Geekbrains

**Использование фреймворков в практике фронтенд-разработки**

**(на примере Vue.js)**

Frontend-программист:

Цифровые профессии

Оберемко В.Ю.

Ставрополь

2024

Оглавление

[**Введение** 3](#_Toc156765151)

[**Глава 1. Фреймворк как неотъемлемая часть процесса веб-разработки.** 4](#_Toc156765152)

[**1.1** **Что такое фреймворк.** 4](#_Toc156765153)

[**1.2** **Типы фреймворков в веб-разработке** 6](#_Toc156765154)

[**1.2.1** **Фронтенд-фреймворки** 6](#_Toc156765155)

[**1.2.2** **Бэкенд-фреймворки** 11](#_Toc156765157)

[**1.2.3** **Кроссплатформенные фреймворки** 13](#_Toc156765158)

[**1.3** **Архитектура фреймворков** 14](#_Toc156765159)

[**1.4** **Основные критерии выбора фреймворка**  18](#_Toc156765160)

[**Глава 2. Vue.js: преимущества, характеристика** 20](#_Toc156765161)

[**2.1** **Ключевые преимущества Vue.js в фронтенд-разработке** 20](#_Toc156765162)

[**2.2** **Vue 3. Принципиальные отличия.** 27](#_Toc156765163)

[**Глава 3. Разработка сайта интернет магазина мебели как пример внедрения Vue.js на практике.** 35](#_Toc156765212)

[**Заключение** 37](#_Toc156765213)

[**Список литературы** 54](#_Toc156765214)

[**Приложения** 55](#_Toc156765216)

# **Введение**

**Тема проекта:** Использование фреймворков в практике фронтенд-разработки (на примере Vue.js).

**Цель:** Изучить проблему использования фреймворков в практике фронтенд-разработки.

**Задачи:**

1. Дать определение фреймворка.
2. Разобрать классификацию фреймворков по области применения в разработке, дать им характеристику.
3. Рассмотреть Vue.js как пример фронтенд-фреймворка, разобрать его плюсы и минусы, возможные преимущества перед конкурентами.
4. Выполнить в графическом редакторе дизайн проект интернет-магазина.
5. Разработать проект интернет-магазина с использованием фреймворка Vue.js.

**Инструменты:** Figma, GitHub, Visual Studio Code, Node.js, Vue.js

# **Глава 1. Фреймворк как неотъемлемая часть процесса веб-разработки.**

* 1. **Что такое фреймворк.**

При создании простого проекта, например, одностраничного сайта, мы, конечно, можем написать код с нуля: сделать скелет будущей страницы на HTML, стилизовать с помощью CSS. Или, при необходимости создания отдельных элементов вроде блоков с иллюстрациями, текстами, можем использовать уже готовый конструктор – библиотеку. Но, если наша задача изваять нечто более серьезное, будь то объемный сайт или приложение, здесь не обойтись без фреймворка.[[1]](#footnote-1)

Фре́ймворк (англицизм, неологизм от framework «остов, каркас, рама, структура») – программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.[[2]](#footnote-2)

Таким образом, выбор фреймворка как каркаса нашего проекта обусловлен желанием воспользоваться некоторыми преимуществами, которые эти самые фреймворки нам могут обеспечить. А именно:

- увеличение скорости разработки

Возможность использования уже готовых конструкций и шаблонов позволяет пользователю (разработчику) сократить время на написание кода собственных системы связей, блоков, контента. Это особенно актуально для начинающих пользователей. Естественно, использование фреймворка ощутимо сокращает время от начала разработки до внедрения новых функций проекта.

- упрощение работы

Все фреймворки схожи, следовательно, имея опыт работы с одним фреймворком, понимая принципы его структурирования, разработчик свободно может освоить и другие.

- обеспечение безопасности.

По уровню безопасности проекты, созданные с помощью фреймворков, превосходят иные решения.

- скорость работы проекта.

Итоговый сайт работает быстрее по сравнению с сайтом, написанным по стандартной методике. Код, которым располагает фреймворк, максимально оптимизирован: отсутствуют лишние строки, бесполезные скрипты.

* 1. **Типы фреймворков в веб-разработке**

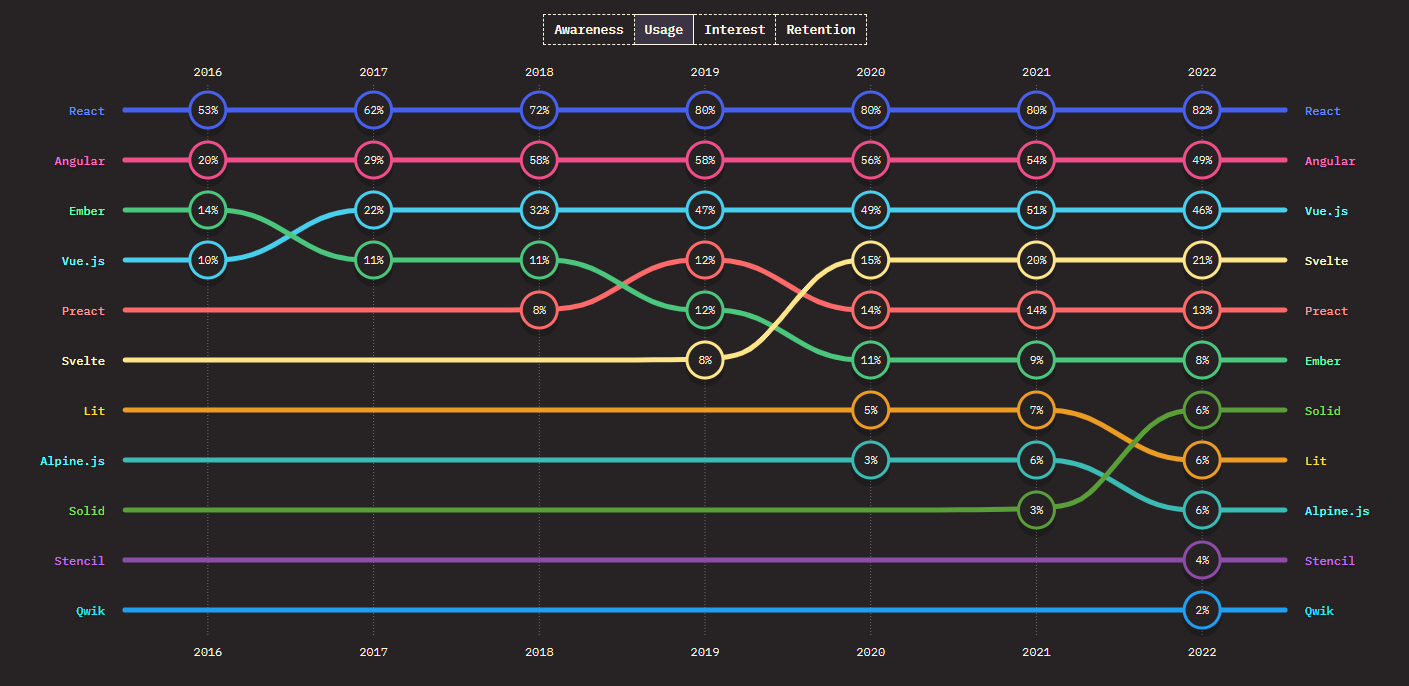
Существует несколько классификаций фреймворков. По области применения в разработке их обычно подразделяют на фронтенд (front-end), бэкенд (back-end) и кроссплатформенные или фуллстэк (fullstack). Рассмотрим их подробнее.

* + 1. **Фронтенд-фреймворки[[3]](#footnote-3)**

Фронтенд-фреймворк - это комплекс инструментов, позволяющий нам использовать готовые элементы и целые конструкции для разработки интерфейса приложений.

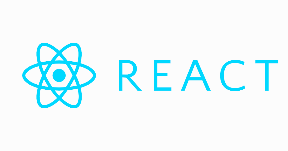
Интерфейсные фреймворки предоставляют вам готовые к использованию компоненты, таким образом нам не нужно создавать каждый элемент интерфейса приложения с нуля.

Приведенный ниже график демонстрирует относительную популярность фронтенд-фреймворков для веб-разработки.

****

Тремя лучшими интерфейсными фреймворками, исходя из процентного соотношения использования по данным, приведенным сайтом 2022.stateofjs.com являются React, Angular и Vue.js. Ещё в 2016-17 годах эти фреймворки завоевали популярность среди сообщества разработчиков и по-прежнему находятся на вершине рейтинга.

Помимо этих трех фреймворков, самыми используемыми в IT-сообществе являются Svelte, Preact и Ember. Однако, несмотря на то, что они все еще уверенно держатся в десятке, отрыв от лидеров настолько существенный, что для нас интереса они практически не представляют, и останавливаться подробно на них смысла нет. Рассмотрим ближе тройку лидеров.



**React** - один из самых простых фреймворков для изучения - был разработан Facebook (в настоящее время Meta - признана экстремистской и запрещена на территории РФ) для устранения проблем с сопровождаемостью кода из-за постоянного добавления функций в приложение. В настоящее время React выделяется как фреймворк с открытым исходным кодом благодаря своей объектной модели виртуального документа (DOM), которая предлагает исключительную функциональность. Часто его рекомендуют для проектов, которые включают создание одностраничных веб-приложений и PWA.

К плюсам можно отнести:

* возможность повторного использования отдельных компонентов в других частях проекта, что, в свою очередь, упрощает совместную работу над проектом;
* использование виртуального DOM;
* возможна интеграция с другими библиотеками;
* предусмотрена возможность разработки мобильных приложений.

Минусы:

* сложности с созданием и поддержанием в надлежащем виде документации, что существенно влияет на процесс обучения новичков из-за постоянных обновлений;
* сложная структура;
* только фронтенд.



**Angular** - единственный фреймворк в этом списке, основанный на языке TypeScript.

Тот Angular, который мы сейчас знаем (а он существенно отличается от своей первой версии, созданной в 2008 году) - Angular 2.0 был выпущен в 2016 году, в то время как предыдущая версия развивалась обособленно под названием AngularJS. В апреле 2022 поддержка AngularJS была официально прекращена. Этот фреймворк разработан Google для преодоления разрыва между растущими требованиями технологий и традиционными концепциями, что в итоге принесло свои плоды.

От отписанного выше React, Angular отличает функция двусторонней привязки данных. Это подразумевает синхронизацию в режиме реального времени модели (model) и представления (view), когда каждое изменение в модели тут же отражается на view, и наоборот.

Angular отлично подходит для создания мобильных или веб-приложений. Кроме того, также можно использовать этот фреймворк для разработки многостраничных веб-приложений. Проекты, созданные на Angular, внедряют такие компании, как Forbes, Xbox, BMW, Blender и др. Но, по сравнению с React, Angular заметно сложнее. Хотя документация доступна в достаточном объеме, она либо слишком сложна, либо сбивает с толку при чтении.

Плюсы:

* встроенная функциональность для обновления корректировок кода, внесенных в модель;
* сокращается объем кода, поскольку большинство важных функций, таких как двусторонняя привязка данных, предоставляются по умолчанию;
* отделяет компоненты от зависимостей, определяя их внешними элементами
* возможность использования компонентов в других частях проекта повторно
* уже сложившееся обширное сообщество единомышленников.

Минусы:

* сложен в освоении.

Иными словами, Angular повышает производительность браузерных приложений за счет быстрого динамического обновления содержимого, поскольку использует двустороннюю привязку данных. Прекрасно подходит для корпоративных приложений и динамических веб-приложений.  
Однако сложный синтаксис и непростой процесс обучения по сравнению с другими популярными фреймворками, существенно ограничивают сферу его применения.



