post.title }}</h3>
           <p class="post_text">{{ post.body }}</p>
        </div>
        <div class="post_btns">
            <MyButton
                @click="$emit('remove', post)"
            >
                Удалить
            </MyButton>
        </div>
    </div>
</template>
<script>
import MyButton from './UI/MyButton.vue';
export default {
    name: 'PostItem',
    props: {
```

```
        post: {
            type: Object,
            required: true
        }
    },
    components: { MyButton }
}
</script>
<style lang="scss" scoped>
.post {
    padding: 12px;
    border: 2px solid rgba(93, 137, 3, 0.30);
    border-radius: 12px;
    margin-top: 12px;
    display: flex;
    align-items: center;
    justify-content: space-between;
    color: #2F4209;
    text-align: center;
    font-style: normal;
    font-weight: 700;
    line-height: normal;
    &_title{
        font-size: 25px;
        margin-bottom: 12px;
    }
    &_text {
        font-size: 18px;
    }
    &_btns {
```

```
            display: flex;
            gap: 8px;
        }
    }
    @media (max-width: 1024px) {
        .post {
            flex-direction: column;
            justify-content: center;
            gap: 12px;
            &_text {
                font-size: 14px;
            }
        }
    }

    </style>
```

PostList.vue

Следующий компонент – это PostList, в который импортируем раннее созданный компонент PostItem и также прослушиваем событие `@remove`, делаем `$emit` и отдаем событие выше в следующий компонент – PostPage. Компонент PostList также может переиспользоваться, поэтому массив постов и модели здесь не содержатся, а принимаются извне пропсами - принимаем массив постов делаем их обязательными. С помощью специальной директивы v-for мы проходим по массиву постов. Обязательно используем для ключа id. Байндим пост из компонента PostItem `:post="post"`

В стилях добавляем анимацию, для того, чтобы посты при загрузке страницы, записи поста и удалении поста плавно сдвигались. Пример берем из документации Vue.js

Во Vue.js <TransitionGroup> — это компонент, предоставляемый Vue.js, который позволяет создавать анимированные переходы для элементов, изменяющихся или добавляемых в список.

<TransitionGroup> используется для группировки компонентов <transition> внутри списка или контейнера и применения анимации к каждому элементу внутри группы при изменении списка.

Когда элемент добавляется в группу или удаляется из нее, <TransitionGroup> применяет анимацию, указанную в компонентах <transition>, к элементу, чтобы создать анимированный переход.

Для того, чтобы в ситуации, когда у нас массив постов пустой, пользователь понимал это, используем директивы v-if v-else. В таком случае у нас отрисовывается тот блок, который соответствует заданным условиям. Если массив пустой – появляется соответствующая надпись.

PostList.vue

```
<template>
    <div class="post-list center">
        <div v-if="posts.length > 0">
            <TransitionGroup name="postlist">
                <PostItem
                    v-for="post in posts"
                    :post="post"
                    :key="post.id"
                    @remove="$emit('remove', post)"
                />
            </TransitionGroup>
        </div>
        <h2 v-else class="post-list_zero-blog">
             Пока в нашем блоге нет записей </h2>
    </div>
```

```
</template>
<script>
import PostItem from '@/components/PostItem';
export default {
    components: {PostItem},
    props: {
        posts: {
            type: Array,
            required: true
        }
    },
    name: 'PostList',
};
</script>
<style lang="scss" scoped>
.post-list {
    width: 100%;
    margin-top: 24px;
    margin-bottom: 24px;
    &_zero-blog {
        color: #2F4209;
    }
}
.postlist-item {
    display: inline-block;
    margin-right: 10px;
}
.postlist-enter-active,
.postlist-leave-active {
    transition: all 0.3s ease;
```

```
}
.postlist-enter-from,
.postlist-leave-to {
    opacity: 0;
    transform: translateX(130px);
}
.postlist-move {
    transition: transform 0.3s ease;
} </style>
```

src/pages

Во Vue.js страницы создаются путем комбинирования компонентов, которые можно подключить внутри другого компонента, используя теги или директивы, такие как <router-view> в App.vue. Таким образом компоненты взаимодействуют между собой, обеспечивая гибкость и модульность кода. Рассмотрим страницы ниже.

MainPage.vue

Главная страница минималистичная. На ней представлена ключевая информация о питомнике и кнопки с переходом на страницы блога и “о нас”. Ниже приводится код.

MainPage.vue

```
<template>
    <div class="content center">
        <main class="main">
            <div class="main_photo">
                <img
                    src="./../../public/mainDog.png"
```

```
                    alt="photo of a dog"
                >
            </div>
            <div class="main_info">
                <div class="main_info__subtitle">
                    <h2
                      class="main_info__subtitle-text">
                       Ризеншнауцеры Цвергшнауцеры
                    </h2>
                </div>
                <div class="main_info__title">
                    <h1
                       class="main_info__title-text">
                       Питомник of eco-style
                    </h1>
                </div>
            </div>
        </main>
        <div class="description">
            <div>
                <p class="description_text">Шнауцеры -
```
собаки с благородным характером. Они являются отличными компаньонами и любящими членами семьи, энергичные, умные и ориентированные на человека, идеальные партнеры для активных владельцев и для семей с детьми.
```
                </p>
            </div>
            <div>
                <div class="description_txt-btn">
                  <p
```

