Величие Vue.js 2

Алекс Кириакидис

Костас Маниатис

Величие Vue.js 2

Kostas Maniatis, Alex Kyriakidis и Alexey Pyltsyn

Эта книга предназначена для продажи на http://leanpub.com/vuejs2-russian

Эта версия была опубликована на 2018-09-01

Leanpub

Это книга с Leanpub book. Leanpub позволяет авторам и издателям участвовать в так называемом Lean Publishing - процессе, при котором электронная книга становится доступна читателям ещё до её завершения. Это помогает собрать отзывы и пожелания для скорейшего улучшения книги. Мы призываем авторов публиковать свои работы как можно раньше и чаще, постепенно улучшая качество и объём материала. Тем более, что с нашими удобными инструментами этот процесс превращается в удовольствие.

© 2018 Kostas Maniatis, Alex Kyriakidis и Alexey Pyltsyn

Оглавление

Предисловие к русскому изданию i

Введение ii

Про Vue.js iii
- Обзор Vue.js iii
- Что говорят люди о Vue.js iii
- Сравнение с другими фреймворками v

Добро пожаловать xiii
- О книге xiii
- Для кого эта книга xiii
- Как с нами связаться xiii
- Домашние задания xiv
- Код примеров xiv
- Опечатки xiv
- Принятые обозначения xiv

О русском издании xvii
- Немного о переводчике xvii
- Опечатки xvii
- Обновления xvii
- Благодарности xviii
- Принятые соглашения xviii
- Примечание к отображению блоков кода xix
- Обратная связь xix

I Основы Vue.js 1

1. Установка Vue.js 2
- 1.1 Автономная версия 2
- 1.2 Загрузка с помощью NPM 2
- 1.3 Загрузка с помощью Bower 3

2. Приступаем к работе **4**
2.1 Привет, Мир 4
2.2 Двусторонняя привязка 6
2.3 Сравнение с jQuery 7
2.4 Домашнее задание 9

3. Вкус директив **10**
3.1 v-show 10
3.2 v-if 13
3.3 v-else 15
3.4 v-if по сравнению с v-show 17
3.5 Домашняя работа 18

4. Отрисовка списков **19**
4.1 Установка и использование Bootstrap 19
4.2 v-for 22
4.3 Отрисовка массива 25
4.4 Объект v-for 30
4.5 Домашняя работа 32

5. Интерактивность **33**
5.1 Обработка событий 33
5.2 Модификаторы событий 36
5.3 Модификаторы клавиш 40
5.4 Вычисляемые свойства 41
5.5 Домашняя работа 47

6. Фильтры **48**
6.1 Отфильтрованные результаты 48
6.2 Отсортированные результаты 57
6.3 Пользовательские фильтры 61
6.4 Служебные библиотеки 62
6.5 Домашняя работа 66

7. Компоненты **68**
7.1 Что такое компоненты? 68
7.2 Использование компонентов 68
7.3 Шаблоны 70
7.4 Входные параметры 71
7.5 Повторное использование 74
7.6 Всё вместе 77
7.7 Домашняя работа 84

8. Пользовательские события **85**

8.1 Генерация и обработка событий . . . 85
8.2 Взаимодействие родительского и дочернего компонента . . . 88
8.3 Передача аргументов . . . 90
8.4 Не взаимодействие родительского и дочернего компонента . . . 94
8.5 Удаление обработчика событий . . . 97
8.6 Обратно к историям . . . 98
8.7 Домашняя работа . . . 101

9. Привязки к классу и стилю . . . 103
9.1 Привязка CSS-класса . . . 103
9.2 Связывание стилей . . . 107
9.3 Привязки в действии . . . 109
9.4 Домашняя работа . . . 112

II Использование API . . . 113

10. Предисловие . . . 114
10.1 CRUD . . . 114
10.2 API . . . 114

11. Работа с реальными данными . . . 120
11.1 Получение данных асинхронно . . . 120
11.2 Рефакторинг . . . 124
11.3 Обновление данных . . . 126
11.4 Удаление данных . . . 128

12. HTTP-клиенты . . . 131
12.1 Введение . . . 131
12.2 Vue-resource . . . 131
12.3 Axios . . . 132
12.4 Интеграция axios . . . 133
12.5 Расширение функциональности . . . 135
12.6 JavaScript-файл . . . 145
12.7 Исходный код . . . 146
12.8 Домашняя работа . . . 151

13. Пагинация . . . 153
13.1 Реализация . . . 154
13.2 Ссылки на пагинацию . . . 157
13.3 Домашняя работа . . . 160

III Создание крупномасштабных приложений 161

14. ECMAScript 6 . . . . . **162**
14.1 Введение 162
14.2 Объявления переменных 163
14.3 Стрелочные функции 164
14.4 Модули 165
14.5 Классы 166
14.6 Значения параметров по умолчанию 167
14.7 Шаблонные литералы 168

15. Продвинутый рабочий процесс **170**
15.1 Компиляция ES6 с помощью Babel 170
15.2 Автоматизация рабочего процесса с помощью Gulp 178
15.3 Сборка модулей с помощью Webpack 182
15.4 Резюме 188

16. Работа с однофайловыми компонентами **189**
16.1 `vue-cli` 189
16.2 Формирование файлов *.vue* 203

17. Устранение повторяющегося состояния **217**
17.1 Совместное использование через входные параметры 217
17.2 Глобальное хранилище 221

18. Замена компонентов **225**
18.1 Динамические компоненты 225

19. Vue Router **232**
19.1 Установка 232
19.2 Использование 233
19.3 Именованные маршруты 236
19.4 Режим HTML5 History 237
19.5 Вложенные маршруты 238
19.6 Автоматическое добавление CSS-класса на активный маршрут 240
19.7 Объект маршрута 243
19.8 Динамические сегменты 243
19.9 Псевдонимы маршрутов 250
19.10 Программная навигация 252
19.11 Переходы 254
19.12 Навигационные хуки 258
19.13 Домашнее задание 260

20. Заключение **263**

21. Дальнейшее обучение **264**
21.1 Руководства 264
21.2 Видеоуроки 264
21.3 Книги 265
21.4 Проекты с открытым исходным кодом 265
21.5 Awesome Vue 266

Полезные русскоязычные ресурсы по Vue.js **267**
Общее 267
Статьи 267
Лучшие практики 267
Видео 268
Аудио 268

Предисловие к русскому изданию

Не проходит и дня без появления нового JavaScript-фреймворка. Но лишь немногие задерживаются в современном мире фронтенда и ещё меньше добиваются популярности.

Vue — один из оставшихся. Появился он сравнительно недавно, но тем не менее продолжает уверенно шагать, конкурируя с широко известными React и Angular.

Чем же он так прельщает разработчиков? При своём небольшом размере, Vue предлагает на старте хороший набор функциональности, с лёгкостью интегрируется в готовые приложения и не ограничивает в выборе библиотек. А когда хочется большего — прекрасно масштабируется (маршрутизация, централизованное хранилище, дополнительные приятные возможности однофайловых компонентов). На мой взгляд, простота и удобство — то, чего иногда так не хватает.

Кто-то называет Vue новым jQuery. Пусть так — в своё время эта библиотека завоевала свою популярность, представив удобный API по работе с DOM, и успешно соответствовала слогану «Write less — do more». Vue, в свою очередь, воплощая концепцию работы от данных, позволяет нам писать ещё меньше кода (связанного с обновлением страницы) и больше сосредоточиться на логике приложения, снимая этот пласт забот с наших плеч.

Vue — это не какой-то новый язык программирования. Всё, что вы увидите, — обычный JS, помните об этом. А значит и знания, на получение которых вы потратите время, сможете применять не только со Vue.

Конечно, эта лёгкость освоения имеет и обратную сторону медали. Некоторые, не освоив новые концепции (или даже основы JavaScript), пытаются использовать Vue на старый лад, а потом сильно удивляются, когда что-то не работает. Поэтому наговнокодить можно не моргнув и глазом :)

В этой книге на простых примерах постараемся разобрать часто встречающиеся ситуации и методы их решения. Но не стоит бояться написать неидеальный код. Куда полезнее по мере изучения возвращаться с новыми знаниями и идеями, и шаг за шагом делать свой код лучше.

Александр Соколов[1], главред переводов документаций команды Translation Gang[2].

[1] *https://github.com/Alex-Sokolov*
[2] *https://github.com/translation-gang*

Введение

Про Vue.js

Обзор Vue.js

Vue (произносится как /vjuː/, примерно как view) — это прогрессивный фреймворк для создания пользовательских интерфейсов. В отличие от монолитных фреймворков, Vue разработан с нуля для постепенного внедрения. Ядро библиотеки ориентировано **только на слой представления**, и его легко взять и интегрировать с другими библиотеками или существующими проектами. С другой стороны, Vue также отлично подходит для создания одностраничных приложений (Single-Page Applications), когда он используется в сочетании с современными инструментами и доступными библиотеками[3].

Если вы опытный фронтенд-разработчик и хотите узнать, чем Vue.js отличается от других библиотек/фреймворков, ознакомьтесь с главой Сравнение с другими фреймворками.

Если вам интересно больше узнать о ядре Vue.js, посмотрите на официальное руководство Vue.js[4].

Что говорят люди о Vue.js

"Vue.js — это то, что заставило меня полюбить JavaScript. Его очень легко и приятно использовать. Он обладает большой экосистемой плагинов и инструментов, позволяющие расширить его основную функциональность. Вы можете быстро включить его в любой проект, маленький или большой, написать несколько строк кода, и вы уже готовы к работе. Vue.js быстрый, лёгкий и это будущее фронтенд-разработки!"

— Алекс Кириакидис (Alex Kyriakidis)

"Когда я начал улучшать свой уровень JavaScript, я был взволнован, посмотрев кучу возможностей, но когда мой друг предложил изучить Vue.js и я последовал его совету, и дела пошли очень хорошо. Во время чтения и просмотра учебных материалов, я всё время думал о том, что я делал до сих пор и насколько это могло быть легче, если бы я ранее потратил время на изучение Vue. Моё мнение таково, что если вы хотите работать быстро, приятно и легко, то Vue — это JS-фреймворк, который вам нужен."

— Костас Маниатис (Kostas Maniatis)

[3] https://github.com/vuejs/awesome-vue#libraries--plugins
[4] https://ru.vuejs.org/v2/guide/

“Запомните мои слова: Vue.js сильно вырастит в популярности в 2016 году. Он того стоит.”

— ***Джеффри Уэй (Jeffrey Way)***

“Vue — это тот фреймворк, который я всегда хотел найти в JavaScript. Этот фреймворк, который масштабируется вместе с вами как разработчиком. Вы можете использовать его на одной странице, или создать продвинутое одностраничное приложение, используя Vuex и Vue Router. Это действительно самый доведённый до совершенства JavaScript-фреймворк, который я когда-либо видел.”

— ***Тейлор Отуэлл (Taylor Otwell)***

“Vue.js — это первый фреймворк, который я нашёл, подходящий как для обычного бекенд-приложения, так и для создания полномасштабного SPA. Вне зависимости от того, нужен мне небольшой виджет на какой-нибудь странице или я создаю сложное JavaScript-приложение, Vue никогда не ощущается как что-то недостаточное или слишком ненужное для этого.”

— ***Адам Ватан (Adam Wathan)***

“Vue.js удалось стать фреймворком, который простой в использовании и лёгкий для понимания. Это глоток свежего воздуха в мире, где люди борются за то, чтобы узнать, кто из них сделает самый сложный фреймворк.”

— ***Эрик Барнс (Eric Barnes)***

“Причина, по которой мне нравится Vue.js, — это то, что я одновременно и дизайнер, и разработчик. Я посмотрел на React, Angular и на несколько других, но кривая обучения и терминология меня всегда отталкивали. Vue.js — это первый JS-фреймворк, который я понимаю. Он не только лёгок на подъём для такого менее уверенного JS-разработчика, как я, но я также заметил, что опытные разработчики на Angular и React обращают на него внимание и испытывают симпатию к Vue.js. Это довольно необычно для мира JS, и именно поэтому я начал митап London Vue.js.”

— ***Джек Бархам (Jack Barham)***

“В поисках альтернативы Angular и jQuery, я вспомнил, что однажды наткнулся на Vue.js. В том время я совсем немного изучил его, но, изучая его подробнее, я нашёл идеальным его для удовлетворения моих основных потребностей: повышения реактивности моих представлений в приложениях на ASP.NET MVC. Поэтому я стал использовать Vue.js вместо Angular и jQuery. Я научился любить и уважать Vue.js, потому что его было легко изучать и просто использовать. Между тем я пришёл к выводу, что мне больше не нужен jQuery или Angular, потому что есть Vue.js, и этот потрясающий фреймворк может использоваться как замена им обоим: как для улучшения представлений или как инструмент для разработки полноценных SPA и даже PWA.”

— ***Майкл Сигер (Michael Seeger)***

Сравнение с другими фреймворками

Angular 1

Некоторый синтаксис Vue выглядит похожим на Angular (к примеру, `v-if` и `ng-if`). Это объясняется тем, что многие идеи в Angular сделаны правильно, и они стали вдохновением на раннем этапе разработки Vue. Однако есть также немало болезненных проблем в Angular, где Vue попытался предложить значительное улучшение.

Сложность

Vue гораздо проще Angular 1, как с точки зрения API, так и проектирования. Знаний, полученных за один день, будет достаточно для создания нетривиальных приложений, что нельзя сказать про Angular 1.

Гибкость и модульность

Приложение на Angular 1 должно следовать жёстко заданной структуре, в то же время Vue более гибкий и модульный в плане структурирования. Поэтому Vue предоставляет шаблон Webpack[5], с помощью которого можно настроить новый проект в течение нескольких минут, а также получить доступ к таким продвинутым возможностям, как горячая замена модулей (hot module reloading), проверка кода (linting), извлечение CSS и т.д.

[5] https://github.com/vuejs-templates/webpack

Связывание данных

Angular 1 использует двунаправленное связывание между двумя областями видимости, тогда как Vue обеспечивает однонаправленный поток данных между компонентами. Это упрощает понимание потока данных в непростых приложениях.

Директивы по сравнению с компонентами

У Vue более чёткое разделение между директивами и компонентами. Директивы предназначены только для инкапсуляции операций с DOM, а компоненты — автономные блоки с собственной логикой и представлением. В Angular много путаницы между ними.

Производительность

Vue показывает лучшую производительность, кроме того, из-за отсутствия *простого сравнения состояний объектов* или dirty checking позволяет намного проще оптимизировать приложения. Angular 1 становится медленным при увеличении количества наблюдателей, потому что каждый раз, когда что-либо в области видимости изменяется, все эти наблюдатели должны быть снова повторно запущены. Кроме того, цикл дайджеста, возможно, придётся запускать несколько раз для «стабилизации», если какой-нибудь наблюдатель запускает следующее обновление. Пользователям Angular часто приходится прибегать к экзотическим техникам для обхода цикла дайджеста, а в некоторых ситуациях попросту нет способа оптимизировать область видимости со многими наблюдателями.

Vue не испытывает таких проблем, потому что использует прозрачную систему отслеживания учёта зависимостей с асинхронной очередью — все изменения происходят независимо, если у них нет явной связи между ними.

Любопытно, существуют довольно много схожего в том, как Angular 2 и Vue решают эти проблемы Angular 1.

Angular 2

Есть отдельный раздел, посвящённый Angular 2, потому что это действительно новый фреймворк. Например, теперь в нём есть полноценная система компонентов, многие детали реализации были полностью переписаны, кроме того, API сильно изменился.

Размер и производительность

Что касается производительности, оба фреймворка исключительно быстрые, и для вынесения окончательного вердикта недостаточно данных, исходя из реальных случаев их использования. Однако, если вы полны решимости увидеть некоторые цифры, то Vue 2.0, похоже, опережает Angular 2, согласно независимому исследованию производительности[6].

[6] https://stefankrause.net/js-frameworks-benchmark4/webdriver-ts/table.html

Хотя Angular 2 с автономной компиляцией и tree-shaking, может значительно уменьшить свой размер, полнофункциональный Vue 2.0 с включённым компилятором (23 Кб) всё равно меньше по размеру по сравнению с готовым каркасом приложения на Angular 2 (50kb).

Гибкость

Vue менее своевольный, чем Angular 2, предлагая официальную поддержку различных систем сборки, без ограничений на то, как должно быть структурировано приложение. Многим разработчикам нравится эта свобода, тогда как некоторые предпочитают иметь один правильный способ создания любого приложения.

Кривая обучения

Для начала работы с Vue нужно знать только HTML и JavaScript ES5 (например, простой JavaScript). Используя эти базовые навыки, можно начать создавать нетривиальные приложения после менее чем одного дня чтения руководства.

Кривая обучения Angular намного круче. API фреймворка просто огромное, и вам, как пользователю, нужно познакомиться с гораздо большим количеством концепций, прежде чем стать продуктивным. Очевидно, что сложность Angular во многом обусловлена его направлённостью на разработку только крупных, сложных приложений, однако это делает фреймворк значительно труднее для менее опытных разработчиков.

React

React и Vue во многом похожи. Оба они:

- используют виртуальный DOM
- предоставляют реактивность и переиспользуемые компоненты
- концентрируют внимание на основной библиотеке, вопросами вроде маршрутизации или управления глобальным состоянием занимаются сопутствующие библиотеки

Быстродействие

В каждом сценарии из реального мира, который был протестирован до этих пор, Vue опережает React с достаточным отрывом.

Производительность отрисовки

При отрисовке пользовательского интерфейса операции с DOM, как правило, являются самой дорогостоящей операцией, и, к сожалению, ни одна библиотека не может быстрее выполнять эти необработанные операции. Чтобы лучше это сделать, нужно:

1. Сократить количество необходимых изменений в DOM. Как React, так и Vue используют абстракции виртуального DOM для достижения этого, и обе реализации работают одинаково хорошо.
2. Добавить как можно меньше накладных расходов поверх этих манипуляций с DOM. Эта область, где Vue и React различаются.

В React, скажем, дополнительные накладные расходы на отрисовку равны 1, а дополнительные накладные расходы средних компонентов равны 2. Во Vue накладные расходы на элемент будут больше похожими на 0.1, но накладные расходы среднего компонента — 4, в результате установки, требуемой для системы реактивности.

Это означает, что в типичных приложениях, где элементов для отрисовки намного больше, чем компонентов, Vue будет превосходить React в значительный степени. Однако в крайних случаях, когда один обычный компонент служит для отрисовки каждого элемента, Vue будет обычно медленнее.

И Vue, и React также предлагают функциональные компоненты, которые могут быть без состояния и не требовать инициализации, и, следовательно, требуют меньше накладных затрат. Когда они используются в критически важных ситуациях, Vue снова оказывается быстрее.

Производительность обновления

В React вам необходимо реализовать `shouldComponentUpdate` всюду и использовать неизменяемые структуры данных для достижения полностью оптимизированных повторных отрисовок. Во Vue зависимости компонента автоматически отслеживаются, так что он

обновляется только при изменении одной из зависимостей. Единственная дополнительная оптимизация, которая иногда может быть полезна во Vue — добавление атрибута key к элементам в длинных списках.

Это означает, что обновления в неоптимизированном Vue будут намного быстрее, чем в неоптимизированном React, и на самом деле из-за улучшенной производительности отрисовки во Vue даже полностью оптимизированный React обычно будет медленнее, чем Vue из коробки.

В разработке

Очевидно, что производительность в продакшене является самой важной, и это то, что обсуждалось до сих пор. Однако эффективность в разработке всё же имеет значение. Хорошей новостью является то, что Vue, как и React, достаточно быстрый для большинства обычных приложений.

Однако, если вы делаете прототип каких-либо высокопроизводительных визуализаций данных или анимации, возможно вам будет интересно знать, что в сценариях, где Vue не может обрабатывать более 10 кадров в секунду при разработке, мы видели, что React замедляется до 1 кадра в секунду.

Это связано со множеством тяжёлых инвариантных проверок React, которые помогают ему предоставлять множество отличных предупреждений и сообщений об ошибках.

Ember

Ember — это полнофункциональный фреймворк, изначально созданный «имеющим и отстаивающим своё мнение». Он требует соблюдения множества соглашений и конвенций, и когда вы с ними освоитесь — он может сделать вас весьма продуктивными. Однако это означает ещё и высокую и крутую кривую обучения, не говоря о страдающей гибкости. Выбор между жёстко структурированным фреймворком и библиотекой со слабосвязанными совместно работающими инструментами — это всегда компромисс. Последняя даёт больше свободы, но и требует принятия большего количества самостоятельных архитектурных решений.

Учитывая вышесказанное, вероятно, будет разумнее сравнивать ядро Vue и слои шаблонизации и объектной модели Ember:

- Vue предлагает ненавязчивую реактивность для простых JavaScript-объектов и полностью автоматические вычисляемые свойства. В Ember от вас ожидается заворачивание всего в «Объекты Ember» и ручное указание зависимостей для вычисляемых свойств.
- Синтаксис шаблонов Vue позволяет использовать все возможности выражений JavaScript, в то время как возможности выражений Handlebars и синтаксис хелперов в Ember намеренно существенно ограничены.
- В вопросах производительности Vue существенно выигрывает — даже с учётом последнего обновления Glimmer engine в Ember 2.x. Vue автоматически объединяет операции обновления, тогда как в Ember требуется ручное управление циклом выполнения в ситуациях, где производительность критична.

Polymer

Polymer — это ещё один проект, спонсируемый Google. В действительности он тоже послужил источником вдохновения для Vue. Компоненты Vue можно приблизительно сравнивать с пользовательскими элементами Polymer. Стиль разработки с использованием обоих фреймворков довольно похож. Самая существенная разница состоит в том, что Polymer базируется на последних возможностях Web Components и требует для работы использования весьма нетривиальных полифилов (с потерей быстродействия в браузерах без нативной поддержки этих возможностей). Vue, напротив, без каких-либо зависимостей или полифилов работает во всех браузерах, начиная с IE 9.

В Polymer 1.0 разработчики существенно ограничили систему связывания данных для улучшения производительности. Например, единственными поддерживаемыми в шаблонах Polymer выражениями являются булево отрицание и одиночные вызовы методов. Реализация вычисляемых свойств тоже не очень гибкая.

Пользовательские элементы Polymer пишутся в HTML-файлах, что ограничивает авторов использованием только простого JavaScript/CSS (и возможностей языка, поддерживаемых браузерами на данный момент). Vue же позволяет использовать в однофайловых компонентах какие угодно пре- и постпроцессоры, включая, разумеется, и ES2015+.

При работе в продакшен-окружении, Polymer рекомендует загружать всё «на лету» при помощи HTML Import, что подразумевает как реализацию этой технологии браузерами, так и поддержку HTTP/2 и сервером, и клиентом. Не для всякой целевой аудитории и не для всякого окружения это может быть выполнимо. В таких случаях предполагается использовать специальный инструмент под названием Vulcanizer, объединяющий элементы Polymer. Vue позволяет комбинировать возможности асинхронной загрузки компонентов с функциями по разделению кода Webpack, организуя «ленивую» загрузку частей приложения. Таким образом, мы сохраняем совместимость со старыми браузерами, не теряя производительности на этапе загрузки приложения.

Riot

Riot 2.0 предлагает похожую модель разработки, основывающуюся на компонентах (в Riot называемых «тегами»), с минималистичным и прекрасно организованным API. Вероятно, Riot и Vue во многом основаны на схожих философских подходах. Однако, несмотря на немного больший размер по сравнению с Riot, Vue имеет некоторые существенные преимущества:

- Условно-истинная отрисовка. Riot отображает все ветви и просто показывает/скрывает их.
- Более мощный маршрутизатор. API маршрутизатор Riot чрезвычайно минимален.
- Более зрелая поддержка инструментов. Vue предоставляет официальную поддержку Webpack, Browserify и SystemJS, в то время как Riot полагается на поддержку сообщества для интеграции системы сборки.
- Система эффектов переходов. У Riot её нет.
- Лучшая производительность. Несмотря на рекламное использование DOM, Riot по факту использует dirty checking и, следовательно, испытывает те же проблемы с производительностью, что и Angular 1.

Для получения обновлённой информации по сравнениям, обратитесь к официальному руководству Vue.js[7].

[7] http://ru.vuejs.org/guide/comparison.html

Добро пожаловать

О книге

Эта книга проведёт вас по пути стремительно распространяющегося JavaScript-фреймворка под названием Vue.js!

Некоторое время назад мы начали новый проект, используя Laravel и Vue.js. После тщательного ознакомления с руководством Vue.js и несколькими учебными материалами обнаружилась нехватка ресурсов по Vue.js во всём вебе. Во время разработки нашего проекта мы получили достаточное количество опыта, поэтому появилась идея написать эту книгу, чтобы поделиться приобретёнными знаниями с миром. Теперь, когда вышел Vue.js 2, мы решили, что пришло время обновить книгу, опубликовав вторую версию, в которой примеры и связанный с ним текст переписаны.

Эта книга написана в неформальном, интуитивно понятном и удобном формате, где все примеры достаточно подробны для обеспечения адекватного руководства для каждого.

Мы начнём с самых основ, и через многие примеры рассмотрим наиболее важные особенности Vue.js. К концу книги вы сможете создавать быстрые фронтенд-приложения и повысить производительность существующих проектов с помощью интеграции Vue.js 2.

Для кого эта книга

Каждый, кто потратил время на изучение современной веб-разработки, работал с Bootstrap, JavaScript и со многими его фреймворками. Эта книга предназначена для всех, кто интересуется изучением лёгкого и простого фреймворка на JavaScript. Никаких особых знаний не требуется, хотя знание HTML и JavaScript было бы кстати. Если вы не знаете, какая разница между строкой и объектом, вам, возможно, стоит сначала изучить основы.

Эта книга полезна как для разработчиков, которые не знают Vue.js, так и тем, кто уже использует Vue.js, но хочет расширить свои знания. Кроме того, книга пригодится тем, кто хочет перейти на Vue.js 2.

Как с нами связаться

Если вы хотите связаться с нами по поводу книги, прислать отзыв или задать вопрос по другой теме, не стесняйтесь обращаться к нам.

Имя	Электронная почта	Твиттер
The Majesty of Vue.js	hello@tmvuejs.com	@tmvuejs
Алекс Кириакидис (Alex Kyriakidis)	alex@tmvuejs.com	@hootlex
Костас Маниатис (Kostas Maniatis)	kostas@tmvuejs.com	@kostaskafcas

Домашние задания

Лучший способ учиться веб-разработке — писать код, поэтому мы подготовили по одному упражнению в конце большинства глав, чтобы вы могли решить их и действительно проверить, что вы изучили. Мы настоятельно рекомендуем как можно больше решить их для лучшего понимания Vue.js. Не бойтесь проверить свои идеи, всё начинается с малого. Возможно, несколько разных примеров или способов дадут вам правильное решение. Однако мы не безжалостны: будут даны подсказки и возможные решения!

Вы можете начать ваше путешествие!

Код примеров

Большинство примеров кода, используемых в книге, можно найти на GitHub. Вы можете посмотреть код здесь[8] (ветка `ru`). Или, если вам так удобнее, скачать его в файле формата `.zip` отсюда[9]. Это избавит вас от копирования и вставки кода из книги, что, вероятно, было бы утомительно.

Опечатки

Несмотря на то, что были приняты все меры для обеспечения точности нашей книги, ошибки всё же случаются. Если вы найдёте одну из них, мы будем благодарны, если вы сообщите нам о ней. Таким образом, вы можете уберечь других читателей от разочарования и заодно поможете нам улучшить последующие версии книги. Если вы обнаружите какие-либо ошибки, пожалуйста, создайте ишью в нашем репозитории на GitHub[10].

Принятые обозначения

В книге используются следующие условные обозначения.

Блок кода будет выглядеть следующим образом:

JavaScript

[8] https://github.com/hootlex/the-majesty-of-vuejs-2/tree/ru
[9] https://github.com/hootlex/the-majesty-of-vuejs-2/archive/ru.zip
[10] https://github.com/hootlex/the-majesty-of-vuejs-2

```
function (x, y) {
    // это комментарий
}
```

Слова с кодом в тексте и данные отображаются так: “Используйте `.container` для адаптивного контейнера с фиксированной шириной.”

Новые термины и **важные слова** выделены жирным шрифтом.

Советы, примечания и предупреждения отображаются подобным образом:

Это предупреждение

Этот элемент указывает на предупреждение или предостережение.

Это совет

Этот элемент означает совет, подсказка или предложение.

Это информационный блок

Некоторая специальная информация.

Это примечание

Примечание по теме.

Это подсказка

Подсказка по теме.

Это команда в терминале

Команды, выполняемые в терминале.

Это текст сравнения

Небольшой текст для сравнения по теме.

Это ссылка на GitHub.

Ссылки ведут на репозиторий книги, где вы можете найти примеры кода и возможные задания к каждой определённой главы.

О русском издании

Немного о переводчике

Меня зовут Алексей Пыльцын[11] и это мой второй перевод обучающей книги по JavaScript-фреймворку. Возможно, вы читали «Путь к изучению React»[12], который я чуть больше месяца назад выпустил. Там были досадные опечатки, которые оперативно исправлялись, и в этом переводе я продолжил делать всё возможное, чтобы предоставить вам, дорогому читателю, возможность наслаждаться изучением нового фреймворка с максимальным комфортом. К сожалению, от ошибок никто не застрахован, поэтому, пожалуйста, будьте благосклонны, если всё-таки их встретите.

Хочу напомнить о прелести Leanpub, вы всегда можете запросить возврат средств, если вам не понравилась книга. Надеюсь, до этого не дойдёт, но если всё-таки такое случится, то прошу прощения за ваше потраченное время, я очень хотел, чтобы на русском языке была обучающая книга по Vue.js.

Я также являюсь участником сообщества переводчиков Translation Gang и приложил руку к переводу и улучшению официальной документации Vue.js и инструмента тестирования Vue Test Utils[13]. Это я к тому, что, купив эту книгу, можете считать, что вы таким образом поддержали меня, чтобы я и дальше занимался этим. Спасибо!

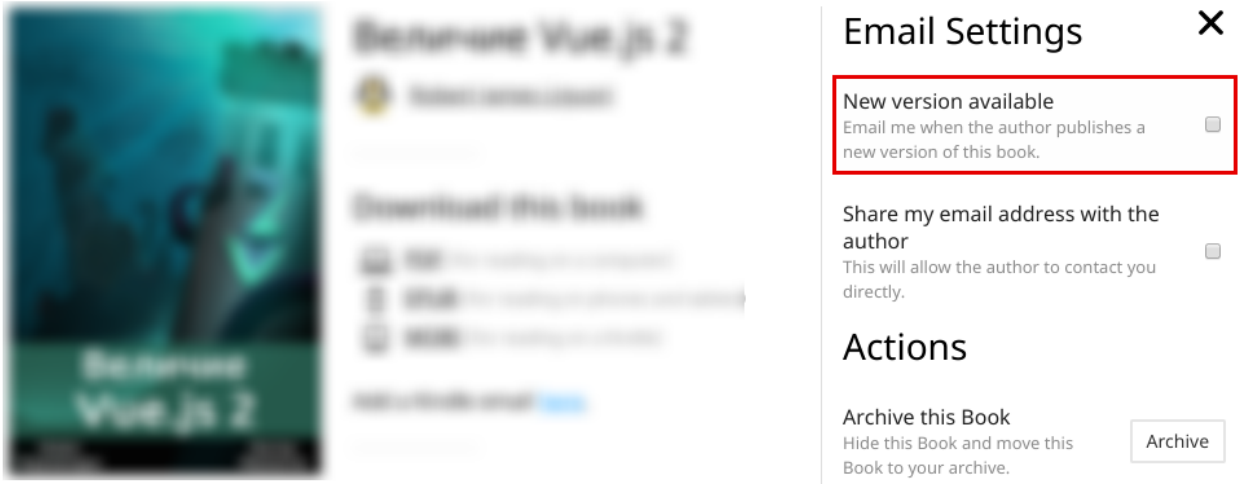
Опечатки

Ни одна книга не идеальна, особенно переводная, поэтому неточности в переводе и опечатки случаются. Пожалуйста, дайте знать о них любым удобным для вас способом, о том как это сделать, смотрите раздел обратной связи.

Обновления

При покупке книги я настоятельно рекомендовал вам подписаться на обновления по электронной почте. Если вы этого не сделали, то есть возможность сделать это сейчас. Зайдите на страницу со всеми вашими книгами на Leanpub[14], выберите эту книгу и справа в блоке «Email Settings» отметьте галочкой «New version available», как показано на скриншоте:

[11] https://github.com/lex111
[12] https://leanpub.com/the-road-to-learn-react-russian
[13] https://vue-test-utils.vuejs.org/ru/
[14] https://leanpub.com/user_dashboard/library



Обязательно будут обновления книги, например, в случае исправления опечаток и неточностей в книге. Поскольку оригинальная книга уже не обновляется, при перевода она была обновлена. Во-первых, используется последняя на момент написания версия Vue.js — 2.5.17. Во-вторых, были исправлены ошибки в оригинальной книге, кроме того, переведно на русский большая часть скриншотов, блоков с кодом, решения домашних заданий, данные для API, а ещё мною был исправлен стиль кода, теперь все «пропущенные» пробелы на своих местах! Всё это вы увидите как в книге, так и в специально созданной для русского перевода ветке в репозитории книги[15].

Благодарности

Прежде всего, хочется поблагодарить Алекса Кириакидиса и Костаса Маниатиса за их книгу[16] и возможность её перевода на русский язык.

Выражаю искреннюю признательность Роману Садженице[17] за его неоценимую помощь в переводе. Александру Соколову[18] за ценные замечания и за всю ту большую работу, которую он делает для русского сообщества Vue.js.

И спасибо тебе, дорогой читатель, что купил данный перевод книги, надеюсь, ты не будешь разочарован, и информация в книге принесёт тебе пользу. Удачи в изучении Vue.js!

Принятые соглашения

В тексте данного перевода книги используется буква «ё», французские кавычки и терминология из замечательного словаря «Веб-стандартов»[19]. Если встретите непонятный термин, то

[15] https://github.com/hootlex/the-majesty-of-vuejs-2/tree/ru
[16] https://leanpub.com/vuejs2
[17] https://github.com/dukkee
[18] https://github.com/Alex-Sokolov
[19] https://github.com/web-standards-ru/dictionary

смело заглядывайте туда, а если там не найдёте разъяснения, то, значит, это слишком общий термин, и возможно, в таком случае вам стоит посмотреть его значение в Google.

Примечание к отображению блоков кода

Некоторые части кода не умещаются в одну строку из-за ограничений ширины страницы. Для обозначения разрыва строки используется обратная коса черта (\). Этот символ выполняет исключительно функцию форматирования. К примеру следующий фрагмент кода:

```
<link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"\
rel="stylesheet">
```

Следует воспринимать так —

<link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">

Именно по этой причине было принято решение разбивать длинные части кода на мелкие, т.е. вышеприведённый фрагмент кода в тексте книги будет отображен следующим образом:

```
<link
  href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
  rel="stylesheet"
>
```

Так оформлены не только данный конкретный блок, который часто встречается в книге, но и другие участки кода. Этим устраняется обозначенная выше проблема с разрывом строки, но это также улучшает читабельность кода, хотя делает его более многословным.

Обратная связь

Если вы хотите задать вопрос по книге или у вас возникли какие-либо проблемы при чтении, или вы просто хотите связаться с нами по какой-либо причине, то вы можете сделать это, создав ишью в GitHub-репозитории[20] либо написав электронное письмо по адресу hello@tmvuejs.com[21]. Кроме того, с большой благодарностью будут приняты ваши предложения с уточнениями и сообщения о найденных ошибках.

Спасибо за покупку перевода и удачи в изучении Vue.js!

[20] https://github.com/hootlex/the-majesty-of-vuejs-2
[21] hello@tmvuejs.com

I Основы Vue.js

1. Установка Vue.js

Теперь, когда пришло время установить Vue.js, есть несколько способов сделать это.

1.1 Автономная версия

1.1.1 Загрузка с vuejs.org

Чтобы установить Vue, вы можете просто скачать его и подключить с помощью тега **`script`**. После этого будет доступна глобальная переменная `Vue`.

Вы можете загрузить две версии Vue.js:

1. Версия для разработки https://vuejs.org/js/vue.js[1]
2. Версия для продакшена https://vuejs.org/js/vue.min.js[2].

Совет: Не используйте минифицированную (сжатую) версию во время разработки. Вы не увидите все полезные предупреждения о распространённых ошибках.

1.1.2 Подключение из CDN

Vue.js.org[3] рекомендует unpkg[4], который будет ссылаться на последнюю версию, как только она будет опубликована в npm.

Вы также можете найти Vue.js на jsdelivr[5] или cdnjs[6]

1.2 Загрузка с помощью NPM

NPM — рекомендуемый способ установки для создания крупных приложений на Vue.js. Он хорошо сочетается с модулем CommonJS, например Webpack[7] или Browserify[8].

[1] https://vuejs.org/js/vue.js
[2] https://vuejs.org/js/vue.min.js
[3] https://ru.vuejs.org/v2/guide/installation.html#CDN
[4] https://unpkg.com/vue/dist/vue.js
[5] https://cdn.jsdelivr.net/npm/vue@2.5.17/dist/vue.min.js
[6] https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.min.js
[7] https://webpack.github.io/
[8] http://browserify.org/

```
# последняя стабильная версия
$ npm install vue
# версия разработки (непосредственно из GitHub):
$ npm install vuejs/vue#dev
```

1.3 Загрузка с помощью Bower

```
# последняя стабильная версия
$ bower install vue
```

Дополнительные инструкции по установке и обновлению смотрите в соответствующем разделе в официальном руководстве Vue.js[9]

В большинстве примеров книги мы включаем Vue.js из CDN, хотя вы можете установить его любым удобным для вас способом.

[9] https://ru.vuejs.org/guide/installation.html

2. Приступаем к работе

Начнём с быстрого ознакомления с возможностями привязки данных Vue. Мы собираемся сделать простое приложение, которое позволит нам ввести сообщение и отобразить его на странице в режиме реального времени. Это приложение продемонстрирует мощь двусторонней привязки данных во Vue. Для создания приложения на Vue, нам потребуется небольшая настройка HTML-страницы.

В процессе разработки вы получите представление о скорости разработки с использованием Vue.js вместо, например, jQuery.

2.1 Привет, Мир

Мы создадим новый файл с каким-нибудь шаблонным кодом. Вы можете назвать его как угодно, но мы назовём его `hello.html`.

```
<html>
<head>
  <title>Привет, Vue</title>
</head>
<body>
  <h1>Приветствую, ваше величество!</h1>
</body>
</html>
```

Это простой HTML-файл с приветственным сообщением.

Теперь мы пойдём дальше и сделаем то же самое, но используя Vue.js. Прежде всего, мы подключим Vue.js и создадим его новый экземпляр.

```
<html>
<head>
    <title>Привет, Vue</title>
</head>
<body>
  <div id="app">
    <h1>Приветствую, ваше величество!</h1>
  </div>
</body>
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
<script>
  new Vue({
    el: '#app',
  })
</script>
</html>
```

Для начала мы подключили Vue.js из cdnjs[1] и внутри тега **`script`** создали экземпляр Vue. Мы используем **`div`** с идентификатором **`id`**, равным **`#app`**, который является элементом, на который мы ссылаемся, поэтому Vue знает, куда «смотреть». Попробуйте думать об этом как о контейнере, в котором работает Vue. Vue не распознаёт ничего вне этого целевого элемента. Используйте опцию **`el`** для указания элемента, в который будет смонтирован Vue.

Теперь присвоим сообщение, которое мы хотим отобразить, в переменную для последующего использования внутри объекта **`data`**. Затем мы передадим объект с данными в качестве опции конструктору Vue.

```
var data = {
  message: 'Приветствую, ваше величество!'
};
new Vue({
  el: '#app',
  data: data
});
```

Чтобы отобразить сообщение на странице, мы просто оборачиваем название свойства в объекте данных (в данном случае это `message`) в двойные фигурные скобки. Поэтому, чтобы наше сообщение автоматически отобразилось внутри тега **`h1`**, мы делаем следующее.

[1] https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js

```
<div id="app">
  <h1>{{ message }}</h1>
</div>
```

Это так просто. Другой способ определить переменную сообщения — сделать это непосредственно внутри конструктора Vue в объекте **data**.

```
new Vue({
  el: '#app',
  data: {
      message: 'Приветствую, ваше величество!'
  }
});
```

Оба способа дают одинаковый результат, поэтому вы снова можете выбрать любой синтаксис, который вам больше нравится.

На заметку

Двойные фигурные скобки — это не HTML, а код скрипта, поэтому что-либо внутри таких тегов называется *выражением привязки*. JavaScript будет выполнять эти выражения, то есть **{{ message }}** возвратит значение JavaScript-переменной. А фрагмент кода **{{ 1 + 2 }}** отобразит число 3.

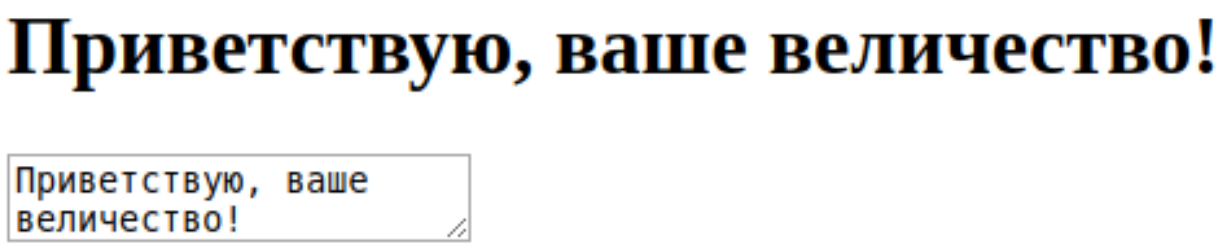
2.2 Двусторонняя привязка

Что круто во Vue, так это то, что он облегчает нашу жизнь. Предположим, мы хотим изменить сообщение, введённое пользователем, как мы можем это легко сделать? В приведённом ниже примере, мы используем директиву Vue **v-model** (мы подробнее рассмотрим директивы в следующей главе). Затем мы используем двустороннюю привязку данных для динамического отображения сообщения при изменении пользователем текста в поле ввода. Данные синхронизируются при каждом изменении поля ввода (событие input) по умолчанию.

```
<div id="app">
  <h1>{{ message }}</h1>
  <textarea v-model="message"></textarea>
</div>
```

```
new Vue({
  el: '#app',
  data: {
    message: 'Приветствую, ваше величество!'
  }
});
```

Вот и всё. Теперь наше сообщение заголовка и пользовательский ввод связаны! Используя **`v-model`** внутри тега **`input`** мы указываем Vue, какая переменная должна связана с элементом **`input`** (в данном случае это **`message`**).



Двусторонняя привязка данных

Двусторонняя привязка означает, что если вы измените значение модели в вашем представлении, все элементы, связанные с этой моделью, будут обновлены.

2.3 Сравнение с jQuery

Возможно, у вас есть определённый опыт работы с jQuery. Если нет, всё хорошо, поскольку использование jQuery в этой книге минимально. Когда мы используем эту библиотеку, то только для демонстрации того, как можно что-то сделать с Vue вместо jQuery, чтобы каждый это понял.

В любом случае, чтобы лучше понять, как привязка данных помогает нам создавать приложения, подумайте, как вы могли сделать предыдущий пример с помощью jQuery. Вероятно, вы создадите поле для ввода и установите ему **`id`** или **`class`**, чтобы вы могли выбрать его и соответствующим образом изменить. После этого вы вызовете функцию, которая изменяет нужный элемент, чтобы он соответствовал введённому значению, всякий раз, когда происходит *событие keyup*. **Это настоящая проблема**.

Более или менее ваш фрагмент кода будет выглядеть так.

```
<html>
<head>
  <title>Привет, Vue</title>
</head>
<body>
<div id="app">
  <h1>Приветствую, ваше величество!</h1>
  <input id="message">
</div>
</body>
<script src="https://code.jquery.com/jquery-2.1.4.min.js"></script>
<script type="text/javascript">
$('#message').on('keyup', function () {
  var message = $('#message').val();
  $('h1').text(message);
})
</script>
</html>
```

Это простой пример сравнения, и, как вы видите, реализация через Vue выглядит намного красивее, меньше по временным затратам и легче для понимания. Конечно, jQuery — мощная библиотека JavaScript для манипулирования объектной моделью документа (Document Object Model, DOM), но у всех есть свои плюсы и минусы!

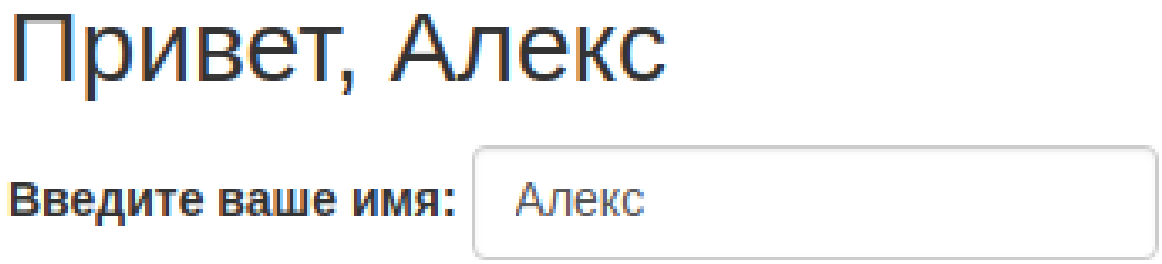
Примеры кода

Вы можете найти примеры кода этой главы на GitHub[2].

[2] https://github.com/hootlex/the-majesty-of-vuejs-2/blob/ru/codes/chapter2.html

2.4 Домашнее задание

Хорошим и очень простым вводным упражнением будет создание HTML-файла с заголовком “Привет, `{{name}}`” Добавьте поля ввода и привяжите его к переменной `name`. Как вы можете себе представить, заголовок должен сразу изменяться каждый раз, когда пользователь что-либо вводит или изменяет в поле ввода. Удачи и приятного времяпрепровождения!



Пример вывода

Примечание

В примере вывода используется Bootstrap. Если вы не знакомы с Bootstrap, вы можете игнорировать эту библиотеку сейчас, она описана в последней главе.

Возможное решение

Вы можете найти потенциальное решение этого упражнения здесь[3].

[3] https://github.com/hootlex/the-majesty-of-vuejs-2/blob/ru/homework/chapter2.html

3. Вкус директив

В этой главе мы рассмотрим некоторые основные примеры директив Vue. Итак, если вы ранее не использовали какие-либо фреймворки, такие как Vue.js или Angular.js, вы, вероятно, не знаете, что такое директива. По сути, директива представляет собой специальный токен (обозначение) в разметке, который указывает библиотеке, что делать с DOM-элементом. Во Vue.js концепция директивы значительно проще, чем в Angular. Некоторые директивы перечислены ниже:

- **`v-show`** используется для условного отображения элемента
- **`v-if`** может использоваться вместо **`v-show`**
- **`v-else`** отображает элемент, когда **`v-if`** вычисляет выражение как `false`.

Кроме того, существует **`v-for`**, который требует специального синтаксиса и предназначен для отображения (например, отображение списка элементов на основе массива). Мы расскажем об использовании каждой из этих директив позднее в этой книге.

Давайте начнём и рассмотрим упомянутые директивы.

3.1 v-show

Для демонстрации первой директивы давайте сделаем что-нибудь простое. Мы дадим вам несколько советов, которые намного облегчат понимание и работу! Предположим, вам нужно переключить отображение элемента на основе некоторого набора критериев. Возможно, кнопка не должна показываться, пока вы не напишите сообщение. Как мы можем это сделать с помощью Vue?

```
1  <html>
2  <head>
3    <title>Привет, Vue</title>
4  </head>
5  <body>
6  <div id="app">
7    <textarea></textarea>
8  </div>
9  </body>
10 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
11 <script>
```

```
12 new Vue({
13   el: '#app',
14   data: {
15     message: 'Наш король мёртв!'
16   }
17 })
18 </script>
19 </html>
```

У нас есть HTML-файл с уже знакомыми элементами **`div id="app"`** и **`textarea`**. Внутри элемента **`textarea`** мы собираемся отображать наше сообщение. Конечно, он ещё не привязан, и к этому моменту вы, возможно, уже поняли это. Также вы, может быть, обратите внимание, что мы больше не используем минифицированную версию Vue.js. Как упоминалось ранее, минифицированная версия не должна использоваться во время разработки, потому что тогда не будут отображаться предупреждения о распространённых ошибках. С этого момента мы собираемся использовать эту версию в книге, но, разумеется, вы можете использовать ту версию, которая нравится.

```
1  <html>
2  <head>
3    <title>Привет, Vue</title>
4  </head>
5  <body>
6  <div id="app">
7    <textarea v-model="message"></textarea>
8    <pre>{{ $data }}</pre>
9  </div>
10 </body>
11 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
12 <script>
13 new Vue({
14   el: '#app',
15   data: {
16     message: 'Наш король мёртв!'
17   }
18 })
19 </script>
20 </html>
```

Настало время привязать значение элемента **`textarea`** к переменной **`message`** с помощью **`v-model`**, чтобы отобразить наше сообщение. Всё, что мы вводим, изменится в режиме реального времени, как мы видели в примере из предыдущей главы, где использовали ввод. Кроме того, здесь используется тег **`pre`** для вывода данных. Что он сделает, так это возьмёт

данные из нашего экземпляра Vue, профильтрует их через **json**, и, наконец, отобразит данные в браузере. Vue будет красиво форматировать вывод автоматически для нас, будь то строка, число, массив или простой объект. Мы считаем, что это даёт гораздо лучший способ создания и манипулирования нашими данными, поскольку лучше видеть всё что происходит прямо перед вами, чем постоянно смотреть в консоль.