**Vue.js** - один из самых популярных интерфейсных фреймворков на сегодняшний день. Vue - прост и интуитивно понятен. Он хорош для устранения сложностей, с которыми сталкиваются, например, разработчики Angular. Он меньше по размеру и также имеет двустороннюю привязку. Vue.js универсален и способен помочь в решении множества задач: создание веб-приложений и приложений для мобильных устройств, а также прогрессивных веб-приложений. Ему по плечу как простые, так и динамичные процессы. Хотя он и создан для решения серьезных задач и работы с массивными приложениями, Vie.js не пользуется широкой популярностью среди больших компаний. Возможно, это дело времени. Ведь пользователями этого фреймворка уже являются Xiaomi, Alibaba, 9gag, Reuters. И число пользователей этого фреймворка продолжает расти.

Что неудивительно, ведь к его плюсам можно отнести:

* Обширную и подробную документацию;
* Простой синтаксис – программисты с опытом работы на javascript могут легко начать работу с Vue.js;
* гибкость в разработке структуры приложения;
* поддержку языка Typescript;

Но есть и минусы:

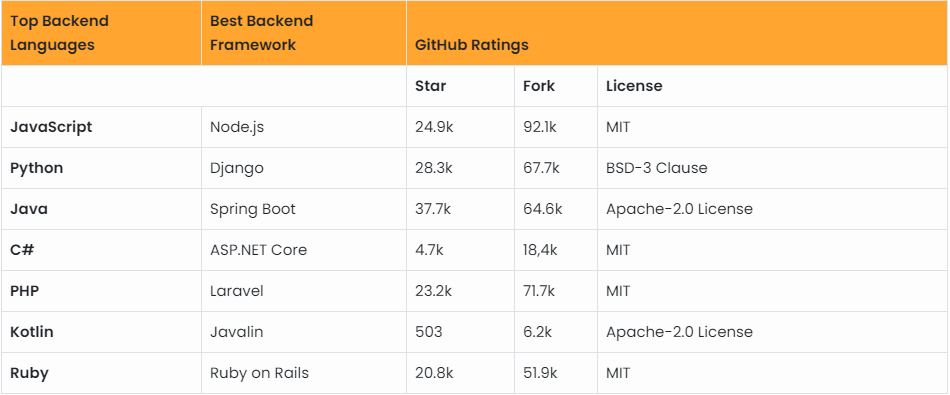
* отсутствие стабильности в компонентах;
* относительно небольшое сообщество поддержки и обучения, в отличие от того же Angular4;
* языковой барьер (большинство плагинов, как ни странно, написано на китайском языке).

Итак, Vue.js рекомендуется для гибких структур дизайна. Он позволяет создавать все с нуля, а также успешно работает при разработке гигантских проектов. Однако есть некоторые сложности со стабильностью компонентов.

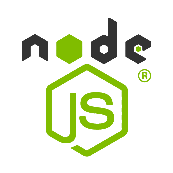
* + 1. **Бэкенд-фреймворки[[4]](#footnote-4)**

Бэкенд – это внутренняя часть приложения, набор различных моделей соответствующих инструментов, помогающих нам как разработчикам настроить основу для любого программного решения, будь то приложение или сайт. Эта сфера отвечает за хранение и структурирование данных, выполнение вызовов API и обеспечение бесперебойной работы на стороне клиента, то есть во внешнем интерфейсе.

Самые востребованные технологии серверной разработки в разрезе используемых языков программирования в 2022 году (Stack Overflow 2022):

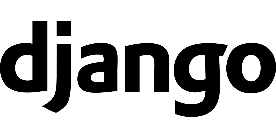


Поскольку целью данной работы не является подробная характеристика фреймворков бэкенд разработки, ограничимся перечислением самых популярных из них и приведем примеры проектов, выпущенных на их основе.



1. Node.js

Продукты: Uber, Twitter, Slack, Netflix.



1. Django

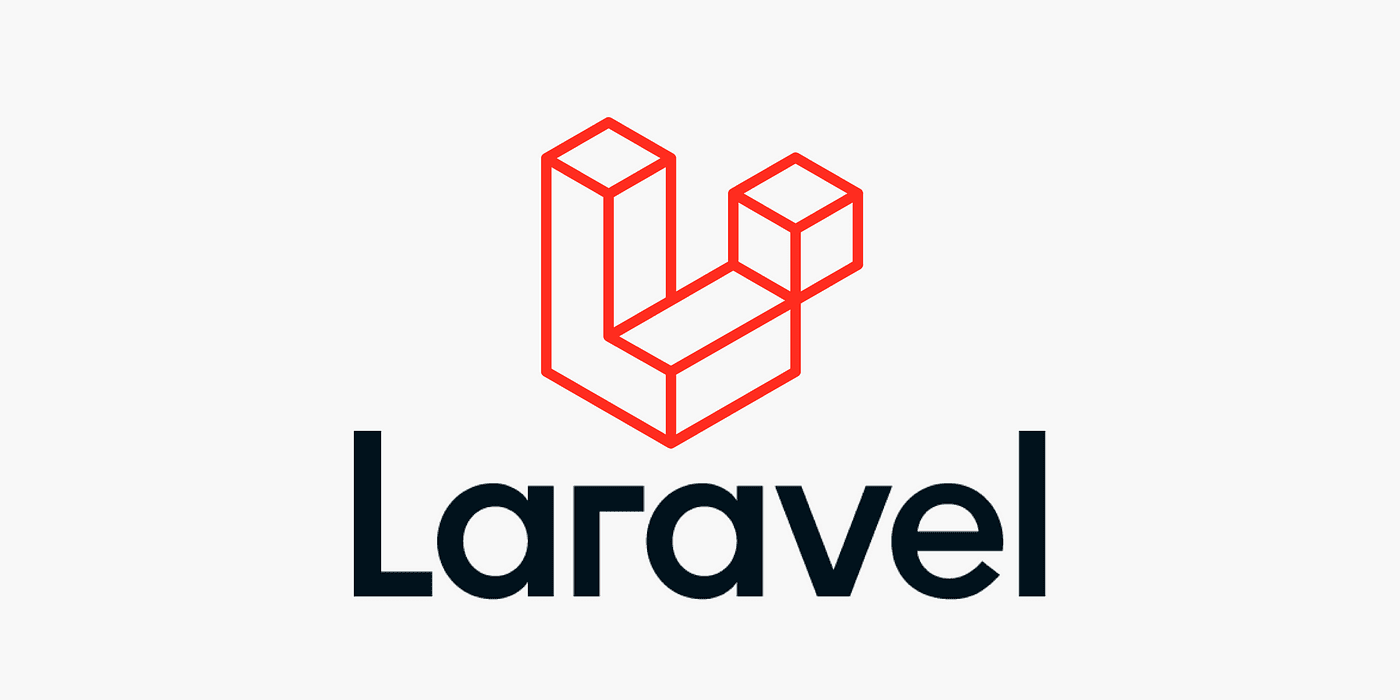
Продукты: Pinterest, Instagram, Trivago.

1. Spring

Продукты: Indeed, Intuit, Zalando.

1. ASP.NET Core

Продукты: Alibaba Traveles, Fonibo.



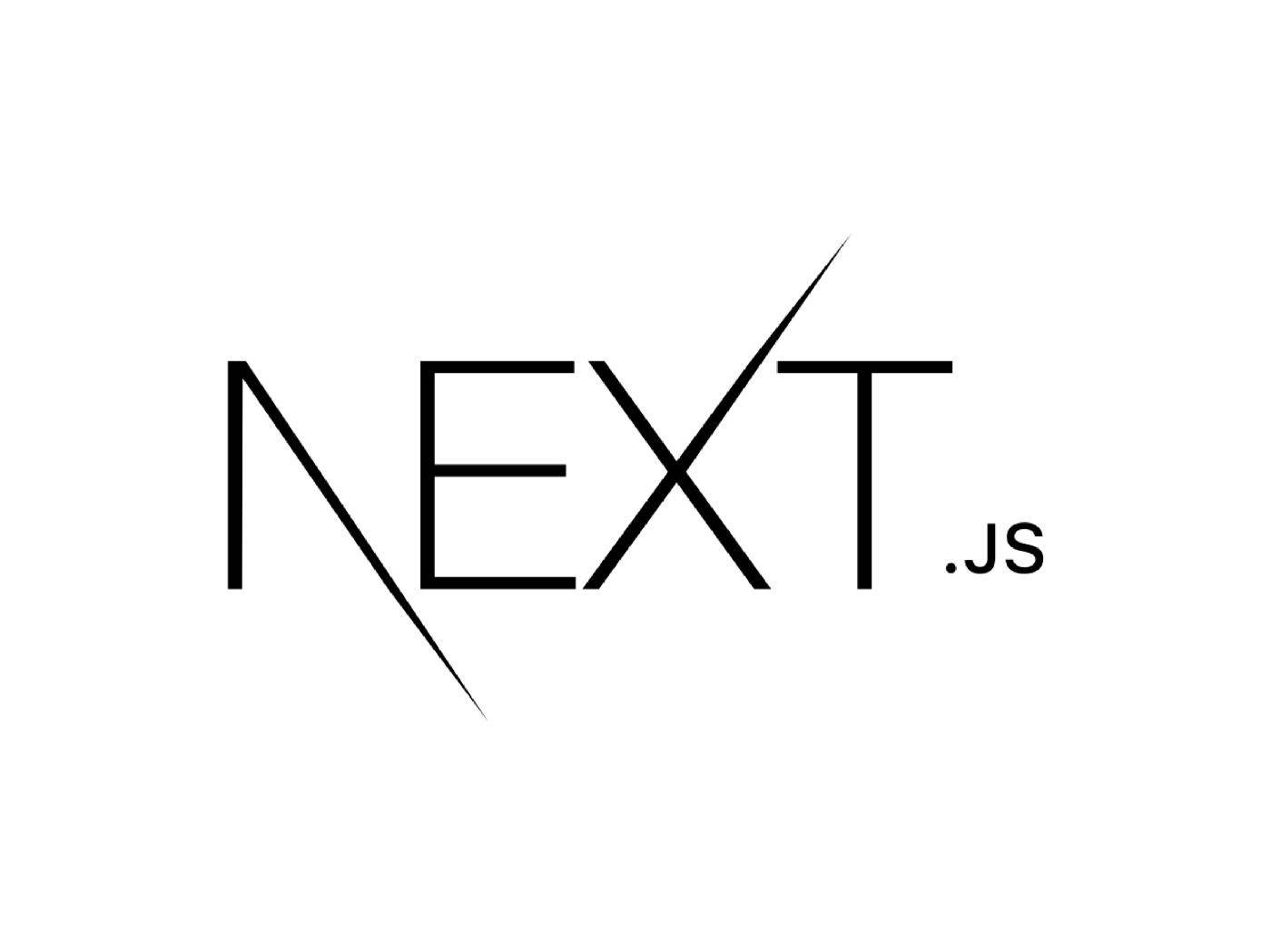
1. Laravel

Продукты: 9GAG, Mastercard.

* + 1. **Кроссплатформенные фреймворки**

Если фреймворк успешно справляется с решением задач как на серверной, так и на клиентской стороне, его можно отнести к категории кроссплатформенных. Примером может послужить фреймворк Meteor, созданный в 2011 году тогда еще под названием Skybreak. Обе его стороны – клиентская и серверная – используют один код - JavaScript. Кроме того, между клиентом и сервером, в основном, передаются данные, а не код разметки PHP/HTML. Meteor – фреймворк, предназначенный для разработки так называемых веб-приложений "реального времени". Это означает, что все изменения в одном интерфейсе влекут за собой изменения и в других.