```
    class="description_text description_txt-btn__txt">
                                Вы можете прочитать статьи о
породе, увидеть наших щенков, узнать расписание выставок и
многое другое в нашем блоге
                        </p>
                        <div
                          class="description_txt-btn__btn">
                             <MyButton
                          @click="$router.push('/posts')"
                              >Блог</MyButton>
                         </div>
                    </div>
                </div>
            </div>
        </div>
    </template>


    <script>
    import MyButton from '@/components/UI/MyButton.vue';

    export default {
        name: 'MainPage',
        components: { MyButton }
    };
    </script>


    <style lang="scss" scoped>
    .content {
```

```
    max-width: auto;//100%;
    height: 900px;
    margin-top: 32px;
    display: flex;
    flex-wrap: wrap;
    gap: 64px;
}
.main {
    max-width: 100%;
    height: 524px;
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin: auto;
    &_photo {
        width: 50%;
        padding-top: 64px;
        padding-bottom: 52px;
        display: flex;
        justify-content: center;
    }
    &_info {
        width: 50%;
        height: 523px;
        padding: 36px 84px 0;
        display: flex;
        flex-direction: column;
        justify-content: center;
        gap: 117px;
        height: 136px;
```

```
            color: #2F4209;
            text-align: center;
            font-style: normal;
            font-weight: 700;
            line-height: normal;
            &__subtitle {
                width: 500px;
                height: 136px;
                display: flex;
                justify-content: center;
            }
            &__subtitle-text {
                font-size: 55px;
            }
            &__title {
                width: 422px;
                height: 234px;
                display: flex;
                justify-content: center;
            }
            &__title-text {
                font-size: 70px;
                 text-shadow: 0px 20px 20px rgba(144, 191,
49, 0.59);
            }
        }
    }
    .description {
        max-height: auto;
        display: flex;
```

```
    flex-direction: column;
    justify-content: center;
    align-items: center;
    gap: 66px;
    &_text {
        color: #2F4209;;
        text-align: center;
        font-size: 25px;
        font-style: normal;
        font-weight: 700;
        line-height: normal;
    }
    &_txt-btn {
        display: flex;
        justify-content: space-between;
        align-items: center;
        flex-wrap: wrap;
        gap: 66px;
        &__txt {
            width: calc(100% / 6 * 4);
        }
        &__btn {
            width: calc(100% / 6);
            display: flex;
            justify-content: flex-end;
        }
    }
}
@media (max-width: 1140px) {
    .content {
```

```
        gap: 0px;
        height: auto;
    }
}
@media (max-width: 1024px) {
    .content {
        height: auto;
        gap: 50px;
    }
    .main {
        // min-height: 344px;
        min-height: 344px;
        flex-direction: column;
        &_photo {
            width: 100%;
            padding-top: 0;
            padding-bottom: 0;
        }
        &_photo img{
            max-width:100%;
            height: auto;
        }
        &_info {
            width: 100%;
            height: 100%;
            padding: 12px 12px 0;
            align-items: center;
            gap: 48px;
            &__subtitle {
                width: 100%;
```

```
            height: auto;
        }
        &__subtitle-text {
            font-size: 32px;
        }
        &__title {
            width: 306px;
            height: auto;;
        }
        &__title-text {
            font-size: 52px;
        }
    }
}
.description {
    margin-bottom: 36px;
    gap: 50px;
    justify-content: center;
    &_text {
        font-size: 18px;
        width: 100%;
    }
    &_txt-btn {
        flex-direction: column;
        width: 100%;
        &__btn {
            width: 100%;
            justify-content: center;
        }
    }
```