На заметку

JSON (JavaScript Object Notation) — простой и лёгкий формат обмена данными. Вы можете получить более подробную информацию о JSON здесь[1]. Вывод **{{ $data }}** связан с данными Vue и будет обновлён при каждом изменении.

```
1  <html>
2  <head>
3    <title>Привет, Vue</title>
4  </head>
5  <body>
6  <div id="app">
7    <h1>Вы должны написать сообщение для получения помощи!</h1>
8    <textarea v-model="message"></textarea>
9    <br>
10   <button type="submit" v-show="message">
11     Отправить послание всем союзникам для помощи!
12   </button>
13   <pre>{{ $data }}</pre>
14 </div>
15 </body>
16 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
17 <script>
18 new Vue({
19   el: '#app',
20   data: {
21         message: 'Наш король мёртв! Пришлите подмогу!'
22   }
23 })
24 </script>
25 </html>
```

Продолжая далее, теперь у нас есть простое предупреждение в теге **h1**, которое будет переключаться позже, основываясь на некотором критерии. Рядом с ним есть кнопка, которая будет отображаться в случае, если заполнено сообщение. Если элемент **textarea** пуст,

[1] https://www.json.org/json-ru.html

следовательно, данные отсутствуют, атрибут **`display`** кнопки устанавливается в значение 'none' и кнопка исчезает.

На заметку

Элемент с директивой **`v-show`** всегда будет отображаться и оставаться в DOM. **`v-show`** просто переключает CSS-свойство **`display`** элемента.

```
<h1 v-show="!message">Вы должны написать сообщение для получения помощи!</h1>
<textarea v-model="message"></textarea>
<button type="submit" v-show="message">
  Отправить послание всем союзникам для помощи!
</button>
```

То, чего мы хотим достичь в этом примере, заключается в переключении различных элементов. На данном этапе нам нужно скрыть предупреждение внутри тега **`h1`**, **если** сообщение написано. В противном случае скрыть кнопку, установив атрибут **`style`** в **`display: none`**.

3.2 v-if

Сейчас вы можете спросить «А как насчёт директивы v-if, которую мы упоминали ранее?». Что ж, мы ещё раз сделаем предыдущий пример, только на этот раз воспользуемся **`v-if`**!

```
<html>
<head>
  <title>Привет, Vue</title>
</head>
<body>
<div id="app">
  <h1 v-if="!message">Вы должны написать сообщение для получения помощи!</h1>
  <textarea v-model="message"></textarea>
  <button type="submit" v-if="message">
      Отправить послание всем союзникам для помощи!
  </button>
  <pre>{{ $data }}</pre>
</div>
</body>
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
<script>
new Vue({
  el: '#app',
```

```
19    data: {
20      message: 'Наш король мёртв! Пришлите подмогу!'
21    }
22  })
23  </script>
24  </html>
```

Как видно, замена **v-show** на **v-if** работает так же хорошо, как мы предполагали. Продолжите работу и попробуйте поэкспериментировать, чтобы увидеть, как это работает! Единственное различие состоит в том, что элемент с **v-if** не остаётся в DOM.

3.2.1 v-if с элементом template

Если когда-нибудь мы окажемся в ситуации, когда захотим одновременно переключить существование нескольких элементов, можем воспользоваться **v-if** в элементе **<template>**. В случаях, когда использование **div** или **span** не представляется возможным, элемент **<template>** может послужить невидимой оболочкой. Элемент **<template>** не будет отображаться в конечном результате.

```
1   <div id="app">
2     <template v-if="!message">
3       <h1>Вы должны написать сообщение для получения помощи!</h1>
4       <p>Немедленно отправьте посыльного!</p>
5       <p>В соседнее королевство Сердец!</p>
6     </template>
7     <textarea v-model="message"></textarea>
8     <br>
9     <button type="submit" v-show="message">
10      Отправить послание всем союзникам для помощи!
11    </button>
12    <pre>
13      {{ $data }}
14    </pre>
15  </div>
```

Директива v-if в элементе template

Используя код из предыдущего примера, мы привязали директиву **`v-if`** к элементу **`template`**, переключая существование всех вложенных элементов.

Предупреждение

Директива **`v-show`** не поддерживает синтаксис **`<template>`**.

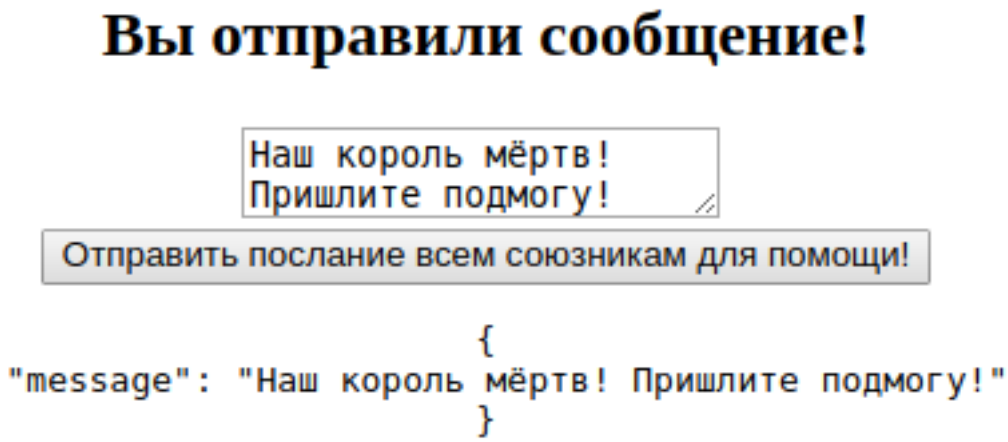
3.3 v-else

При использовании **`v-if`** вы можете использовать директиву **`v-else`** для указания «блока else», как вы уже себе могли представить. Имейте в виду, что директива **`v-else`** должна следовать непосредственно за директивой **`v-if`**, иначе она не будет распознана.

```
1  <html>
2  <head>
3    <title>Привет, Vue</title>
4  </head>
5  <body>
6  <div id="app">
7    <h1 v-if="!message">Вы должны написать сообщение для получения помощи!</h1>
8    <h2 v-else>Вы отправили сообщение!</h2>
9    <textarea v-model="message"></textarea>
10   <br>
11   <button type="submit" v-show="message">
12       Отправить послание всем союзникам для помощи!
13   </button>
14   <pre>{{ $data }}</pre>
15 </div>
16 </body>
17 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
18 <script>
19 new Vue({
```

```
20      el: '#app',
21      data: {
22        message: 'Наш король мёртв! Пришлите подмогу!'
23      }
24  })
25  </script>
26  </html>
```

v-if в действии



v-else в действии

Просто для примера мы использовали тег `h2` с другим предупреждением, чем раньше, который отображается по условию. Если сообщение не предоставлено, мы видим тег `h1`. Если оно есть, мы видим `h2`, используя этот очень простой синтаксис Vue `v-if` и `v-else`. Просто как дважды два!

Предупреждение

Начиная с Vue 2.0, директива `v-show` больше не работает с `v-else`.

3.4 v-if по сравнению с v-show

Хотя мы уже упоминали о различиях между `v-if` и `v-show` , теперь можем немного углубится. Некоторые вопросы могут возникнуть из-за их использования. Есть ли большая разница между использованием `v-show` и `v-if`? Есть ли ситуация, где страдает производительность? Когда лучше использовать ту или иную директиву? Вы можете столкнуться с тем, что использование `v-show` во многих ситуациях приводит к увеличению времени загрузки при отображении страницы. Для сравнения, `v-if` действительно условный в соответствии с руководством Vue.js.

> *При использовании `v-if`, если условие ложно при изначальном отображении, ничего не произойдёт — условной блок не будет отображаться до тех пор, пока условие не станет истинным в первый раз. Вообще говоря, `v-if` имеет большую стоимость переключения, тогда как `v-show` имеет более высокие затраты на отображение. Поэтому предпочитайте использование `v-show`, если ваши элементы будут очень часто переключаться, и используйте `v-if`, если условие вряд ли изменится во время выполнения.*

Итак, когда что использовать на самом деле зависит от ваших потребностей.

Примеры кода

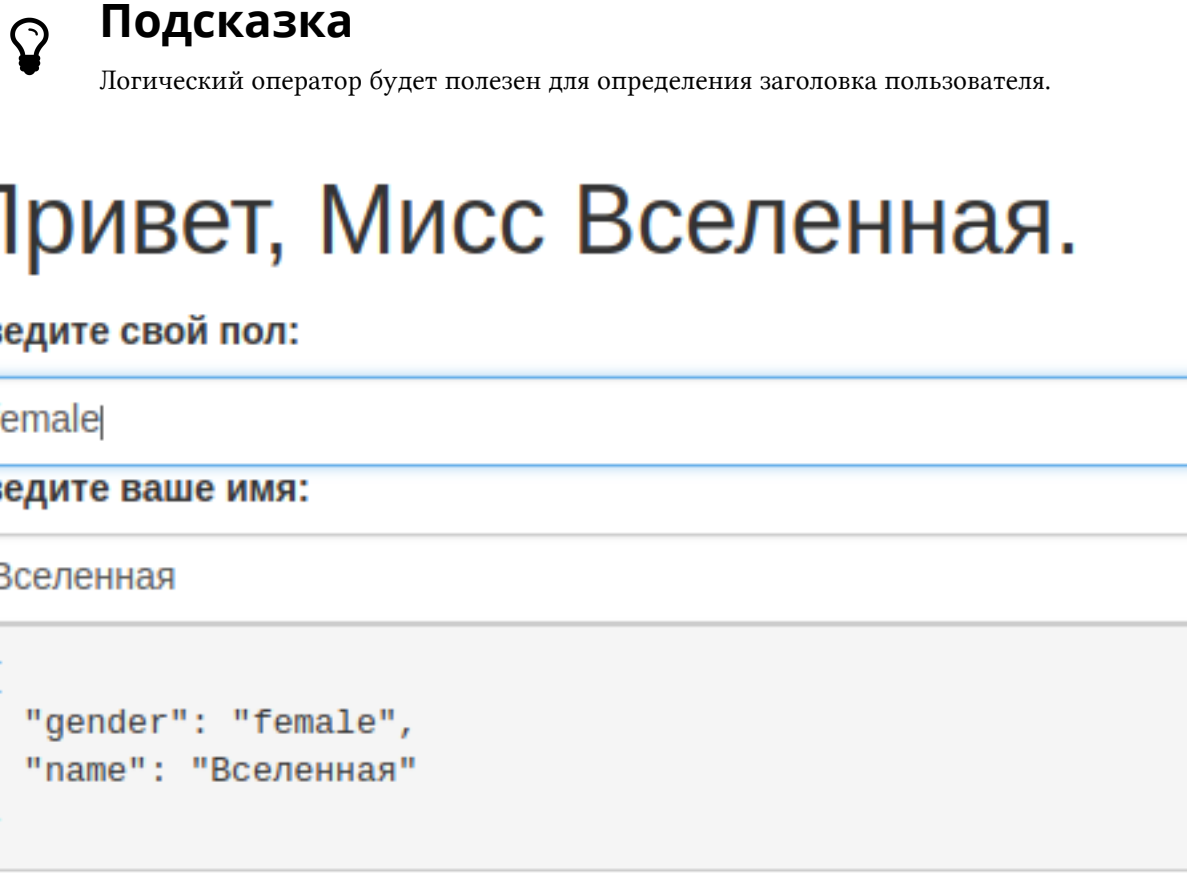
Вы можете найти примеры кода этой главы на GitHub[2].

[2] https://github.com/hootlex/the-majesty-of-vuejs-2/tree/ru/codes/chapter3

3.5 Домашняя работа

Следуя предыдущему уроку домашней работы, вы должны попытаться немного расширить его. Теперь пользователь вводит свой пол или имя. Если пользователь — мужчина, то заголовок приветствует пользователя как “**Привет, мистер {{name}}**”. Если пользователь — женщина, то тогда вместо этого отображается “**Привет, мисс {{name}}**”.

Если пол пользователя ни мужчина, ни женщина, то пользователю отображается заголовок предупреждения с содержимым “**Так, значит, вы не можете решить. Отлично!**”.



Пример вывода

Подсказка

Логический оператор будет полезен для определения заголовка пользователя.

Возможное решение

Вы можете найти потенциальное решение к этому упражнению здесь[3].

[3] https://github.com/hootlex/the-majesty-of-vuejs-2/blob/ru/homework/chapter3.html

4. Отрисовка списков

В четвёртой главе данной книги мы узнаем об отрисовке списков. Используя директивы Vue, мы собираемся продемонстрировать:

1. Отображение списка элементов на основе массива
2. Повторение шаблона
3. Итерация по свойствам объекта для вывода данных

4.1 Установка и использование Bootstrap

Чтобы сделать нашу работу простой и привлекательной для глаз, мы собираемся импортировать Bootstrap.

На заметку

Bootstrap — самый популярный фреймворк для разработки адаптивных проектов с использованием HTML, CSS и JS, реализующий принцип mobile first, т.е. разработка веб-приложений, ориентированных, прежде всего, на мобильные устройства.

Перейдите на https://getbootstrap.com/[1] и нажмите на кнопку загрузки. На данный момент мы просто используем Bootstrap из CDN[2], но вы можете установить его любым удобным для вас способом. Для нашего примера нам нужен только один файл, сейчас это `css/bootstrap.min.css`. При использовании этого файла `.css` в приложении у нас есть доступ ко всем красивым стилям. Просто подключите его внутри тега `head` страницы и мы будем готовы двигаться дальше.

Bootstrap требует наличия содержащего элемента, который оборачивает содержимое сайта и где помимо этого находится наша сетка. Вы можете выбрать один из двух контейнеров для использования в ваших проектах. Обратите внимание, что из-за свойства `padding` и других, ни один контейнер не может быть вложен в другой.

- Используйте `.container` для адаптивного контейнера с фиксированной шириной.

[1] https://getbootstrap.com/
[2] https://www.bootstrapcdn.com/

```
1   <div class="container">
2     <!-- ... -->
3   </div>
```

- Используйте **.container-fluid** для использования контейнера на всю ширину (охватывающего всю ширину вашего вьюпорта)

```
1   <div class="container-fluid">
2     <!-- ... -->
3   </div>
```

На данном этапе мы хотим привести пример использования Vue.js с классами Bootstrap. Конечно, не требуется много экспериментов или детального изучения для того, чтобы использовать вместе Vue и Bootstrap.

```
1  <html>
2  <head>
3    <link
4      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
5      rel="stylesheet"
6    >
7    <title>Привет, Bootstrap</title>
8  </head>
9  <body>
10   <div class="container">
11     <h1>Привет, Bootstrap, сядь рядом с Vue.</h1>
12   </div>
13 </body>
14 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
15 <script type="text/javascript">
16 new Vue({
17   el: '.container'
18 })
19 </script>
20 </html>
```

Обратите внимание, что вместо ранее используемой точки монтирования **app**, теперь мы использовали класс **container** внутри свойства **el** экземпляра Vue.

Совет

В приведённом выше примере мы использовали элемент с классом `.container`. **Будьте осторожны**, когда вы хотите использовать элемент по классу для инициализации Vue, который будет присутствовать в HTML-разметке более одного раза, Vue.js будет примонтирован только к **первому** элементу.

Свойство `el` может быть селектором CSS или фактическим объектом типа HTMLElement. *Обратите внимание: нельзя монтировать основной экземпляр Vue к* `<html>` *или* `<body>`*!*

4.2 v-for

Чтобы обойти каждый элемент в массиве, мы будем использовать директиву **`v-for`**.

Эта директива требует специального синтаксиса в форме **`item in array`**, где **`array`** — исходный массив данных, а **`item`** — псевдоним (переменная) для текущего итерируемого элемента массива.

Предупреждение

Если вы пришли из мира PHP, вы можете заметить, что директива **`v-for`** похожа на PHP-функцию **`foreach`**. Но **будьте внимательны**, если вы привыкли к **`foreach ($array as $value)`**.

Цикл Vue **`v-for`** — полная противоположность, т.е. **`value in array`**.

Первым идёт итерируемый элемент массива, а далее за ним сам массив с данными.

4.2.1 v-for с диапазоном чисел

Директива **`v-for`** может принимать целое число. Всякий раз, когда число передаётся вместо массива или объекта, шаблон повторяется столько же раз, сколько указанное число.

```
1  <html>
2  <head>
3    <link
4      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
5      rel="stylesheet"
6    >
7    <title>Привет, Vue</title>
8  </head>
9  <body>
10   <div class="container">
11     <h1>Таблица умножения на 4</h1>
12     <ul class="list-group">
13       <li v-for="i in 11" class="list-group-item">
14         {{ i }} умножить на 4 равно {{ i * 4 }}.
15       </li>
16     </ul>
17   </div>
18 </body>
19 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
20 <script type="text/javascript">
```

```
21    new Vue({
22      el: '.container'
23    })
24  </script>
25  </html>
```

В приведённом выше примере отображается таблица умножения на 4.

Таблица умножения на 4

1 умножить на 4 равно 4.

2 умножить на 4 равно 8.

3 умножить на 4 равно 12.

4 умножить на 4 равно 16.

5 умножить на 4 равно 20.

6 умножить на 4 равно 24.

7 умножить на 4 равно 28.

8 умножить на 4 равно 32.

9 умножить на 4 равно 36.

10 умножить на 4 равно 40.

11 умножить на 4 равно 44.

Таблица умножения на 4

Примечание

Поскольку мы хотим отобразить всю таблицу умножения на 4 (до 40), мы повторяем шаблон 11 раз так как первое значение `i` принимает значение `1`.

4.3 Отрисовка массива

4.3.1 Обход элементов массива

В следующем примере мы создадим следующий массив историй (stories) внутри нашего объекта данных, и мы отобразим их все, один за другим.

```
stories: [
  'Сегодня я разбил машину!',
  'Вчера кто-то украл мою сумку!',
  'Кто-то съел мой шоколад...',
]
```

Что нам нужно сделать, так это отобразить список. В частности, массив строк.

```
1  <html>
2  <head>
3    <link
4      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
5      rel="stylesheet"
6    >
7    <title>Истории</title>
8  </head>
9  <body>
10   <div class="container">
11     <h1>Давайте послушаем несколько историй!</h1>
12     <div>
13       <ul class="list-group">
14         <li v-for="(story, index) in stories" :key="index" class="list-group-item">
15           Кто-то сказал "{{ story }}"
16         </li>
17       </ul>
18     </div>
19     <pre>{{ $data }}</pre>
20   </div>
21 </body>
22 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
23 <script type="text/javascript">
24     new Vue({
25       el: '.container',
26       data: {
27         stories: [
```

```
28            'Сегодня я разбил машину!',
29            'Вчера кто-то украл мою сумку!',
30            'Кто-то съел мой шоколад...',
31          ]
32        }
33      })
34  </script>
35  </html>
```

Атрибут `key`

По умолчанию используется стратегия «обновления на месте». Если порядок элементов массива или объекта изменился, Vue не будет перемещать элементы DOM, а попросту обновит каждый элемент «на месте», чтобы он отображал новые W> данные по соответствующему индексу. Для достижения этого поведения Vue нужно указать уникальность элемента при итерации с помощью атрибута `key`.

Рекомендуется всегда указывать этот атрибут с уникальным значением (например, идентификатор сущности, либо просто индекс), за исключением случаев, когда итерируемый DOM-контент простой или когда вы сознательно используете стратегию по умолчанию для улучшения производительности.

На заметку

Оба класса, **`list-group`** и **`list-group-item`**, — это классы Bootstrap. Здесь вы можете найти более подробную информацию о стилизации списков[3].

[3] https://getbootstrap.com/css/#type-lists

Давайте послушаем несколько историй!

Кто-то сказал "Сегодня я разбил машину!"

Кто-то сказал "Вчера кто-то украл мою сумку!"

Кто-то сказал "Кто-то съел мой шоколад..."

```
{
  "stories": [
    "Сегодня я разбил машину!",
    "Вчера кто-то украл мою сумку!",
    "Кто-то съел мой шоколад..."
  ]
}
```

Отображение массива, используя v-for

С использованием `v-for` мы сумели отобразить ваши истории в простом неупорядоченном списке. **Это действительно очень просто!**

4.3.2 Обход массива объектов

Теперь мы изменим массив историй, чтобы он содержал объекты `story` (массив с историями). Объект `story` имеет два свойства: `plot` (сюжет) и `writer` (автор). Мы сделаем то же самое, что и раньше, но на этот раз вместо немедленного вывода `story` мы будем отображать `story.plot` и `story.writer` соответственно.

```
1  <html>
2  <head>
3    <link
4      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
5      rel="stylesheet"
6    >
7    <title>Истории</title>
8  </head>
9  <body>
10     <div class="container">
11       <h1>Давайте послушаем несколько историй!</h1>
12       <div>
13         <ul class="list-group">
14           <li
```

```
15            v-for="(story, index) in stories"
16            :key="index"
17            class="list-group-item"
18          >
19            {{ story.writer }} сказал "{{ story.plot }}"
20          </li>
21        </ul>
22      </div>
23      <pre>{{ $data }}</pre>
24    </div>
25 </body>
26 <script src="hhttps://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
27 <script type="text/javascript">
28 new Vue({
29   el: '.container',
30   data: {
31     stories: [
32       {
33         plot: 'Сегодня я разбил машину!',
34         writer: 'Алекс'
35       },
36       {
37         plot: 'Вчера кто-то украл мою сумку!',
38         writer: 'Джон'
39       },
40       {
41         plot: 'Кто-то съел мой шоколад...',
42         writer: 'Джон'
43       },
44       {
45         plot: 'Я съел чей-то шоколад!',
46         writer: 'Алекс'
47       },
48     ]
49   }
50 })
51 </script>
52 </html>
```

Кроме того, когда вам нужно отобразить индекс текущего элемента, вы можете использовать специальную переменную **index**. Он работает следующим образом:

```
<ul class="list-group">
    <li
      v-for="(story, index) in stories"
      :key="index"
      class="list-group-item"
    >
        {{index}} {{ story.writer }} сказал "{{ story.plot }}"
    </li>
</ul>
```

index внутри фигурных скобок, очевидно, представляет индекс итерируемого элемента в данном примере.

Давайте послушаем несколько историй!

0. Алекс сказал "Сегодня я разбил машину!"
1. Джон сказал "Вчера кто-то украл мою сумку!"
2. Джон сказал "Кто-то съел мой шоколад..."
3. Алекс сказал "Я съел чей-то шоколад!"

Отображённый массив с индексом

4.4 Объект v-for

Вы можете использовать **v-for** для итерации по свойствам объекта. Ранее мы упоминали, что вы можете отобразить индекс (**index**) массива, но вы также можете сделать то же самое при итерации объекта. В дополнение к индексу (**index**), каждая область видимости будет иметь доступ к другому специальному свойству, **key** (ключ).

На заметку

При итерации по объекту индекс (**index**) находится в диапазоне **0** ... **n-1**, где **n** — количество свойств объекта.

На этот раз мы изменили структуру наших данных в виде единственного объекта с тремя атрибутами: **plot**, **writer** и **upvotes**.

```
<div class="container">
  <h1>Давайте послушаем несколько историй!</h1>
  <ul class="list-group">
    <li v-for="(value, index) in story" :key="index" class="list-group-item">
      {{ value }}
    </li>
  </ul>
</div>
```

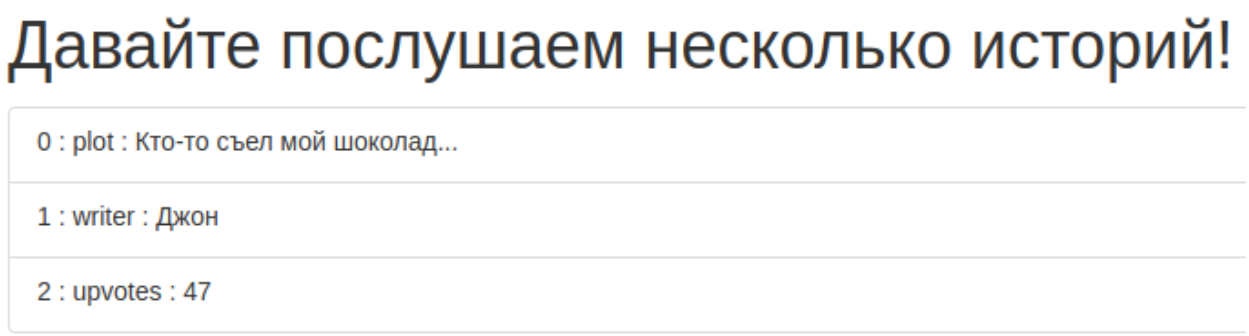
```
new Vue({
  el: '.container',
  data: {
    story: {
      plot: 'Кто-то съел мой шоколад...',
      writer: 'John',
      upvotes: 47
    }
  }
})
```

Мы можем предоставить второй и третий аргумент для **key** и **index** соответственно.

```
<div class="container">
  <h1>Давайте послушаем несколько историй!<</h1>
  <ul class="list-group">
    <li
      v-for="(value, key, index) in story"
      :key="index"
      class="list-group-item"
    >
      {{index}} : {{key}} : {{value}}
    </li>
  </ul>
</div>
```

Как вы можете видеть в приведённом выше примере кода, мы используем **`key`** и **`index`** внутри списка, пары ключ-значение, а также **`index`** для каждой пары.

Результатом будет:



Итерация по свойствам объекта

Примеры кода

Вы можете найти примеры кода для этой главы на GitHub[4].

[4] https://github.com/hootlex/the-majesty-of-vuejs-2/tree/ru/codes/chapter4

4.5 Домашняя работа

Учитывая, что мы рассмотрели в этой главе, в качестве домашней работы создайте объект с атрибутами ваших личных качеств. Под личными качествами я имею в виду имя (`name`), вес (`weight`), рост (`height`), цвет глаз (`eyeColor`) и любимый цвет (`favoriteFood`).

Используйте **`v-for`** для обхода каждого свойства и отобразите в следующем формате: **`index: key = value`**.

Пример вывода

Возможное решение

Вы можете найти потенциальное решение этого упражнения здесь[5].

[5] https://github.com/hootlex/the-majesty-of-vuejs-2/blob/ru/homework/chapter4.html

5. Интерактивность

В этой главе мы собираемся создавать и расширять предыдущие примеры, изучать что-то новое, касающееся методов, обработки событий и вычисляемых свойств. Мы разработаем несколько примеров, используя разные подходы. Пришло время посмотреть, как мы можем реализовать интерактивность Vue, чтобы создать небольшое приложение, такое как калькулятор, работающее красиво и просто.

5.1 Обработка событий

События `HTML` — это то, что происходит с DOM-элементами. Когда Vue.js используется на страницах `HTML`, он может **реагировать** на эти события.

События могут представлять всё, от базовых пользовательских взаимодействий до того, что происходит при формировании отрисовки.

Вот некоторые примеры события `HTML`:

- Веб-страница полностью загрузилась
- Поле ввода было изменено
- Кнопка была нажата
- Форма была отправлена

Смысл обработки событий — это то, что вы можете сделать, когда происходит событие.

Во Vue.js для **обработки** DOM-событий вы можете использовать директиву `v-on`.

Директива `v-on` прикрепляет обработчик события к элементу. Тип события обозначается аргументом, например, `v-on:keyup` обрабатывает событие `keyup`.

На заметку

Событие `keyup` происходит, когда пользователь отпускает клавишу. Полный список событий `HTML` вы можете найти здесь[1].

5.1.1 Встроенная обработка событий

Достаточно слов, давайте перейдём к делу и посмотрим на обработку событий в действии. Ниже находится кнопка «Проголосовать!», которая увеличивает количество голосов при каждом нажатии.

[1] https://www.w3schools.com/tags/ref_eventattributes.asp

```
<html>
<head>
  <link
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
    rel="stylesheet"
  >
  <title>Проголосовать</title>
</head>
<body>
  <div class="container">
    <button type="button" v-on:click="upvotes++">
      Проголосовать! {{upvotes}}
    </button>
  </div>
</body>
<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.1\
7/vue.js"></script>
<script type="text/javascript">
new Vue({
  el: '.container',
  data: {
    upvotes: 0
  }
})
</script>
</html>
```

Счётчик голосов

Внутри наших данных есть свойство `upvotes`. В этом случае мы привязываем обработчик событий нажатия (`click`) с JavaScript-выражением, которое находится рядом с ним. Внутри кавычек всякий раз, когда нажимается кнопка; мы простом увеличиваем счётчик голосов на один, используя оператор инкремента (`upvotes++`).

5.1.2 Обработка событий с использованием методов

Теперь мы будем то же самое, что и в предыдущем примере, но с одним отличием — использовать метод. Метод во Vue.js — это блок кода, предназначенный для выполнения конкретной задачи. Для выполнения метода, вы должны определить и вызвать его.

```
1  <html>
2  <head>
3    <link
4      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
5      rel="stylesheet"
6    >
7  <title>Проголосовать</title>
8  </head>
9  <body>
10   <div class="container">
11     <button type="button" v-on:click="upvote">
12       Проголосовать! {{upvotes}}
13     </button>
14   </div>
15 </body>
16 <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.1\
17 7/vue.js"></script>
18 <script type="text/javascript">
19 new Vue({
20   el: '.container',
21   data: {
22     upvotes: 0
23   },
24   // определить методы в объекте **`methods`**
25   methods: {
26     upvote: function () {
27       // **`this`** внутри методов указывает на Vue-экземпляр
28       this.upvotes++;
29     }
30   }
31 })
32 </script>
33 </html>
```

Мы привязываем обработчик события нажатия к методу **`upvote`**. Это работает так же, как и раньше, но выглядит более чистым и понятным при чтении кода.

Предупреждение

Обработчики событий ограничиваются выполнением **только одного выражения**.

5.1.3 Сокращение для `v-on`

Когда вы всё время используете `v-on` в проекте, то обнаружите что ваша HTML-разметка быстро станет грязной. К счастью, есть сокращение для `v-on` — символ `@`. Символ `@` заменяет весь `v-on:` и при его использовании код выглядит *намного чище*. Использование сокращения абсолютно необязательно.

С использованием `@` кнопка из нашего предыдущего примера будет выглядеть так, сравните сами:

Обработка события 'click', используя `v-on`:

```
<button type="button" v-on:click="upvote">
  Проголосовать! {{upvotes}}
</button>
```

Обработка события 'click', используя сокращение `@`

```
<button type="button" @click="upvote">
  Проголосовать! {{upvotes}}
</button>
```

5.2 Модификаторы событий

Теперь мы перейдём к созданию приложения калькулятора. Для этого мы будем использовать форму с двумя полями ввода и одним выпадающим списком для выбора желаемой операции.

Несмотря на то, что следующий код выглядит прекрасным, наш калькулятор работает не так, как ожидалось.

```
1  <html>
2  <head>
3    <title>Калькулятор</title>
4    <link
5      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
6      rel="stylesheet"
7    >
8  </head>
9  <body>
10   <div class="container">
11     <h1>Введите 2 числа и выберите операцию.</h1>
12     <form class="form-inline">
13       <!-- Обратите внимание на используемый модификатор 'number'
14       который анализирует входные данные в качестве чисел. -->
15       <input v-model.number="a" class="form-control">
16       <select v-model="operator" class="form-control">
17         <option>+</option>
18         <option>-</option>
19         <option>*</option>
20         <option>/</option>
21       </select>
22       <!-- Обратите внимание на используемый модификатор 'number'
23       который анализирует входные данные в качестве чисел. -->
24       <input v-model.number="b" class="form-control">
25       <button
26         type="submit"
27         @click="calculate"
28         class="btn btn-primary"
29       >
30         Вычислить
31       </button>
32     </form>
33     <h2>Результат: {{a}}  {{operator}}  {{b}} = {{c}}</h2>
34     <pre>{{ $data }}</pre>
35 </div>
36 </body>
37 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
38 <script type="text/javascript">
39   new Vue({
40     el: '.container',
41     data: {
42       a: 1,
43       b: 2,
```

```
44        c: null,
45        operator: '+',
46    },
47    methods: {
48      calculate: function () {
49        switch (this.operator) {
50          case '+':
51            this.c = this.a + this.b
52            break;
53          case '-':
54            this.c = this.a - this.b
55            break;
56          case '*':
57            this.c = this.a * this.b
58            break;
59          case '/':
60            this.c = this.a / this.b
61            break;
62        }
63      }
64  },
65  });
66  </script>
67  </html>
```

Если вы запустите данный код самостоятельно, то обнаружите, что при нажатии на кнопку «Вычислить», вместо вычисления произойдёт перезагрузка страницы.

Причина такого поведения ясна, потому что когда вы нажимаете на кнопку «Вычислить», то в фоновом режиме отправляете форму, и, таким образом, страница перезагружается.

Для предотвращения отправки формы, мы должны отменить действие по умолчанию для события **`onsubmit`**. В большинстве случаев вам нужно вызвать **`event.preventDefault()`** внутри нашего метода обработки. В нашем случае метод обработки событий называется **`calculate`**.

Итак, наш метод примет следующий вид:

```
calculate: function (event) {
  event.preventDefault();
  switch (this.operator) {
    case '+':
      this.c = this.a + this.b
      break;
    case '-':
      this.c = this.a - this.b
      break;
    case '*':
      this.c = this.a * this.b
      break;
    case '/':
      this.c = this.a / this.b
      break;
  }
}
```

Введите 2 числа и выберите операцию.

200 / 4

Результат: 200 / 4 = 50

```
{
  "a": 200,
  "b": 4,
  "operator": "/"
}
```

Использование модификаторов событий для создания калькулятора

Хотя мы можем сделать это легко внутри методов, было бы лучше, если эти методы не имели дело с деталями событий DOM.

Vue.js предоставляет четыре модификатора `v-on` для предотвращения поведение по умолчанию события:

1. `.prevent`
2. `.stop`
3. `.capture`
4. `.self`

Итак, используя `.prevent`, наша кнопка отправки изменится с:

```
<button type="submit" @click="calculate">Вычислить</button>
```

на:

```
<!-- событие отправки больше не будет перезагружать страницу -->
<button type="submit" @click.prevent="calculate">Вычислить</button>
```

Теперь мы можем безопасно удалить **`event.preventDefault()`** из метода **`calculate`**.

Примечание

`.capture` и **`.self`** редко используются, поэтому мы больше не будем более подробно касаться этой темы в дальнейшем. Если вам интересно больше узнать о *порядке выполнения событий*, посмотрите на эту обучающую статью[2].

5.3 Модификаторы клавиш

При фокусе на одном из полей ввода и нажатии на кнопку ввода (enter), вы заметите, что вызывается метод вычисления. Если кнопка была не внутри формы или вообще не было подобной кнопки, вы могли бы обработать событие клавиатуры вместо этого.

При обработке событий клавиатуры нам часто потребуется проверять коды клавиш. Код кнопки **`Enter`** — 13. Таким образом, мы можем поступить следующим образом для вызова метода вычисления при нажатии кнопки ввода:

```
<input v-model="a" @keyup.13="calculate">
```

Учитывая, что все коды клавиш сложно запоминать, Vue предоставляет псевдонимы для наиболее используемых клавиш:

- enter
- tab
- delete
- esc
- space
- up
- down
- left
- right

Итак, для выполнения метода `calculate` при нажатии на кнопку ввода в нашем примере, поля ввода будут выглядеть так:

[2] https://www.quirksmode.org/js/events_order.html

```
<input v-model="a" @keyup.enter="calculate">
<input v-model="b" @keyup.enter="calculate">
```

Подсказка

Когда у вас есть форма с большим количеством полей вводов, кнопок и т.д. и вам нужно предотвратить событие отправки формы по умолчанию, вы можете применить модификатор **`submit`** к событию формы.

Например: **`<form @submit.prevent="calculate">`**

Наконец, наш калькулятор готов к работе.

5.4 Вычисляемые свойства

Встроенные выражения Vue.js очень удобны, но для более сложной логики, вы должны использовать вычисляемые свойства. Практически, вычисляемые свойства — это переменные, значение которых зависит от других факторов.

Вычисляемые свойства работают как функции, которые можно использовать в качестве свойств. Но есть существенная разница. Каждый раз, когда изменяется зависимость вычисляемого свойства, значение вычисляемого свойства переоценивается.

Во Vue.js вы определяете вычисляемые свойства внутри объекта **`computed`** экземпляра **`Vue`**.

```
<html>
<head>
  <link
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
    rel="stylesheet"
  >
<title>Привет, Vue</title>
</head>
<body>
<div class="container">
    a={{ a }}, b={{ b }}
   <pre>{{ $data }}</pre>
</div>
</body>
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
<script type="text/javascript">
new Vue({
  el: '.container',
```

```
19    data: {
20      a: 1,
21    },
22    computed: {
23      // вычисляемый геттер
24      b: function () {
25        // **`this`** указывает на Vue-экземпляр
26        return this.a + 1
27      }
28    }
29  });
30  </script>
31  </html>
```

Мы установили две переменные: первая, **a**, установлена в 1, а вторая, **b**, будет установлена возвращаемым значением функции внутри объекта вычисляемых свойств. В этом примере значение **b** будет установлено в 2.

```
1   <html>
2   <head>
3     <link
4       href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
5       rel="stylesheet"
6     >
7     <title>Привет, Vue</title>
8   </head>
9   <body>
10  <div class="container">
11    a={{ a }}, b={{ b }}
12    <input v-model="a">
13    <pre>{{ $data }}</pre>
14  </div>
15  </body>
16  <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
17  <script type="text/javascript">
18  new Vue({
19    el: '.container',
20    data: {
21      a: 1,
22    },
23    computed: {
24      // вычисляемый геттер
25      b: function () {
```

```
26        // **`this`** указывает на Vue-экземпляр
27        return this.a + 1
28      }
29    }
30  });
31  </script>
32  </html>
```

Вышеприведённый пример такой же, как предыдущий, но с одним отличием. Поле ввода связано с переменной `a`. Желательным результатом было бы изменить значение привязанного атрибута и немедленно обновить результат `b`. Но обратите внимание, что он не работает, как мы ожидали.

Если вы запустите данный код и установите переменную `a` на **5**, вы ожидаете, что `b` будет равно **6**. Конечно, это не так, `b` устанавливается значением **51**.

Почему это происходит? Ну, как вы уже могли догадаться, `b` принимает заданное значение из поле ввода `a` в виде строки и добавляет к нему **1** в конец.

Одним из возможных решений — это использовать функцию `parseFloat()`, **которая обрабатывает строку и возвращает число с плавающей точкой**.

```
new Vue({
  el: '.container',
  data: {
    a: 1,
  },
  computed: {
    b: function () {
      return parseFloat(this.a) + 1
    }
  }
});
```

Другой вариант, который приходит на ум, - использовать `<input type="number">`, как раз для полей ввода, предназначенных содержать числовое значение.

Но есть более аккуратный способ. С Vue.js, когда вы хотите, чтобы пользовательское поле ввода автоматически содержало число, вы можете добавить специальный модификатор `.number`.

```
<body>
<div class="container">
  a={{ a }}, b={{ b }}
  <input v-model.number="a">
  <pre>
    {{ $data }}
  </pre>
</div>
</body>
```

Модификатор **number** даст нам желаемый результат без каких-либо дополнительных усилий.

Для демонстрации более широкого представления о вычисляемых свойств, мы будем использовать их в калькуляторе, показанном ранее, но на этот раз используя вычисляемые свойства вместо методов.

Давайте начнём с простого примера, где вычисляемое свойство **c** содержит сумму **a** и **b**.

```
1  <html>
2  <head>
3    <link
4      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
5      rel="stylesheet"
6    >
7    <title>Привет, Vue</title>
8  </head>
9  <body>
10     <div class="container">
11       <h1>Введите 2 числа для вычисления их суммы.</h1>
12       <form class="form-inline">
13         <input v-model.number="a" class="form-control">
14         +
15         <input v-model.number="b" class="form-control">
16       </form>
17       <h2>Результат: {{a}} + {{b}} = {{c}}</h2>
18       <pre> {{ $data }} </pre>
19     </div>
20 </body>
21 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
22 <script type="text/javascript">
23 new Vue({
24   el: '.container',
25   data: {
26     a: 1,
```

```
27      b: 2
28    },
29    computed: {
30      c: function () {
31        return this.a + this.b
32      }
33    }
34  });
35  </script>
36  </html>
```

Начальный код готов, и в этот момент пользователь может ввести 2 числа и получить их сумму. Цель калькулятора — выполнение четырёх основных операций, поэтому давайте приступим к работе!

Поскольку код `HTML` будет таким же, как в калькуляторе, который мы создавали в предыдущем разделе этой главы (за исключением того, что сейчас нам не потребуется кнопка), я собираюсь показать только блок кода с JavaScript.

```
1   new Vue({
2     el: '.container',
3     data: {
4       a: 1,
5       b: 2,
6       operator: '+',
7     },
8     computed: {
9       c: function () {
10        switch (this.operator) {
11          case '+':
12            return this.a + this.b
13          case '-':
14            return this.a - this.b
15          case '*':
16            return this.a * this.b
17          case '/':
18            return this.a / this.b
19        }
20      }
21    },
22  });
```

Калькулятор готов к использованию. Единственное, что мы должны сделать, - так это переместить всё, что было в методе **`calculate`** в вычисляемое свойство **`c`**! Всякий раз, когда

вы изменяете `a` или `b`, результаты обновляются в режиме реального времени! Нам не нужны никакие кнопки, события или что-то ещё. **Разве это не удивительно??**

Примечание

Обратите внимание, что обычный подход состоял бы в том, чтобы иметь выражение `if` для предотвращения ошибок деления. Но на случай такой ошибки у нас есть сценарий использования. Если пользователь вводит 1/0, результат автоматически становится бесконечным! Если пользователь вводит текст, в качестве результате отображается “не число”.

Введите 2 числа и выберите операцию.

125 - 25.1

Результат: 125 - 25.1 = 99.9

```
{
  "a": 125,
  "b": 25.1,
  "operator": "-"
}
```

Калькулятор, созданный с использованием вычисляемых свойств

Примеры кода

Вы можете найти примеры кода этой главы на GitHub[3].

[3] https://github.com/hootlex/the-majesty-of-vuejs-2/tree/ru/codes/chapter5

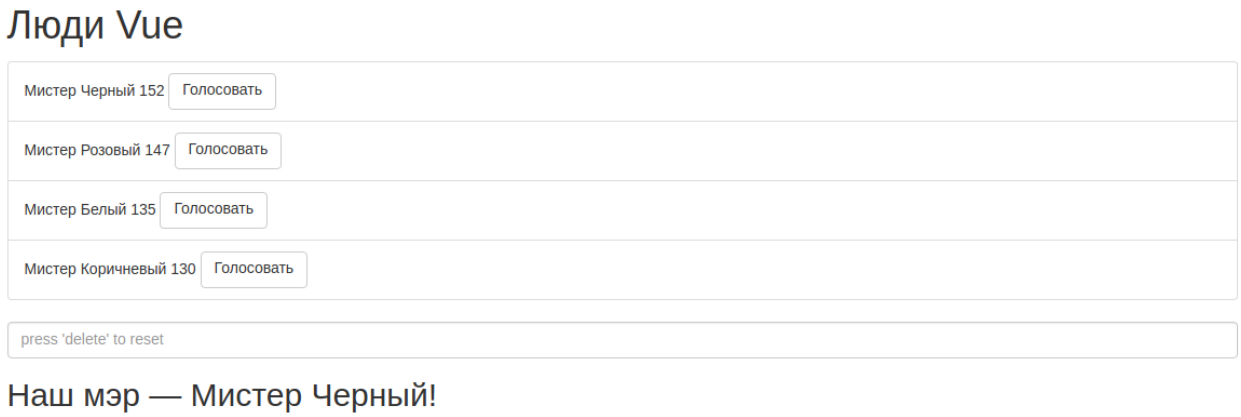
5.5 Домашняя работа

Теперь, когда у вас есть базовое понимание обработки событий, методов, вычисляемых свойств и т.д. во Vue, вы должны попробовать что-то более сложное. Начните с создания массива кандидатов в мэры (`mayor`). У каждого кандидата есть имя (`name`) и количество голосов (`votes`). Используйте кнопку для увеличения количества голосов для каждого кандидата. Используйте вычисляемое свойство для определения, кто сейчас текущий мэр (на основе голосов) и его имя.

Наконец, добавьте поле ввода. При получении фокуса на поле ввода и нажатой клавиши **`delete`** выборы начинаются с самого начала. Это означает, все голоса обнуляются.

Подсказка

Методы JavaScript **`sort()`** и **`map()`** могу оказаться очень полезными и модификаторы ключей помогут вам в этом.



Пример вывода

Возможное решение

Вы можете найти потенциальное решение этого упражнения здесь[4].

[4] https://github.com/hootlex/the-majesty-of-vuejs-2/blob/ru/homework/chapter5.html

6. Фильтры

В двух предыдущих главах мы рассмотрели отрисовку списка, методы и вычисляемые свойства. Теперь самое время применить полученные значения на практике, сделав несколько примеров. В этой главе мы рассмотрим, как:

1. Фильтровать массив элементов.
2. Упорядочивать массив элементов.
3. Применять пользовательский фильтр.
4. Использовать утилитарные библиотеки.

План состоит в том, что разобрать аналогичные примеры, как раньше, объединив все или некоторые техники, которые мы видели.

6.1 Отфильтрованные результаты

Иногда нам нужно отображать отфильтрованную версию массива без фактического изменения или удаления исходных данных. Продолжая предыдущий пример, Обход массива объектов, мы хотели бы показать один список с рассказами от *Alex* и ещё один список рассказов, написанных *John*. Мы можем достичь этого, создав метод, который фильтрует наш массив и возвращает результаты.

На заметку

Начиная с Vue 2.0, фильтры Vue не могут использоваться внутри `v-for`. Фильтры теперь могут использоваться только внутри интерполяций (вставок) текста (`{{ }}`). Команда Vue предлагает переместить логику фильтров в JavaScript так, чтобы её можно было повторно использовать во всём компоненте.

```
1  <html>
2  <head>
3    <link
4      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
5      rel="stylesheet"
6    >
7    <title>Пользовательские истории</title>
8  </head>
9  <body>
10     <div class="container">
11       <h1>Давайте послушаем несколько историй!</h1>
12       <div>
13         <h3>Истории Alex</h3>
14         <ul class="list-group">
15             <li v-for="(story, index) in storiesBy('Алекс')" :key="index"
16                 class="list-group-item"
17             >
18                 {{ story.writer }} сказал "{{ story.plot }}"
19             </li>
20         </ul>
21         <h3>Истории Джона</h3>
22         <ul class="list-group">
23             <li v-for="(story, index) in storiesBy('Джон')" :key="index"
24                 class="list-group-item"
25             >
26                 {{ story.writer }} сказал "{{ story.plot }}"
27             </li>
28         </ul>
29       </div>
30       <pre>{{ $data }}</pre>
31     </div>
32 </body>
33 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
34 <script type="text/javascript">
35 new Vue({
36   el: '.container',
37   data: {
38     stories: [
39       {
40         plot: 'Сегодня я разбил машину!',
41         writer: 'Алекс'
42       },
43       {
```

```
44          plot: 'Вчера кто-то украл мою сумку!',
45          writer: 'Джон'
46        },
47        {
48          plot: 'Кто-то съел мой шоколад...',
49          writer: 'Джон'
50        },
51        {
52          plot: 'Я съел чей-то шоколад!',
53          writer: 'Алекс'
54        },
55      ]
56    },
57    methods: {
58      // метод, фильтрующий истории в зависимости от автора
59      storiesBy: function (writer) {
60        return this.stories.filter(function (story) {
61          return story.writer === writer
62        })
63      },
64    }
65  })
66  </script>
67  </html>
```

На заметку

Внутри метода `storiesBy` мы используем встроенную функцию JavaScript **`filter`**.[1] Функция **`filter()`** создаёт новый массив со всеми элементами, которые проходят тест (условие фильтрации), реализуемый предоставленной функцией.