К кроссплатформенным фреймворкам также можно отнести Nuxt, базой которого является Vue.js., и Next.js, созданный на основе React.js.



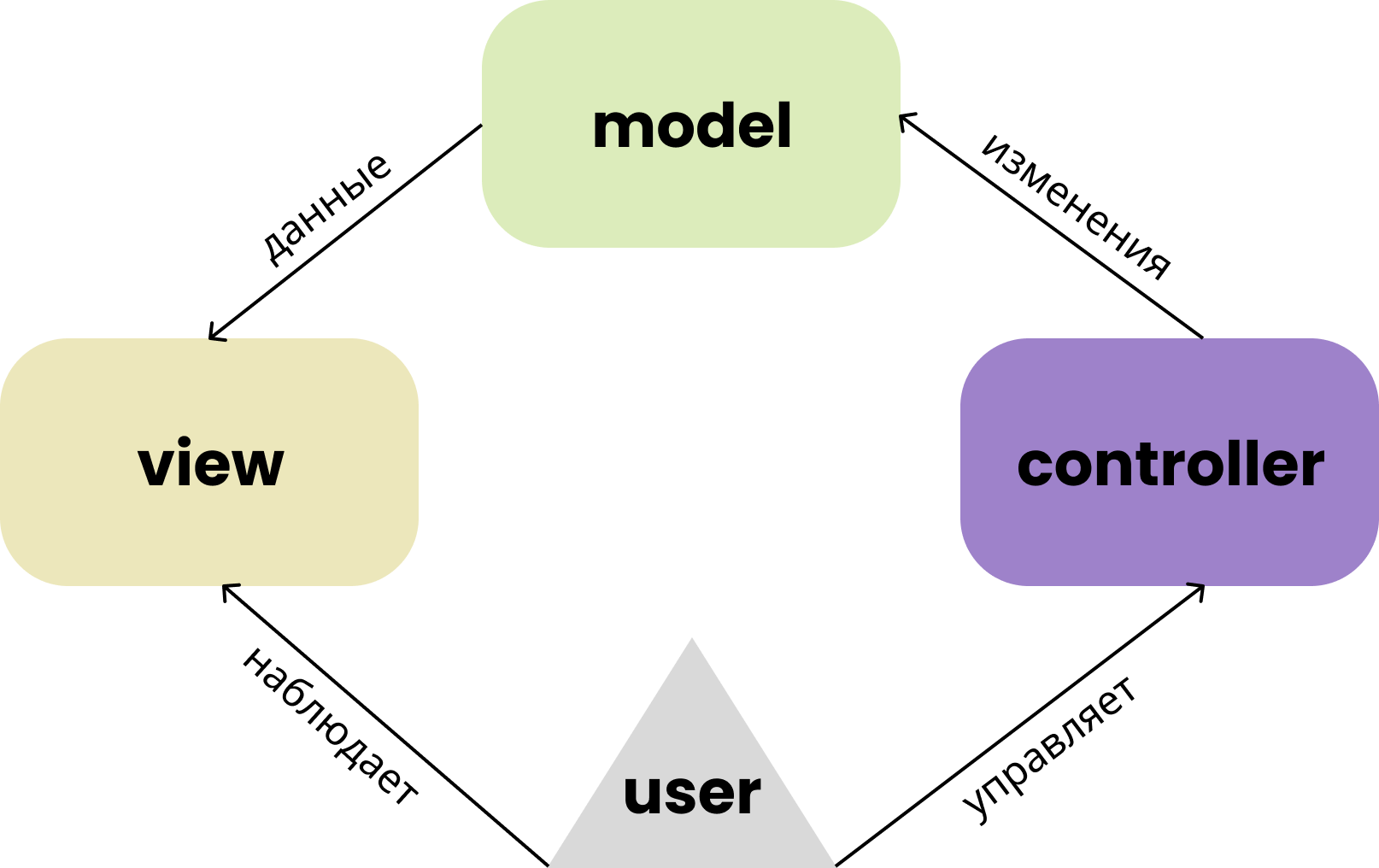
В дополнении к приведенным выше данным в приложении 1 представлена статистика наиболее широко используемых фреймворков среди разработчиков всего мира по состоянию на 2023 год.

* 1. **Архитектура фреймворков[[5]](#footnote-5)**

Абсолютное большинство фреймворков разработано на основе шаблона проектирования, получившего название MVC или Model-View-Controller. Концепция MVC была предложена Трюгве Реенскаугом (Trygve Reenskaug) ещё в 1978 году. Позже другой разработчик Стив Бурбек (Steve Burbeck) реализовал шаблон в объектно-ориентированной системе Smalltalk-80. Окончательный вариант концепции MVC был опубликован в журнале Technology Object только в 1988 году. Впоследствии шаблон MVC естественным образом развивался и эволюционировал, но основная мысль оставалась неизменной:

Model-View-Controller помогает разделить интерфейс и внутренний код на отдельные компоненты. Таким образом, намного проще управлять обеими сторонами и вносить изменения, не мешая их функционированию. Иными словами, шаблон MVC описывает очевидный нам сейчас способ построения каркаса проекта, цель которого - отделение логики бэка от пользовательского интерфейса. В конечном итоге проект легче тестируется, масштабируется, технически сопровождается и используется в дальнейшем.

На схеме ниже приведены компоненты MVC



**Модель** – компонент MVC, включающий в себя бизнес-логику всего приложения и содержащий методы обработки, а также способы предоставления определенных составляющих проекта. Как правило, это самая массивная часть приложения.

Согласно логике MVC, модель не взаимодействует непосредственно с пользователем. Все данные, запрашиваемые пользователем, обрабатываются в иной части - контроллере.

Модель также не отвечает за генерацию какого-либо кода отображения, изменяющегося в зависимости от запросов пользователя. Этот код обрабатывается во view.

**View** – компонент MVC, использующийся для определения внешнего отображения данных, которые получены из других компонентов системы, таких как модель и контроллер. View включает в себя разметку кода HTML и допускает вставки PHP - кода.

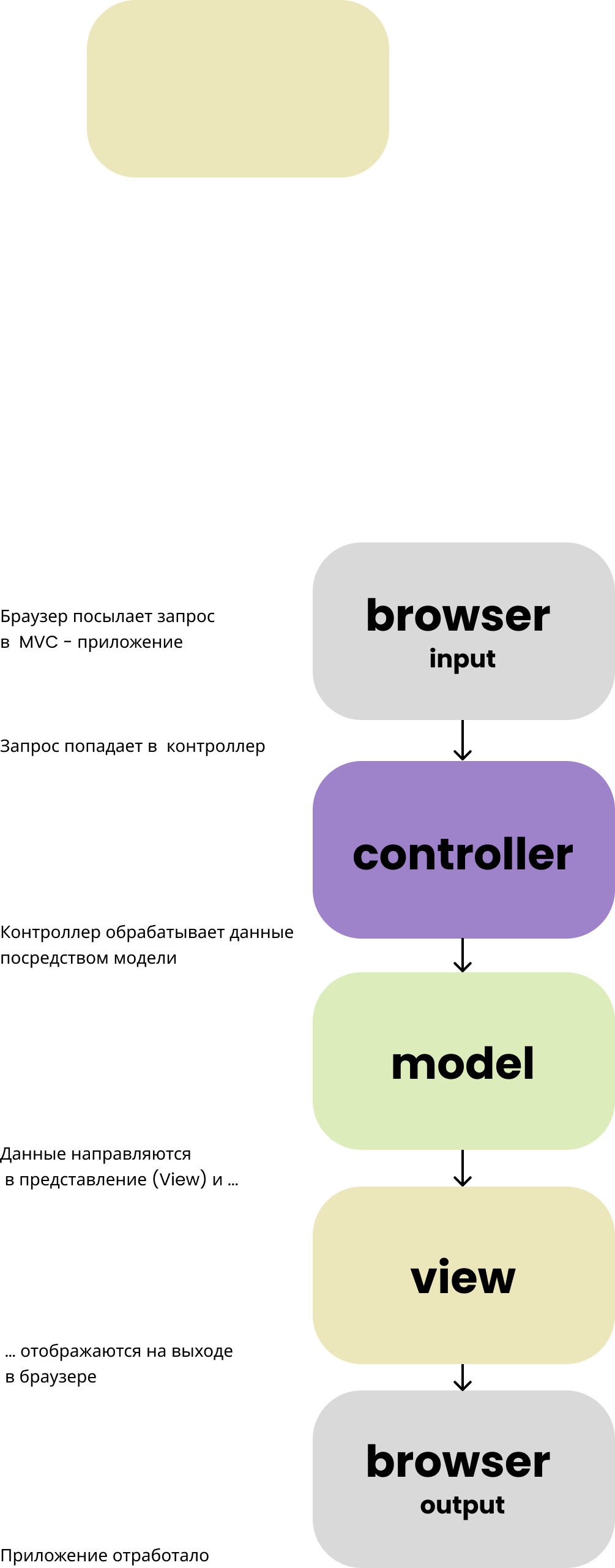
Этот компонент MVC не обращаются напрямую к каким-либо базам данных. Это прерогатива модели.

Также исключается его работа непосредственно с запросами пользователей. Это обязанность контроллера. Что он может, так это обращаться к модели или контроллеру для получения данных, подготовленных ими к выводу. Согласно логике шаблона MVC, View, как правило, разделяется на компоненты.

**Контроллер** – это как раз тот самый компонент MVC, который и связывает между собой модель, view и другие компоненты в нечто работающее согласно логике MVC. Именно он ответственен за обработку запросов пользователя. Контроллер не содержит запросов в базы данных, не включает в себя HTML-код или иную разметку.

Зона его ответственности – коммуникация, следовательно, в отличие от других частей шаблона MVC, контроллеры обычно очень тонкие и могут содержать лишь несколько десятков строк кода.

Схема ниже иллюстрирует выполнение MVC-приложением определенного запроса пользователя.



* 1. **Основные критерии выбора фреймворка [[6]](#footnote-6)**

Проанализировав информацию, изложенную выше, мы можем сформулировать несколько принципов выбора необходимого нам в данный момент фреймворка.

* Популярность фреймворка и размер поддерживающего его IT-сообщества

Чем больше разработчиков обратили в свое время внимание на данный продукт, чем больше его «комьюнити», тем дольше он будет «жить», развиваться, обрастая новыми идеями, модулями и т.д.

* Суть продукта (философия)

Тот самый критерий, который может помочь понять, это ли нам нужно. Фреймворки подразделяются не только по сфере разработки, но и, например, по уровню подготовки пользователя. Один предоставляет возможность для быстрого освоения, другой сложен, но при этом предлагает широчайший спектр возможностей для реализации фишек проекта. Часто это называют философией продукта.

* Стабильность  
   При выборе фреймворка стоит уяснить, будут ли создатели поддерживать его в течение достаточно продолжительного времени. Это упростит модернизацию проекта м его техническое сопровождение.
* Помощь сообщества

Насколько легко будет получить помощь при разработке нашего проекта на основе конкретного фреймворка? Какие источники информации будут нам доступны? Сюда же можно отнести возможные методы повышения квалификации, различные рассылки, профессиональные форумы и т.д.

* Технические приёмы

Насколько современен продукт. Какие практики в разработке он использует. Какие шаблоны применяются системой во взаимодействию с иными системами, [внешними интерфейсами (API)](https://iwsm.ru/?external_url=https%3A%2F%2Fru.wikipedia.org%2Fwiki%2FApi), при [вызове удалённых процедур (RPC)](https://iwsm.ru/?external_url=https%3A%2F%2Fru.wikipedia.org%2Fwiki%2F%D0%A3%D0%B4%D0%B0%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B2%D1%8B%D0%B7%D0%BE%D0%B2_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80).