```
    }
}
@media (max-width: 360px) {
    .main {
        height: auto;
        flex-direction: column;
        justify-content: center;
    }
    .description {
        &_txt-btn {
            flex-direction: column;
            justify-content: center;
            &__txt {
                width: 100%;
            }
            &__btn {
                width: 100%;
                justify-content: center;
            }
        }
    }
}
</style>
```

PostPage.vue

Компонент PostPage является страницей с постами. Сюда импортируются все необходимые для этого компоненты. Ниже будет рассмотрено то, что “происходит” в данном компоненте.

Подписываемся на событие от “ребенка” PostForm `'create'` и создаем функцию (которая отработает по этому событию) по добавлению поста в массив постов `@create="createPost"`. После создания поста диалоговое окно закрывается, поменяв `dialogVisible` на true.

Еще одно событие `'remove'`, прокинутое из PostItem через PostList. В методах создаем функцию `'removePost'` для удаления поста из массива постов.

Здесь же компонент MyDialog, у которого в slot помещен компонент PostForm. Делаем двустороннее связывание с компонентом MyDialog, а пропс show передадим как атрибут v-model `v-model:show="dialogVisible"`. Для этого создаем модель `dialogVisible: false` в data и передаем ее в директиву v-model.

Кнопка для создания поста при нажатии вызывает функцию `showDialog`, в которой меняется значение `dialogVisible` на true. Это позволяет открывать и закрывать диалоговое окно.

Для взаимодействия с сервером создаем асинхронную функцию `fetchPosts()`. Для имитации получения постов с сервера будем использовать сервис jsonplaceholder. Для запросов используется библиотека axios. Устанавливаем ее командой “npm i axios”. Устанавливается значение переменной isPostLoading в true, что означает, что процесс загрузки постов начался. Выполняется запрос на сервис https://jsonplaceholder.typicode.com/posts с использованием метода get. В запросе передаются параметры _page и _limit, которые определяют номер страницы и количество постов, которые нужно загрузить. Полученный ответ сохраняется в переменную response. Высчитывается

количество страниц totalPage. Это делается путем деления общего количества постов на количество постов на одной странице (_limit), с округлением в большую сторону. Полученные посты сохраняются в переменную posts. Если возникает ошибка, выводится сообщение "Ошибка" в виде всплывающего окна. Независимо от успешности загрузки постов, переменная isPostLoading устанавливается в false, указывая, что процесс загрузки закончился.

В шаблоне в PostList используется `v-if="!isPostLoading"` , чтобы показывать загруженые посты, в противном случае появляется соответствующая надпись "загрузка" – для этого используется директива v-else.

Для динамической подгрузки постов использован хук жизненного цикла mounted. Хук жизненного цикла "mounted" во Vue.js срабатывает после создания экземпляра Vue и монтирования компонента в DOM. Когда компонент монтируется, все его дочерние компоненты и директивы также монтируются. Когда монтирование завершено, вызывается хук "mounted". Это означает, что все шаблоны компонента были скомпилированы в виртуальные узлы и добавлены в DOM. В хуке "mounted" можно выполнять действия, которые требуют наличия DOM элементов, такие как инициализация сторонних библиотек, получение данных через API, установка слушателей событий или обновление компонентов и их данные.

В данном случае в хуке "mounted" вызвана функция `fetchPosts()`, чтобы посты подгрузились сразу с открытием страницы с постами.

Для сортировки используется computed: `sortedPosts`. Computed свойства во Vue.js используются для вычисления значений на основе зависимостей от других свойств или данных. Они используются для вычисления значений, которые могут изменяться динамически, и эти значения автоматически обновляются при изменении его зависимостей.

Computed свойства имеют несколько преимуществ:

1. Они кэшируют свои вычисленные значения, что означает, что они не будут пересчитываться каждый раз, когда изменяются его зависимости, если это необходимо.
2. Они предоставляют более декларативный способ вычисления значений, чем методы, их отличие в том, что они используются в шаблоне как свойства.
3. Они автоматически обновляются при изменении их зависимостей, что позволяет легко организовать логику, которая зависит от изменения данных или свойств.

Computed свойства особенно полезны, когда нужно преобразовать или фильтровать значения, построить сложные вычисления на основе нескольких свойств или данных, или организовать сложную логику для отображения данных.

Для поиска используется компонент MyInput. Двустороннее связывание `v-model="searchQuery"` и делаем computed поиска по отсортированному массиву.

Для реализации бесконечной ленты постов используется функция `"loadMorePosts"` , которая похожа на функцию “fetchPosts”, но в которой после получения ответа от сервера посты не перезаписываются, а добавляются в конец массива, когда пользователь пролистал страницу «до края». Для этого используется Intersection Observer API — это API, предоставляемый браузером, который позволяет отслеживать, когда элемент на веб-странице становится видимым при прокрутке страницы или появлении в окне просмотра. Он предоставляет способ асинхронно и эффективно отслеживать изменения взаимодействия элемента с его окружением. Это особенно полезно для создания веб-страниц с многоэлементными областями содержимого, бесконечной прокрутки и других динамических функций. С его помощью можно добавить обработчик событий, который будет вызываться, когда элемент входит в окно просмотра, выходит из него или взаимодействует с ним другим образом. `ref="observer"` в шаблоне указывает на элемент, за которым нужно следить. На сайте developer.mozilla.org представлен пример использования, который

используется в проекте в mounted(). В строке this.fetchPosts(); вызывается функция fetchPosts(), которая делает запрос на сервер и получает посты. Создается объект options, в котором задаются опции для объекта IntersectionObserver. Опции включают rootMargin (отступы от границы корневой области), и threshold (порог). Создается функция callback, которая будет вызвана, когда произойдет пересечение наблюдаемого элемента. В этой функции проверяется, пересеклись ли элементы, и если да, и если this.page меньше this.totalPages, то вызывается функция loadMorePosts(), которая, скорее всего, загружает еще посты. Создается экземпляр IntersectionObserver с передачей в него функции callback и объекта options. Наблюдатель начинает наблюдать за элементом, указанным в this.$refs.observer. Когда происходит пересечение элемента, вызывается функция callback.

PostPage.vue