Как оказалось, мы создали метод с именем **`storiesBy`**, принимающий *writer* в качестве аргумента и возвращающий отфильтрованный массив с рассказами авторами. Затем мы можем использовать это для отображения истории каждого автора, используя директиву **`v-for`** в виде **`(story, index) in storiesBy('Alex')`**, как вы можете видеть в вышеприведённом примере.

[1] https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

Давайте послушаем несколько историй!

Истории Алекса

Алекс сказал "Сегодня я разбил машину!"

Алекс сказал "Я съел чей-то шоколад!"

Истории Джона

Джон сказал "Вчера кто-то украл мою сумку!"

Джон сказал "Кто-то съел мой шоколад..."

Истории, отфильтрованные по автору

Примечание

Как могли заметили, наш тег `li` становится действительно слишком большим, поэтому мы разделили его на несколько строк. Фактический результат остаётся таким же, как с использованием фильтров, введённых во Vue 1.x.

Достаточно просто, не так ли?

6.1.1 Использование вычисляемых свойств

Вычисляемое свойство также может использоваться для фильтрации массива. Использование вычисленного свойства для фильтрации массивов даёт более глубокий контроль и большую гибкость, поскольку это полноценный JavaScript и позволяет получить доступ к фильтруемому результату в другом месте. Например, вы можете получить длину отфильтрованного массива в любом месте кода.

Сначала мы расширим наши истории новым свойством с названием *upvotes*. Затем мы будем фильтровать известные (`famous`) истории. В качестве известной (`famous`) истории , мы определяем истории (свойство *stories*), у которых больше 25 голосов. На этот раз мы создадим вычисляемое свойство, которое возвращает отфильтрованный массив.

```
new Vue({
  el: '.container',
  data: {
    stories: [
      {
        plot: "Сегодня я разбил машину!",
        writer: 'Алекс',
        upvotes: 28
      },
      {
        plot: "Вчера кто-то украл мою сумку!",
        writer: 'Джон',
        upvotes: 8
      },
      {
        plot: "Кто-то съел мой шоколад...",
        writer: 'Джон',
        upvotes: 51
      },
      {
        plot: "Я съел чей-то шоколад!",
        writer: 'Алекс',
        upvotes: 74
      },
    ]
  },
  computed: {
    famous: function () {
      return this.stories.filter(function (item) {
        return item.upvotes > 25;
      });
    }
  }
})
```

В нашем HTML-коде вместо массива **stories** мы будем отрисовывать вычисляемое свойство **famous**.

```
<body>
    <div class="container">
    <h1>Давайте послушаем несколько историй! ({{famous.length}})</h1>
        <ul class="list-group">
            <li
              v-for="(story, index) in famous" :key="index"
              class="list-group-item"
            >
                {{ story.writer }} сказал "{{ story.plot }}"
                и получил {{ story.upvotes }} голосов.
            </li>
        </ul>
    </div>
</body>
```

Давайте послушаем известные истории! (3)

Алекс сказал "Сегодня я разбил машину!" и получил 28 голосов.

Джон сказал "Кто-то съел мой шоколад..." и получил 51 голосов.

Алекс сказал "Я съел чей-то шоколад!" и получил 74 голосов.

Фильтрация массива с использованием вычисляемого свойства

Вот и всё. Мы отфильтровали наш массив, используя вычисляемое свойство. Вы заметили, как легко нам удалось отобразить *количество известных историй* рядом с сообщением заголовка с помощью **`{{famous.length}}`**?

Далее мы реализуем очень простой (но замечательный) поиск. Когда пользователь набирает часть истории, мы можем попытаться определить в режиме реального времени, что это за история и кто её написал. Мы добавим текстовое поле ввода (**`input`**), связанное с пустой переменной **`query`** для возможности динамической фильтрации историй в соответствующем массиве.

```
<div class="container">
  <h1>Давайте послушаем несколько историй!</h1>
  <div>
    // ...
    <div class="form-group">
      <label for="query">
        Что вы ищите?
      </label>
      <input v-model="query" class="form-control">
    </div>
    <h3>Результаты поиска:</h3>
    <ul class="list-group">
        <li
          v-for="(story, index) in search" :key="index"
          class="list-group-item"
        >
          {{ story.writer }} сказал "{{ story.plot }}"
        </li>
    </ul>
  </div>
</div>
```

Затем мы создадим вычисленное свойство с названием **search**. Наряду со встроенной функцией **filter**, мы собираемся использовать JavaScript-функцию `includes`[2], которая определяет, может ли одна строка быть найдена внутри другой строки.

```
new Vue({
  el: '.container',
  data: {
    stories: [...],
      query: ' '
  },
  methods:{
    storiesBy: function (writer) {
      return this.stories.filter(function (story) {
        return story.writer === writer
      })
    }
  },
  computed: {
    search: function () {
```

[2]https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/String/includes

```
16        var query = this.query
17        return this.stories.filter(function (story) {
18          return story.plot.includes(query)
19        })
20      }
21    }
22  })
```

Давайте послушаем несколько историй!

Истории Алекса

Алекс сказал "Сегодня я разбил машину!"

Алекс сказал "Я съел чей-то шоколад!"

Истории Джона

Джон сказал "Вчера кто-то украл мою сумку!"

Джон сказал "Кто-то съел мой шоколад..."

Что вы ищите?

Результаты поиска:

Алекс сказал "Сегодня я разбил машину!"

Джон сказал "Вчера кто-то украл мою сумку!"

Джон сказал "Кто-то съел мой шоколад..."

Алекс сказал "Я съел чей-то шоколад!"

Поиск историй

Давайте послушаем несколько историй!

Истории Алекса

Алекс сказал "Сегодня я разбил машину!"

Алекс сказал "Я съел чей-то шоколад!"

Истории Джона

Джон сказал "Вчера кто-то украл мою сумку!"

Джон сказал "Кто-то съел мой шоколад..."

Что вы ищите?

шоко

Результаты поиска:

Джон сказал "Кто-то съел мой шоколад..."

Алекс сказал "Я съел чей-то шоколад!"

Поиск по ‘шоко’

Разве это не удивительно??

6.2 Отсортированные результаты

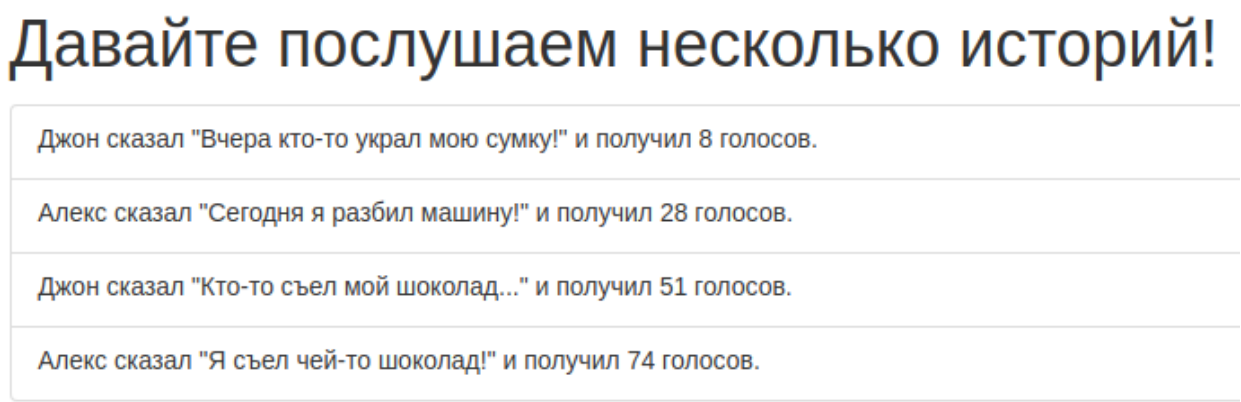
Иногда нам нужно отображать элементы массива, отсортированные по некоторому критерию. Мы можем использовать *вычисляемое свойство* для отображения нашего массива, упорядоченного по количеству голосов (*upvotes*) для каждой истории. Для сортировки мы будем использовать функцию JavaScript `sort`[3], которая сортирует элементы массива, не изменяя его, и возвращает новый массив.

Чем известнее история, тем выше в списке она должна находиться.

```
1  <html>
2  <head>
3    <link
4      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
5      rel="stylesheet"
6    >
7    <title>Известные истории</title>
8  </head>
9  <body>
10   <div class="container">
11     <h1>Давайте послушаем несколько историй!</h1>
12     <ul class="list-group">
13         <li v-for="(story, index) in orderedStories"
14             :key="index"
15             class="list-group-item"
16         >
17           {{ story.writer }} сказал "{{ story.plot }}"
18           и получил {{ story.upvotes }} голосов.
19         </li>
20     </ul>
21     <pre>{{ $data }}</pre>
22   </div>
23 </body>
24 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
25 <script type="text/javascript">
26 new Vue({
27   el: '.container',
28   data: {
29       stories: [
30           // ...
31       ]
```

[3] https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array/sort

```
32    },
33    computed: {
34      orderedStories: function () {
35        return this.stories.sort(function (a, b) {
36          return a.upvotes - b.upvotes;
37        })
38      }
39    }
40  })
41  </script>
42  </html>
```



Отсортированный по голосам массив историй

Хм, массив отсортирован, но это не то, что мы ожидали: **в начале** должны находиться известные истории.

Для изменения порядка отсортированного массива мы должны посмотреть на функцию **`sort`**. В JavaScript-функции **`sort(compareFunction)`**, если **`compareFunction`** предоставлен, элементы массива сортируются в соответствии с возвращаемым значением **`compareFunction`**. Если *a** и **`b`** — два сравниваемых элемента, тогда:

- Если **`compareFunction(a, b)`** меньше 0, сортировка поставит *a** по меньшему индексу, чем **`b`**.
- Если **`compareFunction(a, b)`** равно нулю — оставить **`a`** и **`b`** без изменений.
- Если **`compareFunction(a, b)`** больше 0, сортировка поставит **`b`** по меньшему индексу, чем **`a`**.

В нашем случае функция сравнения `compareFunction` будет следующей:

compareFunction

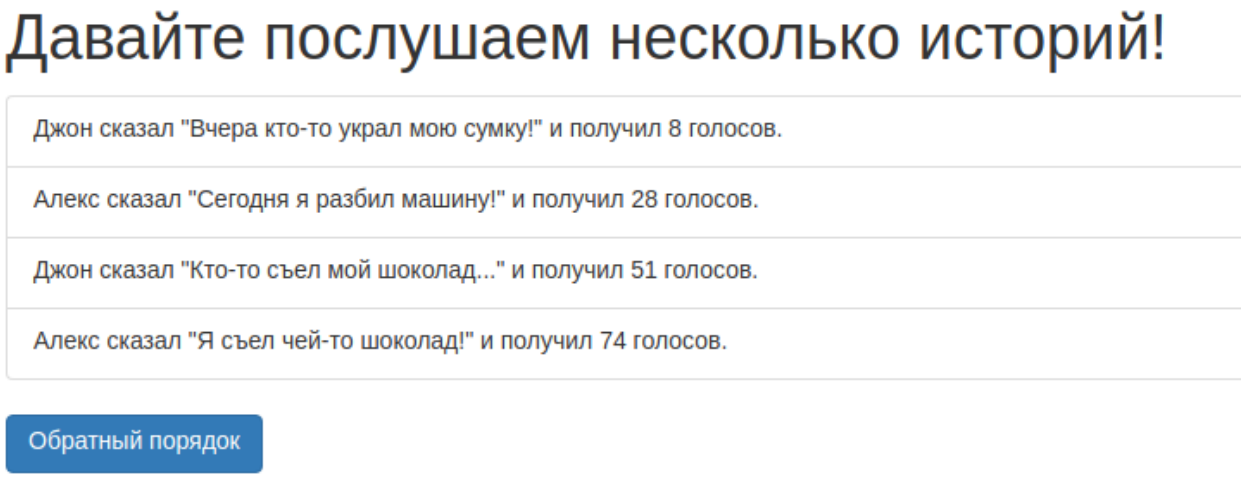
```
function (a, b) {
  return a.upvotes - b.upvotes;
}
```

Таким образом, чтобы изменить порядок от возрастанию по убыванию, мы можем умножить возвращаемое значение на **-1**. (**return (a.upvotes – b.upvotes) * -1**) Мы можем изменить порядок динамически, используя переменную **order**. Будет использоваться элемент **button**, который будет переключать значение новой переменной между **-1** и **1**.

```
1  <div class="container">
2    <h1>Давайте послушаем несколько историй!</h1>
3    <ul class="list-group">
4        <li
5          v-for="(story, index) in orderedStories"
6          :key="index"
7          class="list-group-item"
8        >
9            {{ story.writer }} сказал "{{ story.plot }}"
10           и получил {{ story.upvotes }} голосов.
11       </li>
12   </ul>
13   <button
14     type="button"
15     class="btn btn-primary"
16     @click="order = order * -1"
17   >
18     Обратный порядок
19   </button>
20   <pre>{{ $data }}</pre>
21 </div>
```

```
new Vue({
  el: '.container',
  data: {
    stories: [...],
    order : -1
  },
  computed: {
    orderedStories: function () {
      var order = this.order;
      return this.stories.sort(function (a, b) {
        return (a.upvotes - b.upvotes) * order;
      })
    }
  }
})
```

Мы инициализируем переменную **`order`** со значением **`-1`** а затем передаём её в вычисляемое свойство, поэтому каждый раз, когда нажимается кнопка, переменная меняет своё значение и массив изменяет порядок.



Массив в порядке убывания

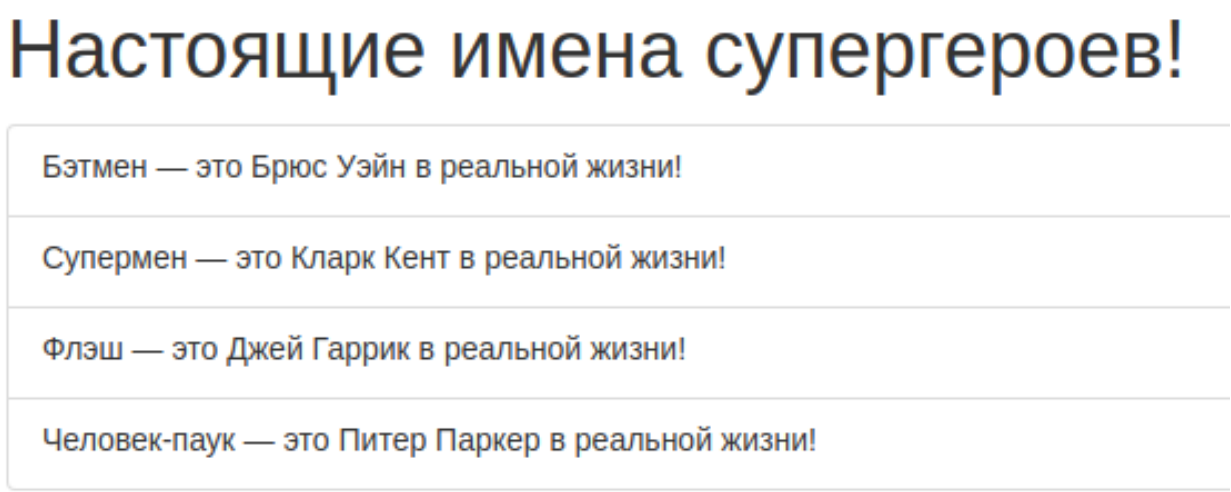
6.3 Пользовательские фильтры

Для демонстрации пользовательских фильтров, мы создадим новый простой пример. Предположим, что мы отвечаем за новостную газету «The Gotham Gazetter» в Готэм-Сити. Наше первое задание — распространять новости о тайных личностях героев. Мы знаем их имя и фамилию, и мы хотим составить неплохой список, в котором будет раскрыто секретное имя. Здесь применяется глобальный метод **`Vue.filter()`** для создания фильтра, который принимает *героя* (*hero*) и возвращает всю информацию по нему для отображения, не засоряя наш код `HTML`.

Для регистрации фильтра мы можем передать в **`filterID`** и **`filterFunction`**, который возвращает обработанное значение. Затем мы будем использовать фильтр внутри интерполяций текста, таким образом:

```
1  <html>
2  <head>
3    <link
4      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
5      rel="stylesheet"
6    >
7    <title>Газета Готэма</title>
8  </head>
9  <body>
10   <div class="container">
11     <h1>Настоящие имена супергероев!</h1>
12     <ul class="list-group">
13       <li
14         v-for="(hero, index) in heroes" :key="index"
15         class="list-group-item"
16       >
17         {{ hero | snitch }}
18       </li>
19     </ul>
20   </div>
21 </body>
22 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js">
23 </script>
24 <script>
25 Vue.filter('snitch', function (hero) {
26   return hero.secretId + ' — это '
27       + hero.firstname + ' '
28       + hero.lastname + ' в реальной жизни!'
29 })
```

```
30
31 new Vue({
32   el: '.container',
33   data: {
34     heroes: [
35       { firstname: 'Брюс', lastname: 'Уэйн', secretId: 'Бэтмен' },
36       { firstname: 'Кларк', lastname: 'Кент', secretId: 'Супермен' },
37       { firstname: 'Джей', lastname: 'Гаррик', secretId: 'Флэш' },
38       { firstname: 'Питер', lastname: 'Паркер', secretId: 'Человек-паук' }
39     ]
40   }
41 })
42 </script>
43 </html>
```



Пользовательский фильтр ‘famous’ в действии

6.4 Служебные библиотеки

На этом этапе мы хотели бы отметить, что вам нужно сортировать/фильтровать данных более продвинутыми способами, вам следует рассмотреть использование утилитарной JavaScript-библиотеки. Есть некоторые замечательные служебные библиотеки, такие как Lodash[4], Underscore[5], Sugar[6] и т.д.

Для лучшего понимания мы собираемся использовать *Lodash* и обновить предыдущий пример.

[4] https://lodash.com
[5] https://underscorejs.org/
[6] https://sugarjs.com/

Если вы последуете за нами, убедитесь, что вы включили *Lodash* из CDN в файл **HTML**.

Метод Lodash **orderBy** возвращает новый отсортированный массив. Итак, мы будем использовать его внутри нашего вычисляемого свойства для сортировки массива историй.

Синтаксис

Синтаксис Lodash **orderBy**:

```
_.orderBy(collection, [iteratees=[_.identity]], [orders])
```

Не позволяйте второму аргументу путать вас. Это действительно просто. Первый аргумент представляет массив, который вы хотите отсортировать. Второй аргумент ожидает массив ключей, на основе которого будет производиться сортировка. Третий аргумент ожидает массив *orders* для каждого ключа.

Например, если у нас есть такой массив:

```
var kids = [
  { name: 'Stan', strength: 70, intelligence: 70 },
  { name: 'Kyle', strength: 40, intelligence: 80 },
  { name: 'Eric', strength: 45, intelligence: 80 },
  { name: 'Kenny', strength: 100, intelligence: 70 }
]
```

И мы выполняем следующее:

```
_.orderBy(kids, ['intelligence', 'strength'], ['desc', 'asc'])
```

Наш массив отсортирован в таком порядке:

```
var kids = [
  { name: 'Kyle', strength: 40, intelligence: 80 },
  { name: 'Eric', strength: 45, intelligence: 80 },
  { name: 'Stan', strength: 70, intelligence: 70 },
  { name: 'Kenny', strength: 100, intelligence: 70 }
]
```

Поскольку массив **в первую очередь** сортируется по **интеллекту ребёнка в порядке убывания** и **только потом** сортируется по **strength в порядке возрастания**.

Мы будем использовать **_.orderBy** внутри вычисляемого свойства следующим образом:

```
computed: {
  orderedStories: function () {
    var order = this.order
    return _.orderBy(this.stories, 'upvotes')
  }
}
```

Это работает, но если аргумент *orders* не передан, массив будет отсортирован в порядке возрастания. Кроме того, должна быть возможность изменить порядок сортировки массива с кнопки, как и раньше. Чтобы сделать это автоматически, мы можем задать свойство данных **`order: 'desc'`** и создать метод для изменения его значения:

```
methods: {
  reverseOrder: function () {
    this.order = (this.order === 'desc') ? 'asc' : 'desc'
  }
},
computed: {
  orderedStories: function () {
    var order = this.order
    return _.orderBy(this.stories, 'upvotes', [order])
  }
}
```

Кнопка будет такой же как и раньше, но с небольшими изменениями.

```
<button type="button" v-on:click="reverseOrder">
```

Этого достаточно. Мы добились ту же самую функциональность, используя *Lodash*.

Подсказка

При использовании утилитарной библиотеки для фильтрации/сортировки данных вы можете перебирать результирующий массив без использования вычисляемых свойств.

Мы можем обновить этот пример для демонстрации этой идеи. Код HTML, отображающий отсортированный массив будет таким:

```
<div class="container">
  <h1>Давайте послушаем несколько историй!</h1>
  <ul class="list-group">
    <li
      v-for="(story, index) in _.orderBy(stories, ['upvotes'], ['desc'])"
      :key="index"
     >
      {{ story.writer }} сказал "{{ story.plot }}"
      и получил {{ story.upvotes }} голосов.
    </li>
  </ul>
</div>
```

Вот и всё. Не нужно писать код *JavaScript*.

Примеры кода

Вы можете найти примеры кода для этой главы на GitHub[7].

[7] https://github.com/hootlex/the-majesty-of-vuejs-2/tree/ru/codes/chapter6

6.5 Домашняя работа

В качестве упражнения этой главы вы должны сделать следующее. Начните с создания массива людей. У каждого человек есть имя и возраст. Используя то, что вы узнали в этой главе, попробуйте отрисовать массив в списке и отсортировать по его возрасту. После этого создайте второй список и создайте вычисляемое свойство с названием «old», возвращающее всех людей старше 65 лет.

Не стесняйтесь заполнять массив собственными данными. **Будьте осторожны**, чтобы добавить людей с возрастом старше и моложе 65 лет для проверки работоспособности фильтра. Вперёд!

Подсказка

Для решения упражнения необходим модификатор `.filter`.

Люди Галлии

Астерикс 32

Обеликс 33

Юлий Цезарь 66

Юлий Цезарь 66

Маджестикс 82

Ветераникс 103

"Старые" люди Галлии

Маджестикс 82

Ветераникс 103

Юлий Цезарь 66

Юлий Цезарь 66

Вывод примера

Возможное решение

Вы можете найти потенциальное решение этого упражнения здесь[8].

[8] https://github.com/hootlex/the-majesty-of-vuejs-2/blob/ru/homework/chapter6.html

7. Компоненты

7.1 Что такое компоненты?

Компоненты — одна из самых мощных возможностей Vue.js. Они помогают вам расширять базовые HTML-элементы для инкапсуляции переиспользуемого кода. На высоком уровне компоненты — пользовательскими элементы, к которым Vue.js присоединяет указанное поведение. В некоторых случаях они могут также отображаться как нативный HTML-элемент, расширенный с помощью атрибута `is`.

Это действительно умный и мощный расширить HTML для создания новых элементов. В этой главе мы начнём с очень простого примера. Затем мы увидим, как компоненты могут помочь нам улучшить созданный нами код из предыдущих главах.

7.2 Использование компонентов

Мы начнём с простого компонента. Чтобы использовать компонент, его нужно сначала зарегистрировать.

Одним из способов регистрации компонента является использование метода `Vue.component` и передача в него `tag` и `constructor`. Думайте про `tag` как о имени компонента, а про `constructor` как об его опциях. В нашем случае, мы назовём компонент `story` и определим свойство `story` (ещё раз). Опция `template` (как мы хотим, чтобы отображалась истории) находится внутри `constructor`, где будут добавлены также другие опции.

Наш компонент `story` будет зарегистрирован следующим образом:

```
Vue.component('story', {
  template: '<h1>Моя лошадь потрясающая!</h1>'
});
```

Теперь, когда мы зарегистрировали компонент, то можем его использовать. Мы добавим пользовательский элемент `<story>` внутри HTML-кода для отображения рассказа.

```
<html>
<head>
  <link
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
    rel="stylesheet"
  >
  <title>Привет, Vue</title>
</head>
<body>
  <div class="container">
    <story></story>
  </div>
</body>
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
<script type="text/javascript">
Vue.component('story', {
  template: '<h1>Моя лошадь потрясающая!</h1>'
});

new Vue({
  el: '.container'
})
</script>
</html>
```

Примечание

Обратите внимание, что вы можете дать имя своему пользовательскому компоненту любое, которое хотите, но обычно рекомендуется, чтобы вы использовали уникальное имя для избежания коллизий с реальными тегами, которые могут появиться в какой-то момент в будущем.

Как мы уже говорили в начале главе, компоненты могут повторно использоваться. Это означает, что вы можете добавить любое количество элементов `<story>`, которое только захотите. Следующий HTML-фрагмент отобразит нашу историю 3 раза.

```
<body>
  <div class="container">
    <story></story>
    <story></story>
    <story></story>
  </div>
</body>
```

Моя лошадь потрясающая!

Моя лошадь потрясающая!

Моя лошадь потрясающая!

Отображение компонента истории

7.3 Шаблоны

Существуют несколько способов объявления шаблонов для компонента. Встроенный шаблон, который мы ранее использовали, очень быстро может стать «грязным».

Другой способ — создать тег **`script`** с типом **`text/template`** и с атрибутом **`id`**, равным **`story-template`**. Для использования такого шаблона нам нужно связать селектор в опции **`template`** нашего компонента component с этим скриптом.

```
<script type="text/template" id="story-template">
  <h1>Моя лошадь потрясающая!</h1>
</script>

<script type="text/javascript">
Vue.component('story', {
    template: '#story-template'
});
</script>
```

На заметку

Тип скрипта **"text/template"** не понимает браузер, поэтому он будет просто игнорировать его. Это позволяет вам помещать что-нибудь туда, которое затем можно извлечь и генерировать сниппеты HTML.

Мой любимый способ определить **template** (и который я собираюсь использовать в примерах этой книги) — создать HTML-тег **template** и назначить ему значение для атрибута **id**. Тогда мы можем сослаться на селектор, как и раньше. Используя этот метод, вышеуказанный компонент будет выглядеть так:

```
<template id="story-template">
  <h1>Моя лошадь потрясающая!</h1>
</template>

<script type="text/javascript">
Vue.component('story', {
    template: '#story-template'
});
</script>
```

7.4 Входные параметры

Давайте посмотрим, как мы можем использовать несколько экземпляров нашего компонента **story** для отображения списка историй. Мы должны обновить **template**, чтобы не отображать всегда одну и ту же историю, а текст истории, которую мы хотим.

```
<template id="story-template">
  <h1>{{ plot }}</h1>
</template>
```

Нам также требуется обновить наш компонент для использования свойства. Для этого мы добавим новое свойство ‘plot’ в атрибут **props** компонента.

```
Vue.component('story', {
  props: ['plot'],
  template: '#story-template'
});
```

Теперь мы можем передавать **plot** каждый раз, когда используем элемент **<story>**.

```
<div class="container">
  <story plot="Моя лошадь потрясающая."></story>
  <story plot="Нарвалы изобрели шашлык."></story>
  <story plot="Тёмная сторона Силы сильнее."></story>
</div>
```

Моя лошадь потрясающая.

Нарвалы изобрели шашлык.

Тёмная сторона Силы сильнее.

Отображение различных историй

Предупреждение

HTML-атрибут не зависят от регистра. При использовании имён свойств в стиле camelCase вам необходимо использовать их эквиваленты в kebab-case (разделённые дефисом).

Итак, camelCase в JavaScript, kebab-case в HTML. Например, `props: ['isUser']` станет HTML-атрибутом `<story is-user="true"></story>`.

Как вы могли себе представить, компонент может иметь более одного свойства. Например, если мы хотим отобразить автора вместе с сюжетом каждой истории, нам вдобавок ещё нужно передать `writer`.

```
<story plot="Моя лошадь потрясающая." writer="Мистер Уибл"></story>
```

Если у вас много свойств и ваши элементы становятся «грязными», вы можете передать объект и отобразить его свойства.

Мы ещё раз улучшим наш пример, чтобы завершить его.

```
<html>
<head>
  <link
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
    rel="stylesheet"
  >
  <title>Удивительные истории</title>
</head>
<body>
  <div class="container">
    <story v-bind:story="{plot: 'Моя лошадь потрясающая.', writer: 'Мистер Уибл'}">
    </story>
    <story v-bind:story="{plot: 'Нарвалы изобрели шашлык.', writer: 'Мистер Уибл'}">
    </story>
    <story v-bind:story="{plot: 'Тёмная сторона Силы сильнее.', writer: 'Дарт Вейдер\
'}">
    </story>
  </div>
  <template id="story-template">
    <h1>{{ story.writer }} сказал "{{ story.plot }}"</h1>
  </template>
</body>
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
<script type="text/javascript">
Vue.component('story', {
  props: ['story'],
  template: '#story-template'
});

new Vue({
  el: '.container'
})
</script>
</html>
```

На заметку

`v-bind` используется для динамического связывания одного или нескольких атрибутов, или свойств компонентов, к выражению.

Поскольку свойство `story` не строка, а объект JavaScript, вместо `story="..."` мы используем `v-bind:story="..."` для привязывания свойства `story` к переданному объекту.

`:` — сокращение для `v-bind`, поэтому с этого момента мы воспользуемся этим `:story="..."`.

7.5 Повторное использование

Давайте ещё раз взглянем на пример **фильтрации результатов**. Предположим, что на этот раз мы берём данные переменной `stories` из внешнего API через HTTP-вызов. Разработчики API решили переименовать свойство `plot` на `body`. Итак, просмотреть весь код и внести необходимые изменения.

На заметку

Позже в этой книге мы рассмотрим, как можно использовать `Vue` создания веб-запросов.

```
1  <div class="container">
2    <h1>Давайте послушаем несколько историй!</h1>
3    <div>
4      <h3>Истории Алекса</h3>
5      <ul class="list-group">
6          <li v-for="(story, index) in storiesBy('Алекс')" :key="index"
7              class="list-group-item"
8          >
9            {{ story.writer }} сказал "{{ story.plot }}"
10           {{ story.writer }} сказал "{{ story.body }}"
11         </li>
12     </ul>
13     <h3>Истории Джона</h3>
14     <ul class="list-group">
15         <li v-for="(story, index) in storiesBy('Джон')" :key="index"
16             class="list-group-item"
17         >
18           {{ story.writer }} сказал "{{ story.plot }}"
```

```
19          {{ story.writer }} сказал "{{ story.body }}"
20        </li>
21    </ul>
22    <div class="form-group">
23      <label for="query">
24        Что вы ищите?
25      </label>
26      <input v-model="query" class="form-control">
27    </div>
28    <h3>Результаты поиска:</h3>
29    <ul class="list-group">
30        <li v-for="story in search"
31            class="list-group-item"
32        >
33          {{ story.writer }} сказал "{{ story.plot }}"
34          {{ story.writer }} сказал "{{ story.body }}"
35        </li>
36    </ul>
37  </div>
38 </div>
```

Примечание

В этом конкретном примере подсветка синтаксиса отключена.

Как вы, возможно, заметили, нам пришлось сделать одно и то же изменение 3 раза, и я не знаю насчёт вас, но я ненавижу повторяться. Если для вас это не имеет большого значения, представьте, что вышеуказанный блок кода нужно добавить в 100 местах, что вы тогда сделаете? К счастью, *Vue* предоставляет решение для подобных ситуаций, и имя у этого решения — **компонент**.

Подсказка

Всякий раз, когда вы обнаруживаете, что повторяете часть функциональности, наиболее эффективным способом борьбы с этим является создание отдельного компонента.

К счастью, в предыдущем примере мы создали компонент `story`, отображающий автора и его текст определённой истории. Мы можем использовать пользовательский элемент `<story>` внутри нашего `HTML` и передавать каждую историю, как и раньше, с тегом `:story`. В этот раз мы будем использовать его внутри директивы `v-for`.

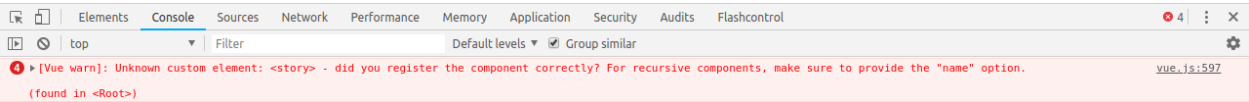
Таким образом, наш код примет следующий вид:

```
<div class="container">
  <h1>Давайте послушаем несколько историй!</h1>
  <div>
    <h3>Истории Алекса</h3>
    <ul class="list-group">
      <story
        v-for="(story, index) in storiesBy('Alex')" :key="index"
        :story="story">
      </story>
    </ul>
    <h3>Истории Джона</h3>
    <ul class="list-group">
      <story
        v-for="(story, index) in storiesBy('John')" :key="index"
        :story="story">
      </story>
    </ul>
    <div class="form-group">
      <label for="query">Что вы ищите?</label>
      <input v-model="query" class="form-control">
    </div>
    <h3>Результаты поиска:</h3>
    <ul class="list-group">
      <story
        v-for="(story, index) in search" :key="index"
        :story="story">
      </story>
    </ul>
  </div>
</div>
```

Если вы попытаетесь выполнить этот код, отобразится следующее предупреждение:

Vue warn: Unknown custom element: `<story>` — did you register the component correctly? For recursive components, make sure to provide the "name" option.

(***Предупреждение Vue: неизвестный пользовательский элемент: `<story>` — вы правильно зарегистрировали компонент Для рекурсивных компонентов обязательно укажите опцию "name"***)



Предупреждение Vue

Для исправления этого нам нужно зарегистрировать снова компонент. На этот раз мы должны внести изменения в шаблон компонента. Мы изменим атрибут `plot` на `body` и тег `<h1>` на `<li>` в соответствии с нашими потребностями.

Итак, шаблон истории будет таким:

```
<template id="story-template">
  <li class="list-group-item">
    {{ story.writer }} сказал "{{ story.body }}"
  </li>
</template>
```

Компонент останется таким же.

```
Vue.component('story', {
  props: ['story'],
  template: '#story-template'
});
```

При выполнении указанного выше кода, вы сможете убедиться сами, что всё работает так же, как и раньше, но на этот раз с использованием пользовательского компонента.

Довольно аккуратно, правда?

Предупреждение

Пожалуйста, будьте ответственными. Не садитесь за руль пьяными.

7.6 Всё вместе

Используя недавно приобретённые знания, мы должны иметь возможность создавать нечто более сложное. Основываясь на структуре приведённого выше примера, мы собираемся создавать систему голосования для историй (`stories`) и добавить функциональность *пометить как избранное*. Способ реализации этого — методы, директивы и, конечно же, компоненты.

Давайте начнём с корректировки данных историй.

```
1  <html>
2  <head>
3    <link
4      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
5      rel="stylesheet"
6    >
7    <title>Привет, Vue</title>
8  </head>
9  <body>
10 <div id="app">
11   <div class="container">
12     <h1>Давайте послушаем несколько историй!</h1>
13     <ul class="list-group">
14         <story v-for="(story, index) in stories" :key="index" :story="story"></story>
15     </ul>
16     <pre>{{ $data }}</pre>
17   </div>
18 </div>
19 <template id="story-template">
20   <li class="list-group-item">
21     {{ story.writer }} сказал "{{ story.plot }}"
22   </li>
23 </template>
24 </body>
25 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
26 <script type="text/javascript">
27 Vue.component('story', {
28   template: '#story-template',
29   props: ['story'],
30 });
31
32 new Vue({
33   el: '#app',
34   data: {
35     stories: [
36       {
37         plot: 'Моя лошадь потрясающая',
38         writer: 'Мистер Уибл',
39       },
40       {
41         plot: 'Нарвалы изобрели шашлык.',
42         writer: 'Мистер Уибл',
43       },
```

```
44        {
45          plot: 'Тёмная сторона Силы сильнее.',
46          writer: 'Дарт Вейдер',
47        },
48        {
49          plot: 'Просто так в Мордор не попадёшь',
50          writer: 'Боромир',
51        },
52      ]
53    }
54  })
55  </script>
56  </html>
```

Следующий шаг позволяет пользователю проголосовать за предпочитаемую историю. Для применения этого лимита (1 голос за историю) вы отобразим кнопку с надписью «Проголосовать», только ещё не проголосовавшим пользователям. Таким образом, у каждой истории должно быть свойство **voted**, которое принимает значение `true` при выполнении функции **upvote**.

```
<template id="story-template">
  <li class="list-group-item">
    {{ story.writer }} сказал "{{ story.plot }}".
    Голосов за историю: {{ story.upvotes }}.
    <button
      type="button"
      v-show="!story.voted"
      @click="upvote"
      class="btn btn-default"
    >
      Проголосовать
    </button>
  </li>
</template>
```

```
Vue.component('story', {
  template: '#story-template',
  props: ['story'],
  methods:{
      upvote: function () {
          this.story.upvotes += 1;
          this.story.voted = true;
      },
  }
});

new Vue({
  el: '#app',
  data: {
    stories: [
      {
        plot: 'Моя лошадь потрясающая',
        writer: 'Мистер Уибл',
        upvotes: 28,
        voted: false,
      },
      {
        plot: 'Нарвалы изобрели шашлык.',
        writer: 'Мистер Уибл',
        upvotes: 8,
        voted: false,
      },
      {
        plot: 'Тёмная сторона Силы сильнее.',
        writer: 'Дарт Вейдер',
        upvotes: 49,
        voted: false,
      },
      {
        plot: 'Просто так в Мордор не попадёшь',
        writer: 'Боромир',
        upvotes: 74,
        voted: false,
      },
    ]
  }
})
```

Давайте послушаем несколько историй!

Мистер Уибл сказал "Моя лошадь потрясающая.". Голосов за историю: 28. Проголосовать

Мистер Уибл сказал "Нарвалы изобрели шашлык.". Голосов за историю: 8. Проголосовать

Дарт Вейдер сказал "Тёмная сторона Силы сильнее.". Голосов за историю: 49. Проголосовать

Боромир сказал "Просто так в Мордор не попадёшь". Голосов за историю: 74. Проголосовать

Готово для голосования!

Мы реализовали с использованием методов систему голосования. Я думаю, это выглядит хорошо, поэтому мы можем продолжить часть с «избранными историями». Мы хотим, чтобы пользователь мог выбрать только одну избранную историю. Первое, что приходит на ум, — добавить один новый пустой объект (favorite) и всякий раз, когда пользователь выбирает одну историю, чтобы поместить её в избранное, обновлять переменную **`favorite`**. Таким образом, мы сможем проверить, является ли это история избранной для пользователя. Давайте сделаем это.

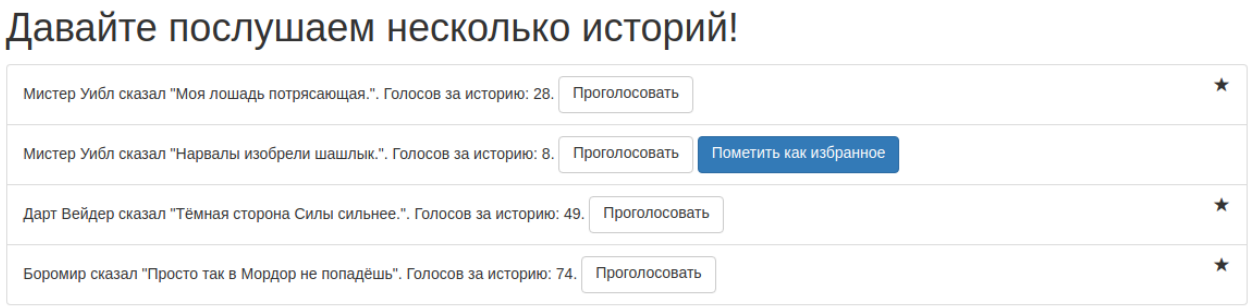
```
<template id="story-template">
    <li class="list-group-item">
      {{ story.writer }} сказал "{{ story.plot }}".
      Голосов за историю: {{ story.upvotes }}.
      <button
        type="button"
        v-show="!story.voted"
        @click="upvote"
        class="btn btn-default"
      >
        Проголосовать
      </button>
      <button
        type="button"
        v-show="!isFavorite"
        @click="setFavorite"
        class="btn btn-primary"
      >
        Пометить как избранное
      </button>
```

```
      <span
        v-show="isFavorite"
        class="glyphicon glyphicon-star pull-right"
        aria-hidden="true">
      </span>
    </li>
</template>


Vue.component('story', {
  template: '#story-template',
  props: ['story'],
  methods:{
    upvote: function () {
      this.story.upvotes += 1;
      this.story.voted = true;
    },
    setFavorite: function () {
      this.favorite = this.story;
    },
  },
  computed:{
    isFavorite: function () {
      return this.story === this.favorite;
    },
  }
});

new Vue({
  el: '#app',
  data: {
    stories: [
      // ...
    ],
    favorite: {}
  }
})
```

При попытке выполнить этот код, вы заметите, что он не работает, как ожидалось. Каждый раз, когда пытаетесь добавить историю в избранное, переменная **`favorite`** внутри **`$data`** остаётся `null`.

Похоже, что наш компонент **`story`** не может обновить объект **`favorite`**, поэтому мы собираемся передавать его каждой истории и добавить свойства компонента **`favorite`**.

```
<ul class="list-group">
    <story
      v-for="(story, index) in stories"
      :key="index"
      :favorite="favorite"
    >
    </story>
</ul>


Vue.component('story', {
  // ...
  props: ['story', 'favorite'],
  // ...
});
```



неправильный метод `setFavorite`

Хм, **`favorite`** до сих пор не обновляется при выполнении **`setFavorite`**. Кнопка исчезает, как ожидалось, и появляется иконка со звездой, но переменная favorite по-прежнему `null`. Это приводит к тому, что пользователь может отметить все истории как избранные.

Проблема с этим подходом заключается в том, что мы не *синхронизируем* данные. По умолчанию все входные параметры образуют одностороннюю привязку между дочерним свойством и родительским. Когда родительское свойство обновляется, её изменённое значение будет доходит до дочернего свойства, но не наоборот.

Мы не можем синхронизировать данные дочерних элементов с родительскими. Итак, пока мы сделаем перерыв и перейдём к изучению **пользовательским событиям** Vue, прежде чем идти дальше.

Примеры кода

Вы можете найти примеры кода для этой главы на GitHub[1].

[1] https://github.com/hootlex/the-majesty-of-vuejs-2/tree/ru/codes/chapter7

7.7 Домашняя работа

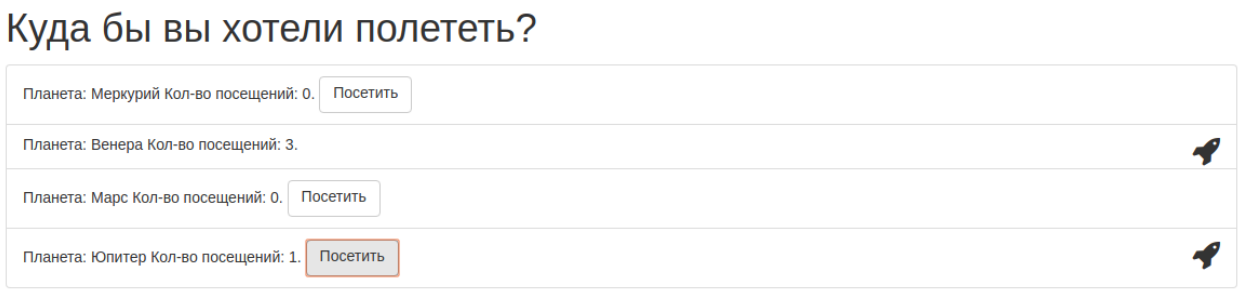
Создайте массив планет. У каждой планеты будут имя (`name`) и количество посещений (`number of visits`).

Вы можете полететь на любую планету, но ограничены только 3 посещениями на каждую планету из-за нехватки топлива.

Для решения этого задания, вам нужен компонент *Planet* с соответствующими методами/-вычисляемыми свойствами.

Каждая планета должна отображать:

- своё имя
- количество посещений
- Кнопка *Посетить* (если максимальное количество посещений этой планеты ещё не было достигнуто)
- Иконка, указывающая, была ли посещена планета хотя бы один раз



Пример вывода

Возможное решение

Вы можете найти потенциальное решение этого упражнения здесь[2].

[2] https://github.com/hootlex/the-majesty-of-vuejs-2/blob/ru/homework/chapter7.html

8. Пользовательские события

Иногда нужно генерировать пользовательское событие. Для этого мы можем использовать методы экземпляра Vue. Каждый экземпляр Vue реализует интерфейс событий[1].

Это означает, что он может:

- Обрабатывать событие, используя `$on(event)`.
- Сгенерировать событие, используя `$emit(event)`.

Он также может:

- Обрабатывать событие только один раз, используя `$once(event)`.
- Удалить обработчика событий, используя `$off()`.

8.1 Генерация и обработка событий

Мы начнём с очень простого примера.

Следующий блок кода представляет страницу со счётчиком и кнопкой *Голосовать*. При нажатии на кнопку, инициируется событие с именем '**voted**'

Также есть *обработчик события* для события, который увеличивает количество голосов при срабатывании события.

```
<html>
<head>
  <title>Генерация и обработка</title>
  <link
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
    rel="stylesheet"
  >
</head>
<body>
  <div class="container text-center">
    <p style="font-size: 140px;">
      {{ votes }}
```

[1] https://ru.vuejs.org/v2/api/#%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D1%8B-%D1%8D%D0%BA%D0%B7%D0%B5%D0%BC%D0%BF%D0%BB%D1%8F%D1%80%D0%B0-%E2%80%94-%D1%81%D0%BE%D0%B1%D1%8B%D1%82%D0%B8%D1%8F

```
13      </p>
14      <button type="button" class="btn btn-primary" @click="vote">Голосовать</button>
15    </div>
16  </body>
17  <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
18  <script type="text/javascript">
19  new Vue({
20    el: '.container',
21    data: {
22      votes: 0
23    },
24    methods: {
25      vote: function (writer) {
26        this.$emit('voted')
27      }
28    },
29    created () {
30      this.$on('voted', function (button) {
31        this.votes++
32      })
33    }
34  })
35  </script>
36  </html>
```

Предупреждение

Здесь и далее используется сокращённый синтаксис определения колбэка из ES6, на примере `created`, т.е. это: {lang="javascript", line-numbers="off"} created () { //... } То же самое, что это (ES5) {lang="javascript", line-numbers="off"} created: function () { //... }

Пример вывода

Мы регистрируем обработчик событий внутри *хука жизненного цикла* `created`. `this` связан с экземпляром Vue внутри метода `vote` и хука `created`. Таким образом, мы можем получить доступ к функциям `$on` и `$emit`, используя `this.$on` и `this.$emit`.

8.1.1 Хуки жизненного цикла

Хуки жизненного цикла — это функции, которые выполняются, когда срабатывают события, связанные с Vue.

Хуки, доступные во Vue 2:

Хук	Когда вызывается
`beforeCreate`	После того, как экземпляр только что был создан, перед отслеживанием данных и настройкой событий/наблюдателей.
`created`	После создания экземпляра.
`beforeMount`	Прямо перед началом монтирования
`mounted`	После того, как экземпляр только был примонтирован в DOM.
`beforeUpdate`	При изменении данных, перед тем, как виртуальный DOM будет повторно отрисован и обновлён.
`updated`	После изменения данных вызывает повторную отрисовку и обновление виртуального DOM
`activated`	При активации компонента kept-alive.
`deactivated`	При деактивации компонента kept-alive.
`beforeDestroy`	Прямо перед уничтожением экземпляра Vue.

Хук	Когда вызывается
`destroyed`	После уничтожения экземпляра Vue.

Вам не обязательно знать про всё это, но хорошо иметь представление об их существовании. Если вы хотите узнать больше о хуках жизненного цикла, посмотрите API Vue[2].

8.2 Взаимодействие родительского и дочернего компонента

Всё становится по-другому, когда родительский компонент должен обрабатывать событие *дочернего* компонента. Мы не можем использовать `this.$on`/`this.$emit`, поскольку `this` будет связан с различными экземплярами.