* Требования безопасности

Любое наше приложение потенциально уязвимо. Для того чтобы снизить риски, всегда лучше выбрать наиболее безопасный фреймворк.

* Документация

Часто можно столкнуться с недостаточным информационным сопровождением продукта. И тем важнее оценить степень его документальной обеспеченности. От этого зависит, насколько доступно будет его использование начинающими разработчиками, а также скорость внедрения новых функций и исправления старых багов.

* Рынок труда

Этот критерий напрямую связан с популярностью продукта. Ведь нам необходимо сформировать команду специалистов, владеющих данным фреймворком для разработки, сопровождения и модернизация нашего приложения.

* Личные предпочтения

В конце концов, очень много зависит от того, насколько нам самим комфортно работать с этим фреймворком. Продукт может лидировать по абсолютному большинству критериев, но совершенно не подходить нам по каким-то личным соображениям.

# **Глава 2. Vue.js: преимущества, характеристика**

* 1. **Ключевые преимущества Vue.js в фронтенд-разработке**

Итак, мы рассмотрели преимущества и недостатки основных игроков на поле фронтенд и бэкенд разработки. Теперь остановимся на одном из самых молодых и, мы уверены, многообещающих фреймворков – Vew.js. Новые фронтенд-фреймворки появляются каждый день, и каждая уважающая себя компания IT-разработки считает делом чести создать что-то свое, что бы полностью удовлетворяла её потребности в разработке. Vue.js также вырос из программы, изначально задумывавшейся как инструмент для внутреннего использования. И он остается одним из самых популярных фреймворков в настоящее время. Чем же объясняется его востребованность?

1. Простота в создании интерфейсов.

Первое и самое существенное преимущество Vue.js - легкость изучения и использования. Ему присущ простой и понятный синтаксис, который легко освоить разработчикам любого уровня опыта. Благодаря этой простоте, Vue.js позволяет быстро приступить к процессу разработки, исключая время на изучение документации и сложных концепций. А декларативный подход к разработке делает создание интерфейсов интуитивным понятным, а значит и эффективным процессом.

1. Гибкая система компонентов.

Компонентный подход – основа Vue.js. Компоненты – относительно небольшие куски кода, на которые фреймворк позволяет разбивать интерфейсы, что значительно упрощает и ускоряет процесс разработки, делает рабочий код более понятным. Кроме того, это существенно упрощает и дальнейшую техническую поддержку нашего кода.

Компоненты совершенно независимы друг от друга и могут быть снова использованы в различных частях нашего приложения, а это, кроме всего прочего, позволяет проекту соответствовать принципам масштабируемости.

Масштабируемость — это свойство системы (приложения) приспосабливаться к растущим объемам работы за счет добавления ресурсов в систему.

Естественно, каждый компонент может быть отлажен вне независимости от остальной части проекта. Благодаря такой структуре, разработчику необходимо лишь взаимодействовать непосредственно с каждым компонентов в отдельности, тогда как Vue.js самостоятельно настраивает связи между ними, поддерживает обновление данных и отображение изменений на страницах.

Кроме того, Vue.js поддерживает создание вложенных структур, то есть мы можем создавать компоненты, содержащие другие компоненты внутри себя, что позволяет существовать в рамках нашего проекта сложным иерархиям, где каждый элемент будет отвечать за конкретный функционал.

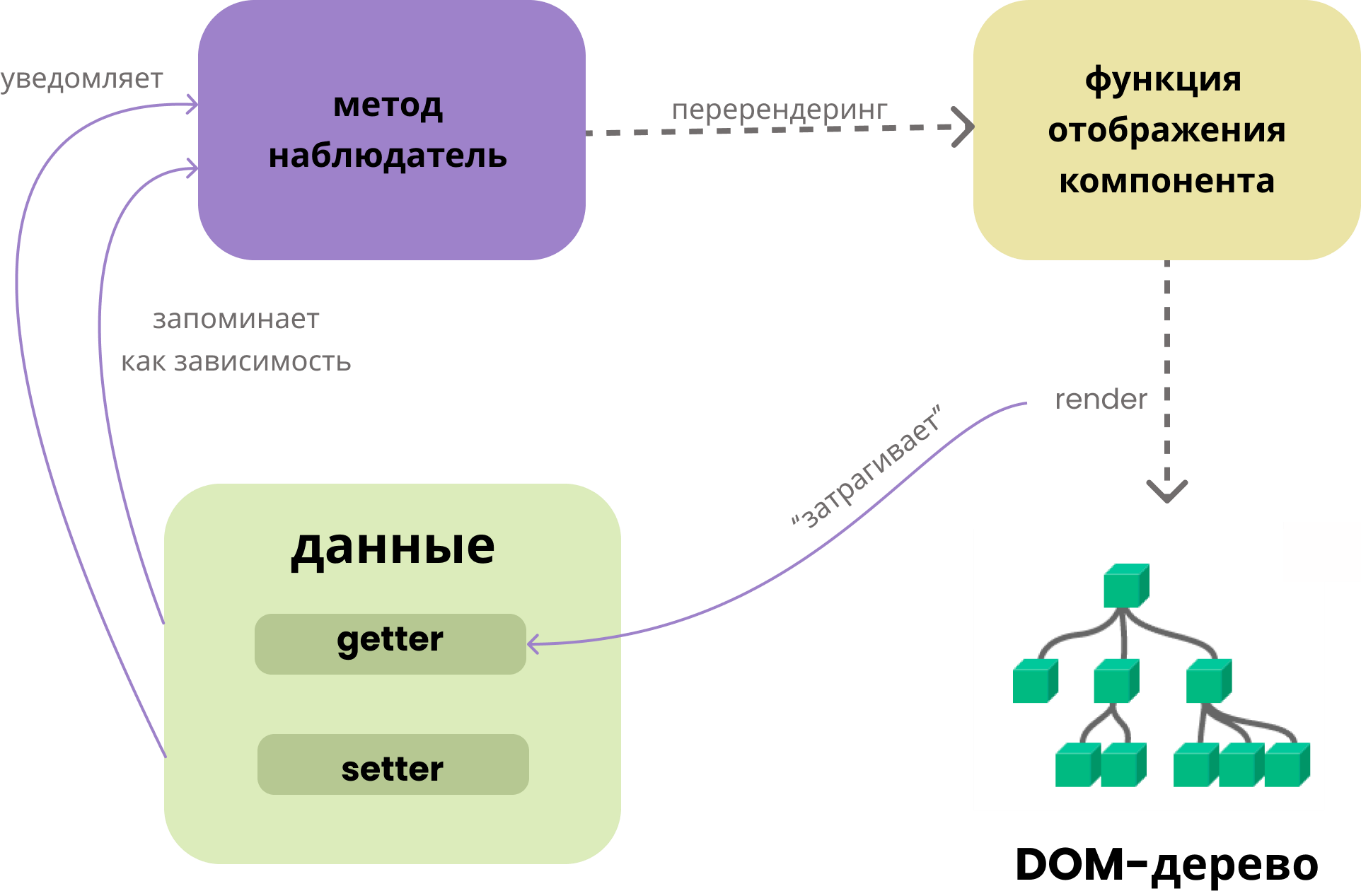
1. Реактивность (отзывчивость) и автоматическое обновление данных.

Во Vue.js компонентный подход сочетается с мощным механизмом реактивности, автоматическим обновляем отображения данных в приложении при их корректировке.

Реактивность — это способность приложения в реальном времени отражать изменения состояния. Реактивные приложения также называют приложениями «реального времени». Рассмотрим, каким образом Vue.js реализует этот принцип.

Модели во Vue.js - простые объекты на языке JavaScript. При их преобразовании обновляется и view. В момент передачи объекта в экземпляр Vue (data), происходит проверка всех полей объекта и преобразование их в пары getter/setter. При этом используется Object.defineProperty. К сожалению, данная опция реализована лишь начиная с ES5, так что Vue не поддерживает браузер версии IE8 и ниже. Доступом к геттерам и сеттерам располагают только разработчики. Именно этот метод и позволяет фреймворку отслеживать зависимости и изменения данных, обеспечивая приложению принцип реактивности.

Проиллюстрируем это на схеме:



1. Производительность.

Примеры инструментов, которые может предложить Vue.js для повышения производительности нашего приложения:

* ленивая загрузка - деление приложения на модули и загрузка их по требованию (директивы v-if и v-show), что позволяет повысить отзывчивость загружаемой страницы. При использовании Vue CLI 3 «Ленивые» ресурсы загружаются по умолчанию.
* предварительная загрузка - Vue предоставляет возможность загружать какие-либо ресурсы заранее, до запроса их браузером.
* кеширование компонентов – тяжелые компоненты кэшируются, что позволяет существенно ускорить отрисовку страницы при повторном переходе.
* виртуализация - позволяет снизить число отображаемых элементов DOM-дерева.
* асинхронная загрузка данных – ускоряет запуск приложения и всю его работу в целом за счет использования параллельной загрузки данных.

1. Использование директив

Директивы – это атрибуты, используемые внутри шаблона vue для взаимодействия с какими-либо элементами DOM. Иными словами, это конкретные действия с элементами кода.

Вот некоторые из директив, которые Vue.js предлагает нам использовать:

|  |  |
| --- | --- |
| v-show | Показывает/скрывает код в зависимости от того, что мы передаем в качестве аргумента (true/false) |
| v-if | Те же true/false, но в случае применения v-if элемент появляется/пропадает из DOM дерева в зависимости от переданного аргумента. Когда же мы используем v-show элемент никуда из DOM-дерева не исчезает. Его положения регулируется через свойство "display". |
| v-else | Используется в комплексе с предыдущей директивой. |
| v-on | Отвечает за добавление события (например, click) на элемент. В качестве аргумента передается название события (как в JavaScript). |
| v-for | Используется для отрисовки массивов. Старые добрые циклы. |
| v-html | Помещает в структуру указанного элемента данные в виде кода HTML. |
| v-bind | Своё существование оправдывает возможностью осуществления привязки выражения к атрибутам или параметрам элемента. |
| v-model | Обнаруживает некоторое сходство с директивой v-bind, но в данном случае привязка двусторонняя. |

1. Использование функционала Vue CLI

Vue нельзя представить без командной строки. Её функционирование и обеспечивает Vue CLI (command line interface - интерфейс командной строки). Он состоит из трех инструментов:

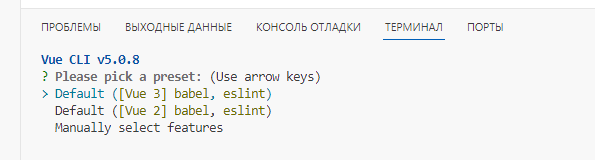
CLI – позволяет быстро создать новый проект или же прототип исходной модели, визуализировать проект при помощи графической (GUI) версии.

CLI Service —обеспечивает не только разработку проекта, но также проверку его конфигурации или возможность компоновки для продакшн.

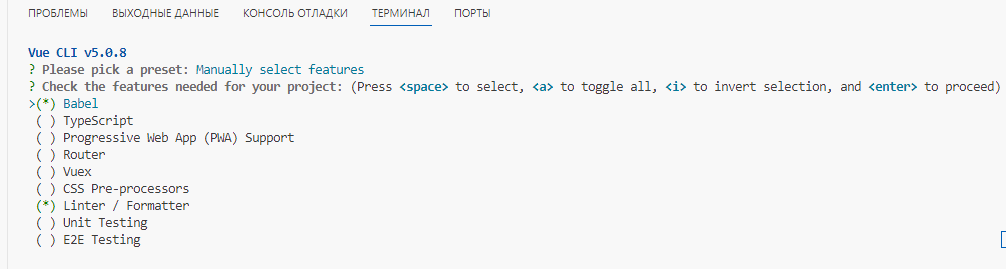
CLI Plugins —предоставляет целый спектр дополнительных функций для создаваемого приложения.