```
<template>
    <div class="center">
        <MyInput
            v-model="searchQuery"
            placeholder="Поиск"
        />
        <div class="search-btn">
            <MyButton
                @click="showDialog"
            >
                Создать пост
            </MyButton>
          <MySelect
                v-model="selectedSort"
                :options="sortOptions"
            />
        </div>
```

```
        </div>
        <MyDialog v-model:show="dialogVisible">
            <PostForm
                @create="createPost"
            />
        </MyDialog>
        <PostList
            :posts="sortedAndSearhedPosts"
            @remove="removePost"
            v-if="!isPostLoading"
        />
        <div v-else>Загрузка...</div>
        <div ref="observer"></div>
    </div>

<script>
import PostForm from '@/components/PostForm.vue';
import PostList from '@/components/PostList.vue';
import MyDialog from '@/components/UI/MyDialog.vue';
import MyButton from '@/components/UI/MyButton.vue';
import axios from 'axios';

export default {
    name: 'PostPage',
    components: { PostForm,
                  PostList,
                  MyDialog,
                  MyButton,
                  },
```

```
data() {
    return {
        posts: [],
        dialogVisible: false,
        isPostLoading: false,
        selectedSort: '',
        searchQuery: '',
        page: 1,
        limit: 20,
        totalPages: 0,
        sortOptions: [
            {value: 'title', name: 'По названию'},
           {value: 'body', name: 'По содержанию'},
        ]
    }
},
methods: {
    createPost(post){
        this.posts.unshift(post);
        this.dialogVisible = false;
    },
    removePost(post) {
        this.posts = this.posts.filter(p =>
                            p.id !== post.id)
    },
    showDialog() {
        this.dialogVisible = true;
    },

    // загружаем данные
```

```
        async fetchPosts() {
            try {
                this.isPostLoading = true;
                const response = await axios.get
        ('https://jsonplaceholder.typicode.com/posts',
                 {
                        params: {
                            _page: this.page,
                            _limit: this.limit
                        }
                 }
        );
                // высчитываем ск-ко страниц
                this.totalPages = Math.ceil(
                     response.headers['x-total-count']
                     / this.limit)
                this.posts = response.data;
            } catch (error) {
                alert('Ошибка');
            } finally {
                this.isPostLoading = false;
            }
        },
        // для загрузки бесконечной ленты постов
        async loadMorePosts() {
            try {
                this.page += 1;
                const response = await axios.get
    ('https://jsonplaceholder.typicode.com/posts', {
                    params: {
```

```
                        _page: this.page,
                        _limit: this.limit
                    }
                }
);
                // высчитываем ск-ко страниц.
                this.totalPages = Math.ceil(
                     response.headers['x-total-count']
                     / this.limit)
            // добавляем посты в конец массива для ленты
                this.posts = [
                                ...this.posts,
                                ...response.data
                            ];
            } catch (error) {
                alert('Ошибка');
            }
        },
    },
    mounted() {
        this.fetchPosts();
        // начинаем следить за перечечением
        const options = {
            rootMargin: "0px",
            threshold: 1.0,
        };
        // отработает, когда пересечем эл-т.Единожды !
        const callback = (entries, observer) => {
            if(entries[0].isIntersecting
             && this.page < this.totalPages) {
```

```
                    this.loadMorePosts();
                }
            };
            const observer = new IntersectionObserver(
                                    callback, options);
            observer.observe(this.$refs.observer);
        },
        computed: {
            sortedPosts() {
              return [...this.posts].sort((post1, post2) =>
                  post1[this.selectedSort]?
                  .localeCompare(post2[this.selectedSort]))
            },
            sortedAndSearhedPosts() {
               return this.sortedPosts.filter(post =>
                  post.title.toLowerCase()
                  .includes(this.searchQuery.toLowerCase()))
            }
        },
}
</script>

<style lang="scss">
.search-btn {
    margin-top: 24px;
}
</style>
```

AboutPage.vue

На странице AboutPage содержится кратко информация о заводчике, представлена фотография. Здесь можно разместить информацию о целях и миссии, основных достижениях, истории, контактных данных и любой другой информации, которая может быть полезной для пользователей, которые хотят узнать больше о том, что представлено на веб-сайте или блоге. Создание такой страницы помогает устанавливать доверие и связь с посетителями, а также продвигать основные сообщения и цели проекта. Временно используется фотография с сайта freepik.com.

AboutPage.vue

```
<template>
 <div class="about center">
   <div class="about_photo">
            <img src="./../../public/shef.jpg"
             alt="заводчик">
   </div>
   <div class="about_freepik">Изображение от
      <a href="https://ru.freepik.com/free-photo/
      medium-shot-woman-and-dog-in-
  bed_18407871.htm#page
  =2&query=%D0%B6%D0%B5%D0%BD%D1%89%D0%B8%D0
  %BD%D0%B0%20%D1%81%20%D1%86%D0%B2%D0%B5%D1
  %80%D0%B3%D1%88%D0%BD%D0%B0%D1%83%D1%86%D0
  %B5%D1%80%D0%BE%D0%BC&position=24&from_view
  =search&track=ais&uuid=947e3849-1c9c-4749-
  95f7-93d24e2bec61">Freepik</a>
```