Помните про обработчик событий? `v-on` (`@`)? Родительский компонент может обрабатывать события, исходящие из дочернего компонента, используя `v-on` непосредственно в шаблоне, где используется дочерний элемент.

Следуя предыдущему примеру, я создам компонент Food со свойством *name*. В его шаблоне будет отображаться кнопка с именем (*name*). При нажатии на кнопку, мы хотим сгенерировать событие *vote*.

Компонент Food

```
Vue.component('food', {
  template: '#food',
  props: ['name'],
  methods: {
    vote: function () {
      this.$emit('voted')
    }
  },
})
```

Шаблон компонента Food

```
<template id="food">
  <button type="button" class="btn btn-default" @click="vote">{{ name }}</button>
</template>
```

В родительском экземпляре `<button type="button">` будет заменён `<food @voted="countVote"></food>`.

`@voted="countVote"` означает, что, когда генерируется событие `voted` дочернего компонента, будет выполняться метод *countVote*. Мы также можем избавиться от слушателя `this.$on`, так как он нам больше не понадобиться.

[2] https://ru.vuejs.org/v2/api/index.html#%D0%9E%D0%BF%D1%86%D0%B8%D0%B8-%E2%80%94-%D1%85%D1%83%D0%BA%D0%B8-%D0%B6%D0%B8%D0%B7%D0%BD%D0%B5%D0%BD%D0%BD%D0%BE%D0%B3%D0%BE-%D1%86%D0%B8%D0%BA%D0%BB%D0%B0

Родительский компонент

```
new Vue({
  el: '.container',
  data: {
    votes: 0
  },
  methods: {
    countVote: function () {
      this.votes++
    }
  }
})
```

Шаблон родительского компонента

```
<div class="container text-center">
  <p style="font-size: 140px;">
    {{ votes }}
  </p>
  <food @voted="countVote" name="Чизбургер"></food>
</div>
```

Если вы запустите данный код в браузере, вы увидите, что вывод один и тот же.

Взаимодействие родительского и дочернего компонента

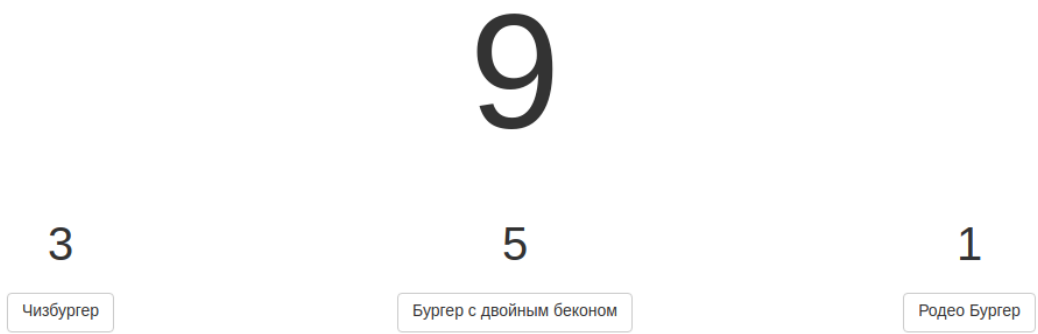
8.3 Передача аргументов

Давайте создадим 3 экземпляра *компонента Food*. Каждый экземпляр будет иметь собственное количество голосов. Когда какой-либо из компонентов Food будет проголосован, он увеличит свои голоса и сгенерирует событие дял обновления общего количества голосов, которое также находится в родительском компоненте.

```
1  <html>
2  <head>
3    <title>Битва между продуктами питания</title>
4    <link
5      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
6      rel="stylesheet"
7    >
8  </head>
9  <body>
10   <div class="container text-center">
11     <p style="font-size: 140px;">
12       {{ votes }}
13     </p>
14
```

```
15      <div class="row">
16        <food @voted="countVote" name="Чизбургер"></food>
17        <food @voted="countVote" name="Бургер с двойным беконом"></food>
18        <food @voted="countVote" name="Родео Бургер"></food>
19      </div>
20    </div>
21
22  </body>
23  <template id="food">
24    <div class="text-center col-lg-4">
25      <p style="font-size: 40px;">
26        {{ votes }}
27      </p>
28      <button type="button" class="btn btn-default" @click="vote">{{ name }}</button>
29    </div>
30  </template>
31  <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
32  <script type="text/javascript">
33  Vue.component('food', {
34    template: '#food',
35    props: ['name'],
36    data: function () {
37      return {
38        votes: 0
39      }
40    },
41    methods: {
42      vote: function () {
43        this.votes++
44        this.$emit('voted')
45      }
46    }
47  })
48  new Vue({
49    el: '.container',
50    data: {
51      votes: 0
52    },
53    methods: {
54      countVote: function () {
55        this.votes++
56      }
57    }
```

```
58 })
59 </script>
60 </html>
```



Несколько экземпляров компонента

Пока ничего нового. Чтобы сделать приложение более интересным, мы добавим *журнал голосований*. Журнал будет обновляться каждый раз, когда проголосовали за продукт питания. Нам нужно обновить дочерний компонент для передачи имя продукта при генерации события `voted`.

На заметку

Функция **`$emit`** вместе с аргументом `event name` передаёт любые дополнительные аргументы в колбэк обработчика. Например: **`vm.$emit('voted', 'Alex', 'Sunday', 'Bob Ross')`**

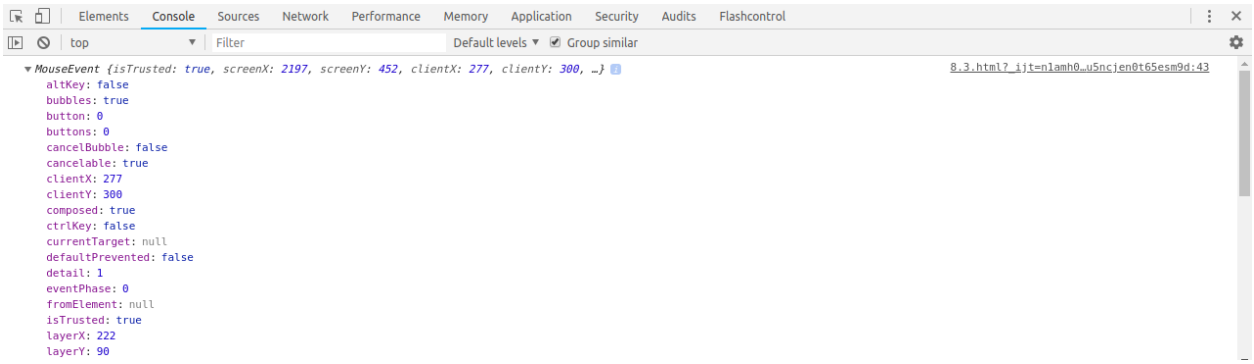
У нас имеется два варианта доступа к имени продукта. Первый — очевидно, по имени свойства компонента. Второй способ — получить доступ к элементу, который вызвал событие и найти его текстовое содержимое. Мы выберем второй способ.

Мы можем залогировать переменную `event` в консоль внутри метода **`vote`** компонента **`Food`**', чтобы узнать, как получить доступ к нажатому элементу.

Компонент Food

```
Vue.component('food', {
  // ...
  methods: {
    vote: function (event) {
      console.log(event)
      this.votes++
      this.$emit('voted')
    }
```

```
  }
})
```



event.srcElement

Если вы сделаете так, то вы увидите, что у нас есть доступ к нажатому элементу внутри атрибута **event.srcElement**. Имя будет как в **event.srcElement.outerText**, так и в **event.srcElement.textContent**.

Итак, теперь давайте один из них в функцию **$emit**.

Компонент Food

```
Vue.component('food', {
  // ...
  methods: {
    vote: function (event) {
      this.votes++
      this.$emit('voted', event.srcElement.textContent)
    }
  }
})
```

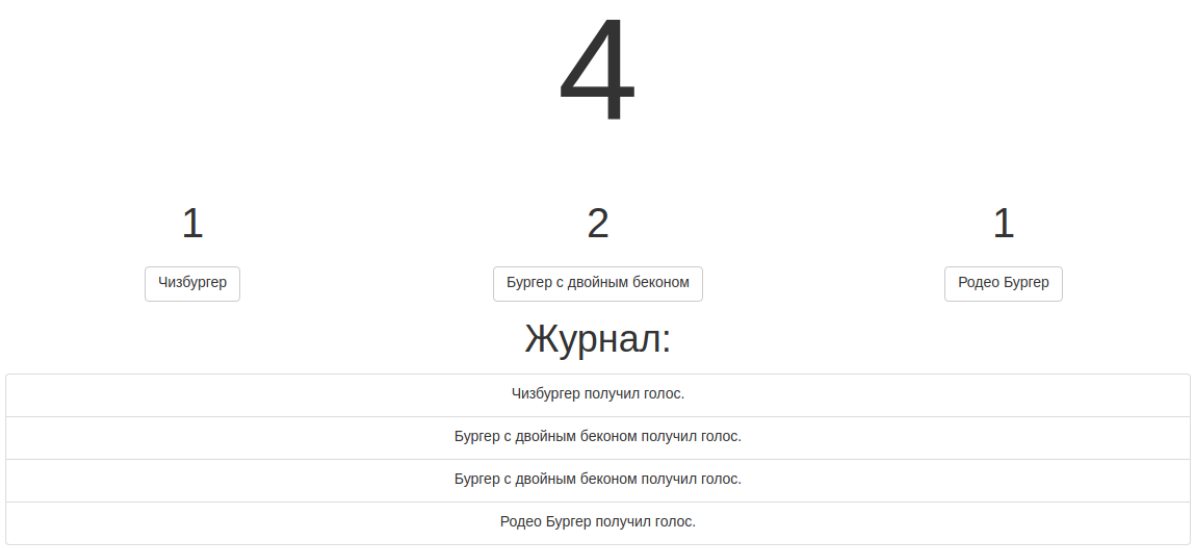
Внутри родительского компонента, мы добавляем поступающий голос в массив `log`.

```
new Vue({
  el: '.container',
  data: {
    votes: 0,
    log: []
  },
  methods: {
    countVote: function (food) {
      this.votes++
      this.log.push(food + ' получил голос.')
```

```
    }
  }
})
```

Для отображения журнала (`log`), мы можем добавить список в шаблон HTML.

```
<h1>Журнал:</h1>
<ul class="list-group">
  <li class="list-group-item" v-for="vote in log"> {{ vote }} </li>
</ul>
```



Журнал голосований

Довольно просто, да?

8.4 Не взаимодействие родительского и дочернего компонента

Мы рассмотрим предыдущий пример ещё раз, добавив кнопку сброса. При её нажатии, она сбросит все счётчики голосов. Как вы можете себе представить, кнопка сброса сгенерирует событие, которое будет обработано всеми компонентами. Но как мы можем поймать событие внутри дочерних компонентов?

Если мы последуем способу с `<food @voted="countVote">`, мы можем обработать событие `voted`, но у нас нет способа генерировать события дочерним компонентам.

Чтобы все компоненты могли взаимодействовать друг с другом, мы будем использовать **пустой экземпляр Vue** в качестве центральной шины событий. Затем внутри хука `created`

компонентов, мы зарегистрируем обработчика событий, используя **bus.$on** вместо **this.$on**. Соответственно, мы будем использовать **bus.$emit** для генерации всех событий.

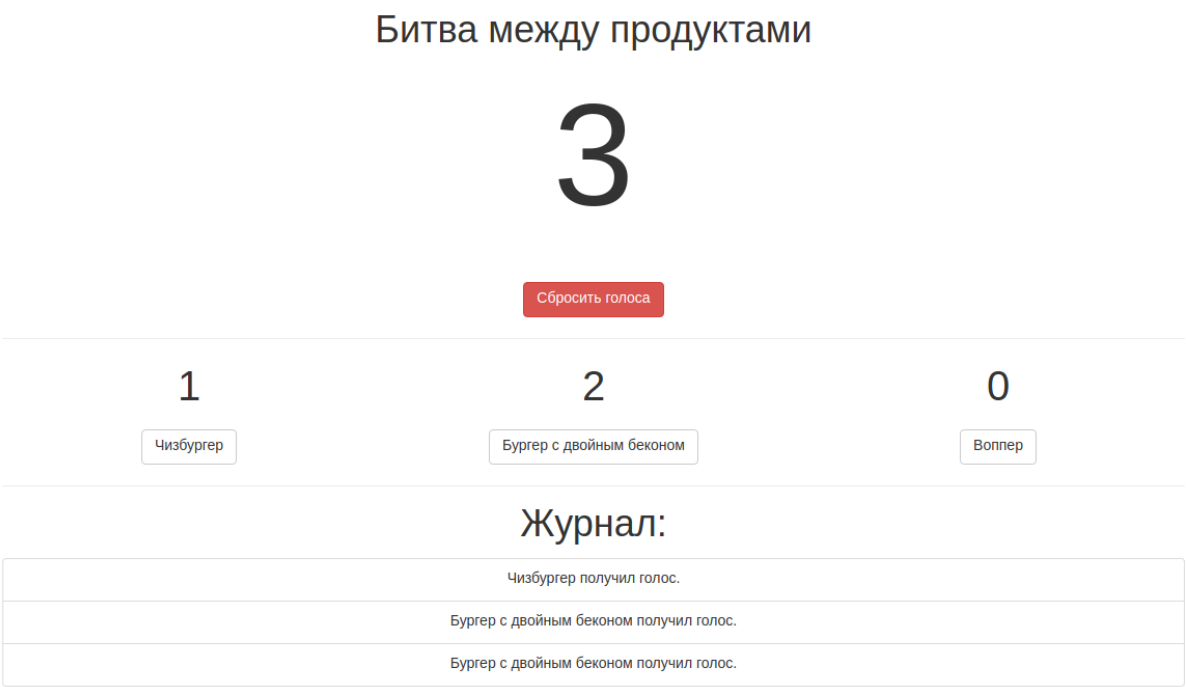
HTML

```
<body>
  <div class="container text-center">
    <h1>Битва между продуктами питания</h1>
    <p style="font-size: 140px;">
      {{ votes.count }}
    </p>
    <button type="button" class="btn btn-danger" @click="reset">Сбросить голоса</but\
ton>
    <hr>

    <div class="row">
      <food name="Чизбургер"></food>
      <food name="Бургер с двойным беконом"></food>
      <food name="Воппер"></food>
    </div>
    <hr>

    <h1>Журнал:</h1>
    <ul class="list-group">
      <li class="list-group-item" v-for="vote in votes.log">{{ vote }}</li>
    </ul>
  </div>
</body>
```

JavaScript

```
var bus = new Vue()

Vue.component('food', {
  template: '#food',
  props: ['name'],
  data: function () {
    return {
      votes: 0
    }
  },
  methods: {
    vote: function (event) {
      // вместо использования this.name
```

```
14        // мы можем получить доступ к элементу текста события
15        var food = event.srcElement.textContent;
16        this.votes++
17        bus.$emit('voted', food)
18      },
19      reset: function () {
20        this.votes = 0
21      }
22    },
23    created () {
24      bus.$on('reset', this.reset)
25    }
26  })
27  new Vue({
28    el: '.container',
29    data: {
30      votes: {
31        count: 0,
32        log: []
33      }
34    },
35    methods: {
36      countVote: function (food) {
37        this.votes.count++
38        this.votes.log.push(food + ' получил голос.')
39      },
40      reset: function () {
41        this.votes = {
42          count: 0,
43          log: []
44        }
45        bus.$emit('reset')
46      }
47    },
48    created () {
49      bus.$on('voted', this.countVote)
50    }
51  })
```



Не взаимодействие родительского и дочернего компонента

Предупреждение

Обратите внимание, что мы используем:

```
bus.$on('voted', this.countVote)
```

Если бы мы пытались использовать его подобным образом:

```
bus.$on('voted', function () {
  this.vote(food)
})
```

Это выбросит ошибку, поскольку `this` будет ограничен *экземпляром шины* вместо экземпляра текущего компонента.

8.5 Удаление обработчика событий

Для удаления одного или более обработчика событий, можно воспользоваться `$off`. Метод `$off([event, callback])` может использоваться по-разному:

1. `$off()`, без аргументов, удаляет все обработчики событий.

2. **`$off([event])`**, удаляет все обработчики событий для указанного события.
3. **`$off([event, callback])`** удаляет обработчик события для указанного колбэка.

Для просмотра все этого в действии, мы добавим кнопку *Стоп*, чтобы остановить подсчёт голосов и соответственно логирования. Мы добавим метод **`stop`** в экземпляре Vue.

```
new Vue({
  // ...
  methods: {
    // ...
    stop: function () {
      bus.$off(['voted'])
    }
  }
})
```

Если мы используем **`bus.$off(['voted'])`**, вы увидите, что после нажатия на *кнопку "Стоп"*, новые голоса не будут добавляться к общему количеству голосов. Также, они не отображаются в журнале. Хотя, если вы нажмёте на кнопку *Сбросить голоса*, голоса компонентов Food будут сброшены в 0.

Для отключения *обработчика сброса*, мы можем удалить всех обработчиков событий с использованием **`bus.$off()`**.

8.6 Обратно к историям

Помните пример с историями из предыдущих глав? Мы собирались синхронизировать *данные компонента* с *данными родительского компонента*. Теперь решение кажется очевидным.

Мы будем использовать компонент *Story* следующим образом:

```
<story
  v-for="story in stories"
  :story="story"
  :favorite="favorite"
  @update="updateFavorite"
>
</story>
```

В компоненте Story мы сгенерируем событие **`update`**, когда история отмечается как избранная. Избранная история будет передана при вызове `emit`.

Компонент Story

```
Vue.component('story', {
  // ...
  methods: {
    // ...
    updateFavorite: function () {
      // 'update' — это просто имя пользовательского события
      // это может быть что угодно, например, fav-update
      this.$emit('update', this.story)
    }
  }
  // ...
});
```

В родительском экземпляре мы добавим переменную `favorite` в данные. Кроме того, мы создадим новый метод, который будет обновлять переменную `favorite` при вызове.

Родительский экземпляр

```
new Vue({
  // ...
  data: {
    // ...
    favorite: {}
  },
  methods: {
    updateFavorite: function (story) {
      this.favorite = story;
    }
  }
})
```

Мистер Уибл сказал "Нарвалы изобрели шашлык.". Голосов за историю: 8. Проголосовать Добавить в избранное

Дарт Вейдер сказал "Тёмная сторона Силы сильнее.". Голосов за историю: 49. Проголосовать Добавить в избранное

Боромир сказал "Просто так в Мордор не попадёшь". Голосов за историю: 74. Проголосовать ★


```
    {
  "stories": [
    {
      "plot": "Моя лошадь потрясающая.",
      "writer": "Мистер Уибл",
      "upvotes": 28,
      "voted": false
    },
    {
      "plot": "Нарвалы изобрели шашлык.",
      "writer": "Мистер Уибл",
      "upvotes": 8,
      "voted": false
    },
    {
      "plot": "Тёмная сторона Силы сильнее.",
      "writer": "Дарт Вейдер",
      "upvotes": 49,
      "voted": false
    },
    {
      "plot": "Просто так в Мордор не попадёшь",
      "writer": "Боромир",
      "upvotes": 74,
      "voted": false
    }
  ],
  "favorite": {
    "plot": "Просто так в Мордор не попадёшь",
    "writer": "Боромир",
    "upvotes": 74,
    "voted": false
  }
}
```


Отметить как избранную можно только одну историю

Теперь желаемый результат достигнут, и пользователь может выбрать одну история для избранного, когда как он может проголосовать за истории, которые он хочет.

На заметку

Во Vue 2 привязки **всегда** односторонние. Для синхронизации между родительским и дочерним компонентами, вы должны использовать *события*.

Примеры кода

Вы можете найти примеры кода для этой главы на GitHub[3].

[3] https://github.com/hootlex/the-majesty-of-vuejs-2/tree/ru/codes/chapter8

8.7 Домашняя работа

Это самое сложное упражнение до сих пор, поэтому обязательно используйте всё, что вы узнал в этой книге. Создайте массив из 4 конных колесниц. Каждая колесница имеет имя (`name`) и количество лошадей (`horses`), от 1 до 4. Создайте компонент с именем “chariot”. Компонент “chariot” должен отображать имя колесницы и количество лошадей в ней. Он также должен иметь кнопку действия. Текст кнопки зависит от текущего выбранной колесницы.

Более конкретно про текст кнопки:

- ‘Выбрать колесницу’ — отображается пока, пользователь не выбрал колесницу
- ‘Отклонить лошадей’ — когда у колесницы меньше лошадей, чем у выбранной колесницы
- ‘Нанять лошадей’ — когда у колесницы больше лошадей, чем у выбранной колесницы
- ‘В путь!’ — когда колесница выбрана (эта кнопка будет отключена)

Пользователь должен иметь возможность выбрать колесницу, а затем выбрать между любыми другими колесницами, которые он хочет.

Пример сценария: Пользователь выбрал колесницу с **2** лошадьми и на кнопке надпись ‘В путь!’. У колесницы с **3** лошадьми есть ещё одна лошадь, поэтому на кнопке текст ‘Нанять лошадей’. У колесницы с **1** лошадью меньше лошадей, чем у колесницы пользователя, поэтому на кнопке будет ‘Отклонить лошадей’. Я думаю, вы поняли идею…

Подсказка

Вам нужно синхронизировать свойство **`currentChariot`** родительского и дочернего компонентов.

Подсказка

Для отключения кнопки используется атрибут **`disabled="true"`**. Вам нужно выяснить, как использовать его условно.

Магазин колесниц

У колесницы "Олимп" **4** лошадей!

Нанять лошадей

У колесницы "Саггита" **3** лошадей!

Нанять лошадей

У колесницы "Икар" **2** лошадей!

В путь!

У колесницы "Абраксас" **1** лошадей!

Отклонить лошадей

Пример вывода

Возможное решение

Вы можете найти потенциальное решение этого упражнения здесь[4].

[4] https://github.com/hootlex/the-majesty-of-vuejs-2/blob/ru/homework/chapter8.html

9. Привязки к классу и стилю

9.1 Привязка CSS-класса

9.1.1 Синтаксис объекта

Обычная необходимость в привязке данных — манипулирование классом элемента и его стилями. Для таких случаев вы можете использовать **`v-bind:class`**. Его можно использовать для применения классов в зависимости от условия, переключения их и/или применять многие из них, используя один связанный объект и др.

Директива **`v-bind:class`** принимает объект в следующем формате в качестве аргумента:

```
{
  'classA': true,
  'classB': false,
  'classC': true
}
```

и применяет все классы к элементу со значением **`true`**. Например, классы следующего элемента будут **`classA`** и **`classC`**.

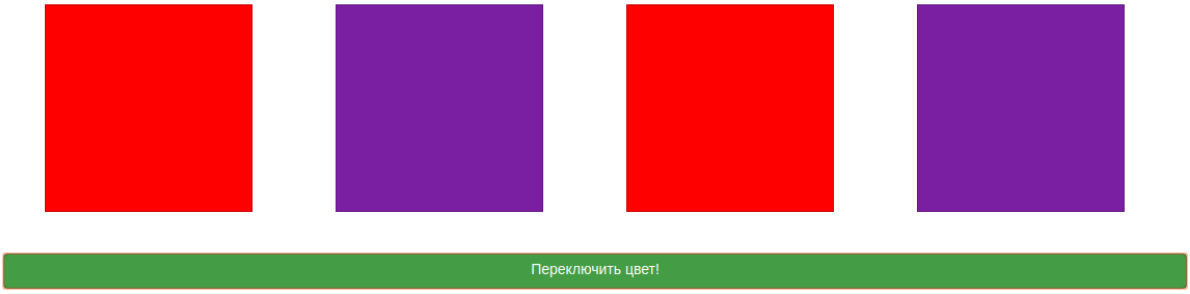
```
<div v-bind:class="elClasses"></div>
```

```
data: {
  elClasses: {
      'classA': true,
      'classB': false,
      'classC': true
    }
}
```

Для демонстрации использования **`v-bind`** с атрибутами класса, мы собираемся сделать пример с переключением классов. Используя директиву **`v-bind:class`**, будем динамически переключать класс элементов **`div`**.

```
1  <html>
2  <head>
3    <link
4      href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
5      rel="stylesheet"
6    >
7    <title>Привет, Vue</title>
8  </head>
9  <body>
10   <div class="container text-center">
11     <div class="box" v-bind:class="{ 'red' : color, 'blue' : !color }"></div>
12     <div class="box" v-bind:class="{ 'purple' : color, 'green' : !color }"></div>
13     <div class="box" v-bind:class="{ 'red' : color, 'blue' : !color }"></div>
14     <div class="box" v-bind:class="{ 'purple' : color, 'green' : !color }"></div>
15     <button type="button" v-on:click="flipColor" class="btn btn-block btn-success">
16       Переключить цвет!
17     </button>
18   </div>
19 </body>
20 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
21 <script type="text/javascript">
22 new Vue({
23   el: '.container',
24   data: {
25     color: true
26   },
27   methods: {
28     flipColor: function () {
29       this.color = !this.color;
30     }
31   }
32 });
33 </script>
34 <style type="text/css">
35 .red {
36   background: #ff0000;
37 }
38 .blue {
39   background: #0000ff;
40 }
41 .purple {
42   background: #7B1FA2;
43 }
```

```
44 .green {
45   background: #4CAF50;
46 }
47 .box {
48   float: left;
49   width: 200px;
50   height: 200px;
51   margin: 40px;
52   border: 1px solid rgba(0, 0, 0, .2);
53 }
54 </style>
55 </html>
```



Цвет переключаемых блоков

Цвет переключаемых блоков

Для нашего удобства мы применили класс **`box`** к каждому **`div`**. Что этот код делает на самом деле, так это «переключает» цвет блоков при нажатии на кнопку. При нажатии вызывается функция **`flipColor`**, которая меняет значение цвета (*color*), первоначально установленного в **`true`**. Затем **`v-bind:class`** будет переключать имя класса с ‘red’ на ‘blue’ или с ‘purple’ на ‘green’ в зависимости от истинности значения *color*. Таким образом, стиль будет применяться к каждому классу и даст нам желаемый цвет.

Директива `v-bind:class` может быть использована вместе с обычным атрибутом класса.

Итак, в нашем примере у элементов `div` всегда есть класс `box` и один из следующих классов в зависимости от условия: `red`, `blue`, `purple` или `green`.

9.1.2 Синтаксис массива

Мы также можем применить список классов к элементу, используя массив с именами.

```
<div v-bind:class="['classA', 'classB', anotherClass]"></div>
```

Применение класса по условию также может быть организовано с помощью встроенного `if` внутри массива.

```
<div v-bind:class="['classA', condition ? 'classB' : '']"></div>
```

На заметку

Встроенный if обычно называют **тернарным оператором**, **условный оператор** или **тернарный if**.

Условный (тернарный) оператор — единственный в JavaScript оператор, принимающий три операнда.

Синтаксис **тернарного оператора** — `condition ? expression1 : expression2`. Если условие (condition) равняется `true`, оператор возвращает значение первого выражения (expression1), в противном случае он вернёт значение второго выражения (expression2).

Использование встроенного `if` в примере с переключением цветов будет выглядит так:

```
<div class="container text-center">
  <div class="box" v-bind:class="[ color ?  'red' : 'blue' ]"></div>
  <div class="box" v-bind:class="[ color ?  'purple' : 'green' ]"></div>
  <div class="box" v-bind:class="[ color ?  'red' : 'blue' ]"></div>
  <div class="box" v-bind:class="[ color ?  'purple' : 'green' ]"></div>
  <button type="button" v-on:click="flipColor" class="btn btn-block btn-success">
    Переключить цвет!
  </button>
</div>
```

```
new Vue({
  el: '.container',
  data: {
    color: true
  },
  methods: {
    flipColor: function () {
      this.color = !this.color;
    }
  }
});
```

Подсказка

Чтобы использовать имя класса вместо переменной внутри массива с классами, используйте одинарные кавычки. `v-bind:class="[variable, 'classname']"`

9.2 Связывание стилей

9.2.1 Синтаксис объекта

Синтаксис объекта для `v-bind:style` довольно прост; он выглядит почти как CSS, за исключением того, что это JavaScript-объект. Мы собираемся использовать сокращение, которое Vue предоставляет для ранее используемой директивы `v-bind(:)`.

```
<!-- сокращение -->
<div :style="niceStyle"></div>
```

```
data: {
  niceStyle: {
    color: 'blue',
    fontSize: '20px'
  }
}
```

Мы также можем объявить свойства стиля внутри встроенного **:style=**”...“.

```
<div :style="{'color': 'blue', fontSize: '30px'}">...</div>
```

Мы можем даже **сослаться на переменную** внутри объекта стиля:

```
<!-- Переменная 'niceStyle' — то же самое, что мы использовали в предыдущем примере \
-->
<div :style="{'color': niceStyle.color, fontSize: niceStyle.fontSize}"></div>
```

Какие приятные стили, милорд!

Привязка к объекту стиля

Часто хорошей идеей будет использовать объект стиля и привязывать его, тогда шаблон становится чище.

9.2.2 Синтаксис массива

При использовании встроенного синтаксиса массива для `v-bind:style` мы можем применить несколько объектов стиля к одному и тому же элементу, что означает, что каждый элемент списка будет иметь `color` и `font-size` в `niceStyle`, и стиль шрифта в `badStyle`.

```
<!-- shorthand -->
<div :style="[niceStyle, badStyle]"></div>
```

```
data: {
  niceStyle: {
    color: 'blue',
    fontSize: '30px'
  },
  badStyle: {
    fontStyle: 'italic'
  }
}
```

Информация для продвинутых

Когда вы используете свойство CSS, требующее браузерных префиксов в **`v-bind:style`**, например, *transform*, Vue автоматически обнаружит и добавит соответствующие префиксы к применяемым стилям.

Вы можете найти дополнительную информацию о префиксах здесь[1].

9.3 Привязки в действии

```
<html>
<head>
  <link
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
    rel="stylesheet"
  >
  <title>Привет, Vue</title>
</head>
<body class="container-fluid">
  <div id="app">
    <ul>
        <li
          :class="{'completed' : task.done}"
          :style="styleObject"
          v-for="(task, index) in tasks"
          :key="index"
        >
          {{task.body}}
          <button type="button" @click="completeTask(task)" class="btn btn-success">
            Просто сделай это!
```

[1] https://developer.mozilla.org/ru/docs/Glossary/Vendor_Prefix

```
21            </button>
22          </li>
23      </ul>
24    </div>
25  </body>
26  <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.1\
27  7/vue.js"></script>
28  <script type="text/javascript">
29  new Vue({
30    el: '#app',
31    data: {
32      tasks: [
33        { body: 'Покормить лошадей', done: true },
34        { body: 'Помыть броню', done: true },
35        { body: 'Поточить меч', done: false },
36      ],
37      styleObject: {
38        fontSize: '25px'
39      }
40    },
41    methods: {
42      completeTask: function (task) {
43        task.done = !task.done;
44      }
45    },
46  });
47  </script>
48  <style type="text/css">
49  .completed {
50    text-decoration: line-through;
51  }
52  </style>
53  </html>
```

В приведённом выше примере представлен массив объектов с задачами *tasks* и **`styleObject`**, ***содержащий только одно свойство. С помощью `**v-for**` отрисовывается список задач, где каждая задача имеет свойство `*done`*** с логическим значением. В зависимости от значения ***done*** применяется класс **`completed`**, как и раньше. Если задача завершена, применяется стиль *CSS* и задача отображается перечёркнутой из-за применения свойства **`text-decoration` со значением **`line-through`**. Каждая задача сопровождается кнопкой, обрабатывающей событие *click*, которая вызывает метод для изменения статуса завершения задачи. Атрибут **`style`** связан с **`styleObject`**, что приводит к изменению свойства **`font-size`** для всех задач. Как вы можете видеть, метод **`completeTasks`** принимает в качестве параметра

`task`.

- ~~Покормить лошадей~~ Просто сделай это!
- ~~Помыть броню~~ Просто сделай это!
- Поточить меч Просто сделай это!

Стилизация выполненных задач

Примеры кода

Вы можете найти примеры кода для этой главы на GitHub[2].

[2] https://github.com/hootlex/the-majesty-of-vuejs-2/tree/ru/codes/chapter9

9.4 Домашняя работа

Весёлое и, возможно, сложное упражнение для этой главы. Создайте поле ввода, в котором пользователь может выбрать цвет. Когда цвет выбран, примените его к элементу на ваш выбор. Вот и всё, **давайте красить!! :)**

Подсказка

Для удобства можно использовать `input type="color"` (поддерживается в большинстве браузеров).

Пример вывода

Возможное решение

Вы можете найти потенциальное решение этого упражнения здесь[3].

[3] https://github.com/hootlex/the-majesty-of-vuejs-2/blob/ru/homework/chapter9.html

II Использование API

10. Предисловие

В этой главе мы копнём немного глубже и продемонстрируем, как мы можем использовать API в контексте Vue.js.

Следуя примерам с *историями* из предыдущих глав, мы теперь собираемся использовать реальные данные, поступающие от внешнего источника.

Для использования реальных данных необходимо использовать базу данных. Предполагается, что вы уже знаете, как создавать базу данных, это не будет описано в книге. Для работы с примерами книги мы обо всём позаботились и создали одну базу данных, чтобы начать её использовать.

10.1 CRUD

Предположим, у нас есть база данных и можем выполнять так называемые операции CRUD (Create, Read, Update, Delete — создание, чтение, обновление, удаление). В частности, мы хотим:

- **Создавать (Create)** новые истории в базе данных
- **Получать (Read)** существующие истории
- **Обновлять (Update)** подробности существующей истории (например, 'upvotes')
- **Удалять (Delete)** истории, которые нам не нравятся

Поскольку Vue.js — фронтенд-фреймворк JavaScript, он не может непосредственно подключаться к базе данных. Для доступа к базе данных нам нужен слой между Vue.js и БД. Этим слоем будет API (Application Program Interface — программный интерфейс приложения).

10.2 API

Поскольку эта книга посвящена Vue.js, а не разработке API, мы предоставим вам демонстрационный API, созданный на Laravel[1]. Laravel — один из самых мощных PHP-фреймворков наряду с Symfony, Nette, CodeIgniter и Yii2. Вы можете создать собственный API, используя *любой язык или фреймворк*, который вам по душе. Я использую Laravel, потому что он простой, у него отличное сообщество и вообще он просто потрясающий! :)

По этой причине мы настоятельно рекомендуем вам использовать *демонстрационный API*, созданный исключительно для демонстрации примеров книги.

[1] https://laravel.com/

10.2.1 Загрузка кода книги

Для использования нашего API, необходимо загрузить код книги и запустить сервер. Для этого следуйте инструкциям ниже.

- Откройте свой терминал и создайте каталог (мы создадим его по пути '~/themajestyofvuejs2')

```
mkdir ~/themajestyofvuejs2
```

- Скачайте исходный код с GitHub

```
cd ~/themajestyofvuejs2
git clone https://github.com/hootlex/the-majesty-of-vuejs-2 .
```

В качестве альтернативного варианта вы можете посетить репозиторий на GitHub[2] и скачать zip-архив. Затем распакуйте его содержимое в созданном каталоге.

- Перейдите к текущей главе 'apis' в корне только созданного каталога.

```
cd ~/themajestyofvuejs2/apis/stories
```

- Выполните установочный скрипт

```
sh setup.sh
```

- Теперь у вас есть база данных, заполненная фиктивными данными, а также полностью рабочий сервер, **доступный по адресу *http://localhost:3000***!

Если вы хотите изменить сервер (хост, порт и т.д.), вы можете произвести установку вручную. Ниже приведён исходный код нашего скрипта.

[2] https://github.com/hootlex/the-majesty-of-vuejs-2

Скрипт установки setup.sh

```
# Перейти в каталог главы
$ cd ~/themajestyofvuejs2/apis/stories

# Установить зависимости
$ composer install

# Создать базу данных
$ touch database/database.sqlite;

# Запустить миграции и заполнение её начальными данными
$ php artisan migrate;
$ php artisan db:seed;

# Запуск сервера
$ php artisan serve --port=3000 --host localhost;
```

Отлично! Теперь у вас *полноценный API* и база данных с добавленными прекрасными историями.

Примечание

Если вы используете Vagrant, вам нужно запустить сервер на **host '0.0.0.0'**. Затем вы можете получить доступ к своему серверу по IP-адресу сборки (box) Vagrant.

Если, к примеру, сборка Vagrant настроена на *192.168.10.10* и вы выполняете в терминале

```
$ php artisan serve --port=3000 --host 0.0.0.0;
```

то сайт будет доступен по адресу `192.168.10.10:3000`.

Если вы загрузили демонстрационный API, вы можете перейти к следующему разделу.

Если вы решили создать собственный API, вам нужно создать таблицу для хранения `историй`. Должны присутствовать следующие столбцы.

Название столбца	Тип
`id`	Целое число с автоинкрементом
`plot`	Строка
`writer`	Строка
`upvotes`	Целое число без знака

Не забудьте заполнить фиктивными данными таблицы, чтобы использовать их в следующих

примерах.

10.2.2 Конечные точки API

Конечная точка (endpoint) — это просто URL-адрес. Когда вы переходите на *http://example.com/foo/bar*, то это конечная точка и вам нужно просто вызывать её `/foo/bar`, потому что домен будет одинаковым для всех конечных точек.

Для управления ресурсом историй (**`Story`**) нам нужно 5 конечных точек. Каждая конечная точка соответствует конкретному действию.

Метод HTTP	**URI**	**Действие**
GET/HEAD	api/stories	*Получает* все истории
GET/HEAD	api/stories/{id}	*Получает* определённую историю
POST	api/stories	*Создаёт* новую историю
PUT/PATCH	api/stories/{id}	*Обновляет* существующую историю
DELETE	api/stories/{id}	*Удаляет* определённую историю

Как указано в приведённой выше таблице, для получения всех 'историй' мы должны сделать HTTP-запрос GET или HEAD на

`api/stories`. Для обновления существующей истории, нам требуется сделать HTTP-запрос PUT или PATCH на `api/stories/{storyID}`, предоставив данные истории, которые мы хотим обновить, заменив `{storyID}` на идентификатор обновляемой истории. Та же самая логика применяется ко всем конечным точкам. Думаю, вы поняли идею.

Предполагая, что ваш сервер работает на `http://localhost:3000`, вы можете посмотреть на список всех историй в JSON-формате, перейдя по адресу `http://localhost:3000/api/stories`.

```
[
  {
    "id": 1,
    "plot": "Вы были правы в одном, Мастер; переговоры были недолгими.",
    "upvotes": "747",
    "writer": "Оби-Ван Кеноби",
    "created_at": "2018-08-29 12:54:22",
    "updated_at": "2018-08-29 12:54:22"
  },
  {
    "id": 2,
    "plot": "Справедливость — это дело, за которое стоит сражаться",
    "upvotes": "896",
    "writer": "Оби-Ван Кеноби",
    "created_at": "2018-08-29 12:54:22",
    "updated_at": "2018-08-29 12:54:22"
  },
  {
    "id": 3,
    "plot": "Всегда найдётся рыба крупней.",
    "upvotes": "882",
    "writer": "Квай-Гон Джин",
    "created_at": "2018-08-29 12:54:22",
    "updated_at": "2018-08-29 12:54:22"
  },
  {
    "id": 4,
    "plot": "Вы были правы в одном, Мастер; переговоры были недолгими.",
    "upvotes": "754",
    "writer": "Оби-Ван Кеноби",
    "created_at": "2018-08-29 12:54:22",
    "updated_at": "2018-08-29 12:54:22"
  },
  {
    "id": 5,
    "plot": "Если противник превосходит тебя численно, лучший способ добиться победы — действовать неожиданно",
    "upvotes": "872",
    "writer": "Квай-Гон Джинн",
    "created_at": "2018-08-29 12:54:22",
    "updated_at": "2018-08-29 12:54:22"
  },
  {
    "id": 6,
    "plot": "Вы проиграли, Ваше величество. Я джедай, каким и мой отец был!",
    "upvotes": "456",
    "writer": "Люк Скайуокер",
    "created_at": "2018-08-29 12:54:22",
    "updated_at": "2018-08-29 12:54:22"
```


Ответ в формате JSON

Подсказка

Чтение необработанных (raw) JSON-данных в браузере может быть трудным. Всегда легче читать **правильно сформулированный JSON**. Для браузера Chrome существуют множество отличных расширений, которые могут форматировать необработанные JSON-данные в удобочитаемом древовидном виде.

Я использую расширение JSON Formatter[3], потому что оно поддерживает подсветку синтаксиса и отображает JSON в древовидном представлении, где узлы дерева могут свёрнуты или раскрыты по клику на иконку с треугольником слева от каждого узла. Оно также предоставляет кнопку для переключения на просмотр исходных (необработанных) данных.

Вы можете выбрать любое расширение, которое вам больше нравится, главное вам следует **обязательно использовать его!**

[3] https://chrome.google.com/webstore/detail/json-formatter/bcjindcccaagfpapjjmafapmmgkkhgoa

11. Работа с реальными данными

Настало время на деле использовать нашу базу данных и выполнять ранее упомянутые операции CRUD. Мы воспользуемся последним примером из главы «Компоненты», но на этот раз, конечно, наши данные будут получены из внешнего исхочника. Для обмена данными с сервером нам нужно выполнять асинхронные запросы HTTP (Ajax).

На заметку

AJAX — технология, позволяющая асинхронно обновлять веб-страницы через обмен небольшими данными с сервером в реальности.

11.1 Получение данных асинхронно

Воспользуемся моментом и взглянем на последний пример из главы «Компоненты». Как вы видите, массив **stories** жёстко задан в коде внутри объекта `data` экземпляра `Vue`.

Жёстко закодированный массив историй

```
new Vue({
  data: {
    stories: [
      {
        plot: 'Моя лошадь потрясающая.',
        writer: 'Мистер Уибл',
      },
      {
        plot: 'Нарвалы придумали кебаб.',
        writer: 'Мистер Уибл',
      },
      // ...
    ]
  }
})
```

Сейчас мы хотим получать существующие истории с сервера.

Для этого мы выполним HTTP-запрос GET, используя сначала jQuery. Позже в этой главе мы перейдём к vue-resource[1], чтобы увидеть разницу между ними.

[1] https://github.com/vuejs/vue-resource

Чтобы сделать вызов AJAX будем использовать jQuery-функцию **`$.get()`**, которая загружает данные с сервера с помощью HTTP-запроса GET. Полную документацию для **`$.get()`** можно посмотреть здесь[2].

Важная информация

vue-resource — плагин для `Vue.js`, предоставляющий возможность совершать веб-запросы и обрабатывать ответы.

Синтаксис метода **`$.get()`** такой:

```
$.get(
  url,
  success
);
```

Который на самом деле является сокращением:

```
$.ajax({
  url: url,
  success: success
});
```

Итак, что мы делаем дальше? Мы хотим получить истории с сервера, используя **`$.get('/api/stories')`**, и сохранить данные ответа в массиве `stories`.

Это распространённый приём, нам надо сделать вызов **после того, как экземпляр будет готов**. Помните ли вы хуки жизненного цикла?

Есть *хук* под названием **`mounted`**, который вызывается сразу после монтирования экземпляра.

Предупреждение

Хук **`mounted`** — это не то же самое, что **`$(document).ready()`** в jQuery. При использовании **`mounted`**, **нет никакой гарантии, что будет в документе**. Если нужно что-то выполнить, как только страница объектной модели документа (Document Object Model, DOM) будет загружена полностью, вы можете использовать это:

```
mounted: function () {
 this.$nextTick(function () {
  // код, предполагающий, что элемент this.$el находится в документе (DOM)
 })
}
```

[2] https://api.jquery.com/jquery.get/

Давайте посмотрим на него в действии.

```
<div id="app">
  <div class="container">
    <h1>Давайте послушаем несколько историй!</h1>
    <ul class="list-group">
      <story v-for="story in stories" :key="story.id" :story="story">
      </story>
    </ul>
    <pre>{{ $data }}</pre>
  </div>
</div>
<template id="template-story-raw">
  <li class="list-group-item">
    {{ story.writer }} сказал "{{ story.plot }}"
    <span>{{story.upvotes}}</span>
  </li>
</template>
```

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
<script src="https://code.jquery.com/jquery-2.1.4.min.js"></script>
<script type="text/javascript">
Vue.component('story', {
  template: "#template-story-raw",
  props: ['story'],
});

var vm = new Vue({
  el: '#app',
  data: {
    stories: []
  },
  mounted: function () {
    var vm = this;
    $.get('/api/stories', function (data) {
      vm.stories = data;
    })
  }
})
</script>
```

Начнём с подключения jQuery из cdnjs[3]. Затем внутри хука `mounted` мы выполняем запрос

[3] https://cdnjs.com/libraries/jquery/

`GET`. После успешного завершения запроса мы сохраняем данные ответа (внутри колбэка) в массив `stories`.

Давайте послушаем несколько историй!

- Йода сказал "Страх — это путь на Тёмную сторону. Страх порождает гнев. Гнев приводит к ненависти. Ненависть приводит к страданиям" 645
- Дарт Вейдер сказал "Старая система мертва, сенатор. И вы поступите достаточно благоразумно, если не будете упорствовать и вольетесь в новую" 213
- Падме Амидала сказал "Вот так и умирает свобода — под гром аплодисментов..." 769
- Хан Соло сказал "Никогда не говори мне о шансах!" 880
- Оби-Ван Кеноби сказал "Все умирает. Даже звезды сгорают" 844
- Йода сказал "Размер не имеет значения. Посмотри на меня. По размеру меня судишь, так? Гмм? Гмм. А не должен ты. Ибо мой союзник — Сила, и могущественный союзник она" 480
- Принцесса Лея сказал "Кто-нибудь уберет этот коврик с моей дороги?" 219
- Шми Скайуокер сказал "Теперь, будь храбр и не оглядывайся назад. Не оглядывайся" 876
- Оби-Ван Кеноби сказал "Если ты наделяешь себя властью забирать жизни, жаждой власти и владением другими, то у тебя нет ничего" 456
- Оби-Ван Кеноби сказал "Справедливость — это дело, за которое стоит сражаться" 591
- Шми Скайуокер сказал "Теперь, будь храбр и не оглядывайся назад. Не оглядывайся" 38
- Дарт Вейдер сказал "Ты недооцениваешь силу Темной стороны" 509
- Йода сказал "Решение принимаешь ты один. Но помнить ты должен, что делаешь это также и за других, кто стоит за твоим плечом" 897
- Йода сказал "Воистину удивителен ребёнка разум." 419
- Квай-Гон Джинн сказал "То, что кто-то говорит, и то, что он испытывает, не обязательно и не всегда одно и то же" 575
- Йода сказал "Боязнь потери к Темной стороне привести может" 729
- Квай-Гон Джин сказал "Способность болтать — ещё не признак интеллекта" 273
- Йода сказал "Мой союзник - Сила - могучий союзник. Жизнь её сотворила и вырастила. Её энергия окружает нас и связывает. Сила вокруг тебя, везде, между мной, тобой, деревом, камнем..." 288

Получение историй

Следует обратить внимание, что внутри колбэка мы ссылаемся на переменную `stories`, используя `vm.stories` вместо `this.stories`. Мы это делаем это потому, что переменная **`this`** не связана с экземпляром **`Vue`** внутри колбэка. Итак, мы сохраняем весь экземпляр **`Vue`** в переменную с именем **`vm`**, чтобы получить к ней доступ из любого места в коде. Чтобы узнать больше о **`this`**, ознакомьтесь с документацией[4].

Предупреждение

Для того, чтобы извлекать данные с API, откройте файл в браузере, после запуска сервера, предоставляющего API. Если вы используете API из нашего репозитория, вам следует открыть файл по адресу `http://localhost:3000/stories.html`. В случае вашего собственного API, убедитесь, что файл `stories.html` участвует в маршруте вашего приложения. Это не сработает, если вы откроете файл `stories.html` непосредственно в браузере.

[4] https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/this

11.2 Рефакторинг

Наличие большого количества кода может сбивать с толку, если оно правильно не отображается, как в вашем текстовом редакторе, так и в браузере. По этой причине мы хотим улучшить код пример, изменив отрисовку списка историй с использованием элемента **`<table>`** вместо **`<ul>`**.