Для создания проекта вводим в терминале команду:

vue create {название проекта},

и Vue CLI предлагает варианты конфигураций для дальнейшей сборки проекта. В терминале выглядит это следующим образом:

Можно воспользоваться предложенными вариантами или настроить проект вручную, отключив ненужные в данном проекте плагины:



Таким образом, Vue CLI предоставляет нам возможность автоматического создания компонентов, сервисов и иных элементов проекта. Использование данного инструментария позволяет существенно уменьшить время команды на разработку проекта. Автоматизация развертывания проекта и создания необходимых элементов позволяет без потери времени и функциональности приступить к написанию пользовательского интерфейса.

1. Возможности использования сторонних библиотек.

Модульная структура построения приложения, созданного на каркасе Vue.js, её покомпонентное разбиение делает возможным интеграцию с многими популярными библиотеками на рынке IT-разработки.

Далее, в третьей главе мы будем использовать такие библиотеки как Vuex и Vue Router, что даст нам возможность на конкретном примере показать, насколько легко Vue координирует взаимодействие составных модулей проекта с инструментами данных библиотек.

1. Большое сообщество и поддержка со стороны разработчиков.

Vue.js – относительно молодой фреймворк. Но уже сейчас сформировалось активное сообщество, распробовавшее все его преимущества и готовое поделиться своим опытом с начинающими разработчиками. Их усилиями создаются новые библиотеки и плагины для написания нового кода и совершенствования функциональности уже существующего. Это помогает не только в поиске решений конкретных технических вопросов, но и способствуют росту фронтенд разработчика в целом.

* 1. **Vue 3. Принципиальные отличия.**

Не будем слишком уж углубляться в историю вопроса. Лишь отметим, что любой более или менее успешный продукт IT-индустрии обязан идти по пути развития и совершенствования. Иначе неприятная перспектива вытеснения с рынка информационных технологий более успешными конкурентами станет печальной реальностью.

Жить можно только в движении, что, как показала практика, очевидно и для разработчиков Vue.

Концепция Vue 3 возникла ещё в 2018 году. Согласно релизу от сентября 2020 года, «версия 3.0 - это более двух лет усилий по разработке, включающих [30+ RFC](https://github.com/vuejs/rfcs/tree/master/active-rfcs), 2600+ коммитов, [628 пулл-реквестов](https://github.com/vuejs/vue-next/pulls?q=is%3Apr+is%3Amerged+-author%3Aapp%2Fdependabot-preview+) от [99 участников](https://github.com/vuejs/vue-next/graphs/contributors), а также огромный объем работы по разработке и написанию документации за пределами основного репозитория». [[7]](#footnote-7)

Так или иначе, мы полагаем, что усилия всей этой массы людей увенчались успехом. Ведь Vue 3 показал существенный рост показателей производительности в сравнении с Vue предыдущей версии: размер пакета уменьшился в среднем на 41% (система сборки с поддержкой tree-shaking), скорость первичного рендеринга выросла более чем в полтора раза, а обновление отображения информации в браузере стало до 133% быстрее. Кроме того, Vue стал рациональнее использовать память: экономия до 54%.

С помощью каких инструментов были достигнуты столь существенные изменения?

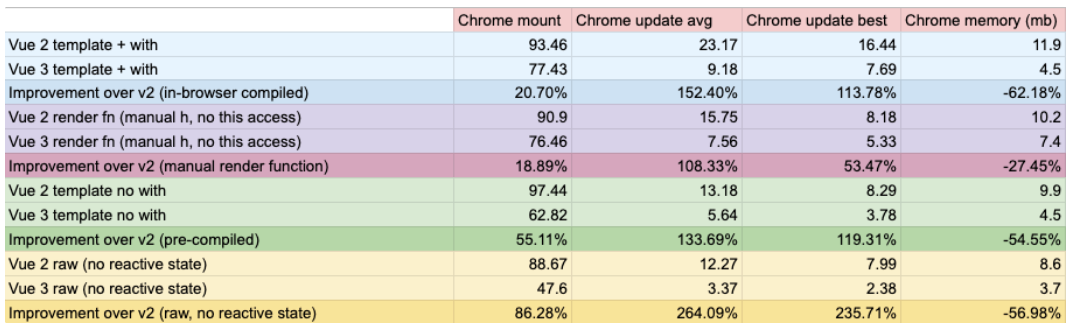
Это и новый API и масштабные корректировки исходного кода, фактически переписанного с нуля. Это и Vetur — подключаемый модуль, обеспечивающий поддержку IntelliSense, а также отладку кода Vue.js. Кроме того, он поддерживает проверку типов и свойств шаблона с использованием улучшенной внутренней типизации новой версии Vue. Система реактивности также претерпела изменения.

Все это несомненно сделало новую вервию Vue.js интересной не только непосредственным пользователям, получившим кроме полезных нововведений в части функционала ещё и улучшенную производительность, но и владельцам бизнеса, которые всё чаще выбирают данный фреймворк для разработки своих продуктов.

Теперь подробнее рассмотрим преимущества Vue 3.0:

1. Улучшение производительности.

Приведенная в релизе от сентября 2020 года таблица наглядно демонстрирует преимущества третьей версии Vue над предыдущей.



1. Поддержка

Судьба большинства устаревших версий любого продукта – прекращение поддержки. Выбор для разработки Vue 3 как более современной версии предполагает осуществление процедур поддержки на более длительный срок.

Однако на сегодняшний день в силу популярности Vue существует огромное количество проектов, реализованных на второй версии, и нет никаких предпосылок к прекращению их информационного обеспечения.

1. Некоторые функциональные преимущества Vue 3:

* Алгоритмы работы с DOM-деревом

Алгоритм работы с DOM-деревом в Vue построен на сопоставлении двух объектов Vnode (oldNode и vNode) с последующим внедрением корректировки в браузере. Во Vue 2 этот процесс осуществлялся в сопоставлении узлов вне фокусировки на их типах. Во Vue3 осуществляется сопоставление Vnode элементов только с привязкой динамического типа, а статические элементы успешно игнорируются.

* Новая концепция реактивности

Как мы помним, реактивность — это способность приложения в реальном времени отражать изменения состояния. Во Vue 2 данный механизм осуществляется на основе системы «getter/setter» переопределением свойств объекта (Статический метод Object.defineProperty()).

Во Vue 3 же принцип реактивности основывается на применении такого явления как Proxy. Прокси — это заменитель реального объекта, у которого есть право выступать от имени и в интересах этого объекта. С помощью Proxy во Vue3 решаются основные проблемы, свойственные системе реактивности в предыдущей версии.

* Инициализация

Vue 3 возвращает новый экземпляр приложения Vue каждый раз, когда для инициализации приложения используется «createApp». Мы можем настроить каждый экземпляр как независимый объект, не затрагивая другие экземпляры. Это удобно в больших проектах или при необходимости разделения функциональных возможностей между экземплярами.

* Фрагменты

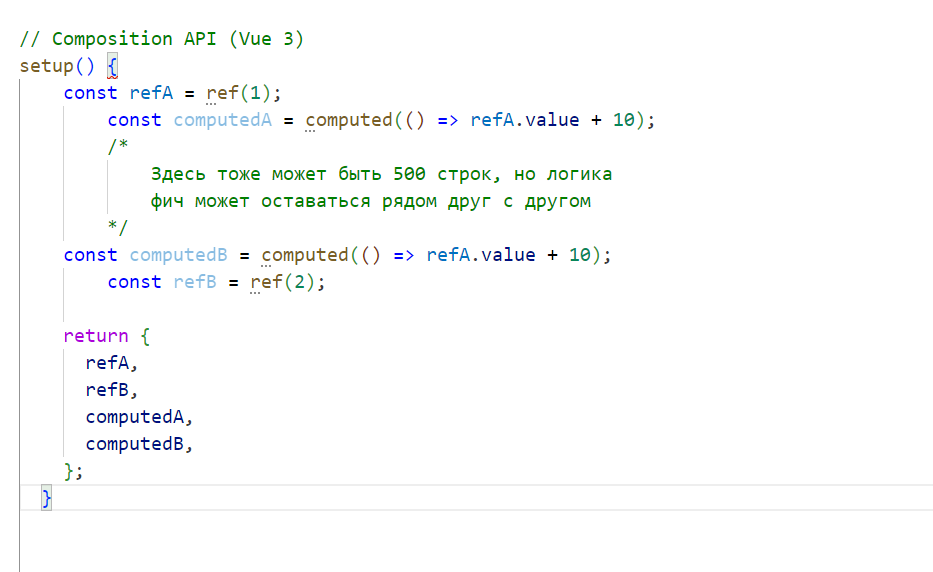
Поскольку компоненты в Vue 3 в отличие от предыдущей версии могут иметь более одного узла (из-за введения фрагментов), процесс создания компонентов в Vue 2 и Vue 3 реализован по-разному. Доступ к данным при создании компонента в Vue 3 осуществляется с помощью опции «reactive», в то время как в Vue 2 ничего подобного не существует.

* Composition API

Одним из самых больших изменений, внесенных в Vue 3, стал Composition API. Во Vue 2 логика построения кода разделялась по опциям: data, methods, computed, created и т.д.

Например:

Composition API это совершенно новое решение организации компонентов в Vue 3. Это решение позволяет группировать код по функциям, а не по типу свойства. [[8]](#footnote-8)

Пример:

* Хуки жизненного цикла в setup

Во Vue 2 хуки жизненного цикла доступны непосредственно из компонента. Но во Vue 3 почти все (включая хуки жизненного цикла) находится внутри метода setup ().

* Computed

Использование вычисляемых свойств в Vue 2 выполняется путем добавления поля «computed» к объекту options. Однако при использовании Vue 3 разработчикам необходимо импортировать «computed» в компонент, прежде чем использовать его внутри метода setup ().

Это изменение было реализовано для предотвращения импорта ненужных пакетов, которые никогда не использовались, как это было принято в Vue 2.

* Доступ к props

Во Vue 2 ключевое слово «this» всегда использовалось для обозначения компонента, а не какого-либо конкретного свойства. И можно было легко получить доступ к «props». Но в Vue 3 «this» больше не обеспечивает доступа к «props». Вместо этого аргументы «props» и «context» нужно передать в метод setup ().

* Обработка событий (events)

Обработка событий в третьей версии также отличается. Во Vue 2 можно просто использовать «this.$emit» для отправки события родительскому компоненту. Но Vue 3 предоставляет больше контроля над тем, как осуществляется доступ к свойствам или методам.

Во Vue 3 это реализовано путем добавления объекта «context» в качестве второго параметра к методу setup (). Теперь мы можем просто вызвать «emit» для отправки события родительскому компоненту.

* Порталы

Функция портала позволяет нам перемещать HTML-шаблон из одного компонента в другой компонент в другом DOM дереве. Порталы, являющиеся общей функцией React, во Vue 2 доступны через использование стороннего плагина под названием portal-vue. [[9]](#footnote-9)

Vue 3 же для достижения того же результата использует встроенный портал. Уникальный тег <teleport> используется для вложения любого кода, который подготавливается к телепортации в другое место. Эта функция очень помогает при создании, например, модальных окон.

1. Поддержка TypeScript

Во второй версии для использования в написании кода языка TypeScript нужно было задействовать декораторы и классовые компоненты. Это естественным образом не делало процесс разработки быстрее.