```
    </div>
    <div class="about_text">
          <p>Я - заводчик собак породы цвергшнауцер и ризеншнауцер.</p>
          <p>Это призвание, любовь и большая ответственность.</p>
       <p>Мои питомцы имеют хорошие породные качества, красивый внешний вид и чудесный характер.</p>
      <p>Если вы хотите узнать больше об этой прекрасной породе, увидеть щенков или узнать новости о мероприятиях, в которых мы учавствуем, вы можете прочитать наш блог.</p>
       <p>Если есть вопросы, напишите: </p>
       <p>schnauzer-eco@po4ta.ru</p>
    </div>
   </div>
</template>

    <script>
    export default {
        name: 'AboutPage',
    };
    </script>

    <style lang="scss" scoped>
    .about {
        width: 100%;
        display: flex;
        flex-direction: column;
        justify-content: center;
        align-items: center;
```

```
    margin: 60px auto;
    gap: 60px;
    color: #2F4209;
    text-align: center;
    font-style: normal;
    font-weight: 700;
    line-height: normal;
    &_photo {
        max-width: 600px;
        max-height:900px;
        & img {
            width: 100%;
            border-radius: 12px;
        }
    }
    &_freepik {
        font-size: 12px;
    }
    &_text {
        border: 2px solid rgba(93, 137, 3, 0.30);
        border-radius: 12px;
        padding: 24px;
        display: flex;
        flex-direction: column;
        justify-content: center;
        gap: 24px;
        font-size: 24px;
    }
}
</style>
```

NotFound.vue

Страница NotFound (или 404 страница) создается для того, чтобы предоставить пользователю информацию о том, что запрашиваемая им страница не найдена на сервере. Она используется для уведомления пользователя о том, что ресурс, к которому он пытается получить доступ, не существует или был перемещен.

NotFound.vue

```
<template>
    <div class="not-found center">
        <h2 class="not-found_num">404</h2>
        <h2>страница не найдена</h2>
    </div>
</template>

<script>
export default {
    name: 'NotFound',
};
</script>

<style lang="scss" scoped>
.not-found {
    display: flex;
    flex-direction: column;
    gap: 50px;
    justify-content: center;
    padding-top: 10%;
    color: #2F4209;
    text-align: center;
```

```
        font-family: Merriweather Sans;
        font-size: 36px;
        font-style: normal;
        font-weight: 700;
        line-height: normal;
        &_num {
            text-shadow: 0px 20px 20px rgba(144, 191, 49,
0.59);
            font-size: 70px;
        }
    }
    </style>
```

src/router

router.js

Во Vue.js, router используется для создания и управления маршрутами веб-приложения. Он позволяет определить пути (URL) и связанные с ними компоненты, а также управлять навигацией между этими компонентами.

С помощью router в Vue.js можно реализовать следующие функции:

1. Определение маршрутов: Можно определить различные маршруты и указать соответствующие компоненты для каждого маршрута. Например, маршрут "/" связан с компонентом MainPage, а "/about" с компонентом AboutPage.
2. Навигация: router позволяет легко переходить между различными маршрутами в приложении. Можно использовать маршруты в виде ссылок или программно переходить по маршрутам с помощью методов, предоставляемых router.

3. Передача параметров: router позволяет передавать параметры в маршруты, которые можно использовать в компонентах. Например, можно передать идентификатор пользователя, который будет использоваться для отображения подробной информации о пользователе.
4. Вложенные маршруты: router также поддерживает вложенные маршруты, что позволяет создавать более сложные маршрутные структуры. Например, можно иметь основной маршрут "/users" и вложенные маршруты "/users/:id" для отображения информации о каждом отдельном пользователе.

В целом, router в Vue.js облегчает управление навигацией и маршрутизацией веб-приложения, делая его более модульным и масштабируемым.

Регистрируем роутер в приложении. Для этого в main.js импортируется router и передается в функции .use router: `app.use(router)`.

router.js

```
import { createRouter, createWebHistory } from 'vue-router'
import MainPage from '@/pages/MainPage'
import PostPage from '@/pages/PostPage'
import AboutPage from '@/pages/AboutPage'
import NotFound from '@/pages/NotFound'

const routes = [
  {
    path: '/',
    name: 'main',
    component: MainPage
  },
  {
    path: '/posts',
```

```
    name: 'posts',
    component: PostPage
  },
  {
    path: '/about',
    name: 'about',
    component: AboutPage
  },
  {
    path: '/:pathMatch(.*)*',
    name: '404',
    component: NotFound,
  },
]