```
<div id="app">
  <table class="table table-striped">
    <tr>
      <th>#</th>
      <th>Сюжет</th>
      <th>Автор</th>
      <th>Кол-во голосов</th>
      <th>Действия</th>
    </tr>
    <tr v-for="story in stories" :key="story.id" is="story" :story="story"></tr>
  </table>
</div>
<template id="template-story-raw">
  <tr>
    <td>{{story.id}}</td>
    <td><span>{{story.plot}}</span></td>
    <td><span>{{story.writer}}</span></td>
    <td>{{story.upvotes}}</td>
  </tr>
</template>
<p class="lead">Вот список всех ваших историй.</p>
<pre>{{ $data }}</pre>
```

Однако есть проблема.

Давайте послушаем несколько историй!

#	Сюжет	Автор	Кол-во голосов	Действия
1	Страх — это путь на Тёмную сторону. Страх порождает гнев. Гнев приводит к ненависти. Ненависть приводит к страданиям	Йода	645	
2	Старая система мертва, сенатор. И вы поступите достаточно благоразумно, если не будете упорствовать и вольетесь в новую	Дарт Вейдер	213	
3	Вот так и умирает свобода — под гром аплодисментов...	Падме Амидала	769	
4	Никогда не говори мне о шансах!	Хан Соло	880	
5	Все умирает. Даже звезды сгорают	Оби-Ван Кеноби	844	
6	Размер не имеет значения. Посмотри на меня. По размеру меня судишь, так? Гмм? Гмм. А не должен ты. Ибо мой союзник — Сила, и могущественный союзник она	Йода	480	
7	Кто-нибудь уберет этот коврик с моей дороги?	Принцесса Лея	219	
8	Теперь, будь храбр и не оглядывайся назад. Не оглядывайся	Шми Скайуокер	876	
9	Если ты наделяешь себя властью забирать жизни, жаждой власти и владением другими, то у тебя нет ничего	Оби-Ван Кеноби	456	
10	Справедливость — это дело, за которое стоит сражаться	Оби-Ван Кеноби	591	
11	Теперь, будь храбр и не оглядывайся назад. Не оглядывайся	Шми Скайуокер	38	
12	Ты недооцениваешь силу Темной стороны	Дарт Вейдер	509	
13	Решение принимаешь ты один. Но помнить ты должен, что делаешь это также и за других, кто стоит за твоим плечом	Йода	897	
14	Воистину удивителен ребёнка разум.	Йода	419	
15	То, что кто-то говорит, и то, что он испытывает, не обязательно и не всегда одно и то же	Квай-Гон Джинн	575	
16	Боязнь потери к Темной стороне привести может	Йода	729	
17	Способность болтать — ещё не признак интеллекта	Квай-Гон Джин	273	
18	Мой союзник - Сила - могучий союзник. Жизнь её сотворила и вырастила. Её энергия окружает нас и связывает. Сила вокруг тебя, везде, между мной, тобой, деревом, камнем...	Йода	288	
19	То, что кто-то говорит, и то, что он испытывает, не обязательно и не всегда одно и то же	Квай-Гон Джинн	569	
20	Ты недооцениваешь силу Темной стороны	Дарт Вейдер	823	

Отрисовка проблем

Примечание

В скриншоте выше скрыт столбец с действиями, т.к. пока их нет.

Наша таблица отрисовывается неправильно, но почему?[5]

> *Некоторые HTML-элементы, к примеру `<table>`, имеют ограничения на то, какие элементы могут быть внутри них. Пользовательские элементы не входят в разрешённый список и поэтому не будут отрисованы корректно. В подобных случаях вам нужно использовать специальный атрибут* **`is`** *для обозначения пользовательского элемента.*

Следовательно, для решения этой проблемы нам нужно использовать **специальный атрибут `is`** во Vue.

[5] https://goo.gl/Xr9RoQ

```
<table>
  <tr is="my-component"></tr>
</table>
```

И таким образом, наш пример будет выглядеть так:

```
<tr v-for="story in stories" :key="story.id" is="story" :story="story"></tr>
```

Истории

#	Сюжет	Автор	Кол-во голосов
1	Страх — это путь на Тёмную сторону. Страх порождает гнев. Гнев приводит к ненависти. Ненависть приводит к страданиям	Йода	645
2	Старая система мертва, сенатор. И вы поступите достаточно благоразумно, если не будете упорствовать и вольетесь в новую	Дарт Вейдер	213
3	Вот так и умирает свобода — под гром аплодисментов...	Падме Амидала	769
4	Никогда не говори мне о шансах!	Хан Соло	880
5	Все умирает. Даже звезды сгорают	Оби-Ван Кеноби	844
6	Размер не имеет значения. Посмотри на меня. По размеру меня судишь, так? Гмм? Гмм. А не должен ты. Ибо мой союзник — Сила, и могущественный союзник она	Йода	480
7	Кто-нибудь уберет этот коврик с моей дороги?	Принцесса Лея	219
8	Теперь, будь храбр и не оглядывайся назад. Не оглядывайся	Шми Скайуокер	876
9	Если ты наделяешь себя властью забирать жизни, жаждой власти и владением другими, то у тебя нет ничего	Оби-Ван Кеноби	456
10	Справедливость — это дело, за которое стоит сражаться	Оби-Ван Кеноби	591
11	Теперь, будь храбр и не оглядывайся назад. Не оглядывайся	Шми Скайуокер	38
12	Ты недооцениваешь силу Темной стороны	Дарт Вейдер	509
13	Решение принимаешь ты один. Но помнить ты должен, что делаешь это также и за других, кто стоит за твоим плечом	Йода	897
14	Воистину удивителен ребёнка разум.	Йода	419
15	То, что кто-то говорит, и то, что он испытывает, не обязательно и не всегда одно и то же	Квай-Гон Джинн	575
16	Боязнь потери к Темной стороне привести может	Йода	729
17	Способность болтать — ещё не признак интеллекта	Квай-Гон Джин	273

Корректная отрисовка таблицы

Что ж, выглядит получше!

11.3 Обновление данных

Раньше у нас была функция, которая позволяла пользователя голосовать за любую понравившуюся историю. Но теперь мы хотим чего-то большего. Наша задача — информирование сервера каждый раз при любом голосовании за историю, тем самым гарантируя, что голос за историю будет обновлён также в базе данных.

Для обновления существующей истории нам нужно сделать HTTP-запрос PATCH или PUT к `api/stories/{storyID}`.

Внутри функции **`upvoteStory`**, которая будет создана, мы собираемся сделать запрос HTTP после увеличения голосов истории. Мы передадим недавно обновлённую переменную **`story`** в данных запроса (Request Payload) для обновления данных в нашем сервере.

```
<td>
  <div class="btn-group">
    <button type="button" @click="upvoteStory(story)" class="btn btn-primary">
      Проголосовать
    </button>
  </div>
</td>
```

```
Vue.component('story', {
  template: '#template-story-raw',
  props: ['story'],
  methods: {
    upvoteStory: function (story) {
      story.upvotes++;
      $.ajax({
        url: '/api/stories/'+story.id,
        type: 'PATCH',
        data: story,
      });
    }
  },
})
```

Мы вернули метод **`upvote`** и разметили его внутри компонента **`story`**. Выполняя PATCH-запрос сейчас, предоставляющий новые данные, сервер обновляет счётчик **`upvotes`**.

Истории

#	Сюжет	Автор	Кол-во голосов	Действия
1	Страх — это путь на Тёмную сторону. Страх порождает гнев. Гнев приводит к ненависти. Ненависть приводит к страданиям	Йода	646	Проголосовать
2	Старая система мертва, сенатор. И вы поступите достаточно благоразумно, если не будете упорствовать и вольетесь в новую	Дарт Вейдер	213	Проголосовать
3	Вот так и умирает свобода — под гром аплодисментов...	Падме Амидала	769	Проголосовать
4	Никогда не говори мне о шансах!	Хан Соло	880	Проголосовать
5	Все умирает. Даже звезды сгорают	Оби-Ван Кеноби	844	Проголосовать
6	Размер не имеет значения. Посмотри на меня. По размеру меня судишь, так? Гмм? Гмм. А не должен ты. Ибо мой союзник — Сила, и могущественный союзник она	Йода	480	Проголосовать
7	Кто-нибудь уберет этот коврик с моей дороги?	Принцесса Лея	219	Проголосовать
8	Теперь, будь храбр и не оглядывайся назад. Не оглядывайся	Шми Скайуокер	876	Проголосовать
9	Если ты наделяешь себя властью забирать жизни, жаждой власти и владением другими, то у тебя нет ничего	Оби-Ван Кеноби	456	Проголосовать
10	Справедливость — это дело, за которое стоит сражаться	Оби-Ван Кеноби	591	Проголосовать
11	Теперь, будь храбр и не оглядывайся назад. Не оглядывайся	Шми Скайуокер	38	Проголосовать
12	Ты недооцениваешь силу Темной стороны	Дарт Вейдер	509	Проголосовать
13	Решение принимаешь ты один. Но помнить ты должен, что делаешь это также и за других, кто стоит за твоим плечом	Йода	897	Проголосовать
14	Воистину удивителен ребёнка разум.	Йода	419	Проголосовать

Голосование за истории

11.4 Удаление данных

Давайте перейдём к другой части функциональности, у списка историй должна быть возможность удаление истории, которая нам не понравилась. Для удаления истории из массива и DOM, нам нужно найти её и удалить из массива **`stories`**.

```
<td>
  <div class="btn-group">
    <button type="button" @click="upvoteStory(story)" class="btn btn-primary">
      Проголосовать
    </button>
    <button type="button" @click="deleteStory(story)" class="btn btn-danger">
      Удалить
    </button>
  </div>
</td>
```

Мы добавляем кнопку *Удалить* в столбец с *действиями*, привязанную к методу удаления истории. Метод **`deleteStory`** будет там:

```
Vue.component('story', {
  // ...
  methods: {
    // ...
    deleteStory: function (story) {
      // ищем нужную историю
      var index = vm.stories.indexOf(story);

      // удаляем её
      vm.stories.splice(index, 1)
    }
  }
  // ...
})
```

Но, естественно, таким образом, мы удалим историю только из временного хранилища в браузере. Для удаления истории из базы данных, нам нужно выполнить AJAX-запрос типа DELETE.

```
Vue.component('story', {
  // ...
  methods: {
    // ...
    deleteStory: function (story) {
      // ищем нужную историю
      var index = vm.stories.indexOf(story);

      // удаляем её
      vm.stories.splice(index, 1)

      // делаем запрос на удаление (DELETE)
      $.ajax({
        url: '/api/stories/' + story.id,
        type: 'DELETE'
      });
    },
  }
  // ...
})
```

Мы передаём URL-адрес, как и раньше. Тип запроса должен быть `DELETE`. Теперь наш метод готов, и мы можем удалить историю из базы данных, также как из DOM.

Истории

#	Сюжет	Автор	Кол-во голосов	Действия
1	Страх — это путь на Тёмную сторону. Страх порождает гнев. Гнев приводит к ненависти. Ненависть приводит к страданиям	Йода	646	Проголосовать Удалить
2	Старая система мертва, сенатор. И вы поступите достаточно благоразумно, если не будете упорствовать и вольетесь в новую	Дарт Вейдер	213	Проголосовать Удалить
3	Вот так и умирает свобода — под гром аплодисментов...	Падме Амидала	769	Проголосовать Удалить
4	Никогда не говори мне о шансах!	Хан Соло	880	Проголосовать Удалить
5	Все умирает. Даже звезды сгорают	Оби-Ван Кеноби	844	Проголосовать Удалить
6	Размер не имеет значения. Посмотри на меня. По размеру меня судишь, так? Гмм? Гмм. А не должен ты. Ибо мой союзник — Сила, и могущественный союзник она	Йода	480	Проголосовать Удалить
7	Кто-нибудь уберет этот коврик с моей дороги?	Принцесса Лея	219	Проголосовать Удалить
8	Теперь, будь храбр и не оглядывайся назад. Не оглядывайся	Шми Скайуокер	876	Проголосовать Удалить
9	Если ты наделяешь себя властью забирать жизни, жаждой власти и владением другими, то у тебя нет ничего	Оби-Ван Кеноби	456	Проголосовать Удалить
10	Справедливость — это дело, за которое стоит сражаться	Оби-Ван Кеноби	591	Проголосовать Удалить
11	Теперь, будь храбр и не оглядывайся назад. Не оглядывайся	Шми Скайуокер	38	Проголосовать Удалить
12	Ты недооцениваешь силу Темной стороны	Дарт Вейдер	509	Проголосовать Удалить
13	Решение принимаешь ты один. Но помнить ты должен, что делаешь это также и за других, кто стоит за твоим плечом	Йода	897	Проголосовать Удалить
14	Воистину удивителен ребёнка разум.	Йода	419	Проголосовать Удалить

Голосование и удаление за истории

Вот и всё на данный момент. Мы продолжим наш пример в следующей главе, путём расширения функциональности с помощью **создания новой истории**, **редактирование существующих историй** и многое другое. Но в первую очередь мы заменим **jQuery** на **axios**.

12. HTTP-клиенты

12.1 Введение

Как вы знаете, Vue ранее имел собственный HTTP-клиент. Однако впоследствии команда разработчиков Vue.js лишила его статуса официальной рекомендации. Вы можете прочитать объявление Эвана Ю, Retiring vue-resource[1], где объясняются причины принятия этого решения.

Мы собираемся снова реализовать все запросы из предыдущей главы, используя axios[2] на этот раз.

Перед погружением в работу с axios, мы хотели бы вам показать один пример с `vue-resource`. Таким образом вы сможете увидеть разницу и понять, что для вас работает лучше. jQuery также хорош, но если вы используете его только для выполнения AJAX-запросов, следует отказаться от него.

12.2 Vue-resource

Vue-resource предоставляет сервисы, использующие XMLHttpRequest или JSONP, для выполнения HTTP-запросов и обработки ответов.

Инструкцию по установке и документацию можно найти на GitHub[3]. Как обычно, вы также можете подключить его из CDN[4].

Для получения данных с сервера вы можете воспользоваться методом `vue-resource` с названием `$http` и следующим синтаксисом:

```
// GET-запрос
this.$http({url: '/someUrl', method: 'GET'})
  .then(function (response) {
      // колбэк, выполняемый при успешном запросе
  }, function (response) {
      // колбэк, выполняемый при неудачном запросе
  });
```

[1] https://medium.com/the-vue-point/retiring-vue-resource-871a82880af4#.lew43e17f
[2] https://github.com/mzabriskie/axios
[3] https://github.com/vuejs/vue-resource
[4] https://cdnjs.com/libraries/vue-resource

Важная информация

При использовании `vue-resource` экземпляр Vue предоставит функцию **`this.$http(options)`**, которая получает объект `options` для генерации HTTP-запроса и возвращает промис. Также экземпляр Vue будет автоматически привязан к **`this`** во всех обработчиках этой функции.

Для более короткой записи существуют отдельные методы для всех типов запросов.

Методы для выполнения различных HTTP-запросов

```
1 this.$http.get(url, [data], [options]).then(successCallback, errorCallback);
2 this.$http.post(url, [data], [options]).then(successCallback, errorCallback);
3 this.$http.put(url, [data], [options]).then(successCallback, errorCallback);
4 this.$http.patch(url, [data], [options]).then(successCallback, errorCallback);
5 this.$http.delete(url, [data], [options]).then(successCallback, errorCallback);
```

Согласно сообщению Эвана, совершенно нормально продолжать использовать `vue-resource`, если вас все устраивает. Мы будем использовать `axios` с данного момента, поскольку его теперь рекомендуют использовать.

12.3 Axios

Axios — это основанный на промисах HTTP-клиент для браузера и платформы Node.js. Кроме того, он покрывает почти всю функциональность `vue-resource` с очень похожим API, универсален, поддерживает отмену запроса, имеет объявления типов TypeScript[5].

Ниже показано, как можно выполнить GET-запрос с помощью `axios`:

```
// Делаем GET-запрос данных пользователя по идентификатору
axios({
  method: 'get',
  url: '/user/kostas'
})
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  });
```

Для удобства существуют псевдонимы для всех поддерживаемых методов запросов.

[5] https://www.typescriptlang.org/

Методы-псевдонимы для выполнения запросов

```
axios.request(config)
axios.get(url [, config])
axios.delete(url [, config])
axios.head(url [, config])
axios.post(url [, data [, config]])
axios.put(url [, data [, config]])
axios.patch(url [, data [, config]])
```

При использовании методов-псевдонимов не нужно определять свойства `url`, `method` и `data` в `config`.

Подсказка

Если вы хотите использовать `axios` через **`this.$http`**, как в случае `vue-resource`, можно установить **`Vue.prototype.$http = axios`**. Это будет очень удобно, если вы решите заменить `vue-resource` на `axios`.

12.4 Интеграция axios

Настало время использовать `axios` в нашем примере. Прежде всего, добавим его. Добавим эту строку в наш HTML-файл.

stories.html

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/axios/0.18.0/axios.min.js"></scr\
ipt>
```

Чтобы получить истории, сделаем GET-запрос в соответствующей форме:

app.js

```
mounted: function () {
  var vm = this;

  // GET-запрос
  axios.get('/api/stories')
    .then(function (response) {
      Vue.set(vm, 'stories', response.data)
      // Или как мы делали раньше
      // vm.stories = response.data
    })
}
```

Наш список историй приходит без каких-либо проблем.

Теперь перейдём к запросам PATCH и DELETE, используя методы-псевдонимы.

app.js

```
// ...
upvoteStory: function (story) {
  story.upvotes++;
  axios.patch('/api/stories/' + story.id, story)
}
```

app.js

```
deleteStory: function (story) {
  var index = this.$parent.stories.indexOf(story);
  this.$parent.stories.splice(index, 1)
  axios.delete('/api/stories/' + story.id)
}
```

Мы заменили AJAX-методы в кратчайшие сроки!

Важная информация

С тех пор, как компонент *story* больше не имеет доступа к массиву *stories*, мы получаем данный массив с помощью **`this.$parent.stories`**. Можно также использовать **`vm.stories`** или создавать событие при обновлении массива в пределах родительского экземпляра Vue.

12.5 Расширение функциональности

Нам следует добавить пару новых возможностей, чтобы сделать наш список историй аккуратным. Можем дать пользователю возможность **изменять сюжет истории**, **её автора**, **а также создавать новые истории**.

12.5.1 Редактирование историй

Начнём с первой задачи и дадим пользователю несколько полей ввода для управления атрибутами истории. Два связанных поля ввода будут выполнять эту работу, но нам следует показывать их **только** в том случае, когда пользователь **редактирует** историю. Кажется, что-то подобное ранее мы уже делали.

Определять, находится ли история (`story`) в **состоянии редактирования**, будем с помощью свойства данных **`editing`**, которое станет `true`, если пользователь нажмёт на кнопку *Редактировать*.

```
1  <td>
2    <!-- если история редактируется, показываем поле ввода для сюжета -->
3    <input v-if="story.editing" v-model="story.plot" class="form-control">
4    <!-- в остальных случаях показываем сюжет истории -->
5    <span v-else>
6      {{story.plot}}
7    </span>
8  </td>
9  <td>
10   <!-- если история редактируется, показываем поле ввода для писателя -->
11   <input v-if="story.editing" v-model="story.writer" class="form-control">
12   <!-- в остальных случаях показываем писателя истории -->
13   <span v-else>
14     {{story.writer}}
15   </span>
16 </td>
17 <td>
18   {{story.upvotes}}
19 </td>
20 <td>
21   <div v-if="!story.editing" class="btn-group">
22     <button type="button" @click="upvoteStory(story)" class="btn btn-primary">
23       Проголосовать
24     </button>
25     <button type="button" @click="editStory(story)" class="btn btn-default">
```

```
26        Редактировать
27      </button>
28      <button type="button" @click="deleteStory(story)" class="btn btn-danger">
29        Удалить
30      </button>
31    </div>
32  </td>
```

```
1  Vue.component('story', {
2    // ...
3    methods: {
4      // ...
5      editStory: function (story) {
6        story.editing = true;
7      },
8    }
9    // ...
10 })
```

Это наша обновлённая таблица с двумя новыми полями ввода и кнопкой. Мы используем функцию `editStory`, чтобы установить `story.editing` значение `true`, тогда директива `v-if` откроет поля ввода для редактирования истории и скроет кнопки *Голосовать* и *Удалить*. Хотя... это решение не будет работать. Кажется, DOM не обновляется после установки значения `true` в `story.editing`. Но почему так происходит?

Получается, в соответствии с этой статьёй из блога Vue.js[6], когда **вы добавляете новое свойство, которого не было в свойствах данных изначально**, DOM не будет обновляться. Лучшая практика — всегда определять **заранее** свойства, которым необходимо быть реактивными. В случаях, когда вам однозначно необходимо добавить или удалить свойство во время выполнения приложения, используйте глобальные методы **`Vue.set`** или **`Vue.delete`**.

По этой причине, мы должны установить атрибуту **`story.editing`** значение `false` для каждой истории **сразу после** получения историй с сервера.

Для этого воспользуемся JavaScript-методом **`.map()`** в обработчике успешного выполнения GET-запроса.

[6] https://vuejs.org/2016/02/06/common-gotchas/

```
mounted: function () {
  var vm = this;

  // GET-запрос
  axios.get('/api/stories')
    .then(function (response) {
      // устанавливаем данные в vm
      var storiesReady = response.data.map(function (story) {
          story.editing = false;
          return story
      })
      vm.stories = storiesReady
      // или так: Vue.set(vm, 'stories', storiesReady)
    });
}
```

Важная информация

Метод `.map()` вызывает определённую функцию обратного вызова для каждого элемента массива и возвращает массив результатов. Вы можете найти больше информации о методе **`.map()`** и его синтаксисе здесь[7].

Эта функция добавляет атрибут **`editing`** для каждого объекта и затем возвращает обновлённую историю. Новая переменная **`storiesReady`** — это обновлённый массив с включённым новым атрибутом.

Когда историю редактируют, мы будет предоставлять пользователю две возможности: обновить историю новыми значениями или отменить редактирование.

[7] https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array/map

Истории

#	Сюжет	Автор	Кол-во голосов	Действия
1	Всегда много путей достичь цель есть. Испробовать их все должны вы	Йода	552	Проголосовать Редактировать Удалить
2	Никогда не говори мне о шансах!	Хан Соло	18	Проголосовать Редактировать Удалить
3	Я не боюсь смерти. Я умирала каждый день до встречи с тобой	Падме Амидала	617	
4	Страх — это инстинкт, который учит нас быть осторожными. Тот, кто говорит, что идет в бой без страха, — самонадеянный глупец	Оби-Ван Кеноби	261	Проголосовать Редактировать Удалить
5	Все умирает. Даже звезды сгорают	Оби-Ван Кеноби	876	Проголосовать Редактировать Удалить

Поля ввода формы для редактирования истории

Давайте теперь пойдём дальше и добавим две кнопки, которые следует показывать только тогда, когда пользователь редактирует историю. Дополнительно будет создан метод `updateStory`. Он будет обновлять историю после нажатия на кнопку *Обновить историю*.

```
<!-- Если история редактируется, показывать группу кнопок -->
<div class="btn-group" v-else>
  <button type="button" @click="updateStory(story)" class="btn btn-primary">
    Обновить историю
  </button>
  <button type="button" @click="story.editing=false" class="btn btn-default">
    Отмена
  </button>
</div>


Vue.component('story', {
  // ...
  methods: {
    // ...
    updateStory: function (story) {
      axios.patch('/api/stories/' + story.id, story)
      // Установить false свойство editing, чтобы показывать снова кнопки действий и\
 скрывать поля ввода
      story.editing = false;
    },
  }
  // ...
})
```

#	Сюжет	Автор	Кол-во голосов	Действия
1	Всегда много путей достичь цель есть. Испробовать их все должны вы	Йода	552	Проголосовать Редактировать Удалить
2	Никогда не говори мне о шансах!	Хан Соло	18	Проголосовать Редактировать Удалить
3	Я не боюсь смерти. Я умирала каждый день до встречи с тобой	Падме Амидала	617	Обновить историю Редактировать
4	Страх — это инстинкт, который учит нас быть осторожными. Тот, кто говорит, что идет в бой без страха, — самонадеянный глупец	Оби-Ван Кеноби	261	Проголосовать Редактировать Удалить

Действия по обновлению истории

Форма готова и работает. После успешного завершения PATCH-запроса, необходимо установить значение `false` в **`story.editing`**, чтобы скрыть поля ввода и вернуть кнопки действий.

12.5.2 Создание новых историй

Усложним задачу, позволив пользователю создавать новую историю и сохранять её на нашем сервере. Прежде всего, мы должны предоставить поля ввода для добавления новой истории. Для этого будем создавать пустой объект истории и добавлять его в *stories* при помощи **`push()`**. Инициализируем все атрибуты истории значениями *null*, кроме **`editing`**. Поскольку мы хотим сразу редактировать новую историю, свойству **`editing`** будет присвоено значение **`true`**.

```
1  var vm = new Vue({
2    // ...
3    methods: {
4      createStory: function () {
5        var newStory = {
6            plot: '',
7            upvotes: 0,
8            editing: true
9        };
10       this.stories.push(newStory);
11     },
12   }
13 })
```

```
<p class="lead">
  Вот список всех ваших историй.
  <button type="button" @click="createStory()" class="btn btn-primary">
    Добавить новую историю?
  </button>
</p>
```

Важная информация

Метод `push()` добавляет новые элементы в конец массива и возвращает новую длину. Вы можете найти больше информации о методе `push()` и его синтаксисе здесь[8].

Назовём новую функцию **`createStory`** и разместим её в нашем экземпляре Vue.

Справа под нашим списком добавим кнопку. При клике на неё будет вызываться метод **`createStory`**. Так как свойство **`newStory.editing`** установлено в значение `true`, связанные поля ввода *plot* и *writer* вместе с *кнопками для редактирования* отрисовываются мгновенно.

Новый объект *story* также должен быть отправлен на сервер для сохранения в базе данных. Для этого выполним POST-запрос внутри метода **`storeStory`**.

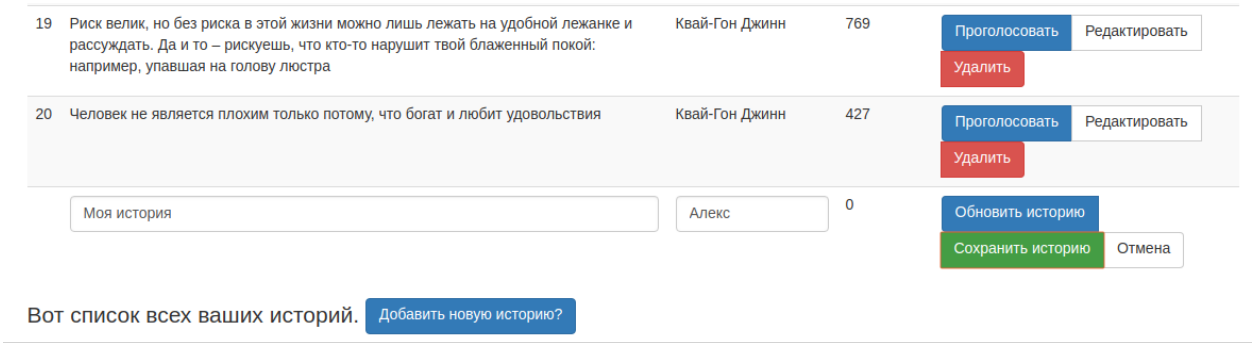
```
Vue.component('story', {
  // ...
  methods: {
    // ...
    storeStory: function (story) {
      axios.post('/api/stories/', story).then(function () {
        story.editing = false;
      });
    },
  }
  // ...
})
```

В случае успешного сохранения истории в соответствующем колбэке выключим режим редактирования, снова показывая кнопки *действий* и скрывая поля ввода и кнопки для *редактирования*. Ниже мы собираемся обновить *группы* кнопок в соответствии с новым методом.

[8] https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array/push

```
<td>
<div class="btn-group" v-if="!story.editing">
  <button type="button" @click="upvoteStory(story)" class="btn btn-primary">
    Проголосовать
  </button>
  <button type="button" @click="editStory(story)" class="btn btn-default">
    Редактировать
  </button>
  <button type="button" @click="deleteStory(story)" class="btn btn-danger">
    Удалить
  </button>
</div>
<div class="btn-group" v-else>
  <button type="button" class="btn btn-primary" @click="updateStory(story)">
    Обновить историю
  </button>
  <button type="button" class="btn btn-success" @click="storeStory(story)">
    Сохранить историю
  </button>
  <button type="button" @click="story.editing=false" class="btn btn-default">
    Отмена
  </button>
</div>
</td>
```

В этом блоке кода имеется небольшая ошибка. Когда включён режим *редактирования* (блок `v-else`), мы видим, что кнопки для обновления и сохранения показываются вместе, что неверно, так как каждая история будет **либо** сохранена, **либо** обновлена. Но не одновременно. Таким образом, если история существует и пользователь редактирует её, необходима кнопка обновления. С другой стороны, если история новая, требуется кнопка сохранения.



Небольшая ошибка

Для решения этой проблемы отредактируем наши кнопки. Кнопка `Обновить историю` будет

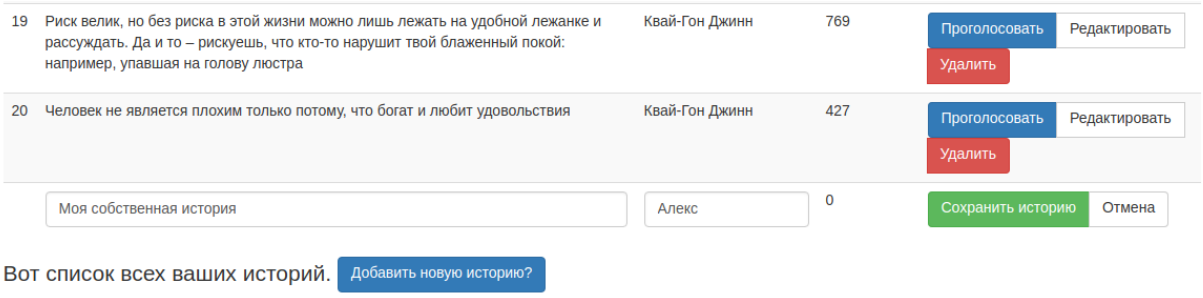
показываться **только** тогда, когда **история существует**. Соответственно кнопка `Сохранить историю` будет показываться, когда история **только создана**.

Возможно, вы обратили внимание, что все полученные с сервера истории имеют атрибут `id`. Мы собираемся использовать значение идентификатора истории для **определения, является ли история новой или нет**.

```
1  <div class="btn-group" v-else>
2    <!--Если история существует, значит мы хотим обновить её
3    ПОДСКАЗКА: если история получена из базы данных, значит она имеет идентификатор -->
4    <button type="button" v-if="story.id" class="btn btn-primary" @click="updateStory(\
5  story)">
6      Обновить историю
7    </button>
8    <!-- Если история новая, значит мы хотим сохранить её -->
9    <button type="button" v-else class="btn btn-success" @click="storeStory(story)">
10     Сохранить историю
11   </button>
12   <!-- Всегда показывать кнопку отмены -->
13   <button type="button" @click="story.editing=false" class="btn btn-default">
14     Отмена
15   </button>
16 </div>
```

Подсказка

Если история получена с базы данных, она будет иметь идентификатор.



Добавление новой истории

Вот пока всё. Это не было сложным, верно?

После завершения данной части, тестирование нашего приложения принесло другую ошибку. После создания, сохранения и попытки отредактировать новую историю, появляется сообщение, что на кнопке написано «Сохранить историю» вместо «Обновить историю»! Так

произошло потому, что мы не получаем только что созданную историю с сервера после её отправки, поэтому отсутствует идентификатор. Для решения этой проблемы можно снова получить истории с сервера сразу после сохранения новой в базе данных.

Так как я не люблю, чтобы мой код повторялся, вынесем эту процедуру в метод **fetchStories()**. После этого я могу использовать данный метод для получения историй в любое время.

Метод fetchStories

```
var vm = new Vue({
  el: '#v-app',
  data : {
    stories: [],
  },
  mounted: function () {
    this.fetchStories()
  },
  methods: {
    createStory: function () {
      var newStory = {
        plot: '',
        upvotes: 0,
        editing: true
      };
      this.stories.push(newStory);
    },
    fetchStories: function () {
      var vm = this;
      axios.get('/api/stories')
        .then(function (response) {
            var storiesReady = response.data.map(function (story) {
                story.editing = false;
                return story
            })
            Vue.set(vm, 'stories', storiesReady)
            // или: vm.stories = storiesReady
        });
    },
  }
});
```

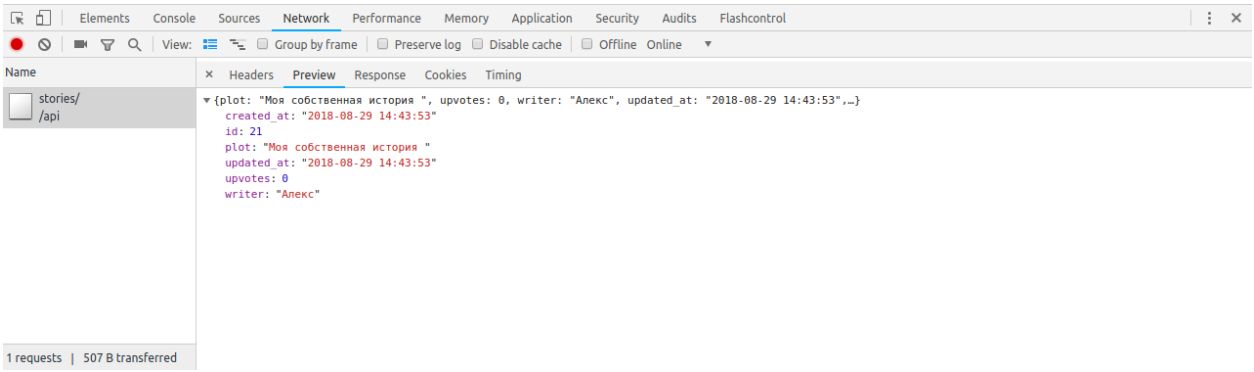
В нашем случае будем вызывать **fetchStories()** при успешном выполнении POST-запроса.

```
Vue.component('story', {
  // ...
  methods: {
    // ...
    storeStory: function (story) {
      axios.post('/api/stories/', story).then(function () {
        story.editing = false;
        vm.fetchStories();
      });
    },
  }
  // ...
})
```

Вот так! Теперь мы можем создавать и редактировать любые истории, какие захотим.

12.5.3 Блок сохранения и обновления

Лучший вариант для исправления предыдущей проблемы — получать только что созданную *историю* из базы данных вместо *всех* существующих. Если вы посмотрите на ответ сервера после POST-запроса, то увидите, что он возвращает созданную *историю* вместе с её *идентификатором*.



Ответ сервера после создания новой истории

Нам осталось только обновить компонент *story* в соответствии с теми данными, которые приходят с сервера. Таким образом, атрибут идентификатора истории будет установлен из данных ответа. Сделаем это в обработчике успешного выполнения POST-запроса.

```
Vue.component('story', {
  // ...
  methods: {
    // ...
    storeStory: function (story) {
      axios.post('/api/stories/', story).then(function (response) {
        Vue.set(story, 'id', response.data.id);
        story.editing = false
      });
    },
  }
  // ...
})
```

Я использую **`Vue.set(story, 'id', response.data.id)`** вместо **`story.id = response.data.id`**, потому что внутри нашей таблицы мы показываем идентификатор (**id**) каждой истории. Раз новая история не имеет **`id`**, когда она добавляется в массив `stories`, DOM не будет обновлён, когда атрибут **`id`** изменится, то есть мы не сможем увидеть новый **`id`**.

Подсказка

Когда вы добавляете **новое свойство, которое не было представлено, когда данные были добавлены**, Vue.js не сможет определить добавление этого свойства. Поэтому, если необходимо добавить или удалить свойства во время выполнения приложения, используйте глобальные методы **`Vue.set`** или **`Vue.delete`**.

12.6 JavaScript-файл

Как вы могли заметить, нашего кода становится больше. Раз наш проект растёт, его труднее поддерживать. Для начала разделим *JavaScript-код* и *HTML*. Я создам файл **`app.js`** и сохраню его в папке **`js`**.

Весь JavaScript-код отныне должен находиться внутри этого файла. Чтобы включить новый скрипт на любую HTML-страницу, просто следует добавить этот тег

```
<script src="/js/app.js" type="text/javascript"></script>
```

и вы **готовы продолжить!**

12.7 Исходный код

Ниже находится весь исходный код предыдущего примера *Управление историями*. Если вы скачали наш репозиторий, предлагаю вам открыть локальные файлы вашим любимым текстовым редактором, потому что код достаточно большой. Файлы расположены в *~/themajestyofvuejs2/apis/stories/public*.

Если вы не скачали репозиторий, можно просмотреть файлы **stories.html**[9] и **app.js**[10] на *GitHub*.

stories.html

```
<html lang="en">
<head>
  <title>Истории</title>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css/b\
ootstrap.min.css">
</head>

<body>
<main>
  <div class="container">
    <h1>Истории</h1>
    <div id="v-app">
      <table class="table table-striped">
        <tr>
          <th>#</th>
          <th>Сюжет</th>
          <th>Автор</th>
          <th>Кол-во голосов</th>
          <th>Действия</th>
        </tr>
        <tr v-for="story in stories" is="story" :key="story.id" :story="story"></tr>
      </table>
      <p class="lead">
        Вот список всех ваших историй.
        <button type="button" @click="createStory()" class="btn btn-primary">
          Добавить новую историю?
        </button>
      </p>
      <pre>{{ $data }}</pre>
    </div>
```

[9] https://github.com/hootlex/the-majesty-of-vuejs-2/blob/ru/apis/stories/public/stories.html
[10] https://github.com/hootlex/the-majesty-of-vuejs-2/blob/ru/apis/stories/public/js/app.js

```
31    </div>
32  </main>
33  <template id="template-story-raw">
34    <tr>
35      <td>
36        {{story.id}}
37      </td>
38      <td class="col-md-6">
39        <input v-if="story.editing"
40        v-model="story.plot"
41        class="form-control">
42        </input>
43        <!-- в остальных случаях показывать сюжет истории -->
44        <span v-else>
45          {{story.plot}}
46        </span>
47      </td>
48      <td>
49        <input v-if="story.editing"
50        v-model="story.writer" class="form-control">
51        </input>
52        <!-- в остальных случаях показывать писателя истории -->
53        <span v-else>
54          {{story.writer}}
55        </span>
56      </td>
57      <td>
58        {{story.upvotes}}
59      </td>
60      <td>
61        <div class="btn-group" v-if="!story.editing">
62          <button type="button" @click="upvoteStory(story)"
63          class="btn btn-primary">
64            Проголосовать
65          </button>
66          <button type="button" @click="editStory(story)" class="btn btn-default">
67            Редактировать
68          </button>
69          <button type="button" @click="deleteStory(story)"
70          class="btn btn-danger">
71            Удалить
72          </button>
73        </div>
```

```
74        <div class="btn-group" v-else>
75          <!-- Если история получена с базы данных, она имеет идентификатор -->
76          <button type="button" v-if="story.id"
77          class="btn btn-primary"
78          @click="updateStory(story)">
79            Обновить историю
80          </button>
81
82          <!-- Если история новая, значит мы хотим сохранить её -->
83          <button type="button" v-else class="btn btn-success"
84          @click="storeStory(story)">
85            Сохранить историю
86          </button>
87
88          <!-- Всегда показывать кнопку отмены -->
89          <button type="button" @click="story.editing=false"
90          class="btn btn-default">
91            Отмена
92          </button>
93        </div>
94      </td>
95      </tr>
96  </template>
97  <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.17/vue.js"></script>
98  <script src="https://cdnjs.cloudflare.com/ajax/libs/axios/0.18.0/axios.min.js">
99  </script>
100 <script src="/js/app.js" type="text/javascript"></script>
101 </body>
102 </html>
```

app.js

```
1  Vue.component('story', {
2    template: '#template-story-raw',
3    props: ['story'],
4    methods: {
5      deleteStory: function (story) {
6        var index = this.$parent.stories.indexOf(story);
7        this.$parent.stories.splice(index, 1)
8        axios.delete('/api/stories/' + story.id)
9      },
10     upvoteStory: function (story) {
11       story.upvotes++;
```

```
12        axios.patch('/api/stories/' + story.id, story)
13      },
14      editStory: function (story) {
15        story.editing = true;
16      },
17      updateStory: function (story) {
18        axios.patch('/api/stories/' + story.id, story)
19        /* Установим false в свойство editing, чтобы показывать снова
20        кнопки действий и скрывать поля ввода */
21        story.editing = false;
22      },
23      storeStory: function (story) {
24        axios.post('/api/stories/', story).then(function (response) {
25          /*
26          После сохранения новой истории в базе данных, получаем снова все истории
27          с помощью vm.fetchStories();
28          Или лучше обновим идентификатор созданной истории
29          */
30          Vue.set(story, 'id', response.data.id);
31
32          /* Установим false в свойство editing, чтобы показывать снова
33          кнопки действий и скрывать поля ввода */
34          story.editing = false;
35        });
36      },
37    }
38  })
39
40  // Vue.prototype.$http = axios
41
42  new Vue({
43    el: '#v-app',
44    data: {
45      stories: [],
46      story: {}
47    },
48    mounted: function () {
49      this.fetchStories()
50    },
51    methods: {
52        createStory: function () {
53          var newStory = {
54            plot: '',
```

```
55          upvotes: 0,
56          editing: true
57        };
58        this.stories.push(newStory);
59      },
60      fetchStories: function () {
61        var vm = this;
62        axios.get('/api/stories')
63          .then(function (response) {
64              // установить данные в vm
65              var storiesReady = response.data.map(function (story) {
66                  story.editing = false;
67                  return story
68              })
69              // vm.stories = storiesReady
70              Vue.set(vm, 'stories', storiesReady)
71          });
72      },
73  }
74 });
```

Примеры кода

Вы можете найти примеры кода этой главы на GitHub[11]

12.8 Домашняя работа

Для того, чтобы освоиться с созданием веб-запросов и обработкой ответов, вам следует воспроизвести то, что мы делали в этой главе.

Всё, что вам нужно сделать, — выполнить запросы к API, чтобы:

- создавать таблицу и **показывать существующие** фильмы
- **модифицировать** существующие фильмы
- **сохранять** новые фильмы в базе данных
- **удалять** фильмы из базы данных

Я подготовил **базу данных и API** для вас. Вам остаётся только написать HTML и JavaScript.

12.8.1 Введение

Если вы следовали инструкциям из десятой главы, откройте терминал и запустите команду:

```
cd ~/themajestyofvuejs/apis/movies
sh setup.sh
```

В противном случае вам следует сделать следующее:

```
mkdir ~/themajestyofvuejs
cd ~/themajestyofvuejs
git clone https://github.com/hootlex/the-majesty-of-vuejs .
cd ~/themajestyofvuejs/apis/movies
sh setup.sh
```

Теперь у вас есть **база данных, заполненная прекрасными фильмами** вместе с полноценным сервером, **запущенным на *http://localhost:3000***!

Чтобы удостовериться, что все работает как нужно, откройте *http://localhost:3000/api/movies*. Вы должны увидеть массив с фильмами в формате JSON.

[11] https://github.com/hootlex/the-majesty-of-vuejs-2/tree/ru/codes/chapter12

12.8.2 Конечные точки API

Вам понадобятся следующие **конечные точки API**:

Метод HTTP	URI	Действие
GET/HEAD	api/movies	*Получает* все фильмы
GET/HEAD	api/movies/{id}	*Получает* определённый фильм
POST	api/movies	*Создаёт* новый фильм
PUT/PATCH	api/movies/{id}	*Обновляет* существующий фильм
DELETE	api/movies/{id}	*Удаляет* определённый фильм

12.8.3 Ваш код

Добавьте ваш HTML-код в файл `~/themajestyofvuejs2/apis/movies/public/movies.html`, который мы создали. Вы можете разместить там и ваш JavaScript-код, либо внутри `js/app.js`.

Для проверки вашей работы откройте в браузере *http://localhost:3000/movies.html*.

Надеюсь, вам понравится. Удачи!

Фильмы

#	Название	Режиссер	Действия
1	Звёздные войны: Эпизод 6 – Возвращение Джедая	Ричард Маркуэнд	Редактировать Удалить
2	Перед закатом	Ричард Линклейтер	Редактировать Удалить
3	Спасти рядового Райана	Стивен Спилберг	Редактировать Удалить
4	Эта замечательная жизнь	Фрэнк Капра	Редактировать Удалить
5	Лицо со шрамом	Брайан Де Пальма	Редактировать Удалить
6	Другие	Алехандро Аменабар	Редактировать Удалить
7	Заводной апельсин	Стэнли Кубрик	Редактировать Удалить
8	Зомби по имени Шон	Эдгар Райт	Редактировать Удалить
9	Гарри Поттер и Дары Смерти: Часть II	Дэвид Йейтс	Редактировать Удалить
20	Гарри Поттер и Дары Смерти: Часть II	Дэвид Йейтс	Редактировать Удалить

Вот список всех ваших фильмов. Добавить новый фильм?

Пример вывода

Возможное решение

Вы можете найти потенциальное решение этого упражнения здесь[12].

[12] https://github.com/hootlex/the-majesty-of-vuejs-2/tree/ru/homework/Chapter12

13. Пагинация

В предыдущей главе мы добились получения всех записей из базы данных и отобразить их внутри таблицы. Эта реализация хороша для множества записей но в реальности, когда вам приходится работать с тысячами или миллионами записей, вы не можете просто получить их и поместить их в массив. Если вы это сделаете, ваш браузер не будет рад загрузить такой объём данных, но даже если это он сделает, то я уверяю вас, что никому не понравится работать с таблицей с 100 000 строк.

На заметку

Пагинация или постраничная навигация используется в той или иной форме в почти каждом веб-приложении для разделения возвращённых данных и отображения их на нескольких страницах. Пагинация также включает в себя логику подготовки и отображения ссылок на различные страницы, и её можно обрабатывать на стороне клиента и сервера. Пагинация на стороне сервера используется чаще.

В ситуациях, похожую на эту, разработчики API разделяют (я надеюсь на это) возвращаемые данные на страницы.

HTTP-ответ будет содержать некоторые простые **`метаданные`** наряду с основными данными, содержащие *все* элементы *на каждую страницу* и т.д. Для облегчения навигации по страницам, они предоставляют такую информацию как **`текущая страница`**, **`следующая страница`** и **`предыдущая страница`**.

Пример запроса с данными пагинации

```
{
  "total": 10000,
  "per_page": 50,
  "current_page": 15,
  "last_page": 200,
  "next_page_url": "/api/stories?page=16",
  "prev_page_url": "/api/stories?page=14",
  "from": 751,
  "to": 800,
  "data": [...]
}
```

Метаданные пагинации также могут находиться внутри объекта рядом с **`data`** или в том месте, в котором решили разработчики API.

Пример запроса с данными пагинации

```
{
  "data": [...],
  "pagination": {
    "total": 10000,
    "per_page": 50,
    "current_page": 15,
    "last_page": 200,
    "next_page_url": "/api/stories?page=16",
    "prev_page_url": "/api/stories?page=14",
    "from": 751,
    "to": 800,
  }
}
```

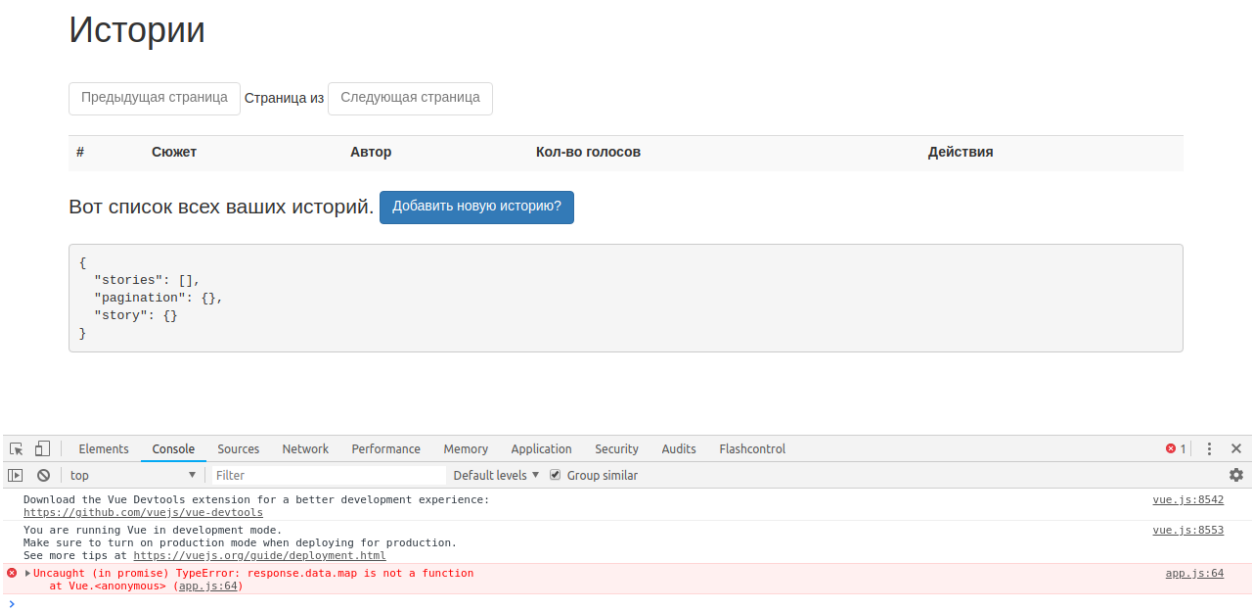
13.1 Реализация

Мы продолжим работу с примерами *story* из предыдущей главы, используя слегка улучшенный API пагинации. Итак, мы собираемся изменить код, чтобы иметь возможность доступа и использования этих данных.