Vue 3 же целиком реализован на TypeScript, все официальные пакеты Vue 3 поставляются уже со встроенными объявлениями типов, то есть по сути работают без предварительных танцев с бубном.

1. Порог внедрения

Vue с момента создания придерживался консервативного подхода к реализации шаблонов и стилей, которые создаются отдельно от логики построения самого приложения. Кроме того, Vue 3, как и ранее, использует язык разметки HTML и CSS. Разработчики, особенно только начинающие осваивать фреймворки, могут это только приветствовать, так как это куда более привычно и существенно облегчает постепенную интеграцию в работу существующих приложений.

Следовательно, разработчику требуется существенно меньше опыта кодинга для внесения доработок, а также последующего обслуживания проекта на Vue.

Итак, в качестве краткого вывода мы можем отметить, что Vue 3 был анонсирован как альтернатива Vue 2, его компактная, более быстрая и более обслуживаемая версия. Vue 3 не только привносит в фреймворк новые функции, но и устраняет некоторые недостатки, свойственные Vue 2.

Во Vue 2 с увеличением размера код усложняется и становится менее читаемым. Vue 3 пытается решить эту и другие проблемы Vue 2, не теряя при этом схожести по синтаксису с предшественником, и в то же время сохраняя преимущества перед конкурентами.

# **Глава 3. Разработка сайта интернет магазина мебели как пример внедрения Vue.js на практике.**

В данной главе мы покажем фреймворк Vue.js в работе. Используем вторую версию фреймворка. Несмотря на все преимущества Vue 3, описанные нами ранее, предыдущая версия более дружелюбна к новичкам и требует минимальных изменений в построении проекта.

В качестве примера разберем создание небольшого интернет-магазина мебели. Начнем с нуля.

В основе нашего проекта - адаптированный дизайн-макет Maddy Beard[[10]](#footnote-10), созданный в графическом редакторе Figma[[11]](#footnote-11).

Итак, макет нарисован:

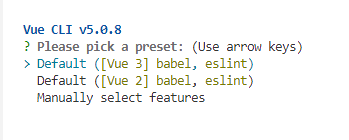
https://github.com/viktoriya2022/Graduation\_Project.git.

Следующий шаг – настройка среды разработки.

Первым делом - устанавливаем Vue CLI. Согласно документации для установки Vue CLI 4х рекомендуется [Node.js](https://nodejs.org/)  v10+. Сейчас нами используется [Node.js](https://nodejs.org/) v18.17.1.

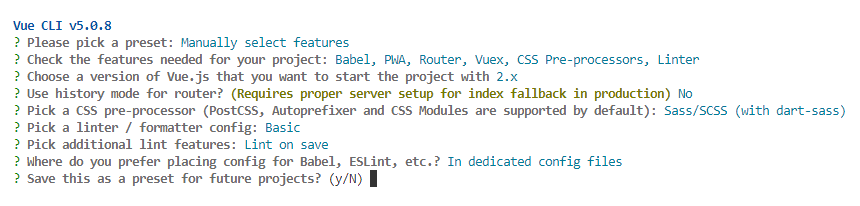
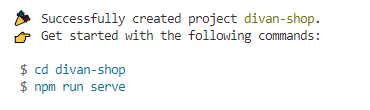
Устанавливаем Vue CLI (на данный момент v5.0.8):

npm install -g @vue/cli

Vue предлагает нам выбрать один из вариантов комплектации нашего будущего проекта.

Выбираем «Manually select features», что позволит нам настроить проект вручную:

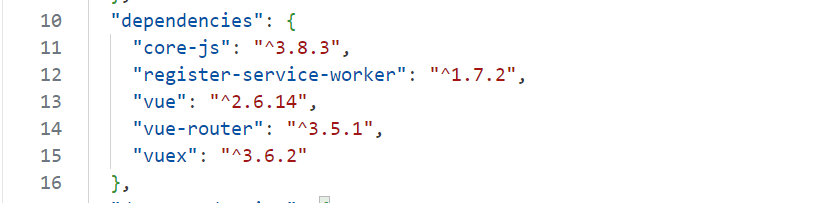
Конфигурацию не сохраняем.

Проект создан.

Если изначально проект разворачивался без установки Vue-router, то его нужно будет в последствии поставить дополнительно, например, следующим образом:

npm install vue-router

В случае дополнительной установки, следует отдельно проконтролировать соответствие установленных версий. С Vue2 будет работать Vue-router и Vuex не выше третьей версии. В нашем случае, проект развернут вместе с библиотеками:

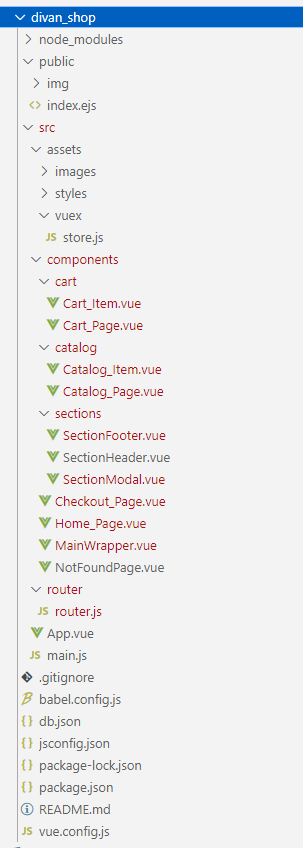


В нашем интернет магазине должна быть предусмотрены:

1. удобная демонстрация товара, его сортировка;
2. возможность детально рассмотреть каждую карточку товара;
3. функционал корзины., наглядность покупки.

Для этого, предварительно продумав структуру приложения, мы создали несколько компонентов и объединили в папки: cart, catalog, sections.

Структура нашего приложения следующая:

****

Создаем оболочку (контейнер) для нашего проекта: MainWrapper.

Его структура:

<template>

  <div>

    <keep-alive>

      <router-view> </router-view>

    </keep-alive>

  </div>

</template>

<script>

export default {

  name: 'MainWrapper',

  data() {

    return {

      title: 'Main Wrapper',

    };

  },

};

</script>

<style lang="scss" scoped></style>

Неотъемлемая часть проекта на Vue.js – router. Напишем и его:

import Vue from 'vue';

import Router from 'vue-router';

import Catalog from '../components/catalog/Catalog\_Page.vue';

import Cart from '../components/cart/Cart\_Page.vue';

import HomePage from '../components/Home\_Page.vue';

import NotFoundPage from '../components/NotFoundPage.vue';

import CheckoutPage from '../components/Checkout\_Page.vue';

Vue.use(Router);

let router = new Router({

  routes: [

    {

      path: '/',

      name: 'home',

      component: HomePage,

    },

    {

      path: '/catalog',

      name: 'catalog',

      component: Catalog,

    },

    {

      path: '/cart',

      name: 'cart',

      component: Cart,

      props: true,

    },

    {

      path: '/checkout',

      name: 'checkout',

      component: CheckoutPage,

      props: true,

    },

    {

      path: '/\*',

      name: '404',

      component: NotFoundPage,

      props: true,

    },

  ],

});

export default router;

Структура App.vue:

<template>

  <div id="app">

    <main-wrapper />

  </div>

</template>

<script>

import MainWrapper from './components/MainWrapper.vue';

export default {

  name: 'App',

  components: {

    MainWrapper,

  },

};

</script>

В качестве компонента импортируем только нашу оболочку: MainWrapper. Больше нам ничего не понадобится.

Структура main.js:

import Vue from 'vue';

import App from './App.vue';

import store from './assets/vuex/store';

import router from './router/router';

import './assets/styles/styles.scss';

Vue.config.productionTip = false;

new Vue({

  render: (h) => h(App),

  store,

  router,

}).$mount('#app');

Привязываем store и router. Библиотеки типа axios мы использовать не будем. Обойдемся только store. По традиции это самая внушительная часть проекта.

Она включает себя следующие элементы:

**state** – базу данных нашего проекта:

state: {

    products: [..

    ],

    products\_box: [],

    catalogTags: [

      { id: 0, tag: 'bedroom', title: 'Спальни' },

      { id: 1, tag: 'living\_room', title: 'Мягкая мебель' },

      { id: 2, tag: 'kitchen', title: 'Кухни' },

      { id: 3, tag: 'play\_room', title: 'Детские' },

    ],

    cart: [],

  },

products – массив, содержащий все продукты нашего онлайн магазина;

products\_box – пустой массив для вывода карточек товаров через цикл;

catalogTags – массив тэгов для отрисовки блока кнопок;

cart – пустой массив для вывода продуктов в корзине.

В блоке **mutations** содержатся методы для изменения состояния данных в хранилище, а именно: добавление и удаление товара, изменение его количества в корзине и сортировка товаров на странице каталога.

  mutations: {

    SET\_CART: (state, product) => {

      if (state.cart.length) {

        let isProductExists = false;

        state.cart.map(function (item) {

          if (item.article === product.article) {

            isProductExists = true;

            item.quantity++;

          }

        });

        if (!isProductExists) {

          state.cart.push(product);

        }

      } else {

        state.cart.push(product);

      }

    },

    REMOVE\_FROM\_CART: (state, index) => {

      state.cart.splice(index, 1);

    },

    INCREMENT: (state, index) => {

      state.cart[index].quantity++;

    },

    DECREMENT: (state, index) => {

      if (state.cart[index].quantity > 1) {

        state.cart[index].quantity--;

      }

    },

    SET\_PRODUCTS\_BOX: (state) => {

      state.products\_box = [];

      while (state.products\_box.length < 3) {

        let random = Math.floor(Math.random() \* (state.products.length + 1));

        state.products.map(function (item) {

          if (random === item.article) {

            state.products\_box.push(item);

          }

        });

      }

    },

  },

В блоке **actions** объявляются методы, которые будут вызывать какие-либо изменения в хранилище.

  actions: {

    ADD\_TO\_CART({ commit }, product) {

      commit('SET\_CART', product);

    },

    DELETE\_FROM\_CART({ commit }, index) {

      commit('REMOVE\_FROM\_CART', index);

    },

    INCREMENT\_CART\_ITEM({ commit }, index) {

      commit('INCREMENT', index);

    },

    DECREMENT\_CART\_ITEM({ commit }, index) {

      commit('DECREMENT', index);

    },

  },

И, наконец, **getters**:

  getters: {

    PRODUCTS(state) {

      return state.products;

    },

    CART(state) {

      return state.cart;

    },

    CATALOG\_TAGS(state) {

      return state.catalogTags;

    },

    PRODUCTS\_BOX(state) {

      return state.products\_box;

    },

  },

Посредством getters мы можем получить данные из хранилища в любом компоненте приложения.

Разберем компонент Home\_Page.