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

export default router
```

App.vue

App.vue – корневой компонент приложения. Ниже представлен код данного компонента. Его отличием является то, что только в нём помещены Header и Footer так, чтоб затем в остальных страницах приложения не повторять эти блоки и router-view, который рассматривался выше. Так же в этом компоненте записаны те стили, которые будут использованы для всего проекта.

App.vue

```
<template>
  <HeaderComp/>
  <div class="app">
    <router-view></router-view>
  </div>
  <FooterComp/>
</template>
<script>
import HeaderComp from '@/components/HeaderComp.vue';
import FooterComp from '@/components/FooterComp.vue';

export default {
  components: { HeaderComp, FooterComp },
}
</script>
<style lang="scss">
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: 'Merriweather Sans', sans-serif;
```

```
}
a {
    text-decoration: none;
}
.app {
  min-height: calc(100vh - 140px);
}
.center {
  padding-left: calc(50% - 700px);
  padding-right: calc(50% - 700px);
}
@media (max-width: 1140px) {
  .center {
    padding-left: calc(50% - 510px);
    padding-right: calc(50% - 510px);
  }
}
@media (max-width: 1024px) {
  .center {
    padding-left: calc(50% - 450px);
    padding-right: calc(50% - 450px);
  }
}
@media (max-width: 1024px) {
  .center {
    padding-left: calc(50% - 172px);
    padding-right: calc(50% - 172px);
  }
}
</style>
```

Коротко о VueX

Vuex — это библиотека для управления состоянием приложения во фреймворке Vue.js. Она предоставляет централизованное хранилище данных, доступное для всех компонентов приложения, при этом используются правила, гарантирующие изменение состояния приложения определенным образом.

Vuex используется в проектах, где необходимо управлять сложным состоянием приложения, иметь доступ к данным из разных компонентов и обеспечить синхронизацию данных между компонентами.

Преимущества использования Vuex:

1. Централизованное хранилище: Vuex создает единую точку доступа ко всем данным приложения. Это позволяет упростить управление состоянием, взаимодействие компонентов и отслеживание изменений данных.
2. Реактивность: Vuex обеспечивает реактивность данных, что означает, что всякая модификация состояния будет отслеживаться автоматически. Это упрощает реагирование на изменения и обновление компонентов при необходимости.
3. Отделение логики состояния: Vuex позволяет явно определить и разделить логику состояния приложения от компонентов. Это повышает понятность кода, его модульность и повторное использование.
4. Удобство отладки: Vuex предоставляет инструменты для удобной отладки состояния приложения на высоком уровне. Это включает в себя инструменты разработчика с возможностью просмотра текущего состояния, мутаций и действий.
5. Возможность расширения: Vuex можно легко расширить с помощью плагинов для реализации различных функций, таких как сохранение состояния в localStorage, асинхронные операции и другие.

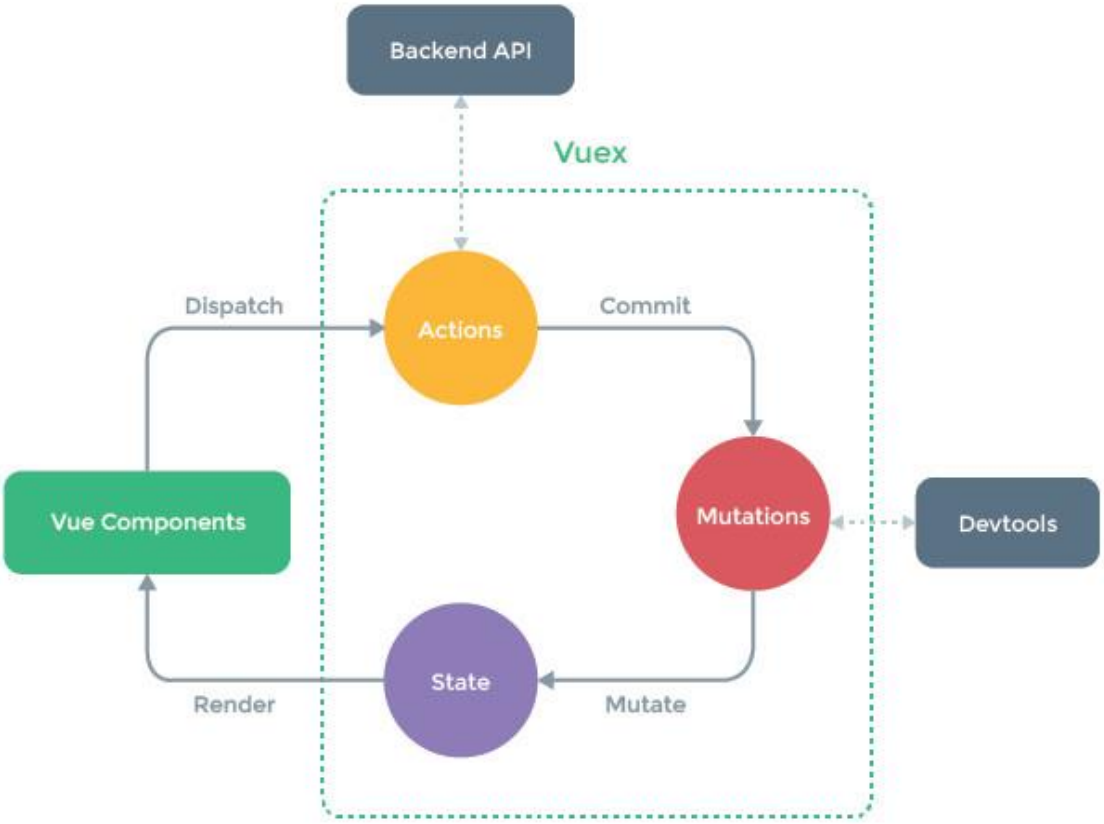

Рис 8. Схема работы VueX (https://vuex.vuejs.org)

На рис. 8 представлено схематичное изображение работы VueX. State – это состояние приложения, в котором содержатся определенные данные. Эти данные state отправляет вью компоненту (Vue Components). В свою очередь, вью компонент их принимает и отрисовывает с помощью рендера (Render). В компоненте могут происходить события, тогда при помощи функции Dispatch можем вызвать Actions, которые м.б. асинхронные, так как они работают с бэкендом (Backend API). Затем, когда данные подгружаются или изменяется что-либо в экшенах, мы можем вызвать мутации, используя метод коммит (commit). Мутации изменяют напрямую стэйт и т.д. далее по схеме.

Некоторые из минусов использования Vuex:

1. Повышенная сложность: Использование Vuex может повысить сложность разработки приложения, особенно для небольших проектов, где его применение может быть избыточным.
2. Дополнительные шаги при разработке компонентов: Для использования состояний Vuex, вам потребуется сначала специальным образом объявить его в компонентах и использовать экшены и мутации для изменения состояния, что добавляет дополнительные шаги и требует дополнительного понимания концепций Vuex.
3. Потребление памяти: Использование Vuex может привести к увеличенному потреблению оперативной памяти, поскольку данные хранятся в централизованном хранилище и для доступа к ним требуется дополнительный уровень абстракции.

В целом, использование Vuex может быть очень полезным для средних и больших проектов, где требуется эффективное управление состоянием и совместное использование данных между несколькими компонентами. Однако, для маленьких или простых проектов его использование может быть излишним и сложно поддерживаемым.