Если вы посмотрите на код из предыдущей главы, вы заметите, что метод **fetchStories** очень похож на это:

```
new Vue({
  // ...
  methods: {
    // ...
    fetchStories: function () {
      var vm = this;
      this.$http.get('/api/stories')
        .then(function (response) {
            var storiesReady = response.data.map(function (story) {
                story.editing = false;
                return story
            })
            Vue.set(vm, 'stories', storiesReady)
        });
    },
    // ...
  }
});
```

Если мы откроем наш HTML-файл в браузере, как вы, возможно, уже догадались, таблица не будет отображаться корректно.



Истории не отображаются

Это происходит потому, что необходимые нам истории (`stories`) находятся внутри массива `data`. Для исправления этого нам нужно изменить `response.data` на `response.data.data` (я знаю, что это странно, но...).

Кроме массива *stories*, мы также хотим сохранить данные пагинации внутри объекта для облегчения реализации функциональности пагинации.

Чтобы узнать, как мы можем получить доступ к этим данным, давайте посмотрим на ответ сервера.

```
{
    "total": 200,
    "per_page": 15,
    "current_page": 1,
    "last_page": 14,
    "next_page_url": "http://localhost:3000/api/stories?page=2",
    "prev_page_url": null,
    "from": 1,
    "to": 15,
    "data": [
        {
            "id": 1,
            "plot": "Вы были правы в одном, Мастер; переговоры были недолгими.",
            "upvotes": "201",
            "writer": "Оби-Ван Кеноби",
            "created_at": "2018-08-29 13:15:33",
            "updated_at": "2018-08-29 13:15:33"
        },
        {
            "id": 2,
            "plot": "Страх — это путь на Тёмную сторону. Страх порождает гнев. Гнев приводит к ненависти. Ненависть приводит к страданиям",
            "upvotes": "857",
            "writer": "Йода",
            "created_at": "2018-08-29 13:15:33",
            "updated_at": "2018-08-29 13:15:33"
        },
        {
            "id": 3,
            "plot": "Видимо, наша судьба — страдать. Это наше предназначение в жизни",
            "upvotes": "313",
            "writer": "С 3PO",
            "created_at": "2018-08-29 13:15:33",
            "updated_at": "2018-08-29 13:15:33"
        },
        { … }, // 6 items
        { … }, // 6 items
        { … }, // 6 items
        { … }, // 6 items
```


Ответ сервера

Для начала нам не нужны все эти данные. Итак, мы будем работать с *current_page*, *last_page*, *next_page_url* и *prev_page_url*.

Наш объект пагинации будет примерно таким:

```
pagination: {
  "current_page": 15,
  "last_page": 200,
  "next_page_url": "/api/stories?page=16",
  "prev_page_url": "/api/stories?page=14"
}
```

Давайте изменим наш метод **fetchStories** для обновления объекта **pagination** каждый раз, когда извлекаются данные из базы данные.

```
new Vue({
  // ...
  methods: {
    // ...
    fetchStories: function () {
      var vm = this;
      this.$http.get('/api/stories')
      .then(function (response) {
          var storiesReady = response.data.data.map(function (story) {
              story.editing = false;
              return story
          })
          // здесь мы используем response.data
          var pagination = {
              current_page: response.data.current_page,
              last_page: response.data.last_page,
              next_page_url: response.data.next_page_url,
              prev_page_url: response.data.prev_page_url
          }
          Vue.set(vm, 'stories', storiesReady)
          Vue.set(vm, 'pagination', pagination)
      });
    },
    // ...
  }
});
```

13.2 Ссылки на пагинацию

К настоящему моменту у нас есть объект **pagination**, но мы не всегда получаем первую страницу историй, когда делаем HTTP-запрос GET к *api/stories*. Нам требуется изменить запрашиваемую страницу на основе пользовательского взаимодействия (переход на следующую или предыдущую страницу).

Сначала мы обновим метод **fetchStories** для добавления аргумента с предпочитаемой страницей. Если аргумент не передан — извлекается первая страница. Мы также создадим новый метод **makePagination** для того, чтобы сделать код чище.

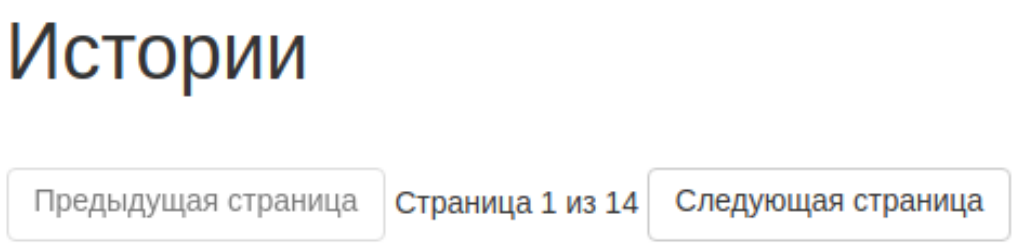
```
new Vue({
  // ...
  methods: {
    // ...
    fetchStories: function (page_url) {
      var vm = this;
      page_url = page_url || '/api/stories'
      this.$http.get(page_url)
        .then(function (response) {
            var storiesReady = response.data.data.map(function (story) {
                story.editing = false;
                return story
            })
            vm.makePagination(response.data)
            Vue.set(vm, 'stories', storiesReady)
        });
    },
    makePagination: function (data) {
      // здесь мы используем response.data
      var pagination = {
          current_page: data.current_page,
          last_page: data.last_page,
          next_page_url: data.next_page_url,
          prev_page_url: data.prev_page_url
      }
      Vue.set(vm, 'pagination', pagination)
    }
    // ...
  }
}
```

Теперь наш метод готов, нам нужен способ вызвать их правильно. Мы добавим две кнопки в верхней части **`#app div`**: одну для следующей страницы, а вторую для предыдущей. Каждая кнопка вызывает метод **`fetchStories`** при нажатии, передавая соответствующий **URL-адрес страницы**.

```
<div class="pagination">
  <button type="button" @click="fetchStories(pagination.prev_page_url)">
    Предыдущая страница
  </button>
  <button type="button" @click="fetchStories(pagination.next_page_url)">
    Следующая страница
  </button>
</div>
```

Великолепно! При попытке нажатия на кнопки, вы увидите, что они работают, как они ожидаются. У нас есть наша пагинация в одно мгновение ока. Полезно информировать пользователя о том, на какой странице он находится, а также показывать общее количество страниц. Кроме того, мы можем отключить кнопку с **предыдущей страницей**, если пользователь находится на первой странице, а кнопку на **следующую страницу** на последней странице.

```
<div class="pagination">
  <button
    type="button"
    @click="fetchStories(pagination.prev_page_url)"
    :disabled="!pagination.prev_page_url"
  >
      Предыдущая страница
  </button>
  <span>Страница {{pagination.current_page}} из {{pagination.last_page}}</span>
  <button
    type="button"
    @click="fetchStories(pagination.next_page_url)"
    :disabled="!pagination.next_page_url"
  >
      Следующая страница
  </button>
</div>
```



Отключённая кнопка «Назад»

Примеры кода

Вы можете найти примеры кода для этой главы на GitHub[1].

13.3 Домашняя работа

В данной домашней работе нет ничего особенного для этой главы. Если вы действительно хотите поработать над этим примером, я предоставил вам API с пагинацией.

Если вы сделали домашнюю работу из предыдущей главы (скачали код и запустили сервер), вы уже облегчили себе решение данного домашнего задания. Если этого сделано не было, просто следуйте этим инструкциям.

API с пагинацией находится внутри каталога *'~/themajestyofvuejs2/apis/pagination/stories'*.

Файл HTML вы найдёте в каталоге *'~/themajestyofvuejs2/apis/pagination/stories/public'*.

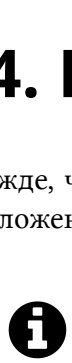
Если вы хотите посмотреть на законченный код, вы можете посмотреть его на GitHub[2].

[1] https://github.com/hootlex/the-majesty-of-vuejs-2/tree/ru/codes/chapter13
[2] https://github.com/hootlex/the-majesty-of-vuejs-2/tree/ru/apis/pagination/stories/public

III Создание крупномасштабных приложений

14. ECMAScript 6

Прежде, чем мы пойдём дальше, посмотрим, как мы можем создавать крупномасштабные приложения; я бы хотел познакомить вас с ECMAScript 6.



Важная информация

ECMAScript — спецификация для скриптовых языков на стороне клиента, которой следуют несколько языков программирования, включая JavaScript, ActionScript и JScript.

ECMAScript 6 (ES6), также известный как ES2015 — последняя версия стандарта ECMAScript. Спецификация ES6 завершена в июне 2015 года. Это большое изменение языка и первое крупное обновление с момента принятия стандарта ES5 в 2009 году. В настоящее время[1] ведётся разработка над реализацией функциональных возможностей ES6 для следующего крупного обновления JavaScript.

14.1 Введение

В ES6 множество новых возможностей. Мы рассмотрим те, которые будем использовать в следующих главах. Если интересно узнать подробнее о новых возможностях в ES6, я крайне рекомендую книгу «Understanding ECMAScript 6» от Николаса Закаса (Nicholas Zakas), которая доступна на Leanpub[2]. Также есть бесплатная онлайн-версия[3] этой книги.

Кроме этого, есть и другие полезные ресурсы и обучающие материалы, например, на Babel[4], статья на Tuts+[5], пост в блоге[6] Николаса Закаса и ещё куча другой информации во всём интернете!

[1] https://kangax.github.io/compat-table/es6/
[2] https://leanpub.com/understandinges6/
[3] https://leanpub.com/understandinges6/read
[4] https://babeljs.io/docs/learn-es2015/
[5] http://code.tutsplus.com/articles/use-ecmascript-6-today--net-31582
[6] https://www.nczonline.net/blog/2013/09/10/understanding-ecmascript-6-arrow-functions/

14.1.1 Совместимость

Неудивительно, что поддержка сильно отличается от версии к версии, а браузер Mozilla стремится идти впереди.

Таблица совместимости ES6[7] — хорошее начало для определения того, какие возможности ECMAScript 6 поддерживает и какие не поддерживает ваш браузер.

Примечание

Если вы используете Chrome, большинство возможностей ES6 скрыты за настройками переключения в браузере. Откройте *chrome://flags*, найдите раздел «Enable Experimental JavaScript» и выберите «Enabled» для включения этой поддержки.

Впредь мы будем использовать возможности ES6 в наших примерах.

14.2 Объявления переменных

14.2.1 Объявление Let

`let` — это новый **`var`**. Вы можете с лёгкость заменить `var` на `let`, это ограничит область видимости переменной в пределах текущего блока кода. Поскольку объявления `let` не поднимаются в верхнюю часть окружающего блока, лучше всегда помещать эти объявления в первую очередь в блоке кода, чтобы они были доступны для всего блока. Например:

Переменная, объявленная через let внутри конструкции if

```
1 let age = 22
2 if (age >= 18) {
3   let adult = true;
4   console.log(adult); // выведет true
5 }
6 // переменная adult недоступна здесь
7 console.log(adult);
8 // ERROR: Uncaught ReferenceError: adult is not defined
```

[7] https://kangax.github.io/compat-table/es6/

Переменная, объявленная через let вверху

```
let age = 22
let adult
if (age >= 18) {
  adult = true;
  console.log(adult); // выведет true
}
// теперь переменная adult доступна здесь
console.log(adult); // выведет true
```

14.2.2 Объявления констант

Константы, такие как объявления `let` — объявления на уровня блока. Есть одна большая разница между `let` и `const`. Когда вы объявляете переменную, используя `const`, она определяется как константа, что значит, **вы не можете изменить её значение**.

```
const name = 'Алекс'

name = 'Костас' // выбрасывает ошибку
```

На заметку

По аналогии с константами в других языках, их значение не может измениться. Однако в вашем случае значение константы **может быть изменено, если это объект**.

14.3 Стрелочные функции

Одной из самой интересной новой частью ECMAScript 6 является стрелочные функции. Стрелочные функции — это функции, определённые через новый синтаксис, использующий «стрелку» (⇒). Они поддерживают как выражения, так и утверждения. В отличие от функций, стрелочные функции разделяют тот же самый лексический объект `this`, что их окружающий код.

Например, следующая стрелочная функция принимает один аргумент и возвращает значение, увеличенное на 1:

```
var increment = value => value + 1;
increment(5) // возвращает 6

// эквивалентно этому:

var increment = function (value) {
  return value + 1;
};
```

Ещё один пример стрелочной функции, но уже с двумя аргументами, которая возвращает сумму двух переданных:

```
var sum = (a, b) => a + b;
sum(5, 10) // возвращает 15

// эквивалентно этому:

var sum = function (a, b) {
  return a + b;
};
```

И, наконец, пример стрелочной функции без аргументов и использующее выражение.

```
var sayHiAndBye = () => {
  console.log('Привет!');
  console.log('Пока!');
};
sayHiAndBye()

// эквивалентно этому:

var sayHiAndBye = function () {
  console.log('Привет!');
  console.log('Пока!');
};
```

14.4 Модули

Это для меня самое большое улучшение языка. ES6 теперь поддерживает экспорт и импорт модулей через различные файлы.

Самый простой пример — создать файл с типом `.js` с переменной, и использовать её в другом файле следующим образом:

module.js

```
export name = 'Алекс'
```

main.js

```
import {name} from './module'
console.log('Привет, ', name)
// выводит "Привет, Алекс"
```

Вы также можете экспортировать переменные наряду с функциями **поочерёдно**

module.js

```
export var name = 'Алекс'
export function getAge() {
  return 22;
}
```

main.js

```
import {name, getAge} from './module'
console.log(name, ' — ', getAge())
```

Или **внутри объекта**:

module.js

```
var name = 'Алекс'
function getAge() {
  return 22;
}
export default {name, getAge}
```

main.js

```
import person from './module'
console.log(person.name, ' — ', person.getAge() )
// выведет "Алекс — 22"
```

14.5 Классы

Классы JavaScript представлены ECMAScript 6 и по сути являются синтаксическим сахаром для существующего наследования на основе прототипов.

Синтаксис класса **не** представляет новую объектно-ориентированную модель наследования в JavaScript. Классы JavaScript предоставляют гораздо более простой и чистый синтаксис для создания объектов и работы с наследованием.

Пример класса

```
// родительский класс
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }

  calcArea() {
    return this.height * this.width;
  }

  // Для создания геттера используйте ключевое слово get.
  get area() {
    return this.calcArea();
  }
  // Для создания сеттера делайте то же самое, но используя ключевое слово set.
}

// дочерний класс
class Square extends Rectangle {
  constructor(side) {
    // вызов родительского конструктора
    super(side, side)
  }
}

var square = new Square(5);

console.log(square.area); //outputs 25
```

14.6 Значения параметров по умолчанию

С помощью ES6 вы можете определить значения параметров по умолчанию.

```
function divide(x, y = 2) {
  return x/y;
}

// эквивалентно этому:

function divide(x, y) {
  y = y == undefined ? 2 : y;
  return x/y;
}
```

14.7 Шаблонные литералы

Шаблонные литералы — строковые литералы, позволяющие использовать внутри встроенные выражения. С помощью них можно использовать многострочные строки и возможности интерполяции строки. Они назывались «шаблонные строки» (template strings) в предыдущих выпусках спецификации ES2015/ES6.

Шаблонные литералы оборачиваются в символы обратных кавычек (`) вместо двойных или одинарных кавычек. Внутри обратных кавычек вы можете использовать **${expression}**, где expression — функция, переменная или корректное выражение.

Шаблонные литералы — использование переменных

```
let name = 'Alex'
console.log(`Привет, ${name}`)

// эквивалентно этому:
console.log('Привет, ' + name )
```

Шаблонные литералы — использование функций

```
add = (a, b) => a + b

let [a, b] = [10, 2]

console.log(`Если у вас ${a} яиц и вы покупаете ${b}
всего у вас будет ${add(a, b)} яиц!`)

// эквивалентно этому:
console.log('Если у вас ' + a +' яиц и вы покупаете ' + b +
'\nвсего у вас будет '+ add(a, b) + ' яиц!')
```

Шаблонные литералы — использование выражений

```
let [a, b] = [10, 2]

console.log(`Если у вас ${a} яиц и вы покупаете ${b}
всего у вас будет ${a + b} яиц!`)

// эквивалентно этому:
console.log('Если у вас ' + a +' яиц и вы покупаете ' + b +
'\nсего у вас будет '+ (a + b)*1 + ' яиц!')
```

Также можно разделить сообщение на несколько строк, используя `'\n'`.

15. Продвинутый рабочий процесс

Все эти возможности ES6 (и многое другое) могут вас возбудить, но есть одна загвоздка. Как упоминалось ранее, **не** все браузеры полностью поддерживают возможности ES6/ES2015.

Чтобы сегодня писать код JavaScript, используя новый синтаксис и возможности, нам нужен своего рода посредника, который возьмёт наш код и переведёт его в привычный нам JavaScript-код или Vanilla JS[1], который исполнит *каждый браузер*. Это процедура на самом деле **важна** в продакшене, хотя у вас может быть иное мнение.

Позвольте мне рассказать вам поучительную историю. Несколько лет назад мой коллега использовал некоторые классные возможности JS, которые не всеми браузерами поддерживались. Спустя несколько дней наши пользователи начали жаловаться, что некоторые страницы сайта некорректно отображаются, но мы не могли понять, в чём проблема. Мы тестировали сайт на разных компьютерах, Android- и iPhone-смартфонах и т.д., и сайт на 100% работал во всех наших браузерах. Позже он узнал, что старые версии мобильного браузера Safari не поддерживали его код. *Так вот, не будьте таким парнем!*

Иногда бывает **трудно** знать наверняка, будет ли написанный код работать хорошо во **всех браузерах**, включая мобильный браузер от Facebook, которого я вызывает у меня самое страшное опасение.

15.1 Компиляция ES6 с помощью Babel

Babel станет нашим *посредником*. Babel — компилятор JavaScript одной версии в другую (например, ES6 в ES5), который позволяет сегодня использовать следующего поколения JavaScript.

Важная информация

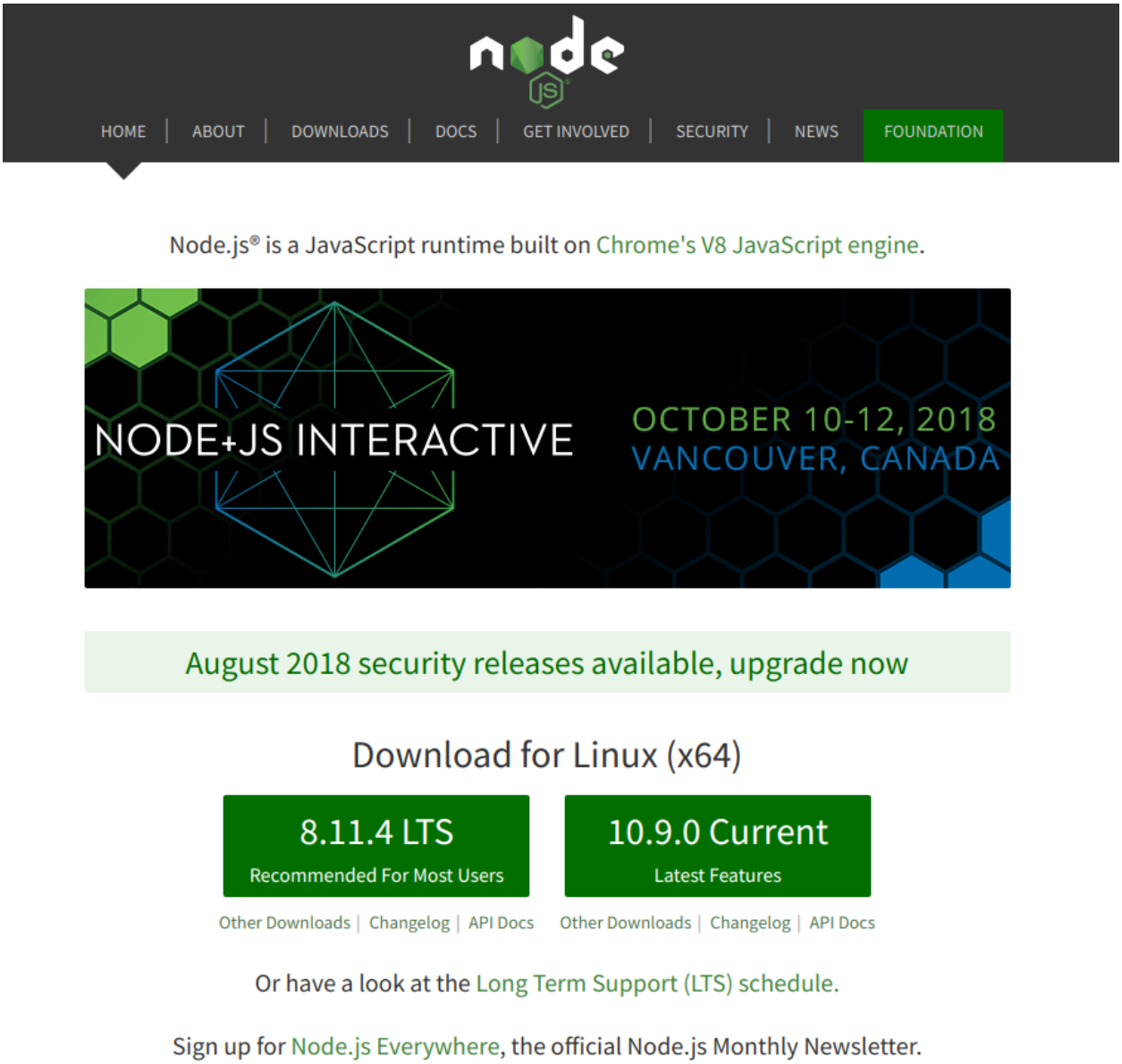
Компилятор от одного исходного кода в другой (source-to-source compiler), транскомпилятор (transcompiler) или транспилер (transpiler) — тип компилятора, принимающий исходный код программы, написанной на одном языке программирования в качестве входного значения, и создающий эквивалентный исходный код на другом языке программирования.

[1] http://vanilla-js.com/

Babel

Перед установкой Babel, нужно установить **Node.js**. Для этого перейдите на сайт Node[2] и загрузите последнюю стабильную версию. Будет предложено загрузить файл *.pkg* (или *.msi* на Windows). Когда скачивание файла завершено, откройте файл и следуйте инструкциям. Затем выполните требуемую перезагрузку, и всё готово!

[2] https://nodejs.org/en/



Node.js

15.1.1 Установка

Создайте новый каталог и поместите внутри него файл с именем `package.json`, содержащий пустой JSON-объект (`{}`). Вы можете сделать его вручную или выполнив определённые команды в вашем терминале.

```
mkdir babel-example
echo {} > package.json
```

Затем запустите следующую строку для установки Babel:

```
npm install @babel/core @babel/cli --save-dev
```

```
lex@lex-desktop: ~/vue_book/babel-example
Файл  Правка  Вид  Поиск  Терминал  Справка
  ├── @babel/template@7.0.0
  ├── @babel/traverse@7.0.0
  │ ├── @babel/helper-function-name@7.0.0
  │ │ └── @babel/helper-get-function-arity@7.0.0
  │ ├── @babel/helper-split-export-declaration@7.0.0
  │ ├── debug@3.1.0
  │ └── globals@11.7.0
  ├── @babel/types@7.0.0
  │ ├── esutils@2.0.2
  │ └── to-fast-properties@2.0.0
  ├── debug@3.1.0
  │ └── ms@2.0.0
  ├── json5@0.5.1
  ├── resolve@1.8.1
  │ └── path-parse@1.0.6
  └── semver@5.5.1

npm WARN optional Skipping failed optional dependency /chokidar/fsevents:
npm WARN notsup Not compatible with your operating system or architecture: fsevents@1.2.4
npm WARN babel-example No description
npm WARN babel-example No repository field.
npm WARN babel-example No license field.
```

Вывод терминала

Когда всё будет сделано, ваш файл `package.json` будет выглядеть похожим образом:

package.js

```
{
  "devDependencies": {
    "@babel/cli": "^7.0.0",
    "@babel/core": "^7.0.0",
  }
}
```

Что такое `package.json`?

Файл package.json содержит метаданные приложения или модуля. Самое главное, он включает список зависимостей для установки из npm при выполнении `npm install`. Если вы знакомы с Composer, этот файл похож на `composer.json`.

Для получения подробной информации о `package.json` взгляните на документацию npm[3].

[3] https://docs.npmjs.com/files/package.json

Каталог вашего проекта должен выглядеть так:

Имя	Размер	Последнее изменение
node_modules	130 объектов	09:43
package.json	87 байт	09:43

Каталог проекта

15.1.2 Конфигурация

Теперь, когда мы установили Babel, нам нужно явно указать, какие преобразования нужно выполнять при сборке. Поскольку мы хотим преобразовать код ES2015, мы установим @babel/preset-env[4].

Мы также создадим конфигурационный файл (`.babelrc`) для включения пресета.

```
npm install @babel/preset-env --save-dev

echo { "presets": ["@babel/preset-env"] } > .babelrc
```

Подсказка

Если вторая команда терпит неудачу, заключите содержимое файла в кавычки подобным образом:

```
echo '{ "presets": ["@babel/preset-env"] }' > .babelrc
```

15.1.3 Псевдоним для сборки

Вместо непосредственно выполнения Babel из командной строки, мы собираемся поместить наши команды внутри **npm-скриптов**.

Мы добавим поле `scripts` в файл `package.json` и зарегистрируем команду для запуска `babel` как скрипт с именем `build`.

Наш файл `package.json` будет выглядеть так:

[4] https://babeljs.io/docs/en/babel-preset-env

package.js

```
{
  "scripts": {
    "build": "babel src -d assets/js"
  },
  "devDependencies": {
    "@babel/cli": "^7.0.0",
    "@babel/core": "^7.0.0",
    "@babel/preset-env": "^7.0.0"
  }
}
```

Это действует как псевдоним, что означает, когда мы выполняем **`npm run build`**, фактически выполняется **`babel src -d assets/js`**. Эта команда указывает Babel транспилировать код из каталога **`src`** в каталог *`assets/js`*.

Но перед выполнением команды `build`, нам нужно сделать ещё кое-что. Для начала продолжайте и создайте вышеупомянутые каталоги (*src* и *assets/js*).

15.1.4 Использование

Давайте двигаться вперёд и поместим некоторые файлы в каталог *src*. Я создам файл `sum.js` с простой функцией `sum` и вызову её.

src/sum.js

```
const sum = (a, b) => a + b;
console.log(sum(5,3));
```

Вот и всё. Теперь мы можем запустить:

```
npm run build
```

При запуске этого, вы увидите в своём терминале, что файл **`src\sum.js`** скомпилирован в **`assets\js\sum.js`** и выглядит так:

assets/js/sum.js

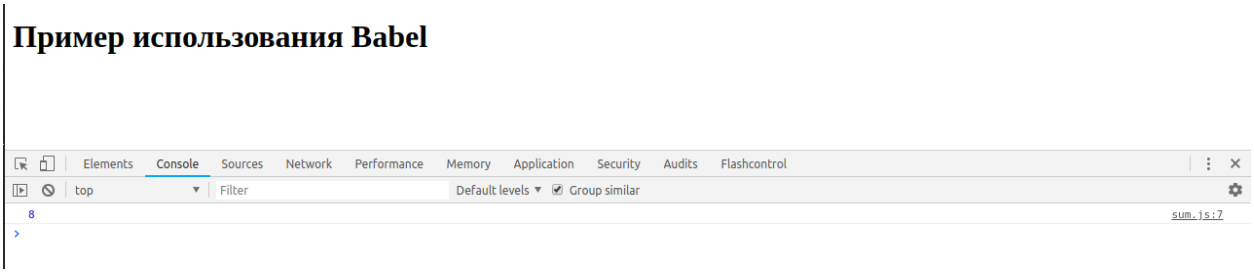
```
"use strict";

var sum = function sum(a, b) {
  return a + b;
};
console.log(sum(5, 3));
```

Теперь всякий раз, когда вы хотите скомпилировать код на ES6, вам нужно запустить команду `build`. **Довольно изящно, да?**

Пришло время просмотреть полученный файл `sum.js` в браузере. Я создам `sum.html` и подключу наш `js`.

sum.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Пример использования Babel</title>
</head>
<body>
  <h1>Пример использования Babel</h1>

  <script src="assets/js/sum.js"></script>
</body>
</html>
```



Вывод браузера

Как видите результат функции `sum` успешно распечатался в консоли.

Важная информация

Если вы захотите протестировать файл *.js*, но не хотите привлекать для этого браузер, вы можете выполнить его с помощью Node.js.

В примере *sum.js* есть строка `console.log(sum(5, 3))`, поэтому когда вы введёте `node sum.js` в терминале, вы увидите результат (**8**)!

15.1.5 Домашняя работа

Данное домашнее задание предназначено для того, чтобы помочь вам вспомнить всё, что вы узнали, путём создания примера, которые мы уже делали. Вместо **`sum.js`** используйте *классы из ES6* и создайте файл *`Ninja.js`* с определением класса `Ninja`.

У класса `Ninja` должно быть свойство *`name`* и метод *`announce`*, который будет предупреждать о наличии Ninja.

Пример

```
new Ninja('Леонардо').announce()
// alerts "Ниндзя Леонардо здесь!"
```

Не забудьте скомпилировать *js* с использованием Babel, прежде чем включать его в разметку *HTML*.

Подсказка

Пример создания классов вы можете найти в предыдущей главе.

Вторая подсказка

Не забудьте выполнить **`npm run build`** каждый раз при внесении изменений в файл *js*, в противном случае сборка приложения не обновится!

Возможное решение

Вы можете найти потенциальное решение к этому упражнению здесь[5].

[5] https://github.com/hootlex/the-majesty-of-vuejs-2/tree/ru/homework/Chapter15/chapter15.1

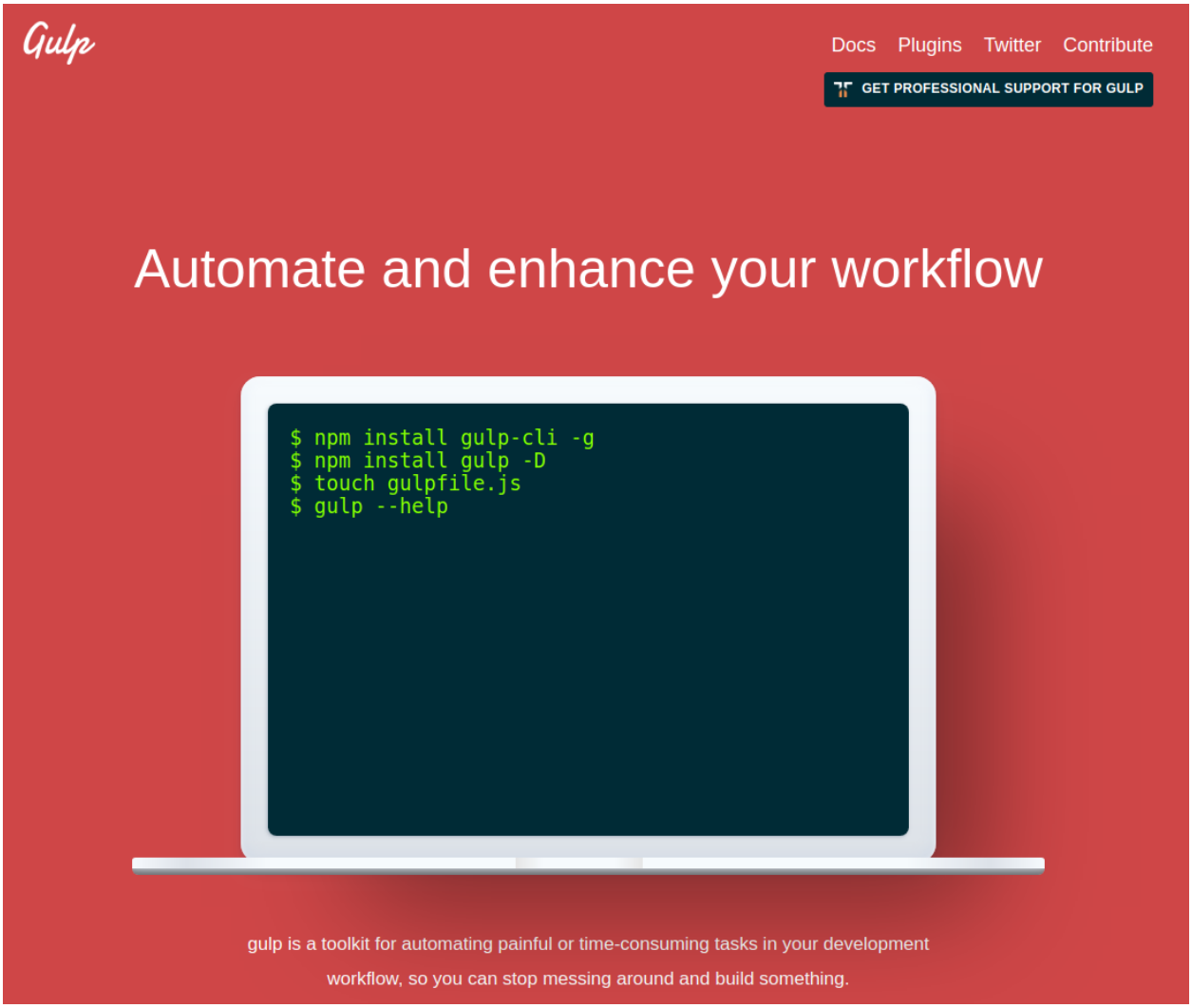
15.2 Автоматизация рабочего процесса с помощью Gulp

15.2.1 Менеджеры задач

Если потратили время приложению из предыдущего раздела домашней работы, вы, скорее всего, обратили внимание, что немного раздражает запускать `npm run build` каждый раз, когда вы изменяете код.

Вот где **менеджеры задач (task runners)**, такие как Gulp[6] или Grunt[7], приходят на помощь. Менеджеры задач позволяют автоматизировать и улучшить рабочий процесс.

[6] https://gulpjs.com/
[7] https://gruntjs.com/



Gulp

Зачем использовать менеджер задач?

Одним словом: **Автоматизация**. Чем меньше усилий вы прилагаете для выполнения целого ряда повторяющихся задач, таких как минификация (сжатие), компиляция, модульное тестирование, проверка кода и т.д., тем легче становится ваша работа.

Gulp или Grunt

Grunt, как и Gulp — инструмент для определения и выполнения задач. Основное различие между Grunt и Gulp в том, что Grunt определяет задачи с помощью **объектов конфигурации**, когда как Gulp делает то же самое через **JavaScript-функции**. Поскольку Gulp выполняет JavaScript, это обеспечивает большую гибкость при написании задач.

У обоих есть огромная библиотека плагинов, где можно найти тот плагин, который подходит для нужной вам задачи.

15.2.2 Установка

Я покажу код примера использования Gulp для отслеживания изменения в файлах с расширением `js` и автоматического выполнения команда **build**.

Для начала нужно установить Gulp глобально:

```
npm install gulp-cli --global
```

Затем установим Gulp в зависимостях разработки (`devDependencies`) в проекте:

```
npm install gulp --save-dev
```

Теперь, когда Gulp установлен, мы создадим **`gulpfile.js`** в корне проекта:

gulpfile.js

```
const gulp = require('gulp');

gulp.task('default', function () {
  // код для задачи по умолчанию
});
```

15.2.3 Использование

Когда мы запускаем **`gulp`** в консоли, он выполняется, но ничего не делает. Нам нужно настроить задачу по умолчанию.

Для выполнения `babel` напрямую, я установлю npm-плагин gulp-babel[8].

```
npm install gulp-babel --save-dev
```

Я добавлю новую *задачу gulp* с именем *`babel`* и установлю её как задачу по умолчанию. Мой `gulpfile` будет выглядеть так:

[8] https://www.npmjs.com/package/gulp-babel

gulpfile.js

```
const gulp = require('gulp');
const babel = require('gulp-babel');

gulp.task('default', ['babel']);

// основная задача Babel
gulp.task('babel', function () {
  return gulp.src('src/*.js')
  .pipe(babel({
    presets: ['@babel/preset-env']
  }))
  .pipe(gulp.dest('assets/js/'))
})
```

Эта задача в основном указывает Babel преобразовывать все файлы *js* в каталоге *src* с использованием пресета `es2015` в каталог *assets/js*.

15.2.4 Отслеживание

В настоящее время у команды **`gulp`**, введённой в консоли, тот же эффект, что у **`npm run build`**. Мы хотим выполнить эту задачу каждый раз, когда файл *js* изменился. Для этого мы создадим *наблюдатель* внутри *gulpfile* следующим образом:

gulpfile.js

```
const gulp = require('gulp');
const babel = require('gulp-babel');

gulp.task('default', ['watch']);

// основная задача Babel
gulp.task('babel', function () {
  return gulp.src('src/*.js')
  .pipe(babel({
    presets: ['@babel/preset-env']
  }))
  .pipe(gulp.dest('assets/js/'))
})

// задача отслеживания
gulp.task('watch', function () {
```

```
  gulp.watch('src/*.js', ['babel']);
})
```

Когда мы запускаем **`gulp watch`** в консоли, Gulp отслеживает изменения в файлах с расширением *`.js`* в указанном каталоге. Каждый раз, когда мы вносим изменения, gulp выполняет задачу *`babel`* и файлы в *`assets/js`* будут обновлены. **Разве это не круто?**

15.2.5 Домашняя работа

Это домашняя работа следует за предыдущей. Если вы не сделали предыдущее, никогда не поздно начать!

Поскольку эта часть главы посвящена , нам нужно настроить наблюдателя с помощью Gulp и скомпилировать код, используя *Babel*, когда обнаружено изменение.

Примечание

Возможно вы заметили, что при выполнении Gulp печатает сообщение в терминале (***"Starting"*** - ***"Finished"***), поэтому не торопитесь и дождитесь применения изменений.

Возможное решение

Вы можете найти потенциальное решение к этому упражнению здесь[9].

```
lex@lex-desktop  ~/vue_book/babel-example  gulp watch
[10:05:43] Using gulpfile ~/vue_book/babel-example/gulpfile.js
[10:05:43] Starting 'watch'...
[10:05:43] Finished 'watch' after 5.42 ms
[10:05:52] Starting 'babel'...
[10:05:52] Finished 'babel' after 258 ms
```

Gulp отслеживает изменения!

15.3 Сборка модулей с помощью Webpack

15.3.1 Сборщики модулей

Наш рабочий процесс хорошо работает с текущим кодом *`sum.js`*.

Мы расширим его возможности для расчёта стоимости пиццы и пива с сообщением об этом клиенту.

[9] https://github.com/hootlex/the-majesty-of-vuejs-2/tree/ru/homework/Chapter15/chapter15.2

src/sum.js

```
const pizza = 10
const beer = 5

const sum = (a, b) => a + b + '$';
console.log(`Алекс, вам надо заплатить ${sum(pizza, beer)}`)
```

Этот код выглядит неплохо, но если предположить, что не у всех людей имя *Алекс*, в таком случае создадим файл **`client.js`**, предоставляющий имя клиента.

src/client.js

```
export const name = 'Алекс'
```

Теперь мы импортируем имя из недавно созданного файла.

src/sum.js

```
import { name } from './client'

const pizza = 10
const beer = 5

const sum = (a, b) => a + b + '$';
console.log(`${name}, вам надо заплатит ${sum(pizza, beer)}`)
```

Отлично! Если выполнить **`node assets/js/sum.js`**, мы получим ожидаемый результат.

```
lex@lex-desktop  ~/vue_book/babel-example  node assets/js/sum.js
Алекс, вам надо заплатит 15$
lex@lex-desktop  ~/vue_book/babel-example
```

Вывод sum.js

Вы ожидаете, что подобный вывод при открытии файла *`html`* в браузере, **но это не произойдёт**! Вместо это появляется ошибка.

Пример использования Babel

```
Elements  Console  Sources  Network  Performance  Memory  Application  Security  Audits  Flashcontrol
top  Filter  Default levels  Group similar
Uncaught ReferenceError: require is not defined
    at sum.js:3                                                    sum.js:3
>
```

Require не определён

Проверьте `node assets/js/sum.js` и обратите внимание на `var _client = require('./client');` вверху. Причина ошибки в браузере заключается в том, что **`require()` не существует в браузере/JavaScript на стороне клиента**. Нам нужно *объединить* (*bundle*) модули в один файл, для включения его в тег `<script>`.

```
JS sum.js ×
1  "use strict";
2
3  var _client = require("./client");
4
5  var pizza = 10;
6  var beer = 5;
7
8  var sum = function sum(a, b) {
9    return a + b + '$';
10 };
11
12 console.log("".concat(_client.name, ", \u0432\u0430\u043C \u043D\u0430\u0434\u043E
   \u0437\u0430\u043F\u043B\u0430\u0442\u0438\u0442 ").concat(sum(pizza, beer)));
```

assets/js/sum.js

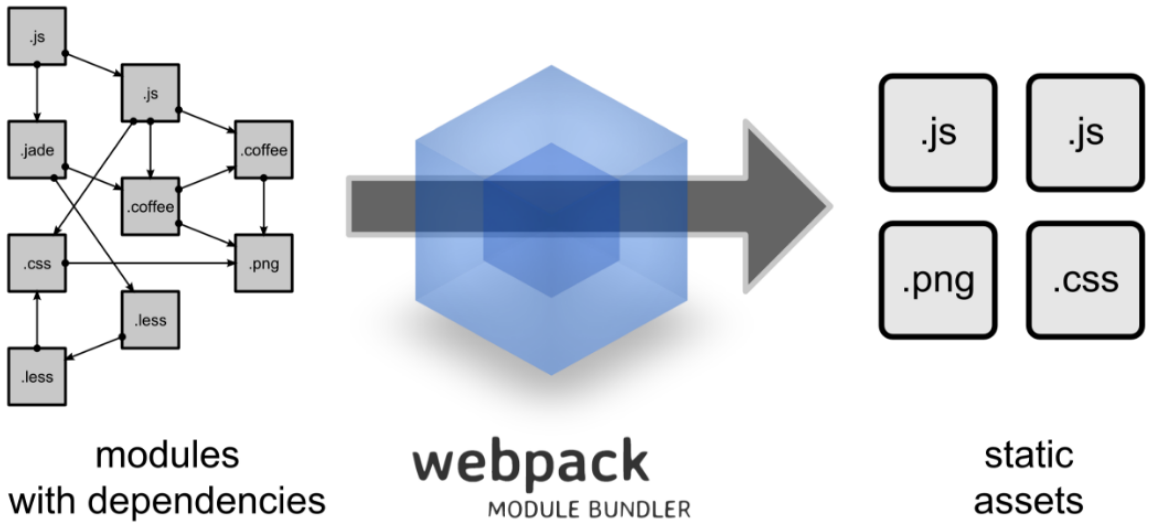
Именно в этом случае нам необходимы **сборщики модулей** (**module bundlers**), такие как Webpack[10] или Browserify[11].

15.3.2 Webpack

Webpack — сборщик модулей. Он принимает JavaScript-модули, анализирует их зависимости, а затем объединяет и создаёт статичные ресурсы, представляющие эти модули.

Я буду использовать Webpack в следующих примерах. Это потому, что я считаю, что в будущем это будет полезно для других целей, поскольку этот инструмент может сделать больше, чем просто сборка модулей. Использование *загрузчиков*, мы можем научить Webpack преобразовывать все типы файлов так, как мы хотим, перед тем, как будет готов окончательный бандл.

[10] https://webpack.github.io/
[11] http://browserify.org/



Что делает Webpack

15.3.3 Установка

Сначала я собираюсь установить Webpack глобально, а затем добавить его в качестве зависимости проекта.

```
npm install webpack -g

npm install webpack --save-dev
```

Подсказка

На момент написания есть известный баг при использовании Vagrant[12] на Windows, так что выполнение `npm install` может завершиться неудачно. Для решения этой проблемы, закройте Vagrant, перейдите (`cd`) в ваш проект в своём терминале Windows и выполните там `npm install`.

15.3.4 Использование

Чтобы скомпилировать JavaScript, нам нужно предоставить Webpack исходную точку и точка вывода. В нашем случае в качестве исходной точкой будет *`assets/js/sum.js`*, а точкой вывода будет *`assets/webpacked/app.js`*.

[12] https://www.vagrantup.com/

```
webpack assets/js/sum.js assets/webpacked/app.js
```

```
alex@192: /babel-example
/babel-example » webpack assets/js/sum.js assets/webpacked/app.js
Hash: 7d079f20fab1a5cd6563
Version: webpack 1.13.0
Time: 79ms
 Asset     Size  Chunks             Chunk Names
app.js  1.78 kB       0  [emitted]  main
   [0] ./assets/js/sum.js 192 bytes {0} [built]
   [1] ./assets/js/client.js 113 bytes {0} [built]

/babel-example »
```

Вывод Webpack

После завершения сборки мы можем использовать готовый файл *assets/webpacked/app.js* внутри *sum.html*. Мы также можем выполнить его в терминале, используя **node assets/webpacked/app.js**.

15.3.5 Автоматизация

Если вы ленивы, как и я, вам не понравится запускать *webpack* каждый раз при новых изменениях, поэтому мы можем автоматизировать этот процесс. Можно настроить Webpack для отслеживания исходной точки и пересобирать бандл, когда какой-либо из её файлов изменится. Вместо это я собираюсь интегрировать Webpack в Gulp для демонстрации, как можно совместно использовать несколько инструментов.

Дальнейшее обучение

Если вы хотите узнать больше о том, как работает Webpack и как его можно настроить, прочитайте "Beginner's guide to Webpack"[13] by Nader Dabit[14].

Для интеграции Webpack в Gulp я буду использовать плагин webpack-stream[15].

[13] https://medium.com/@dabit3/beginner-s-guide-to-webpack-b1f1a3638460
[14] https://twitter.com/dabit3
[15] https://www.npmjs.com/package/webpack-stream

```
npm install webpack-stream --save-dev
```

После установки я создам новую задачу с именем ‘webpack’ и укажу Gulp запускать её каждый раз, когда он обнаруживает изменение, сразу после выполнения задачи ‘babel’.

gulpfile.js

```
const gulp = require('gulp');
const babel = require('gulp-babel');
const webpack = require('webpack-stream');

gulp.task('default', ['watch']);

// основная задача babel
gulp.task('babel', function () {
  return gulp.src('src/*.js')
    .pipe(babel({
      presets: ['@babel/preset-env']
    }))
    .pipe(gulp.dest('assets/js/'))
})

// основная задача webpack
gulp.task('webpack', ['babel'], function () {
  return gulp.src('assets/js/sum.js')
    .pipe(webpack({
      output: {
        path: "/assets/webpacked",
        filename: "app.js"
      }
    }))
    .pipe(gulp.dest('assets/webpacked'));
})

// задача отслеживания
gulp.task('watch', function () {
  gulp.watch('src/*.js', ['babel', 'webpack']);
})
```

Это не идеальное решение, а скорее просто демонстрация того, как можно связать всё вместо то, что мы изучили. В продакшене есть намного лучшие способы автоматизации ваших webpack-задач.

15.4 Резюме

Когда вы хотите компилировать ES6, вы можете использовать **Babel**[16].

Для автоматизации таких операций как минификация, компиляция SASS/LESS и др., вам нужны менеджеры задач, такие как **Gulp**[17] или **Grunt**[18].

Для сборки модулей используйте **Webpack**[19] или **Browserify**[20].

В следующей главе мы погрузимся в изучение однофайловых компонентов во Vue и будем использовать несколько инструментов, предоставляемых Vue вместе с теми, которые мы изучили.

Примечание

Если эта глава показалась вам сложной для понимания, не беспокойтесь по этому поводу. Вам не нужно помнить обо всём этом. Это была просто демонстрация, чтобы дать вам как можно понятное объяснение, как это работает. В следующей главе мы будем использовать *project-templates*. В нём, например, **сборка модулей**, **автоматизация**, **сборка при изменении** и многое другое, уже реализованы, и мы воспользуемся ими.

[16] http://babeljs.io/
[17] https://gulpjs.com/
[18] https://gruntjs.com/
[19] https://webpack.github.io/
[20] http://browserify.org/

16. Работа с однофайловыми компонентами

Как и обещали, в этой главе мы собираемся рассмотреть однофайловые компоненты. Чтобы использовать однофайловые Vue-компоненты, нам необходимы инструменты вроде **Webpack** с **vue-loader** или **Browserify** с **vueify**. Для наших примеров мы будем использовать Webpack, который мы уже видели в работе. Если вы предпочитаете Browserify или что-то ещё, не стесняйтесь использовать его.

Однофайловые компоненты (или Vue-компоненты) совмещают CSS-стили, шаблон и JavaScript-код в одном файле с расширением `.vue`. Webpack занимается связыванием этого нового типа файла с другими файлами.

Webpack использует vue-loader[1], чтобы преобразовывать Vue-компоненты в обычные JavaScript-модули. *vue-loader* также предоставляет очень хороший набор функций, например, включённый по умолчанию стандарт ES2015, стили с ограниченной областью видимости для каждого компонента и т.д.

16.1 `vue-cli`

Чтобы избежать настройки Webpack и создания нового рабочего процесса с нуля, мы воспользуемся официальным инструментом **`vue-cli`**.

С момента выхода последней версии оригинальной книги **`vue-cli`** претерпел значительные изменения и обновился до 3.x версии, которая будет далее использоваться в книге. Однако процесс создания нового Vue-проекта с использованием ветки 2.x также будет сохранён в связи с наличием в оригинальной версии книги. Каких-либо причин использовать vue-cli версии 2.x вместо 3.x нет.

16.1.1 Устаревшая версия (2.x)

Важная информация

vue-cli[2] — это простой *интерфейс командной строки* для создания шаблона проектов на Vue.js.

[1] https://github.com/vuejs/vue-loader
[2] https://github.com/vuejs/vue-cli

Этот потрясающий инструмент является самым быстрым способом получить предварительно настроенную сборку. Он предоставляет шаблоны с горячей заменой модулей, проверкой (линтингом) кода при сохранении, модульным тестированием и ещё многим другим. На данный момент предоставляются шаблоны каркаса с webpack и browserify, но при необходимости вы можете создать ваш собственный[3].

16.1.1.1 Шаблоны

CLI берет шаблоны из официального репозитория шаблонов Vue.js[4], где сейчас доступно 6 шаблонов. Вы можете проверить на GitHub, что они представляют из себя.

Все шаблоны включают файл *package.json*, который управляет зависимостями проекта и поставляется с предустановленными NPM-скриптами.

Используя шаблоны проекта, вы сразу получаете множество возможностей. Например, описание шаблона «webpack» говорит, что это *«Полнофункциональная настройка Webpack + vue-loader с горячей заменой модулей, линтингом кода, тестированием и извлечением CSS-стилей.»*

16.1.1.2 Установка

Мы собираемся придерживаться решения с настроенным Webpack и устанавливаем глобально **`vue-cli`** следующей командой.