<template>

  <div class="body main">

    <HeaderBlock />

    <section class="home\_\_picture center">

      <div class="home\_\_title">

        <div

          class="home\_\_sale"

          @click="$router.push({ path: '/catalog' })"

        >

          <a…

          </a>

        </div>

        <span class="home\_\_title-text">Выбери Свой Стиль</span>

      </div>

    </section>

<section class="home-block center"> …

    </section>

    <section class="product-box center">

      <CatalogItem

        v-for="product in PRODUCTS\_BOX"

        :key="product.article"

        v-bind:product\_data="product"

        @addToCart="addToCart"

      />

    </section>

    <section class="ad center"> …

    </section>

    <FooterBlock />

  </div>

</template>

На данной странице использованы дополнительные блоки. Здесь же они импортируются

import HeaderBlock from '@/components/sections/SectionHeader.vue';

import FooterBlock from '@/components/sections/SectionFooter.vue';

import CatalogItem from '@/components/catalog/Catalog\_Item.vue';

и подключаются

components: { HeaderBlock, FooterBlock, CatalogItem },

Кроме того, нужно «вытащить» из store методы:

import { mapGetters, mapActions, mapMutations } from 'vuex';

Запрашиваем данные из store:

  computed: {

    ...mapGetters(['PRODUCTS', 'PRODUCTS\_BOX']),

  },

Используя запрошенные нами методы

methods: {

    ...mapActions(['ADD\_TO\_CART']),

    ...mapMutations(['SET\_PRODUCTS\_BOX']),

    addToCart(data) {

      this.ADD\_TO\_CART(data);

    },

    showBlock() {

      this.SET\_PRODUCTS\_BOX();

    },

  },

через цикл v-for

<CatalogItem

        v-for="product in PRODUCTS\_BOX"

        :key="product.article"

        v-bind:product\_data="product"

        @addToCart="addToCart"

      />

выводим три случайных элемента Catalog\_Item на страницу (метод SET\_PRODUCTS\_BOX), даем пользователю возможность добавлять товары в корзину (биндим метод addToCart), а хук жизненного цикла beforeMount

beforeMount() {

    this.showBlock();

  },

позволяет Vue без труда отрисовывать этот блок перед загрузкой страницы.

На кнопку навешиваем действие по клику

@click="showBlock"

и в компоненте Catalog\_Item через props передаем данные товара

 props: {

    product\_data: {

      type: Object,

      default() {

        return {};

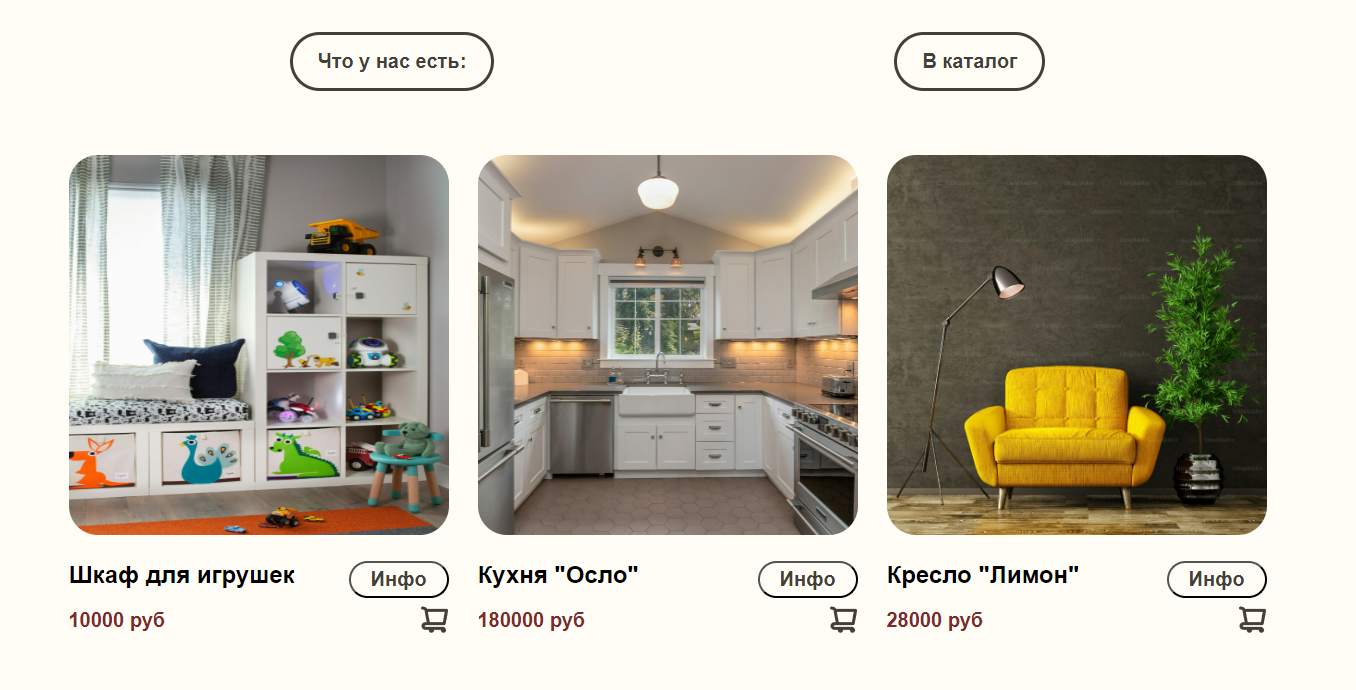
      },

    },

  },

Теперь при нажатии конки каждый раз будет отображаться блок из случайных карточек.

Вот так:



Кроме того, на странице каталога реализована функция модального окна.

Для этого в компонент Catalog\_Item импортируем следующий блок – SectionModal и используем методы

    showModalInfo() {

      this.isInfoModalVisible = true;

    },

    closeModalInfo() {

      this.isInfoModalVisible = false;,

позволяющие нам показывать и скрывать модальное окно по клику. Реализуем функцию закрытия окна по клику на свободную область wrapper:

  mounted() {

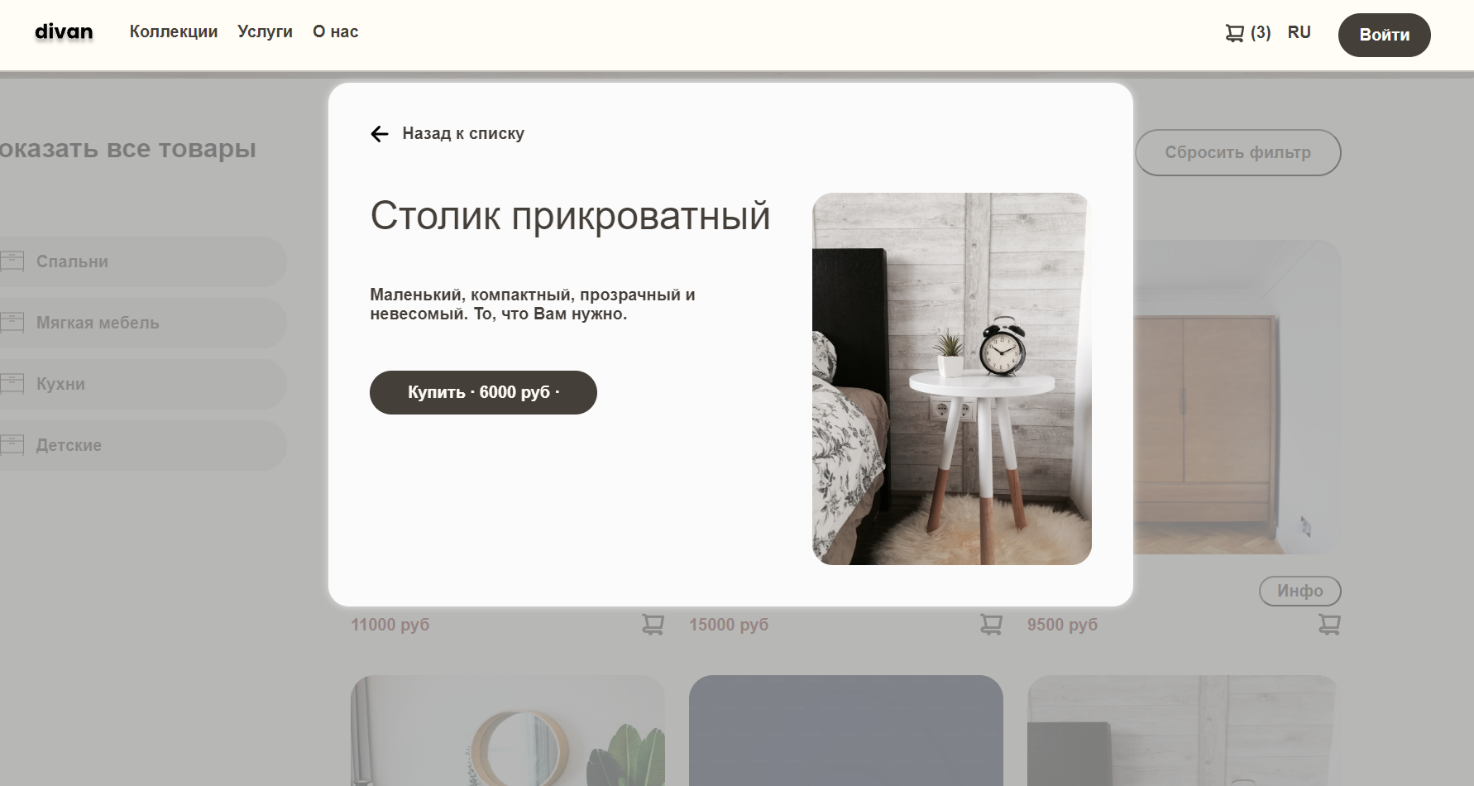
    let vm = this;

    document.addEventListener('click', function (item) {

      if (item.target === vm.$refs['modal\_wrapper']) {

        vm.closeModal();

      }



Похожим образом реализована страница корзины. Не будем подробно на ней останавливаться, укажем лишь метод подсчета итоговой стоимости в компоненте Cart\_Page:

  computed: {

    totalCartCost() {

      let result = [];

      if (this.cart\_data.length) {

        for (let item of this.cart\_data) {

          result.push(item.price \* item.quantity);

        }

        result = result.reduce(function (sum, el) {

          return sum + el;

        });

        this.$emit('totalCartCost', this.result);

        return result;

      } else {

        return 0;

      }

    },

  },

Зв счет методов

methods: {

    ...mapActions([

      'DELETE\_FROM\_CART',

      'INCREMENT\_CART\_ITEM',

      'DECREMENT\_CART\_ITEM',

    ]),

    deleteFromCart(index) {

      this.DELETE\_FROM\_CART(index);

    },

    increment(index) {

      this.INCREMENT\_CART\_ITEM(index);

    },

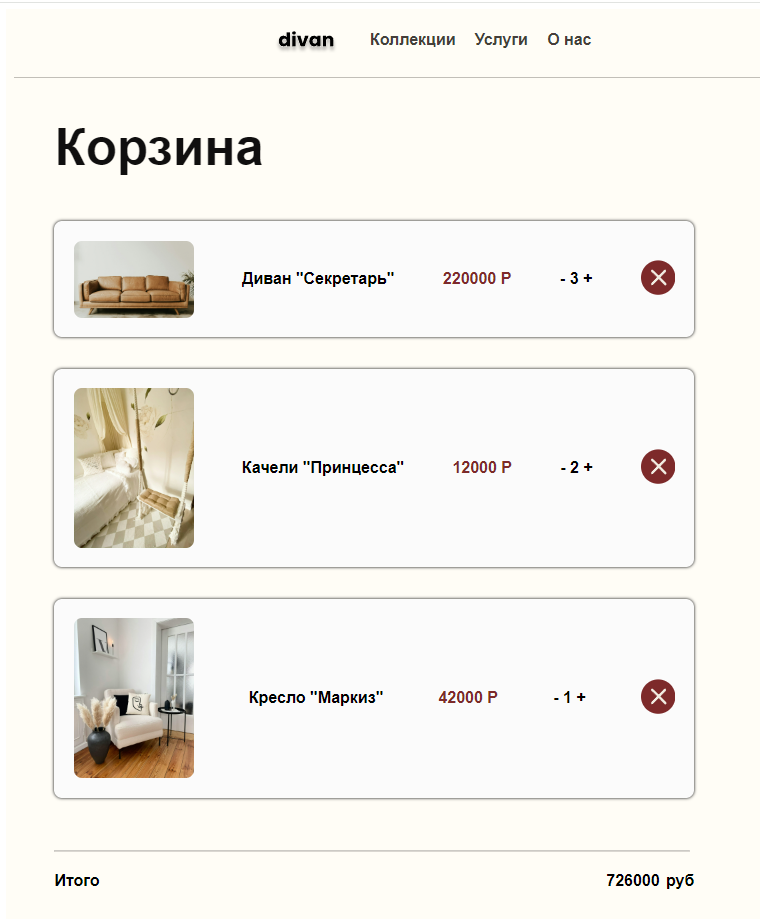
    decrement(index) {

      this.DECREMENT\_CART\_ITEM(index);

    },

  },

реализована возможность удалять товары из корзины и изменять их количество. В результате страница корзины выглядит таким образом:



Для написания CSS-кода проекте использовался препроцессор Sass. Как пример приведем часть кода компонента Cart\_Item:

.cart {

  &\_\_title {

    display: flex;

    flex-direction: column;

    justify-content: center;

    color: #000;

    font-size: 20px;

    font-style: normal;

    font-weight: 600;

    line-height: normal;

  }

  &\_\_item {

    background: #fbfbfb;

    display: flex;

    flex-wrap: wrap;

    justify-content: space-between;

    align-items: center;

    box-shadow: 0 0 4px 0;

    padding: 24px;

    margin-bottom: 16px;

    width: 40vw;

    min-width: 550px;

    border-radius: 10px;

  }

  &\_\_item-image {

    max-width: 150px;

    border-radius: 10px;

  }

}

Таким образом, на примере проекта онлайн магазина «divan» мы убедились в способности фреймворка Vue максимально упростить работу над приложением и помочь даже начинающему разработчику сделать структуру кода стройной и наглядной.