В процессе разработки данного проекта было решено отказаться от использования vuex, поскольку это приводило к усложнения. Возможно, при развитии проекта, и добавление дополнительных опций, например реализация сопутствующих товаров, или запись на груминг, или добавление развернутых веток комментариев, возможностью добавлять и посетителям сайта свои записи в блоге, модерацией и т.д. будет удобным использовать vuex. В этом случае фреймворк Vue.js позволяет модернизировать проект достаточно пластично.

Рис. 9. Целесообразность использования VueX (https://vuex.vuejs.org)

На рис.9 дан скриншот официального сайта vuex.vuejs.org/guide/state, когда может быть удобен данный инструмент. Изначально vuex он был подключен к проекту при создании, планировалось использование store, и была создана директория store с index.js . Данный файл будет добавлен в приложение в том виде, в котором была закончена работа над ним.

Однако в конкретном примере простого блога использование vueX приводила не к упрощению, а, по мнению автора, к усложнению разработки и было принято решение отказаться от него.

ЗАКЛЮЧЕНИЕ

В заключение дипломного проекта на тему "Создание одностраничного сайта питомника собак с блогом" разработанного на Vue.js, можно отметить следующие преимущества использования этого фреймворка.

Во-первых, Vue.js является одним из наиболее популярных фреймворков для разработки веб-приложений. Он имеет небольшой размер, что позволяет быстро загружать страницы и улучшить пользовательский опыт. Vue.js также обеспечивает быструю реакцию на изменения данных благодаря использованию двустороннего связывания данных. Также стоит отметить возможность компонентного подхода в разработке на Vue.js. Это позволяет нам создавать отдельные компоненты, которые могут быть повторно использованы в разных частях приложения. Кроме того, Vue.js упрощает работу с динамическими контентом и адаптивным дизайном, благодаря возможности использования директив, фильтров и компонентов.

Однако, необходимо отметить, что в нашем проекте мы не использовали бэкенд для аутентификации, что может представлять некоторые уязвимости в системе. Аутентификация является важной частью любого сайта, особенно если на сайте предусмотрена возможность взаимодействия пользователей, например, в комментариях блога или в форме для заказа щенков. Для обеспечения безопасности и защиты данных пользователей, рекомендуется добавить бэкенд, который будет отвечать за аутентификацию пользователей и обработку запросов.

Однако, наш проект имеет потенциал для улучшения и дальнейшей разработки. Например, мы можем добавить дополнительный функционал, такой как возможность регистрации пользователей, личные кабинеты для каждого пользователя, возможность оставлять комментарии, а также дополнительные страницы с более подробной информацией о питомнике и собаках. Также, мы можем улучшить дизайн сайта, сделав его более интересным и современным. Для этого можно применить современные технологии веб-дизайна, использовать

анимацию, специальные эффекты и другие элементы, которые создадут приятное визуальное впечатление для пользователей.

В целом, создание одностраничного сайта питомника собак с блогом на Vue.js имеет много преимуществ. Vue.js — это легкий и гибкий фреймворк, который позволяет разрабатывать веб-приложения быстро и эффективно. Однако, для обеспечения безопасности и надежности аутентификации рекомендуется добавить бэкенд. Наш проект имеет потенциал для дальнейшего улучшения и развития, включая дополнительный функционал и современный дизайн. В общем, разработка на Vue.js позволяет создавать мощные и интерактивные веб-приложения, открывая широкие возможности для разработчиков.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Axios. Официальная документация - https://axios-http.com
2. HTML Academy. Статьи о JavaScript и Vue.js - https://htmlacademy.ru/blog/js
3. JSONPlaceholder. Бесплатный поддельный API для тестирования и прототипирования - https://jsonplaceholder.typicode.com
4. MDN Web Docs - https://developer.mozilla.org
5. METANIT. Сайт о программировании - https://metanit.com
6. Stack Overflow. Система вопросов и ответов о программировании - https://stackoverflow.com/
7. Vue.js. Официальная документация - https://v3.ru.vuejs.org/
8. W3Schools. Школа веб-разработчиков - https://www.w3schools.com
9. WEB Учебники - https://www.schoolsw3.com/
10. Webdevblog. Еще один блог веб-разработчика - https://webdevblog.ru
11. Webformyself все о создании сайтов - https://webformyself.com
12. Современный учебник JavaScript - https://learn.javascript.ru/
13. Статья на Хабр: Есть много способов сделать это: Vue 3 и взаимодействие компонентов - https://habr.com/ru/articles/668072/
14. Статья на Хабр: Ох уж эти модальные окна или почему я полюбил render-функции в VueJs - https://habr.com/ru/articles/350232/
15. Ютуб канал Archakov Blog - https://www.youtube.com/@ArchakovBlog
16. Ютуб канал itgid - https://www.youtube.com/@itgid
17. Ютуб канал Ulbi TV - https://www.youtube.com/@UlbiTV
18. Ютуб канал Владилен Минин - https://www.youtube.com/@VladilenMinin

Приложение

store/index.js