```
npm install vue-cli -g
```

16.1.1.3 Использование

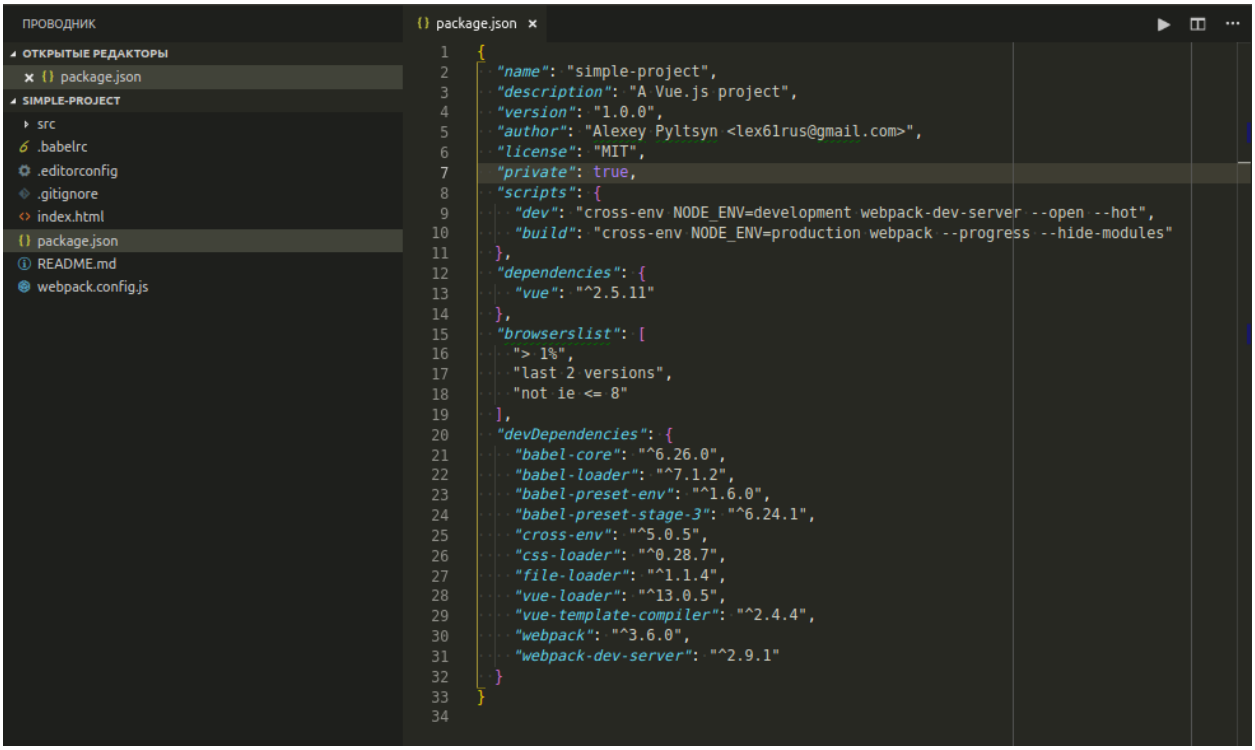
С помощью CLI вы можете запустить **`vue init <template-name> <project-name>`**, где **`<template-name>`** — это названием шаблона (официального или пользовательского), и **`<project-name>`** — название папки/проекта, который вы собираетесь создать.

Таким образом, если вы запустите следующую команду...

```
vue init webpack-simple simple-project
```

...то получите папку `simple-project` со следующей структурой:

[3] https://github.com/vuejs/vue-cli/tree/v2/#custom-templates
[4] https://github.com/vuejs-templates



структура шаблона webpack-simple

В нашем примере мы будем использовать полнофункциональный шаблон webpack, поэтому команда будет выглядеть примерно так:

```
vue init webpack stories-project
```

Подсказка

Используйте `vue list`, чтобы увидеть все доступные официальные шаблоны.

На заметку

При создании нового проекта вам придётся ввести некоторые детали, например, название, версию, автора, сборку и т.д. Каждый раз, когда вы будете использовать cli для создания нового проекта, выбирайте сборку `Runtime + Compiler`, потому что нам необходимо компилировать шаблоны на лету (например, передавая строку в опцию `template` или монтируя к элементу, используя встроенный в DOM HTML-код как шаблон). Вы можете найти подробные объяснения различных сборок в официальном руководстве[5].

[5] https://ru.vuejs.org/v2/guide/installation.html#%D0%9E%D0%B1%D1%8A%D1%8F%D1%81%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5-%D1%80%D0%B0%D0%B7%D0%BB%D0%B8%D1%87%D0%BD%D1%8B%D1%85-%D1%81%D0%B1%D0%BE%D1%80%D0%BE%D0%BA

В какой-то момент вам будет предложено **Выбрать ESlint preset**. К доступным вариантам относятся feross/standard[6] и airbnb/javascript[7].

Я создал таблицу для сравнения этих двух стилей, чтобы вы могли лучше понять, какие правила каждый стиль применяет.

Правила	feross/standard	airbnb/javascript
Отступ	2 пробела	2 пробела
Точки с запятой	Нет!	Да
Неиспользуемые переменные	Не допускается	Не допускается
Строковые кавычки	Одинарные	Одинарные
Использование === вместо ==	Да	Да
Допустимое число пустых строк	1	2
Пробел после имени функции	Да	Нет
Начало строки с [	Нет	Да
Окончание файлов символом новой строки w/	Да	Да
Разрешены завершающие запятые	Нет	Нет

Сравнение пресетов Standard и Airbnb

Правила в таблице являются частью многих принятых в каждом стиле. Чтобы рассмотреть и решить, что подходит вам лучше всего, ознакомьтесь с полным списком правил в их GitHub-репозиториях.

После выбора стиля вас спросят об установке некоторых инструментов вроде Karma-Mocha[8] и Nightwatch[9]. Нам не нужны эти инструменты прямо сейчас, так что отвечайте на эти вопросы отрицательно и заканчивайте с настройкой.

```
lex@lex-desktop  ~/vue_book  vue init webpack stories-project

? Project name stories-project
? Project description A Vue.js project
? Author Alexey Pyltsyn <lex61rus@gmail.com>
? Vue build standalone
? Install vue-router? No
? Use ESLint to lint your code? Yes
? Pick an ESLint preset Standard
? Set up unit tests No
? Setup e2e tests with Nightwatch? No
? Should we run `npm install` for you after the project has been created? (recommended) npm

   vue-cli · Generated "stories-project".


# Installing project dependencies ...
# ========================
```

Установка Vue-шаблона

[6] https://github.com/feross/standard
[7] https://github.com/airbnb/javascript
[8] https://github.com/karma-runner/karma-mocha
[9] http://nightwatchjs.org/

На заметку

Karma — это плагин, адаптер для фреймворка для тестирования Mocha[10].

Nightwatch позволяет писать автоматические тесты для браузера, которые выполняются на сервере Selenium[11].

16.1.1.4 Шаблон Webpack

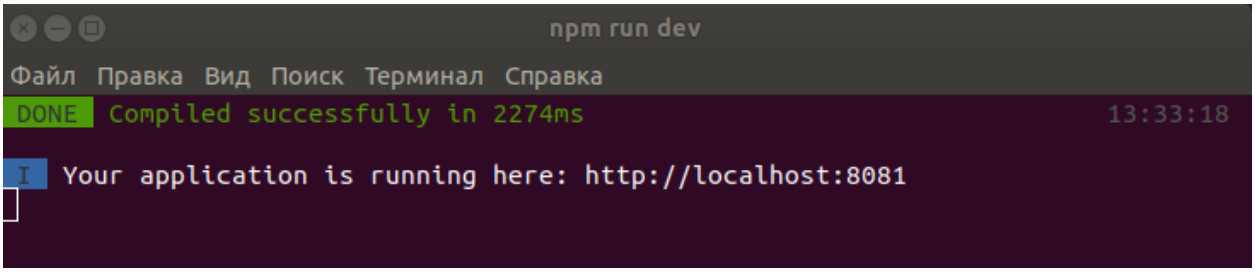
На заметку

Вы можете найти больше информации о структуре проекта шаблона Webpack в его документации[12]

Для завершения настройки нашего проекта нам необходимо установить зависимости. Давайте двигаться дальше и запустим:

```
cd stories-classic-project

npm run dev
```

Терминал выведет что-то вроде ***Your application is running here: http://localhost:8080***.



Запуск сервера

Предупреждение

Будьте осторожны, вам следует быть точным в вашем коде. Иначе вы будете получать ошибки о лишних пустых строках между блоками, пробелах в конце строк, отступах, которые больше или меньше двух пробелов, и о многом другом, что не соответствует правилам выбранного стиля.

[10] https://mochajs.org/
[11] https://www.seleniumhq.org/
[12] http://vuejs-templates.github.io/webpack/structure.html

16.1.2 Новая версия (3.x)

На заметку

vue-cli[13] третьей версии - это уже не просто *интерфейс командной строки* для быстрого создания проекта Vue.js, а полноценная система для быстрой разработки на Vue.js. Более подробно узнать, что из себя представляет новая версия `vue-cli` можно на официальном сайте[14]

Важная информация

Имя пакета изменилось с `vue-cli` на `@vue/cli`. Если у вас установлена глобально предыдущая версия пакета vue-cli (1.x или 2.x), то необходимо сначала удалить её командой `npm uninstall vue-cli -g` или `yarn global remove vue-cli`.

16.1.2.1 Установка

Для установки Vue CLI требуется Node.js версии 8.9 или выше, выполните следующую команду ниже:

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

Проверить корректность установки можно выполнением команды `vue`.

16.1.2.2 Настройка проекта

Для создания заготовки для нашего проекта выполните команду ниже:

```
vue create stories-project
```

Вам будет предложено выбрать пресет настроек (по аналогии со старой версией). Можно выбрать пресет по умолчанию, который добавляет Babel и ESLint, или настроить пресет самостоятельно («Manually select features»), но мы не будем показывать доступные опции, если вам это интересно, то вы можете узнать об этом в документации[15]. Упомянем только про

[13] https://github.com/vuejs/vue-cli
[14] https://cli.vuejs.org/ru/
[15] https://cli.vuejs.org/ru/guide/creating-a-project.html#vue-create

различия в стиле написания JavaScript-кода, небольшую сравнительную таблицу вы можете посмотреть здесь.

```
lex@lex-desktop: ~/vue_book
Файл  Правка  Вид  Поиск  Терминал  Справка
lex@lex-desktop  ~/vue_book  vue create stories-project


Vue CLI v3.0.1
? Please pick a preset: default (babel, eslint)


Vue CLI v3.0.1
✨  Creating project in /home/lex/vue_book/stories-project.
🗃  Initializing git repository...
⚙  Installing CLI plugins. This might take a while...


> yorkie@2.0.0 install /home/lex/vue_book/stories-project/node_modules/yorkie
> node bin/install.js

setting up Git hooks
done

stories-project@0.1.0 /home/lex/vue_book/stories-project
```

Создание проекта с пресетом по умолчанию

16.1.2.3 Запуск приложения

После создания нового проект с пресетом по умолчанию, перейдите в каталог `stories-project` и выполните следующую команду:

```
npm run serve
```

```
npm run serve
Файл  Правка  Вид  Поиск  Терминал  Справка
DONE  Compiled successfully in 2652ms                    14:59:33


  App running at:
  - Local:   http://localhost:8081/
  - Network: http://192.168.1.9:8081/

  Note that the development build is not optimized.
  To create a production build, run npm run build.
```

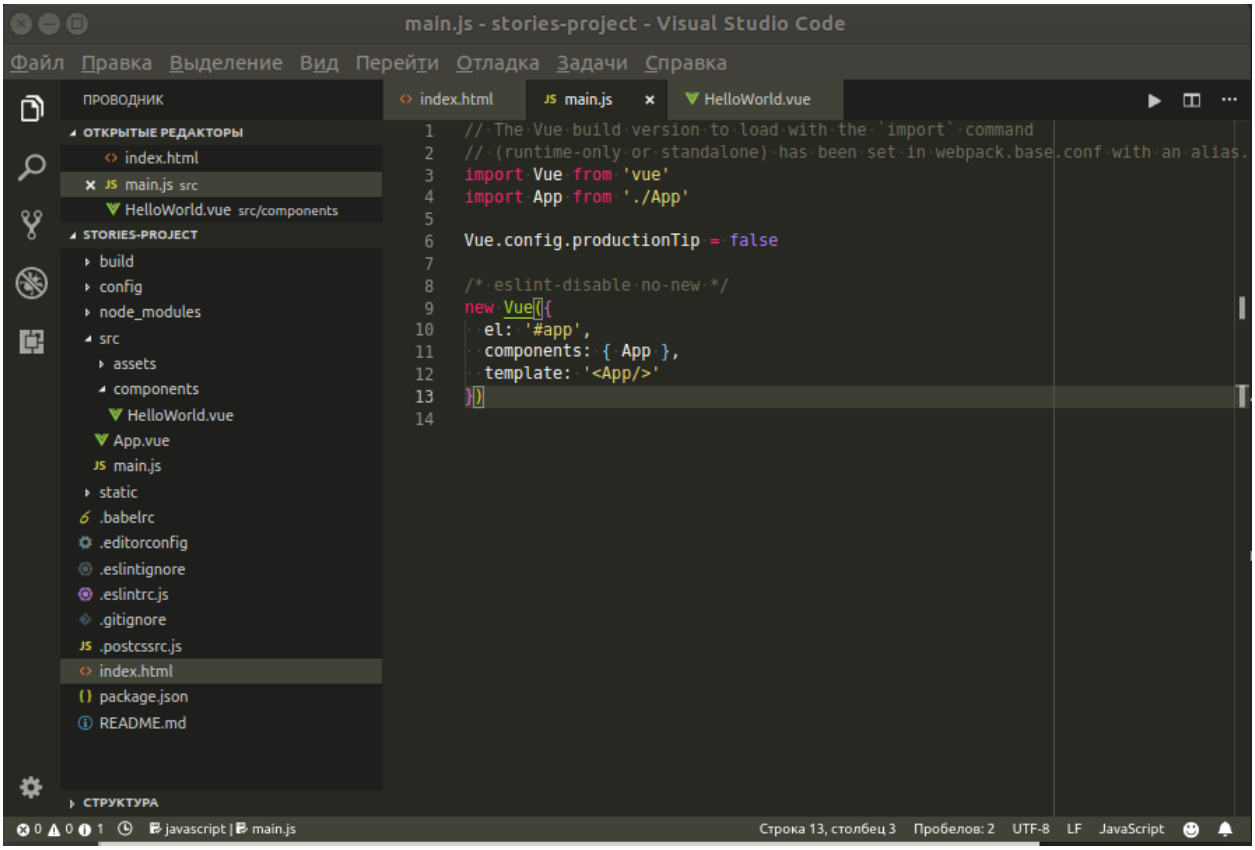
Запуск приложения в Vue CLI 3

Сравнение Vue CLI 2.x и 3.x

В старой версии команде `npm run serve` из новой версии соответствовала команда `npm run dev`.

16.1.3 Структура проекта

После завершения вышеуказанных шагов, вне зависимости от использованной версии Vue CLI, у вас должен быть каталог проекта, заполненный всеми необходимыми файлами.



Структура шаблона Webpack

Сравнение Vue CLI 2.x и 3.x

Структура проекта, созданная Vue CLI старой и новой версиями, — аналогичная, за исключением некоторых изменений в содержании файлов, но в целом многое похоже.

Файлы, с которыми вы обычно будете работать:

1. **`public/index.html`** (в старой версии находится в корне проекта)
2. файлы с каталогах **`src`** и **`src/components`**

16.1.4 index.html

Давайте начнём с **`index.html`** в каталоге `public`. Он должен выглядеть примерно так

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <link rel="icon" href="<%= BASE_URL %>favicon.ico">
    <title>stories-project</title>
  </head>
  <body>
    <noscript>
      <strong>We're sorry but stories-project doesn't work properly without JavaScri\
pt enabled. Please enable it to continue.</strong>
    </noscript>
    <div id="app"></div>
    <!-- built files will be auto injected -->
  </body>
</html>
```

Как видите, это довольно простая настройка с уже добавленным элементом с идентификатором `app`, в котором монтируется Vue-приложение. Комментарий относится к скрипту, **`app.js`**, который является выводом Webpack. Это означает, что после того, как Webpack соберёт скрипты, они будут автоматически добавлены здесь, т.е. вам не нужно добавлять их вручную.

16.1.5 HelloWorld.vue

Далее переходите в *`src/components`* и откройте файл *`HelloWorld.vue`* (*`Hello.vue`* в старой версии Vue CLI), чтобы увидеть, как выглядит файл с расширением *`.vue`*.

src/components/HelloWorld.vue

```
<template>
  <div class="hello">
    <h1>{{ msg }}</h1>
    <!-- ... -->
  </div>
</template>

<script>
export default {
  name: 'HelloWorld',
  props: {
```

```
    msg: String
  }
}
</script>

<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped>
h3 {
  margin: 40px 0 0;
}
/* ... */
</style>
```

Внутри тега **`<script>`**, компонент включает только входные параметры (свойство *`props`*). Нет нужды определять шаблон. Он будет автоматически прикреплён, если существует элемент **`<template>`**. Элемент **`<template>`** определяет шаблон компонента, конечно. Думайте о **`<template>`** как о теге **`<HelloWorld>`** в вашем HTML.

У нас мог бы быть только элемент **`<script>`**, но в этом случае мы не смогли бы получить преимуществ однофайловых компонентов.

Помните, что **`export`**, возможность из ES6, управляется установленными нами ранее прекрасными инструментами (транспиляторами и сборщиками модулей).

Предупреждение

Каждый файл с расширением *`.vue`* должен включать не более одного блока **`<script>`**. Каждый шаблон должен включить только **один корневой элемент** вроде **`<div id=hello>...</div>`**, который содержит все остальные элементы.

Скрипт должен экспортировать *объект опций компонента Vue.js*. Также поддерживается экспорт расширенного конструктора, созданного через **`Vue.extend()`**, но обычный объект предпочтительнее.

ES6 позволяет нам иметь любое количество выражений экспорта (**`export`**), но в однофайловых компонентах выражение экспорта может быть только одно.

При попытке добавить дополнительные выражения экспорта вы получите предупреждение, похожее на это: **`[vue-loader] src/components/HelloWorld.vue: named exports in /*.vue files are ignored`**.

Блок **`<style>`** определяет CSS-стили, как и ожидалось.

В старой версии Vue CLI компонент создавался сл свойством с данными (опция **`data`**) вместо использования в новой версии входных параметров, но в старой версии было установлено значение по умолчанию, что было весьма удобно, давайте сделаем аналогичное и изменим

компонент *HelloWorld.vue*, добавив значение по умолчанию для входного параметра msg, кроме того для краткости и удобства переименуем компонент в *Hello.vue*.

src/components/Hello.vue

```
<!-- ... -->

<script>
export default {
  name: 'HelloWorld',

  props: {
    msg: String
    msg: {
      type: String,
      default: 'Добро пожаловать в ваше приложение на Vue.js'
    }
  }
}
</script>

<!-- ... -->
```

16.1.6 App.vue

Файл *App.vue* расположен в папке *src* и включает основной шаблон приложения. Этот компонент обычно ответственный за включение других компонентов.

App.vue имеет ещё немного строк с текстами и стилями, но раз мы сконцентрированы на структуре, то немного сократили его.

src/App.vue

```
<template>
  <div id="app">
    <img alt="Vue logo" src="./assets/logo.png">
    <hello></hello>
  </div>
</template>

<script>
import Hello from './components/Hello.vue'

export default {
```

```
  name: 'app',
  components: {
    Hello
  }
}
</script>

<style>
/* ... */
</style>
```

У него такая же структура, как у файла *Hello.vue*, который мы видели выше. По умолчанию, существует объект **components**, который включает компонент Hello. Внутри этого объекта мы будем импортировать любые новые компоненты. В шаблоне существует тег **<hello></hello>**, и поэтому шаблон компонента Hello будет отображён.

Welcome to Your Vue.js App

For guide and recipes on how to configure / customize this project, check out the vue-cli documentation.

Installed CLI Plugins

babel eslint

Essential Links

Core Docs Forum Community Chat Twitter News

Ecosystem

vue-router vuex vue-devtools vue-loader awesome-vue

Домашняя страница проекта

16.1.7 main.js

Файл *main.js* в *src*, как вы понимаете, является нашим основным скриптом.

import Vue from ‘vue’ import App from ‘./App.vue’

Vue.config.productionTip = false

new Vue({ render: h ⇒ h(App) }).$mount('#app')

Он импортирует *Vue* как модуль из *node_modules* и компонент *App* из папки *src*. Ниже находится наш экземпляр *Vue* и в объекте `components` — компонент *App*.

Всякий раз, когда вам нужно импортировать глобально скрипт или компонент, вы можете помещать его в *main.js*.

Заметка

Опция **`render: h => h(App)'`** представляет вывод компонента `App`, который будет встроен в *index.html*, как мы упоминали ранее.

`<App/>` имеет такой же вывод, как **`<App></App>`** и **`<app></app>`**.

16.2 Формирование файлов *.vue*

Мы увидели, как выглядит однофайловый компонент и как он используется в проекте. Настало время создать несколько новых компонентов для сценария из реальной жизни. Предположим, вы хотите создать некоторую социальную сеть или форум, где пользователи публикуют свои истории и эксперименты. Для создания нашего стартапа нам необходимо создать 2 формы, страницу для регистрации и входа, страницу для отображения историй пользователей.

Перед тем, как начать, мы включим глобально Bootstrap, чтобы можно было использовать его стили во всех наших компонентах. Чтобы сделать это, необходимо обновить наш *index.html*.

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <link rel="icon" href="<%= BASE_URL %>favicon.ico">
    <title>stories-project</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css\
/bootstrap.min.css">
  </head>
  <body>
    <!-- ... -->
  </body>
</html>
```

А ещё в элементе `template` удалите часть со ссылками в компоненте *`Hello.vue`*, т.е. оставив только вывод `msg` в элементе `h1`.

index.html

```
<template>
  <div class="hello">
    <h1>{{ msg }}</h1>
    <p>
      For guide and recipes on how to configure / customize this project,<br>
      check out the
      <a href="https://cli.vuejs.org" target="_blank" rel="noopener">vue-cli documen\
tation</a>.
    </p>
    <h3>Installed CLI Plugins</h3>
    <ul>
      <li><a href="https://github.com/vuejs/vue-cli/tree/dev/packages/%40vue/cli-plu\
gin-babel" target="_blank" rel="noopener">babel</a></li>
      <li><a href="https://github.com/vuejs/vue-cli/tree/dev/packages/%40vue/cli-plu\
gin-eslint" target="_blank" rel="noopener">eslint</a></li>
    </ul>
    <h3>Essential Links</h3>
    <ul>
      <li><a href="https://vuejs.org" target="_blank" rel="noopener">Core Docs</a></\
li>
      <li><a href="https://forum.vuejs.org" target="_blank" rel="noopener">Forum</a>\
</li>
      <li><a href="https://chat.vuejs.org" target="_blank" rel="noopener">Community \
Chat</a></li>
      <li><a href="https://twitter.com/vuejs" target="_blank" rel="noopener">Twitter\
</a></li>
      <li><a href="https://news.vuejs.org" target="_blank" rel="noopener">News</a></\
li>
    </ul>
    <h3>Ecosystem</h3>
    <ul>
      <li><a href="https://router.vuejs.org" target="_blank" rel="noopener">vue-rout\
er</a></li>
      <li><a href="https://vuex.vuejs.org" target="_blank" rel="noopener">vuex</a></\
li>
      <li><a href="https://github.com/vuejs/vue-devtools#vue-devtools" target="_blan\
k" rel="noopener">vue-devtools</a></li>
      <li><a href="https://vue-loader.vuejs.org" target="_blank" rel="noopener">vue-\
loader</a></li>
```

```
      <li><a href="https://github.com/vuejs/awesome-vue" target="_blank" rel="noopen\
er">awesome-vue</a></li>
    </ul>
  </div>
</template>
```

Теперь давайте создадим форму входа в новом файле `Login.vue`.

src/components/Login.vue

```
<template>
  <div id="login">
    <h2>Вход</h2>
    <input type="email" placeholder="Электронная почта">
    <input type="password" placeholder="Пароль">
    <button type="submit" class="btn">Войти</button>
  </div>
</template>

<script>
export default {
  created () {
    console.log('логин')
  }
}
</script>
```

Вот так. Чтобы отобразить файл в браузере, нам нужно добавить наш компонент *Login* куда-нибудь. Поэтому мы импортируем его в основной компонент *App* и добавим его в объект `components`.

src/App.vue

```
<template>
  <div id="app">
    <img src="./assets/logo.png">
    <hello></hello>
  </div>
</template>

<script>
import Login from './components/Login.vue'
import Hello from './components/Hello.vue'
```

```
export default {
  name: 'app',
  components: {
    Hello,
    Login
  }
}
</script>

<style>
/* ... */
</style>
```

Если вы обновите страницу в браузере, то ещё не увидите компонент *Login*, потому что нам нужно сослаться на него. Разместите его под `<hello></hello>` и вы увидите прекрасную форму входа!

src/App.vue

```
<template>
  <div id="app">
    <img src="./assets/logo.png">
    <hello></hello>
    <login></login>
  </div>
</template>
<!-- ... -->
```

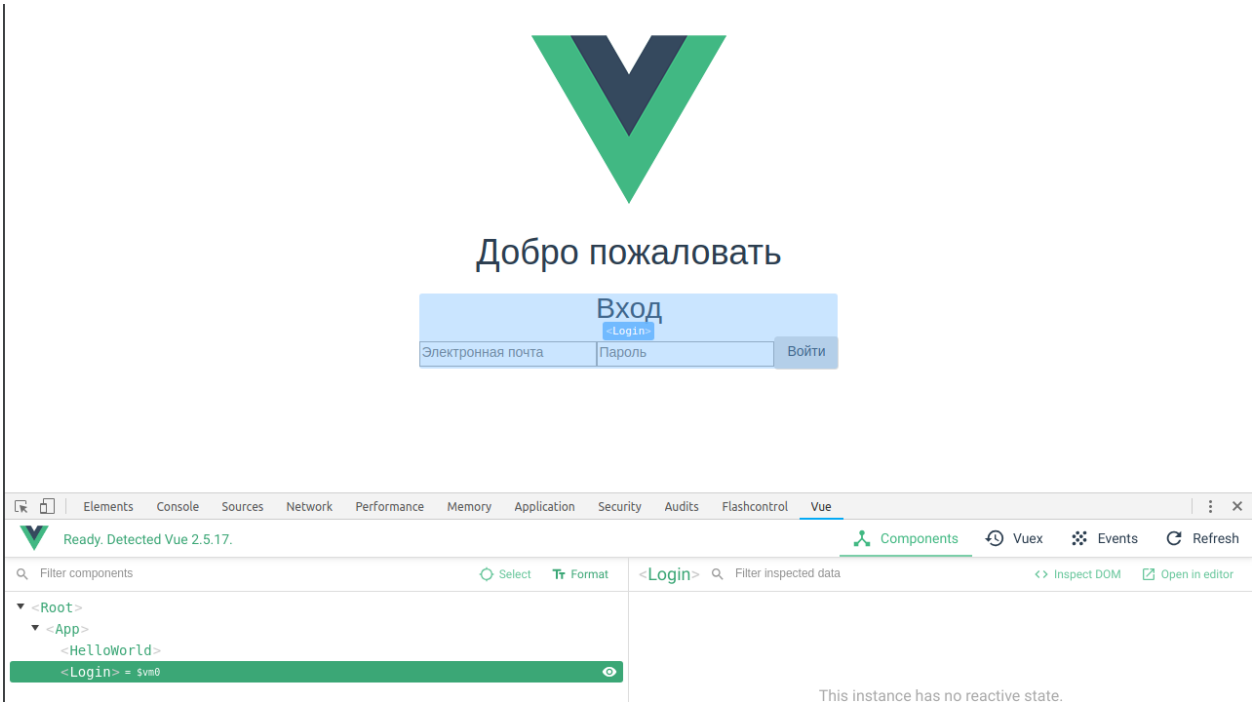
Добро пожаловать

Вход

Электронная почта | Пароль | Войти

Компонент Login

Если открыть консоль браузера, то можно увидеть сообщение **логин**, которое мы залогировали при создании компонента. Если открыть `vue-devtools`, которые мы рекомендуем, вы также увидите его в дереве компонентов.



Представление дерева

Давайте создадим другой компонент, на этот раз для регистрации.

src/components/Register.vue

```
<template>
<div id="register">
  <h2>Форма регистрации</h2>
  <input placeholder="Имя" class="form-control">
  <input placeholder="Фамилия" class="form-control">
  <input placeholder="Электронная почта" class="form-control">
  <input placeholder="Пароль" class="form-control">
  <input placeholder="Подтверждение пароля" class="form-control">
  <button type="submit" class="btn">Зарегистрироваться</button>
</div>
</template>

<script>
export default {
  created () {
    console.log('регистрация')
  }
}
</script>
```

Теперь можно импортировать его в файл *App.vue*.

src/App.vue

```
<template>
  <div id="app">
    <!-- ... -->
    <!-- <hello></hello> -->
    <!-- <login></login> -->
    <register></register>
    ...
  </div>
</template>

<script>
// import Hello from './components/Hello'
// import Login from './components/Login'
import Register from './components/Register'

export default {
  name: 'app',
  components: {
    // Hello,
    // Login,
    Register,
  }
}
</script>
<!-- ... -->
```

Шаблон компонента `Register` появится, когда мы проверим браузер.

Форма регистрации

Имя

Фамилия

Электронная почта

Пароль

Подтверждение пароля

Зарегистрироваться

Компонент Register

Заметка

Другие компоненты закомментированы, потому что мы не хотим отображать их друг под другом. Компонент `Hello` здесь по умолчанию, но мы не собираемся использовать его в наших дальнейших примерах, так что удалим его.

Мы говорили, что работаем над социальной сетью (или чем-то подобным), то есть хотим иметь место для отображения историй. Создадим тогда компонент **Stories**, который при отображении будет показывать все рассказанные пользователями истории.

src/components/Stories.vue

```
<template>
  <ul class="list-group">
    <li v-for="(story, index) in stories" :key="index" class="list-group-item">
      {{ story.writer }} сказал "{{ story.plot }}"
      Голосов за историю: {{ story.upvotes }}.
    </li>
  </ul>
</template>

<script>
export default {
  data () {
    return {
      stories: [
        {
          plot: 'Моя лошадь потрясающая.',
          writer: 'Мистер Уибл',
          upvotes: 28,
          voted: false
        },
        {
          plot: 'Нарвалы изобрели шашлык.',
          writer: 'Мистер Уибл',
          upvotes: 8,
          voted: false
        },
        {
          plot: 'Тёмная сторона Силы сильнее',
          writer: 'Дарт Вейдер',
          upvotes: 52,
          voted: false
        },
        {
          plot: 'Просто так в Мордор не попадёшь',
          writer: 'Боромир',
          upvotes: 74,
          voted: false
        }
```

```
      ]
    }
  }
}
</script>
```

Это файл *Stories.vue*. Мы можем использовать его в нашем основном *App.vue*. В данный момент истории жёстко заданы для простоты. Настало время импортировать его, как и другие компоненты.

src/App.vue

```
<template>
  <div id="app">
    <img class="logo" src="./assets/logo.png">
    <!-- <login></login> -->
    <!-- <register></register> -->
    <stories></stories>
  </div>
</template>

<script>
// import Login from './components/Login.vue'
// import Register from './components/Register.vue'
import Stories from './components/Stories.vue'
export default {
  components: {
    Login,
    Register,
    Stories
  }
}
</script>

<style>
/* ... */
</style>
```

Мистер Уибл сказал "Моя лошадь потрясающая." Голосов за историю: 28.

Мистер Уибл сказал "Нарвалы изобрели шашлык." Голосов за историю: 8.

Дарт Вейдер сказал "Тёмная сторона Силы сильнее." Голосов за историю: 52.

Боромир сказал "Просто так в Мордор не попадёшь" Голосов за историю: 74.

Компонент Stories

Великолепно! Теперь у нас есть страница для отображения всех историй.

16.2.1 Вложенные компоненты

Мы хотели бы иметь возможность отображать самые “популярные” истории в любом месте. То есть после создания компонента *Famous* мы сможем использовать его везде.

src/components/Famous.vue

```
<template>
  <div id="famous">
    <h2>Популярные истории <strong>({{famous.length}})</strong></h2>
    <ul class="list-group">
      <li v-for="(story, index) in famous" :key="index" class="list-group-item">
        {{ story.writer }} сказал "{{ story.plot }}".
        Голосов за историю: {{ story.upvotes }}.
      </li>
    </ul>
  </div>
</template>

<script>
```

```
export default {
  computed: {
    famous () {
      return this.stories.filter(function (item) {
        return item.upvotes > 50
      })
    }
  },
  data () {
    return {
      stories: [
        {
          plot: 'Моя лошадь потрясающая.',
          writer: 'Мистер Уибл',
          upvotes: 28,
          voted: false
        },
        {
          plot: 'Нарвалы изобрели шашлык.',
          writer: 'Мистер Уибл',
          upvotes: 8,
          voted: false
        },
        {
          plot: 'Тёмная сторона Силы сильнее',
          writer: 'Дарт Вейдер',
          upvotes: 52,
          voted: false
        },
        {
          plot: 'Просто так в Мордор не попадёшь',
          writer: 'Боромир',
          upvotes: 74,
          voted: false
        }
      ]
    }
  }
}
</script>
```

Это весь файл *Famous.vue*. У нас есть **отфильтрованный** с помощью вычисляемого свойства

массив `stories`, как мы говорили в предыдущих главах, и **`template`** для его отображения.

Заметка

Массив **`stories`** здесь снова жёстко задан и данные такие же, как и раньше. Это плохая практика, мы найдём позже способ, чтобы определять массив **`stories`** однажды и затем использовать его в разных компонентах.

Но где мы могли бы использовать этот компонент? Идея в том, чтобы добавить его на страницу регистрации, чтобы пользователь мог читать самые популярные истории и был заинтригован. В текущем проекте это означает, что необходимо добавить компонент `Famous` внутрь компонента `Register`. Хорошо, это может быть сделано точно так же, как мы делали внутри `App.vue`.

Поэтому откроем **`Register.vue`**, импортируем `Famous` сюда и добавим ссылку на него в шаблон.

src/components/Register.vue

```
<template>
  <div id="register">
    <h2>Форма регистрации</h2>
    <!-- ... -->
    <famous></famous>
  </div>
  </template>

  <script>
  import Famous from './Famous.vue'

  export default {
    components: {
      Famous
    },
    created () {
      console.log('регистрация')
    }
  }
  </script>
```

Форма регистрации

Имя

Фамилия

Электронная почта

Пароль

Подтверждение пароля

Зарегистрироваться

Популярные истории (2)

Дарт Вейдер сказал "Тёмная сторона Силы сильнее.". Голосов за историю: 52.

Боромир сказал "Просто так в Мордор не попадёшь". Голосов за историю: 74.

Страница регистрации с лучшими историями

Обратите внимание на путь импорта файла. Сейчас, поскольку два файла находятся в одной и той же папке, вы можете использовать `./Famous` вместо полного пути. Такую ошибку очень легко допустить, особенно если вы не знакомы с этим!

Примеры кода

Вы можете найти примеры кода этой главы на GitHub[16].

[16] https://github.com/hootlex/the-majesty-of-vuejs-2/tree/ru/codes/chapter16/16.3

17. Устранение повторяющегося состояния

В предыдущих примерах мы жёстко устанавливали данные — массив историй — внутри каждого компонента. **Это неправильный способ работы с данными**.

Когда более чем один компонент использует одни и те же данные, хорошей практикой является один раз создавать или получать массив, а затем искать варианты того, чтобы он был доступен компонентам приложения.

`Stories.vue` и *`Famous.vue`* используют одинаковый массив `stories`. Мы разберём 2 способа поделиться данными:

1. Используя входные параметры компонента.
2. Используя глобальное хранилище.

17.1 Совместное использование через входные параметры

Первое, что мы собираемся сделать, — перенести массив `stories` в компонент `App`.

src/App.vue

```
<script>
// ...

export default {
  components: {
      // ...
  },
  data () {
    // здесь мы располагаем массив историй
    return {
      stories: [
        {
          plot: 'Моя лошадь потрясающая',
          writer: 'Мистер Уибл',
          upvotes: 28,
```

```
16          voted: false,
17        },
18        {
19          plot: 'Нарвалы изобрели шашлык.',
20          writer: 'Мистер Уибл',
21          upvotes: 8,
22          voted: false,
23        },
24        {
25          plot: 'Тёмная сторона Силы сильнее.',
26          writer: 'Дарт Вейдер',
27          upvotes: 49,
28          voted: false,
29        },
30        {
31          plot: 'Просто так в Мордор не попадёшь',
32          writer: 'Боромир',
33          upvotes: 74,
34          voted: false,
35        },
36      ]
37    }
38  }
39 }
40 </script>
```

Следующий шаг — удалить `data()` из компонентов `Stories` и `Famous` и определить входной параметр `stories`.

Давайте сделаем это для первого компонента.

src/components/Stories.vue

```
1 <script>
2   export default {
3     props: ['stories']
4   }
5 </script>
```

Теперь нужно обновить шаблон компонента *App.vue*.

src/App.vue

```
<template>
  <div id="app">
    <!-- ... -->
    <stories :stories="stories"></stories>
    <!-- ... -->
  </div>
</template>
```

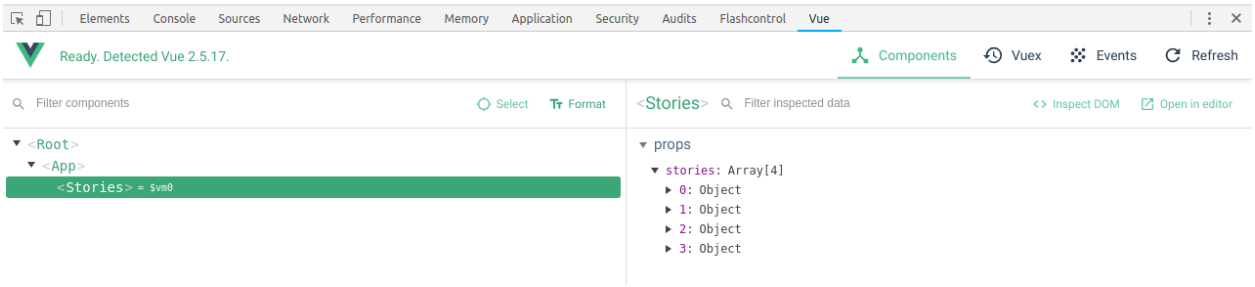
Здесь мы связываем свойство `stories` с массивом `stories`.

Мистер Уибл сказал "Моя лошадь потрясающая." Голосов за историю: 28.

Мистер Уибл сказал "Нарвалы изобрели шашлык." Голосов за историю: 8.

Дарт Вейдер сказал "Тёмная сторона Силы сильнее." Голосов за историю: 52.

Боромир сказал "Просто так в Мордор не попадёшь" Голосов за историю: 74.



Тот же вывод, используя входные параметры

Успех, мы снова получили наши `stories`, извлечённые из родительского компонента!

Мы не можем сделать то же самое с компонентом `Famous`, потому что он не используется внутри *App.vue*. Будем передавать наш массив в компонент `Register`, чтобы отправить его в `Famous`.

src/App.vue

```
<template>
  <div id="app">
    <!-- ... -->
    <register :stories="stories"></register>
    <!-- ... -->
  </div>
</template>
```

src/components/Register.vue

```
<template>
  <h2>Форма регистрации</h2>
  <!-- ... -->
  <famous :stories="stories"></famous>
</template>

<script>
import Famous from './Famous'

export default {
  components: {
    Famous
  },
  props: ['stories']
}
</script>
```

src/components/Famous.vue

```
<script>
export default {
  props: ['stories'],

  computed: {
    famous () {
      return this.stories.filter(function (item) {
        return item.upvotes > 50
      })
    }
  }
}
</script>
```

Такая реализация работает, но она не эффективна, потому что **делает зависимым** компонент `Famous`. Это означает, что мы не сможем использовать его везде, где захотим, до тем пор, пока передаём данные от корневого компонента *App.vue*.

В случае, когда зависимый компонент глубоко вложен, вам придётся передавать бесполезное свойство от компонента к компоненту, просто чтобы использовать его. В нашем случае, если мы хотим использовать компонент `Famous` внутри виджетов боковой панели `Register`, придётся иметь дело с массивом `stories` всё время.

App -> Register -> Sidebar -> WidgetX -> Famous

17.2 Глобальное хранилище

«Способ с входными параметрами» выглядит прекрасным на первый взгляд, но как продемонстрировано на примере компонента `Famous`, проект становится больше и компоненты вкладываются друг в друга, и управление данными и совместное использование ими становится действительно трудным для отслеживания.

Таким образом, давайте немного упростим управление данными в наших примерах. Мы можем извлечь массив `stories` в *.js* файл, сохраняя его **в одном месте**, а далее импортируя при необходимости.

Я назову наш *js*-файл *store.js* и добавлю в папку */src*.

src/store.js

```
1  export const store = {
2    stories: [
3      {
4        plot: 'Моя лошадь потрясающая',
5        writer: 'Мистер Уибл',
6        upvotes: 28,
7        voted: false,
8      },
9      {
10       plot: 'Нарвалы изобрели шашлык.',
11       writer: 'Мистер Уибл',
12       upvotes: 8,
13       voted: false,
14     },
15     {
16       plot: 'Тёмная сторона Силы сильнее.',
17       writer: 'Дарт Вейдер',
18       upvotes: 49,
19       voted: false,
```

```
20      },
21      {
22        plot: 'Просто так в Мордор не попадёшь',
23        writer: 'Боромир',
24        upvotes: 74,
25        voted: false,
26      },
27    ]
28  }
```

Предупреждение

Входной параметр `stories` должен быть удалён из всех файлов, потому что мы изменили способ хранения данных и могут быть конфликты, способные прервать сборку.

После того, как мы сохранили все данные в *`store.js`*, можно импортировать их в *`Stories.vue`*, используя синтаксис ES6-модулей.

src/components/Stories.vue

```
1  <script>
2  import {store} from '../store.js'
3
4  export default {
5    data () {
6      return {
7        // даст нам доступ к store.stories
8        store
9      }
10   },
11   created () {
12     console.log('истории')
13   }
14 }
15 </script>
```

Раз мы импортируем объект `store`, следует изменить также и шаблон компонента.

src/components/Stories.vue

```
<template>
  <ul class="list-group">
    <li v-for="(story, index) in store.stories" :key="index" class="list-group-item">
      {{ story.writer }} сказал "{{ story.plot }}"
      Кол-во голосов: {{ story.upvotes }}.
    </li>
  </ul>
</template>
```

Мы используем **`v-for`** для отрисовки элементов массива (`store.stories`). Наш список историй отображается как раньше.

Мы могли бы сделать то же самое без изменения шаблона с помощью прямого связывания атрибута `stories` компонента с `store.stories`.

src/components/Stories.vue

```
<script>
data () {
  return {
    // Привязка напрямую со stories
    stories: store.stories,
  }
}
</script>
```

То же применяем и к *`Famous.vue`*.

src/components/Famous.vue

```
<script>
import {store} from '../store.js'

export default {
  data () {
    return {
      stories: store.stories
    }
  },
  computed: {
    famous () {
      return this.stories.filter(function (item) {
        return item.upvotes > 50
```

```
14        })
15      }
16    }
17  }
18  </script>
```

Если бы мы не сделали связывание, следовало бы обновить вычисляемое свойство `famous()` для фильтрации `this.store.stories`.

Как только вы привыкните к работе с глобальными объектами, я верю, вы **полюбите их! :)**

Примеры кода

Вы можете найти примеры кода этого раздела на GitHub[1].

[1] https://github.com/hootlex/the-majesty-of-vuejs-2/tree/ru/codes/chapter17

18. Замена компонентов

Использование однофайловых компонентов — это самый простой способ создания одностраничных приложений с использованием Vue. Мы уже видели, как настроить новый проект, создавать файлы *.vue* и управлять дублирующимся состоянием. Теперь пришло время рассмотреть способ замены компонентов в зависимости от представления.

Для справки, в предыдущих примерах, у нас было 3 компонента внутри *App.vue*, а некоторые другие были вложены внутри. Нам нужно найти способ динамической замены компонентов так, что они не будут отрисовываться на странице одновременно.

18.1 Динамические компоненты

18.1.1 Специальный атрибут `is`

Мы можем использовать зарезервированный элемент **`<component>`** и одинаковую точку монтирования для динамического переключения между несколькими компонентами, используя специальный атрибут **`is`**.

src/App.vue

```
<template>
  <div id="app">
    <component is="hello"></component>
    <p>Что-то очень полезное...</p>
  </div>
</template>