# **Заключение**

Сложно представить современную IT – индустрию без такого явления как фреймворк. Это уже давно неотъемлемая её часть. Попробуйте найти программиста, принципиально игнорирующего фреймворки в своей практике и поинтересуйтесь, легко ли ему кодится.

В данной работе мы предприняли попытку объяснить, что же такое фреймворки, зачем они нужны современному разработчику, чем отличаются друг от друга, разберем их плюсы и минусы (если найдем). Чуть подробнее остановились на теме архитектуры фреймворков, а также затронули проблему выбора на этапе планирования проекта.

Вторая глава посвящена одному из лидеров индустрии фронтенд-разработки - фреймворку Vue.js. Здесь мы раскрыли его особенности, возможные недостатки, разобрали преимущества перед конкурентами. Кроме того, рассмотрели принципиальные отличия сравнительно новой третьей версии Vue.

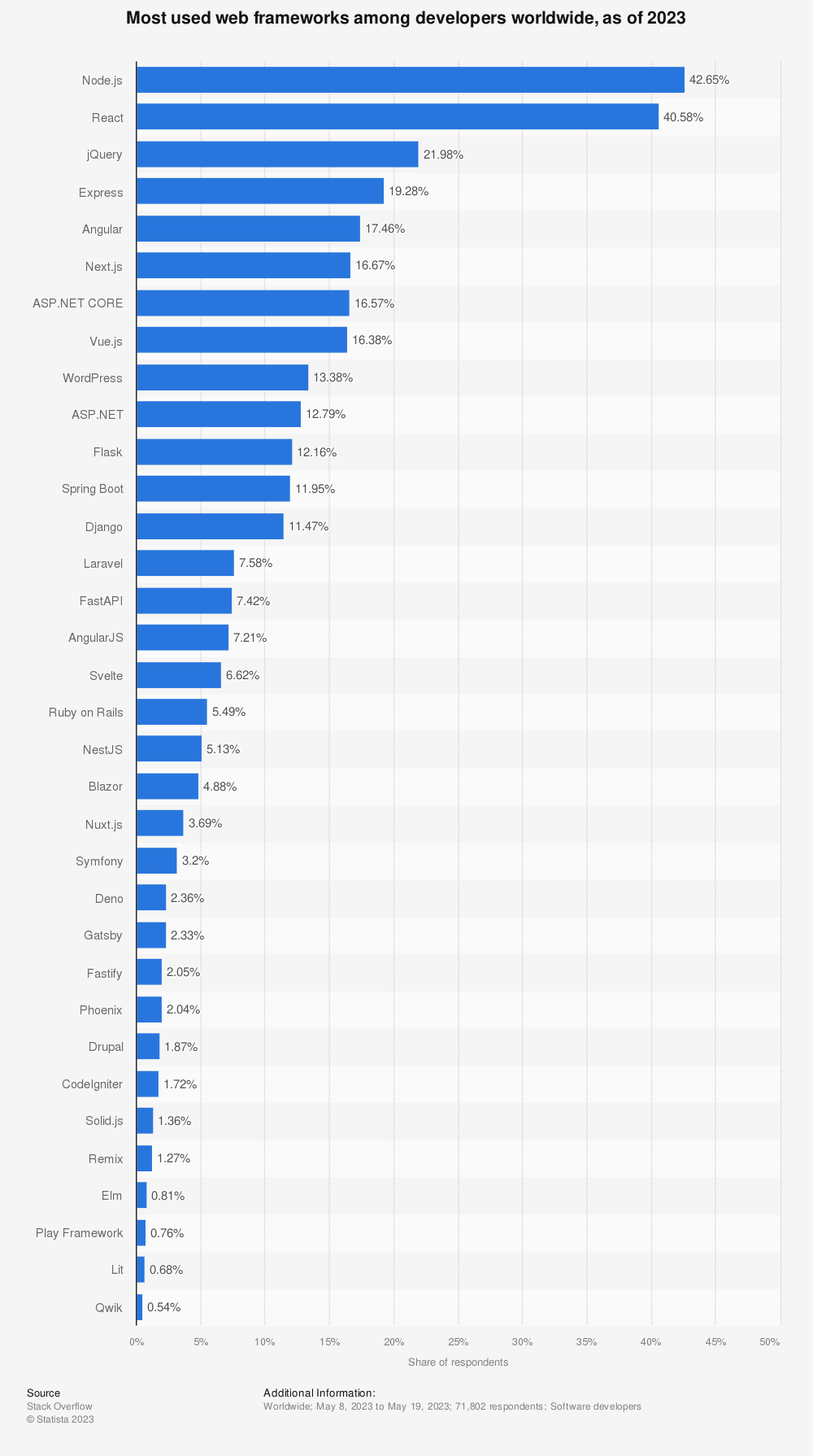
И, наконец, в третьей главе с помощью Vue.js создали небольшой интернет магазин на несколько страниц. Таким образом, мы на практике убедились в одном из преимуществ Vue, заявленных разработчиками, а именно дружелюбии к новичкам.

# **Список литературы**

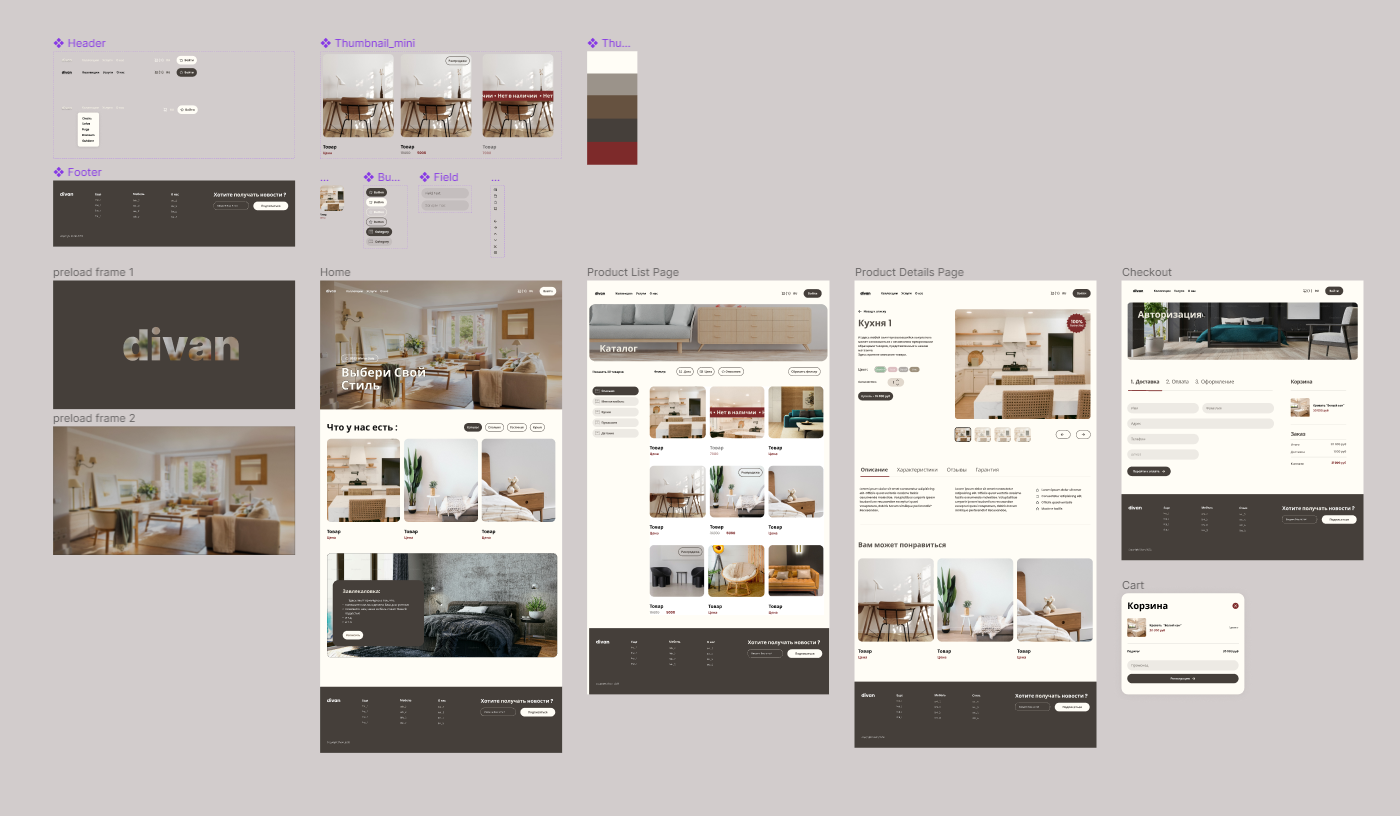
1. <https://ru.hexlet.io/blog/posts/chto-takoe-framework>
2. <https://ru.wikipedia.org/wiki/Фреймворк>
3. https://www.simform.com/blog/best-frontend-frameworks/
4. <https://aglowiditsolutions.com/blog/most-popular-backend-frameworks/>
5. <https://www.tutorialspoint.com/mvc_framework/mvc_framework_quick_guide.htm>
6. <https://symfony.com/ten-criteria>
7. https://github.com/vuejs/core/releases/tag/v3.0.0
8. https://tproger.ru/articles/kak-sozdat-prilozhenie-vokrug-composition-api-vo-vue-3
9. https://www.trio.dev/blog/vue-2-vs-vue-3

# **Приложения**

Приложение 1



Приложение 2



1. Список литературы, п.1 [↑](#footnote-ref-1)
2. Список литературы, п.2 [↑](#footnote-ref-2)
3. Список литературы, п.3 [↑](#footnote-ref-3)
4. Список литературы, п.4 [↑](#footnote-ref-4)
5. Список литературы, п.5 [↑](#footnote-ref-5)
6. Список литературы, п.6 [↑](#footnote-ref-6)
7. Список литературы, п.7 [↑](#footnote-ref-7)
8. Список литературы, п.8 [↑](#footnote-ref-8)
9. Список литературы, п.9 [↑](#footnote-ref-9)
10. https://www.maddybeard.com/ [↑](#footnote-ref-10)
11. приложение 2 [↑](#footnote-ref-11)