```
import { createStore } from 'vuex'
import axios from 'axios';

export default createStore({
  state: {
    posts: [],
    dialogVisible: false,
    isPostLoading: false,
    selectedSort: '',
    searchQuery: '',
    page: 1,
    limit: 10,
    totalPages: 0,
    sortOptions: [
        {value: 'title', name: 'По названию'},
        {value: 'body', name: 'По содержанию'},
    ]
  },
  getters: {
    sortedPosts(state) {
      return [...state.posts].sort((post1, post2) =>
           post1[state.selectedSort]?
          .localeCompare(post2[state.selectedSort]))
    },
    sortedAndSearhedPosts(state, getters) {
        return getters.sortedPosts.filter(post =>
           post.title.toLowerCase()
          .includes(state.searchQuery.toLowerCase()))
```

```
  }
},
mutations: {
  // для изменения состояния
  setPosts(state, posts) {
    state.posts = posts;
  },
  dialogVisible(state, bool) {
    state.dialogVisible = bool
  },
  setLoading(state, bool) {
    state.isPostLoading = bool
  },
  setPage(state, page) {
      state.page = page
  },
  setSelectedSort(state, selectedSort) {
      state.selectedSort = selectedSort
  },
  setTotalPages(state, totalPages) {
      state.totalPages = totalPages
  },
  setSearchQuery(state, searchQuery) {
      state.searchQuery = searchQuery
  },
  createPost(state ,post) {
    state.posts.unshift(post);
    state.dialogVisible = false;
},
},
```

```
  actions: {
   // https://ru.vuejs.org/v2/cookbook/using-axios-to-consume-apis.html
    async fetchPosts({state, commit}) {
      try {
        commit('setLoading', true);
        const response = await axios.get(
      'https://jsonplaceholder.typicode.com/posts', {
              params: {
                _page: state.page,
                _limit: state.limit
              }
        });
        commit('setTotalPages', Math.ceil(
             response.headers['x-total-count']
            / state.limit));
        commit('setPosts', response.data);
      } catch (error) {
         // alert('Ошибка');
         console.log(error);
      } finally {
         commit('setLoading', false);
      }
    },
    // подгрузка свежих постов
    async loadMorePosts({state, commit}) {
      try {
        commit('setPage', state.page += 1);
        const response = await axios.get(
'https://jsonplaceholder.typicode.com/posts', {
          params: {
```

```
            _page: state.page,
            _limit: state.limit
          }
        });
        commit('setTotalPages', Math.ceil(
            response.headers['x-total-count']
            / state.limit));
        commit('setPosts',
               [...state.posts, ...response.data]);
      } catch (error) {
          // alert('Ошибка');
          console.log(error);
      }
    },
  },
  modules: {
  }
})
```