<script>
import Hello from './components/Hello'
// Компонент Hello возвращает шаблон, содержащий свойство данных «msg»
export default {
  components: {
    Hello
  }
}
</script>
```

Мы создали новый проект и изменили файл *Hello.vue*. У нас такой результат, что и раньше, но здесь мы используем элемент **`<component is="hello">`**. Компонент `Hello` привязан

к атрибуту **is**. Чтобы посмотреть, как это будет работать динамически, посмотрите на следующий пример, в котором мы переключаем отображение двух разных компонентов по нажатию по ссылкам.

Во-первых, создайте аналогичный компонент с другим сообщением с именем *Greet.vue*.

src/Greet.vue

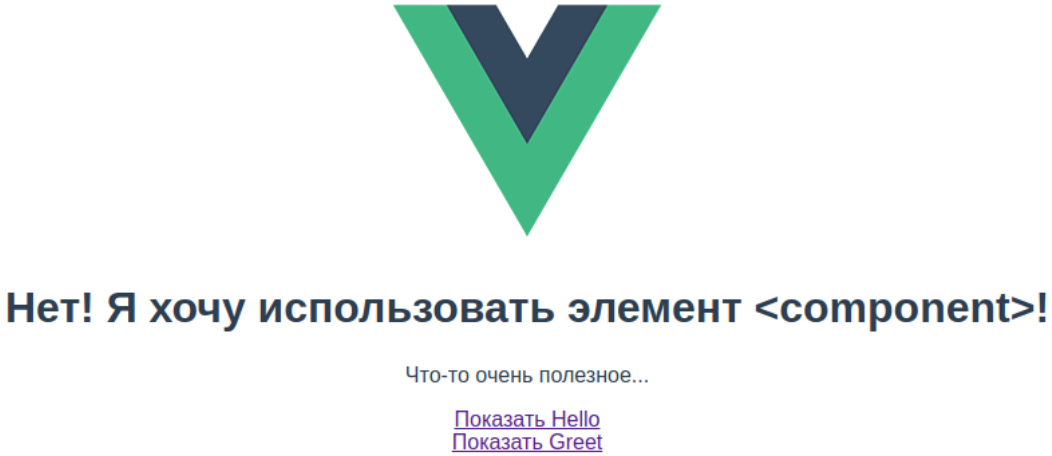
```
<template>
  <div class="greet">
    <h1>{{ msg }}</h1>
  </div>
</template>

<script>
export default {
  data () {
    return {
      msg: 'Нет! Я хочу использовать элемент <component>!'
    }
  }
}
</script>
```

Мы создали компонент `Greet` для отображения сообщения, чтобы продемонстрировать его наличие! Давайте импортируем его в `App` и дадим пользователю возможность поменять эти два компонента местами.

src/App.vue

```
<template>
  <div id="app">
    <img class="logo" src="./assets/logo.png">
    <component :is="currentComponent">
        <!-- отображение компонента поменяется,
             когда изменится this.currentComponent! -->
    </component>
    <p>Что-то очень полезное...</p>
    <a href="#" @click="currentComponent = 'hello'">Показать Hello</a>
    <br>
    <a href="#" @click="currentComponent = 'greet'">Показать Greet</a>
  </div>
</template>

<script>
```

```
import Hello from './components/Hello'
import Greet from './components/Greet'

export default {
  components: {
    Hello,
    Greet
  },
  data () {
    return {
      currentComponent: 'hello'
    }
  }
}
</script>
```



Greet.vue

Итак, как вы видите, мы привязываем специальный атрибут **`is`** к `currentComponent`, поэтому при его изменении, отображаемый компонент также изменится. Чтобы поменять представление, пользователю просто нужно нажать по любой ссылке для изменения значения `currentComponent`.

Этот динамический способ переключения между несколькими компонентами может оказаться удобным.

18.1.2 Навигация

В предыдущих примерах мы использовали файлы с расширением *.vue* для имитации социальной сети, где у нас были такие компоненты, как `Login`, `Registration` и т.д. Теперь мы можем использовать систему табов для навигации между этими компонентами.

Мы поместим *`Stories.vue`* на одной вкладке, *`Register.vue`* на другой, а *`Login.vue`* на третьей. Не забывайте, что `Register` содержит компонент `Famous`, возвращающий самые известные истории.

Тщательно изучите следующий пример.

src/App.vue

```
1  <template>
2    <div id="app">
3      <img class="logo" src="./assets/logo.png">
4      <h1>Добро пожаловать в динамические компоненты!</h1>
5      <ul class="nav nav-tabs">
6        <!-- установка класса 'active' в зависимости от условия -->
7        <li
8          v-for="(page, index) in pages"
9          :key="index"
10         :class="isActivePage(page) ? 'active' : ''"
11       >
12         <!-- используем ссылки для изменения табов -->
13         <a @click="setPage(page)">{{page | capitalize}}</a>
14       </li>
15     </ul>
16     <component :is="activePage"></component>
17   </div>
18 </template>
19
20 <script>
21 import Vue from 'vue'
22 import Login from './components/Login.vue'
23 import Register from './components/Register.vue'
24 import Stories from './components/Stories.vue'
25
26 Vue.filter('capitalize', function (value) {
27   return value.charAt(0).toUpperCase() + value.substr(1)
28 })
29
30 export default {
31   components: {
```

```
32      Login,
33      Register,
34      Stories
35    },
36    data () {
37      return {
38        // страницы, которые мы хотим отображать каждый раз
39        pages: [
40          'stories',
41          'register',
42          'login'
43        ],
44        activePage: 'stories'
45      }
46    },
47    methods: {
48      setPage (newPage) {
49        this.activePage = newPage
50      },
51      isActivePage (page) {
52        return this.activePage === page
53      }
54    }
55  }
56  </script>
```

Добро пожаловать в динамические компоненты!

Stories Register Login

Мистер Уибл сказал "Моя лошадь потрясающая." Голосов за историю: 28.

Мистер Уибл сказал "Нарвалы изобрели шашлык." Голосов за историю: 8.

Дарт Вейдер сказал "Тёмная сторона Силы сильнее." Голосов за историю: 52.

Боромир сказал "Просто так в Мордор не попадёшь" Голосов за историю: 74.

Страница для каждого компонента

Давайте во всём разберёмся.

Массив `pages` содержит компоненты, которые мы хотели бы отобразить. Мы используем директиву **`v-for`** для создания таба для каждого из них.

Для навигации между табами, мы создали метод `setPage`.

Свойство `activePage` изначально установлено в **`'stories'`**. При нажатии на вкладку значение `activePage` изменится на имя компонента, который мы хотим показать.

Чтобы определить, какой таб должен быть активным, применяется тернарный оператор, который устанавливает класс `active`, сравнивая текущее значение свойства `activePage` с именем каждого текущего компонента.

Для того, чтобы сделать первую букву каждого таба заглавной, мы создали фильтр `capitalize`, используя **`Vue.filter()`**, который используется внутри текстовых вставках.

С помощью этих нескольких и простых строк кода мы создали простой блок переключения содержимого по табам, который изменяет отображение наших компонентов в зависимости от условия.

Примеры кода

Вы можете найти примеры кода для этой главы на GitHub[1].

[1] https://github.com/hootlex/the-majesty-of-vuejs-2/tree/ru/codes/chapter18

19. Vue Router

Маршрутизация, в общем, относится к определению того, как ваше приложение реагирует на запрос клиента. Запрос браузера не будет направлен к вашему приложению без наличия какой-либо формы маршрутизации. Маршрутизатор помогает веб-серверу получить подходящую и точную для пользователя информацию. Он похож на начальника железнодорожной станции, который информирует оператора поезда, когда менять маршруты.

Способ смены представлений мы изучим в процессе настраивания маршрутизацию. Официальный маршрутизатор для Vue.js называется **vue-router**. Он глубоко интегрирован с ядром Vue.js для создания одностраничных приложений на одном дыхании. Этот плагин относительно прост в понимании, установке и использовании.

Его основные возможности:

- Вложенные маршруты/представления
- Модульная конфигурация маршрутизатора
- Доступ к параметрам маршрута, запроса
- Анимация переходов представлений на основе Vue.js
- Удобный контроль навигации
- Автоматическая установка активного CSS-класса для ссылок
- Режимы работы истории HTML5 или в режиме хеша, с автопереключением в IE 9
- Восстановление позиции прокрутки при возвращении назад в режиме истории HTML5

Информация для дальнейшего изучения

Это были лишь некоторые из предоставляемых возможностей, ознакомиться с полным набором возможностей можно на GitHub[1], а также в официальной документации[2].

19.1 Установка

Существуют следующие способы установить плагин: CDN, NPM и Bower. Мы воспользуемся терминалом для установки через NPM.

[1] https://github.com/vuejs/vue-router
[2] https://router.vuejs.org/ru/

```
>_  npm install vue-router
```

Введите эту команду в ваш терминал, чтобы установить плагин в *node_modules* в папке вашего проекта. После этого откройте файл `main.js` и добавьте следующие строки.

src/main.js

```
import Vue from 'vue'
import VueRouter from 'vue-router'
Vue.use(VueRouter)
```

Вы можете установить плагин Vue.js, используя **`Vue.use()`**, как показано выше. Для большей информации о **`Vue.use`** обратитесь к руководству[3].

19.2 Использование

Первый шаг — создание экземпляра маршрутизатора, куда мы будет передавать дополнительные опции позже, но пока оставим его простым.

src/main.js

```
// ...
const router = new VueRouter({
  routes // короткая форма для routes: routes
})
```

Теперь мы должны определить несколько маршрутов. Каждый маршрут следует отображать компонент, то есть мы собираемся создать маршруты для файлов с расширением **`*.vue`**, которые мы использовали в наших примерах.

Основной метод определения отображения маршрутов для маршрутизатора — создание массива `routes`, куда мы можем передать объекты маршрутов.

[3] https://ru.vuejs.org/api/#Vue-use

src/main.js

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import Hello from '../src/components/Hello.vue'
import Login from '../src/components/Login.vue'

Vue.use(VueRouter)

const routes = [
  { path: '/', component: Hello },
  { path: '/login', component: Login }
]
```

Внутри массива определяем 2 маршрута.

1. Когда будет вызван `http://localhost:3000/` (вы можете использовать другой порт), будет отображён `Hello.vue` по умолчанию.
2. Когда будет вызван `http://localhost:3000/login`, будет отображён компонент `Login.vue`.

На заметку

Мы используем **`{ path: '/login', component: Login }`**, потому что компонент **`Login`** импортирован в коде выше. В противном случае вы можете использовать функцию **`require()`** следующим образом: **`{ path: '/login', component: require('Login') }`**

Следующий шаг — использовать маршрутизатор. Руководство утверждает, что для маршрутизатора необходим корневой компонент. В нашем случае мы используем компонент `App.vue` как корневой.

Давайте создадим и примонтируем корневой экземпляр. Помните, что мы должны добавить маршрутизатор с опцией `router`, чтобы всему приложению *стало известно* о нём.

src/main.js

```
// ...
/* eslint-disable no-new */
new Vue({
  el: '#app',
  router,
  template: '<app></app>',
  components: {
    App
  }
})
```

Установим шаблон **<app></app>**, так что Vue заменит div с идентификатором app на шаблон компонента App.vue. Поскольку мы уже импортировали **App** в наш файл main.js, и всё настроено для работы.

Если вы будете пренебрегать добавлением комментария **/* eslint-disable no-new */**, то получите сообщение об ошибке от eslint:

https://eslint.org/docs/rules/no-new Do not use 'new' for side effects

Поэтому что мы должны сделать, чтобы этого избежать, так это выключить это правило.

Подсказка

Каждый раз, когда вы боретесь с правилом eslint, можно выключить его, добавив комментарий, как показано выше. Например, **/* eslint-disable eqeqeq */**. Полный список правил можно найти здесь[4].

Сейчас необходимо определить несколько ссылок для навигации. Откройте **App.vue**, чтобы сделать требуемые изменения.

src/App.vue

```
<template>
  <div id="app">
    <img class="logo" src="./assets/logo.png">
    <h1>Добро пожаловать в маршрутизацию!</h1>
    <router-link to="/">Главная</router-link>
    <router-link to="/login">Логин</router-link>
    <!-- вывод маршрута -->
    <router-view></router-view>
  </div>
</template>
```

<router-view></router-view> — место, где происходит вся магия, когда нам отображаются компоненты.

router-link — это компонент, позволяющий пользователям перемещаться по приложению с маршрутизатором. Свойство **to** определяет местоположение цели, соответствующей определённому в main.js маршруту. По умолчанию **router-link** будет отображаться как тег **<a>** с атрибутом **href**, в который устанавливается требуемый URI. Для более сложных перемещений может понадобиться больше аргументов, которые мы разберём позже.

Заметка

Если вы используете шаблон vue-cli с включённым маршрутизатором, то должны определять массив маршрутов внутри router/index.js.

[4] https://eslint.org/docs/rules/

19.3 Именованные маршруты

Пока опции, которые мы рассмотрели для маршрутизации, покрывают нужды для небольшого проекта, вроде нашего примера, вероятно, вам понадобится больше опций в процессе роста проекта. Например, если позже вы примете решение изменить URL с `/login` на `/signin`, то придётся обновлять все ссылки, ведущие на страницу входа. Чтобы предотвратить это, можно давать каждому маршруту имя.

Чтобы задать маршруту имя, мы должны переделать его конфигурацию.

src/main.js

```
// ...
const routes = [
  {
    path: '/',
    name: 'home',
    component: Hello
  },
  {
    path: '/login',
    name: 'login',
    component: Login
  }
]
```

Чтобы задать маршруту имя, добавьте свойство **`name`** и используйте его как идентификатор для ссылок в дальнейшем.

src/App.vue

```
<template>
  <div id="app">
    // ...
    <router-link :to="{ name: 'home'}">Главная</router-link>
    <router-link :to="{ name: 'login'}">Логин</router-link>
    <!-- вывод маршрута -->
    <router-view></router-view>
  </div>
</template>
```

Заметим, что вместо использования строки для определения назначения ссылки (**`to="/home"`**), мы используем объект (**`:to="{ name: 'home'}"`**). Мы подробно остановимся на этом позже.

19.4 Режим HTML5 History

Прежде чем двигаться дальше, я хотел бы обратить внимание на то, что касается URL браузера при использовании Vue Router. Как видите, когда изменяется маршрут, символ '**#**' добавляется к URL. Например, когда мы перемещаемся на страницу входа, URL становится **/#/login**.

Это вызвано тем, что по умолчанию Vue Router работает в режиме хеша, который соответственно использует хеш URL для имитации полного URL, чтобы страница не перезагружалась при изменении URL-адреса. Чтобы избавиться от хеша, можно заменить его на режим истории (**history mode**). Также можем установить другую опцию, **base**, которая определяет корневой путь для всех навигационных маршрутов. Её изменение на что-нибудь, что отличается от первоначального значения `default`, будет отображено в путях, которые всегда будут содержать новое значение в фактическом URL-адресе.

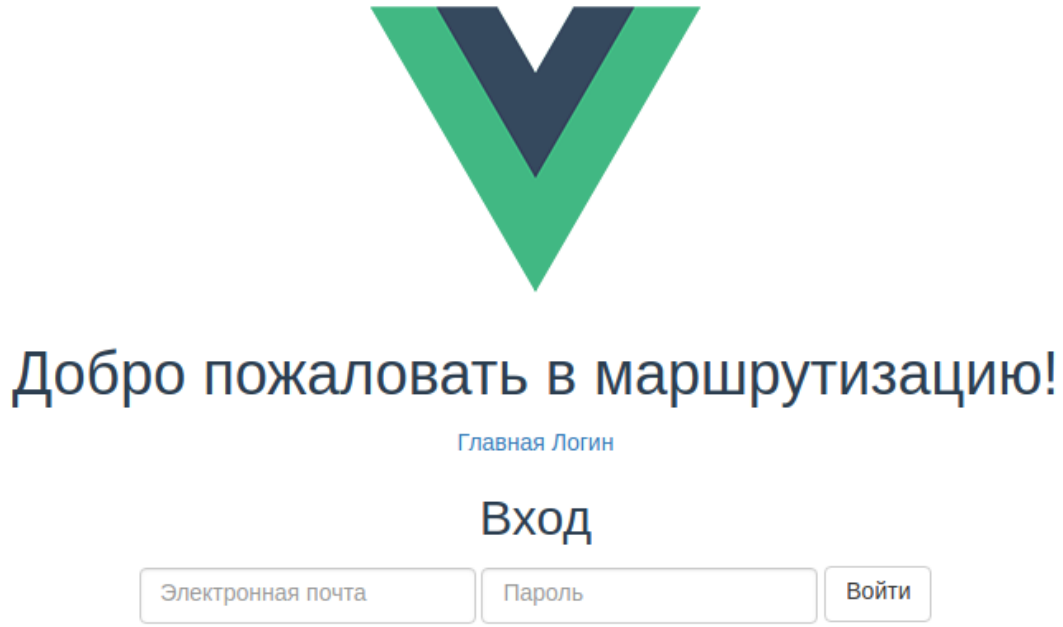
Например, если вы установите **/vuejs** в **base**, то страница входа будет доступна по адресу **/vuejs/login**.

Вы можете установить все что угодно в **base**, сейчас давайте просто вставим **/**.

src/main.js

```
const router = new VueRouter({
  mode: 'history',
  base: '/',
  routes
})
```

Это избавит нас от '**#**' в ссылках.



Аккуратные URL-адреса

Информация

Проверьте подробный список всех доступных опций Vue Router в официальной документации[5]

19.5 Вложенные маршруты

Вложенные маршруты — маршруты, которые находятся внутри других маршрутов. Связывание вложенных маршрутов с компонентами является обычной необходимостью, что очень просто сделать с Vue Router.

Чтобы продемонстрировать это, мы собираемся добавить страницу для отображения историй на двух страницах.

- Для отображения всех историй (`StoriesAll.vue`)
- Для отображения известных историй (`StoriesFamous.vue`)

Создадим вышеупомянутые компоненты и ещё один, который будет обёрткой, похожей на `App.vue`.

[5] https://router.vuejs.org/ru/api/

Начнём с регистрации новых маршрутов. Всё, что мы должны сделать, — использовать опцию `children` в массиве `routes` и добавить вложенный **`<router-view>`** внутрь нашего компонента-обёртки `StoriesPage.vue`.

src/main.js

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

import Hello from './components/Hello.vue'
import Login from './components/Login.vue'
import StoriesPage from './components/StoriesPage.vue'
import StoriesAll from './components/StoriesAll.vue'
import StoriesFamous from './components/StoriesFamous.vue'

import VueRouter from 'vue-router'

Vue.use(VueRouter)

const routes = [
  {
    path: '/',
    component: Hello
  },
  {
    path: '/login',
    component: Login
  },
  {
    path: '/stories',
    component: StoriesPage,
    children: [
      {
        path: '',
        name: 'stories.all',
        component: StoriesAll
      },
      {
        path: 'famous',
        name: 'stories.famous',
        component: StoriesFamous
      }
    ]
  }
```

```
]
```

Обратите внимание, что `path` для `StoriesAll` установлен в значении `''`. Это означает, что существует дочерний маршрут по умолчанию, который будет отображён, когда будет загружен маршрут `/stories`. Вы можете также использовать `'/'` для определения маршрута по умолчанию.

Содержимое `StoriesFamous` будет отображено при выборе маршрута `/stories/famous`.

На данные момент нет необходимости показывать, что происходит внутри этих компонентов. Они оба просто отображают массив историй.

Наш компонент-обёртка `StoriesPage` включает 2 ссылки и тег **`<router-view>`** для отображения содержимого дочерних компонентов.

src/StoriesPage.vue

```
<template>
  <div>
    <h2>Истории</h2>
    <!-- навигация -->
    <router-link :to="{name: 'stories.all'}">Все</router-link>
    <router-link :to="{name: 'stories.famous'}">Известные</router-link>
    <!-- вывод маршрута -->
    <router-view></router-view>
  </div>
</template>
```

19.6 Автоматическое добавление CSS-класса на активный маршрут

Было бы неплохо, если бы мы выделили ссылку, которая указывает на активную страницу. Маршрутизатор Vue.js достаточно умён, чтобы добавить CSS-класс `vue-router-active` к активной ссылке.

Всё, что вы должны сделать, — добавить правило для его оформления в CSS-стилях. Я добавлю его в компонент `App.vue`.

src/App.vue

```
// ...
<style type="text/css">
.router-link-active {
  color: green;
}
</style>
```

Так что теперь всякий раз при посещении страницы соответствующая ссылка зеленеет..

Если вы попробуете это в браузере, то обратите внимание, что ссылка на `Главная` всегда зелёная. Так случилось потому, что путь к `Главная` — `/`, поэтому когда вы посещаете, например, `/login`, `Главная` остаётся активным. Чтобы избавиться от этого поведения, можно добавить свойство `exact` к этой ссылке.

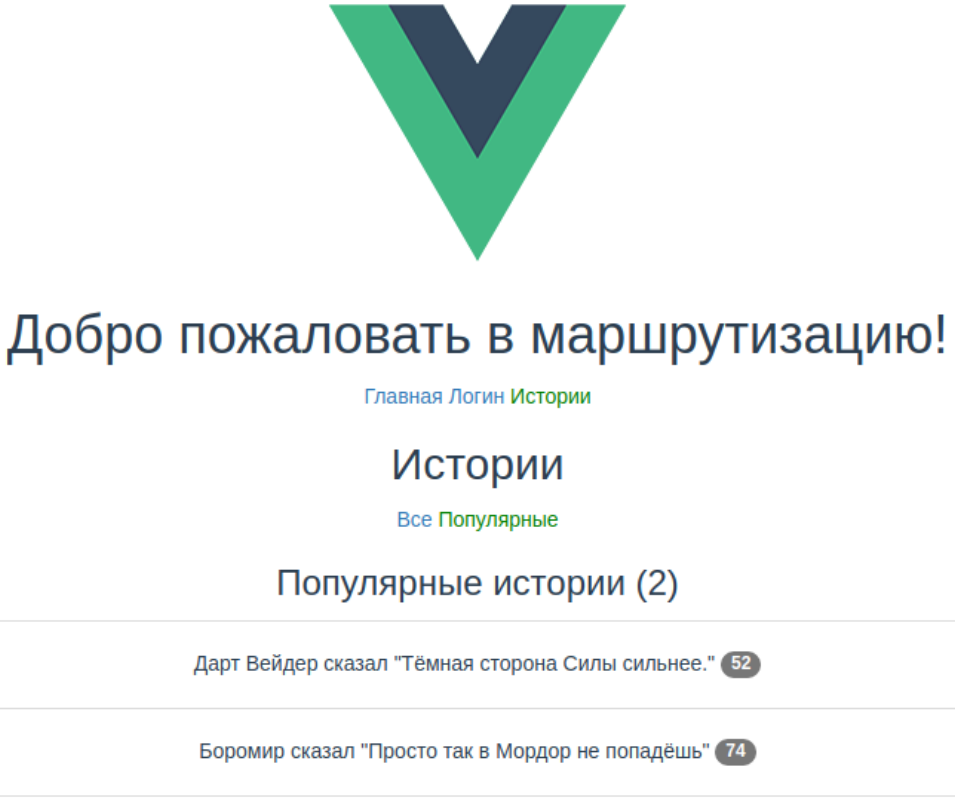
Наши ссылки навигации выглядят примерно так:

src/App.vue

```
<template>
  <div>
    <!-- ... -->
    <router-link :to="{ name: 'hello'}" exact>Главная</router-link>
    <router-link :to="{ name: 'login'}">Логин</router-link>
    <router-link :to="{ name: 'stories.all'}">Истории</router-link>
    <router-view></router-view>
  </div>
</template>
```

Мы также добавили `exact` к первой ссылке внутри `StoriesPage.vue`.

Приятный момент — активная ссылка выделяется также и в навигации второго уровня.



Активный класс

19.6.1 Пользовательский активный класс

Вы можете изменить имя активного класса (`router-link-active`) для **определённой ссылки**, используя свойство `active-class`, или **глобально** с помощью опции `linkActiveClass` в конструкторе маршрутизатора.

Пользовательский класс для активного маршрута, заданный конкретно для одного маршрута

```
<router-link :to="{ name: 'hello'}" active-class="my-active-class" exact>
  Главная
</router-link>
```

Пользовательский класс для активного маршрута, заданный для всех маршрутов

```
const router = new VueRouter({
  mode: 'history',
  base: '/',
  linkActiveClass: 'my-active-class',
  routes
})
```

19.7 Объект маршрута

Вся информация о текущем маршруте доступна в открытом **объекте контекста маршрута** или **объекте маршрута**.

Объект маршрута будет добавлен в каждый компонент в приложении с маршрутизатором и будет доступен как **`this.$route`**. Данные в нем обновляются после выполнения перехода по новому маршруту.

Ниже представлен список свойств объекта **`$route`**.

Свойство	Описание
path	Строка, идентичная пути текущего маршрута, всегда в абсолютном формате, например `/foo/bar`.
params	Объект, который содержит пары ключ/значение динамических сегментов маршрута (включая *-сегменты). Больше деталей ниже.
query	Объект, который содержит пары ключ/значение для строки запроса. Например, для пути `/foo?user=1`, мы получим `$route.query.user == 1`.
hash	Хеш текущего маршрута (вместе с символом **#**), если присутствует. Если хеша нет, то значением будет пустая строка.
fullPath	Полная запись URL-адреса, включая строку запроса и хэш.
matched	Массив, содержащий записи маршрутов для всех вложенных сегментов текущего маршрута. Записи маршрутов — это копии объектов из массива конфигурации routes (и вложенных массивов `children`).
name	Имя текущего маршрута, если такое указано.

19.8 Динамические сегменты

Маршрутизатор Vue.js предоставляет возможность формировать пути с помощью динамических сегментов. Динамические сегменты — это сегменты, в начале которых находится двоеточие. Динамическими они называются, потому что их значение можно изменить.

Важная информация

URL-сегменты — части URL-адреса или пути, разделённые слешами. Если вы получили путь **/user/:id/posts**, то user, :id или posts — сегмент.

В этом пути **:id** является динамическим сегментом и будет соответствовать любому предоставленному значению, например, **/user/11/posts**, **/user/37/posts** и т.д.

Когда путь, включающий динамический сегмент, найден, подобные сегменты доступны внутри **$route.params**.

В нашем примере мы можем использовать динамический сегмент для доступа к конкретной истории по её **id**, чтобы создать представление для возможности редактирования.

src/main.js

```
const routes = [
    // другие
    {
      path: ':id/edit',
      name: 'stories.edit',
      component: StoriesEdit
    }
    // ...
]
```

Сейчас необходим способ связать StoriesAll.vue с StoriesEdit.vue. Давайте изменим файл.

Заметка

Я создал файл StoriesEdit.vue в фоне. Вы увидите его позже, после окончания маршрутизации для него.

src/component/StoriesAll.vue

```
<template>
  <div>
    <h3>Все истории ({{stories.length}})</h3>
    <ul class="list-group">
      <li v-for="story in stories" :key="story.id" class="list-group-item">
        <div class="row">
          <h4>{{ story.writer }} сказал "{{ story.plot }}"
            <span class="badge">{{ story.upvotes }}</span>
          </h4>
          <router-link
```

```
11            :to="{ name: 'stories.edit'}" tag="button"
12            class="btn btn-default" exact
13          >
14            Редактировать
15          </router-link>
16        </div>
17      </li>
18    </ul>
19  </div>
20 </template>
21
22 <script>
23 import {store} from '../store.js'
24
25 export default {
26     data () {
27         return {
28         stories: store.stories
29         }
30     },
31     mounted () {
32         console.log('истории')
33     }
34 }
35 </script>
```

Заметка

Наши файлы примера почти такие же, как раньше, с небольшими изменениями, добавленными стилями, которые не влияют на их функциональность. Примечательное изменение — добавление **`id`** к каждой истории в `store.js`.

Мы добавили кнопку для связи с маршрутом `stories.edit`. Хорошо, но этого недостаточно, потому что нам также необходимо передавать **`id`** истории в маршрут.

Для этого необходимо отредактировать свойство **`to`** и сделать, чтобы **`:id`** превращался в соответствующий `id` каждой истории.

src/component/StoriesAll.vue

```
<template>
  <div>
    <h3>Все истории ({{stories.length}})</h3>
    <ul class="list-group">
      <li v-for="story in stories" :key="story.id" class="list-group-item">
        <h4>{{ story.writer }} сказал "{{ story.plot }}"
          <span class="badge">{{ story.upvotes }}</span>
        </h4>
        <router-link
          :to="{ name: 'stories.edit', params: { id: story.id }}"
          tag="button"
          class="btn btn-default"
          exact
        >
          Редактировать
        </router-link>
      </li>
    </ul>
  </div>
</template>
```

Здесь мы хотим, чтобы **`<router-link>`** отображался как тег **`<button>`** вместо **`<a>`**. Можем использовать свойство **`tag`** для определения тега, который будет обрабатывать события клика для навигации.

Когда мы достигнем нашей цели, в **`$route.params`** будет доступен **`id`** выбранной истории. С этой возможностью мы можем определить, какую историю пользователь хочет отредактировать, и предоставить её.

Настало время показать файл `StoriesEdit.vue`, где будет размещено редактирование.

src/components/StoriesEdit.vue

```
<template>
  <div class="row">
    <h3>Редактирование</h3>
    <form>
      <div class="form-group col-md-offset-2 col-md-8">
        <input class="form-control" v-model="story.plot">
      </div>
      <div class="form-group col-md-12">
        <button type="button" @click="saveChanges(story)" class="btn btn-success">
          Сохранить изменения
```

```
11          </button>
12        </div>
13      </form>
14    </div>
15  </template>
16
17  <script>
18  import {store} from '../store.js'
19
20  export default {
21    data () {
22      return {
23        story: {}
24      }
25    },
26    methods: {
27      isTheOne (story) {
28        return story.id === this.id
29      },
30      saveChanges (story) {
31        // мы будем использовать это позже
32      }
33    },
34    mounted () {
35      this.story = store.stories.find(this.isTheOne)
36    }
37  }
38  </script>
```

Мы используем идентификатор истории, чтобы выбрать нужную `story` из массива `stories` с помощью JavaScript-метода find[6].

Выбранная история готова к редактированию. Обратите внимание, что `id` истории показан в URL-адресе.

[6] https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array/find

Добро пожаловать в маршрутизацию!

Главная Логин Истории

Истории

Все Популярные

Редактирование

Тёмная сторона Силы сильнее.

Сохранить изменения

Редактирование выбранной истории

Это хорошо работает, если вы посетите страницу с подключённым компонентом `StoriesEdit`, нажав на кнопку `Редактировать`, но если попробовать ввести URL-адрес непосредственно в адресную строку браузера, например, *`/stories/2/edit`*, вернётся ошибка и компонент не будет отображён.

Причина этому заключается в том, то мы использовали строгое равенство[7] (`===`) для нахождения подходящей активной истории. При непосредственном переходе на страницу, **`id`** передаётся как строка (не число). Поэтому **`isTheOne`** всегда возвращает `false`.

src/components/StoriesEdit.vue

```
isTheOne (story) {
  // возвращает false, когда this.is не является числом
  return story.id === this.id
}
```

Чтобы исправить это, преобразуем **`id`** к числу с помощью объекта-обёртки **`Number`**.

[7] https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Identity

src/components/StoriesEdit.vue

```
export default {
  // ...
  mounted () {
    this.id = Number(this.$route.params.id)
  }
}
```

Если проверить это в браузере, вы увидите, что история появляется, как если бы мы кликали на кнопку редактирования.

Несмотря на это, существует решение получше, которое помогает нам разделить `StoriesEdit` и **`$route`**. Мы можем преобразовать параметр **`id`** к числу внутри конфигурации маршрута и использовать его как `prop` в компоненте `StoriesEdit`.

src/main.js

```
const routes = [
  // ...
  {
    path: ':id/edit',
    props: (route) => ({ id: Number(route.params.id) }),
    name: 'stories.edit',
    component: StoriesEdit
  },
  // ...
]
```

Теперь, когда путь `stories/:id/edit` работает, маршрутизатор будет передавать числовое значение **`:id`** как входной параметр в компонент **`StoriesEdit`**. Осталось определить это свойство и удалить **`this.$route.params.id`**. В итоге компонент будет выглядеть примерно так:

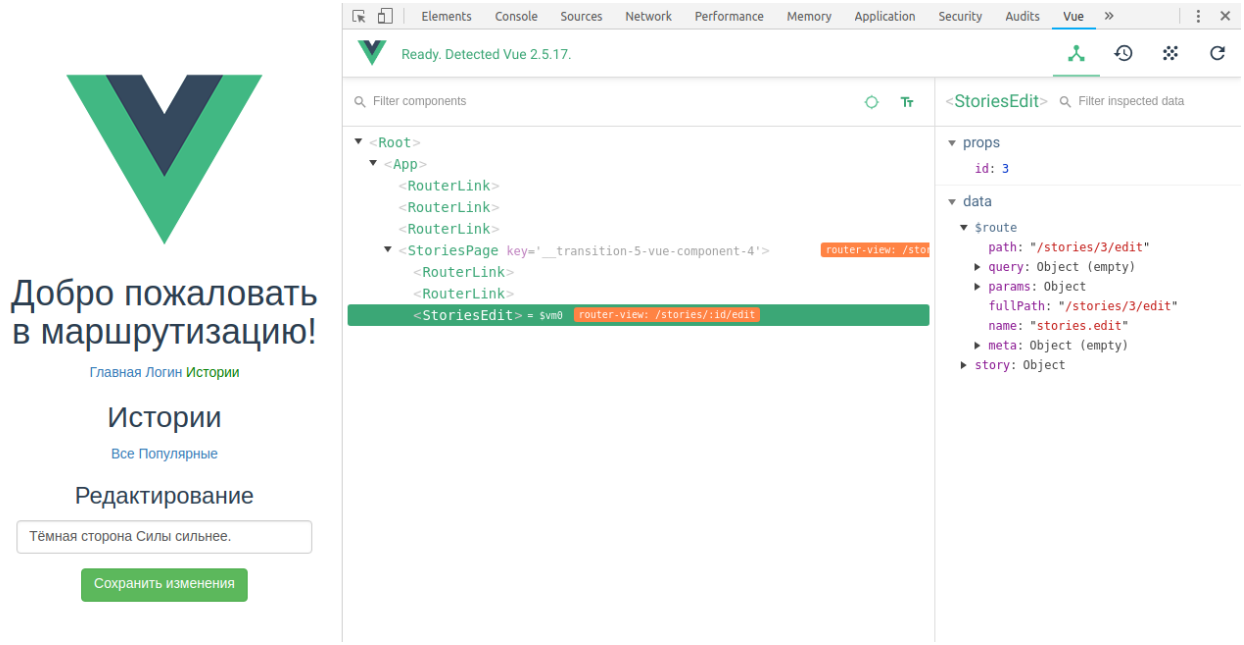
src/components/StoriesEdit.vue

```
<script>
import {store} from '../store.js'

export default {
  props: ['id'],
  data () {
    return {
      story: {}
    }
```

```
  },
  methods: {
    isTheOne (story) {
      return story.id === this.id
    }
  },
  mounted () {
    this.story = store.stories.find(this.isTheOne)
  }
}
</script>
```

Сейчас компонент `StoriesEdit` отделён от маршрута, и мы можем использовать его везде, например, так: **`<stories-edit :id="story.id"></stories-edit>`**.

Вы можете посмотреть атрибуты объекта **`$route`** с помощью Vue.js Devtools.



Внутри $route

19.9 Псевдонимы маршрутов

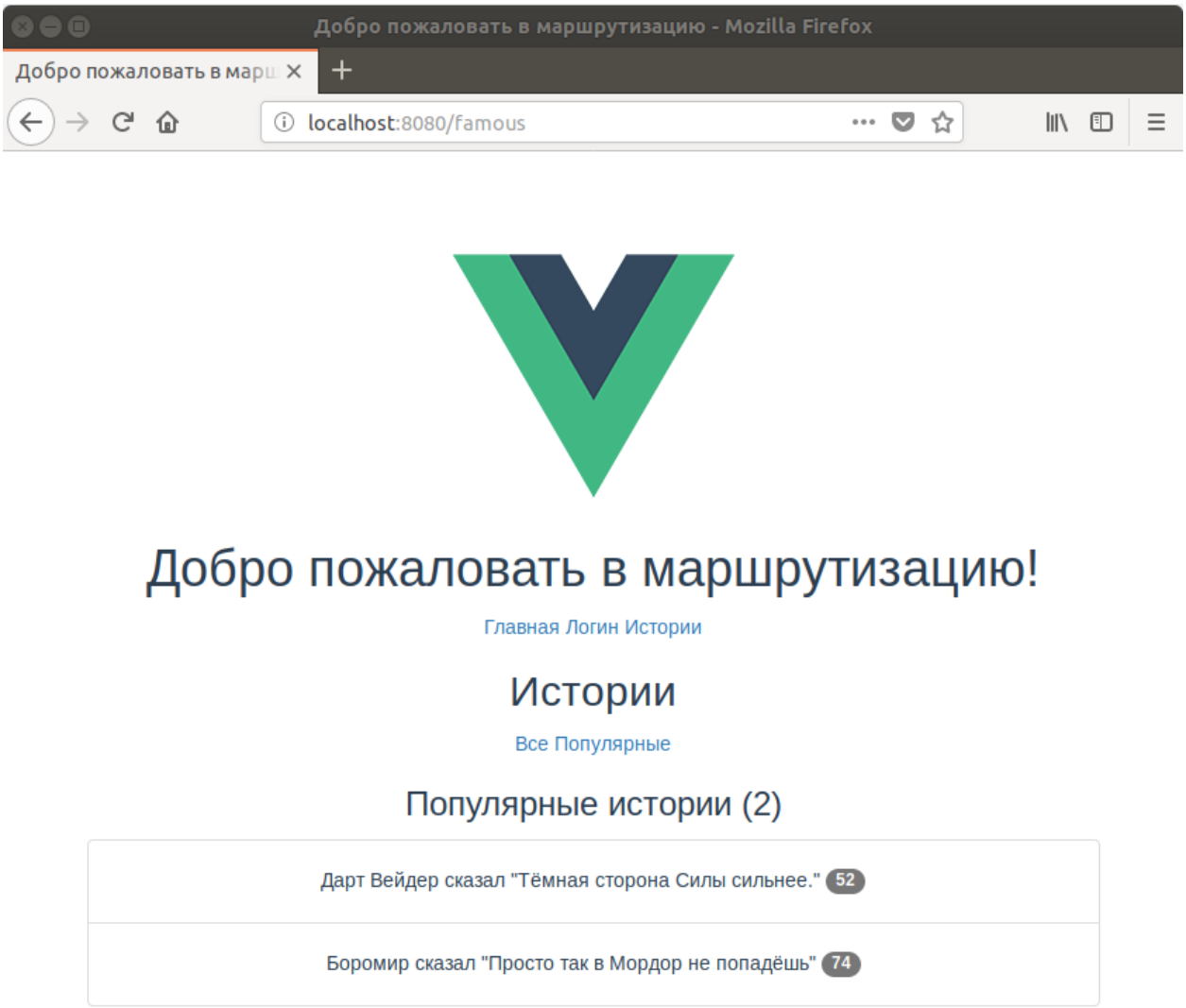
Когда мы определяем новые маршруты, то обычно стараемся сделать их понятными и образцовыми. Хотя иногда мы можем остаться с длинными и сложными путями, которыми может быть трудно управлять позднее.

Если мы хотим увидеть известные истории, то должны открыть `'/stories/famous'` через браузер. Можно сделать этот маршрут короче через определение глобального псевдонима, где более короткий URL-адрес приведёт нас к тому же месту.

src/main.js

```
{
  path: 'famous',
  name: 'stories.famous',
  // находит '/famous' как будто это '/stories/famous'
  alias: '/famous',
  component: StoriesFamous
}
```

Используя эту конфигурацию, мы можем просто использовать `alias` вместо `path`.



Использование псевдонима маршрута

19.10 Программная навигация

В то же время, мы хотим переходить по маршруту не по ссылкам, а программно.

Чтобы перейти по маршруту программным путём, можно использовать `router.push(path)`. Путь может быть строкой или объектом.

Если это строка, путь должен быть в форме обычного пути, т.е. он не может содержать динамических сегментов. Например, `router.push('/stories/11/edit')`.

Если это объект, то можно передавать любые необходимые аргументы.

Programmatic navigation

```
router.push({ path: '/stories/11/edit' })
router.push({ name: 'stories.edit', params: {id: '11'} })
```

Мы можем использовать **`router.push`** для перехода на страницу со списком историй после завершения редактирования.

src/components/StoriesEdit.vue

```
1  <script>
2  import {store} from '../store.js'
3
4  export default {
5    props: ['id'],
6    data () {
7      return {
8        story: {}
9      }
10   },
11   methods: {
12     saveChanges (story) {
13       console.log('Сохранено!')
14       this.$router.push('/stories')
15     },
16     isTheOne (story) {
17       return story.id === this.id
18     }
19   },
20   mounted () {
21     this.story = store.stories.find(this.isTheOne)
22   }
23 }
24 </script>
```

Мы обновили метод **`saveChanges`**. При вызове он регистрирует сообщение в консоли и с помощью **`this.$router.push()`** перемещается обратно к `/stories`.

Если вы хотите направить пользователя на URL-адрес, который был перед этим, вместо конкретного URL-адреса можно просто вызвать **`router.back()`**.

В нашем случае мы можем добавить кнопку и вызывать функцию **`router.back`** вместо **`router.push`**, тогда пользователь перейдёт на предыдущую страницу (какой бы она ни была, например, `https://google.com`).

src/components/StoriesEdit.vue

```
// ...
<button type="button" @click="goBack">Вернуться назад</button>

methods: {
  // ...
  goBack () {
    this.$router.back()
  },
  // ...
}
```

Другой способ сделать это — использовать метод **`router.go(n)`**, который получает в качестве параметра число для определения количества шагов, на которое нужно перейти вперёд или назад по истории навигации[8].

```
goBack () {
  this.$router.go(-1)
}
```

Предупреждение

При использовании **`$router.back()`** или любого другого метода маршрутизатора мы привязываем компонент к маршрутизатору. Если вы не хотите этого, то можете использовать **`window.history.back()`**.

19.11 Переходы

19.11.1 Введение

Каждый раз, когда мы перемещаемся на другую страницу нашего приложения, ничего особенного не происходит. Мы можем изменить это при помощи анимации появления нового компонента и исчезновения предыдущего.

Vue предоставляет различные способы для применения эффектов переходов, когда элементы добавляются, обновляются или удаляются из DOM. Они включают инструменты для:

- автоматического применения классов для переходов и анимация на CSS

[8] https://router.vuejs.org/ru/guide/essentials/history-mode.html#%D1%80%D0%B5%D0%B6%D0%B8%D0%BC-html5-history

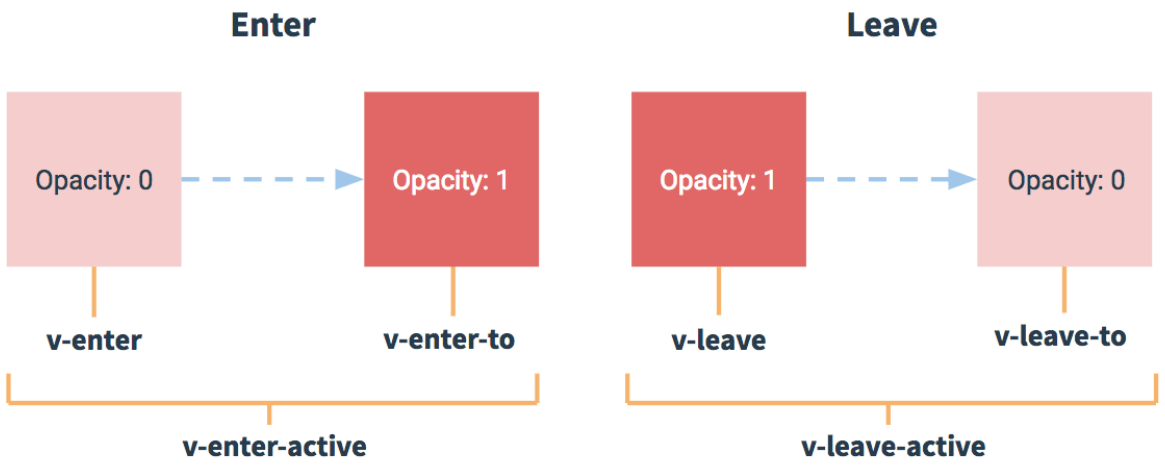
- интеграции CSS-кода или JavaScript-библиотек для анимаций от третьих лиц, таких как Animate.css[9], Velocity.js[10] и т.д.
- использования JavaScript, чтобы напрямую управлять DOM через хуки переходов

В то же время мы рассмотрим только появление и исчезновение с помощью CSS-классов. Если вы заинтересовались в более подробном изучении переходов, проверьте руководство[11].

Чтобы использовать анимацию, следует обернуть соответствующий элемент в компонент `transition`. В нашем случае компонент `router-view`.

Vue добавит CSS-класс `v-enter` к элементу перед тем, как он будет добавлен, и `v-enter-active` в конце. `v-enter-to` — последний класс, который будет добавлен перед завершением анимации. Это конечное состояние для появления.

Соответственно когда элемент удаляется из DOM, будут применены `v-leave`, `v-leave-active` и `v-leave-to`.



Классы перехода

Если указано имя для `transition`, все упомянутые выше классы будут включать это имя вместо 'v'. Например, `fade-enter`, `fade-leave-to` и т.д.

19.11.2 Использование

Давайте используем переход с именем `fade` для нашего маршрута.

[9] https://daneden.github.io/animate.css/
[10] http://velocityjs.org/
[11] https://ru.vuejs.org/v2/guide/transitions.html

src/App.vue

```
<template>
  <div>
  <!-- ... -->
  <transition name="fade">
    <router-view></router-view>
  </transition>
  </div>
</template>

<style type="text/css">
.fade-enter{
  opacity: 0
}
.fade-enter-active {
  transition: opacity 1s
}
.fade-enter-to {
  opacity: 0.8
}
</style>
```

Посмотрите на CSS-классы. Переход начнётся с прозрачностью 0, которая будет постепенно увеличиваться в течение 1 секунды. `.fade-enter-to` не обязателен, его добавление, как в данном случае, создаст прямое исчезновение в период с 0.8 до 1, прямо перед завершением перехода.

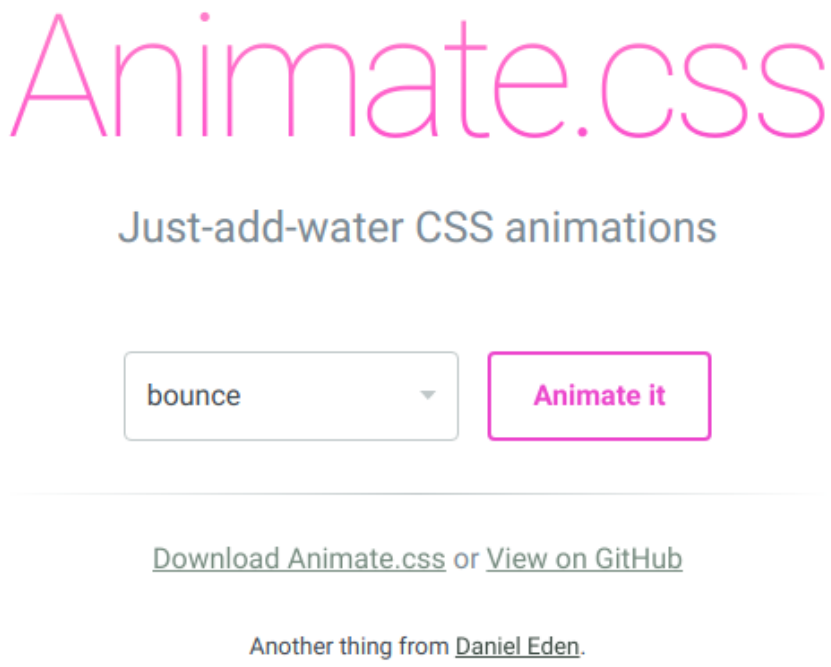
Для создания обратной анимации, когда компонент исчезает, мы должны переделать наши CSS-стили примерно так:

```
<style type="text/css">
.fade-enter, .fade-leave-to{
  opacity: 0
}
.fade-enter-active, .fade-leave-active {
  transition: opacity 1s
}
.fade-enter-to, .fade-leave {
  opacity: 0.8
}
</style>
```

19.11.3 Сторонние средства для CSS-анимаций

Создание анимаций с нуля и проектирование в целом является для меня тяжёлой задачей. Я всегда предпочитаю полагаться на сторонние библиотеки. К счастью для меня (и вас, может быть), Vue-переходы очень легко связывать со сторонними библиотеками для анимаций через CSS и JS.

В этом примере я буду использовать Animate.css[12].



Animate.css

В соответствии с документацией для использования `Animate.css` мы должны:

1. Включить стили в тег `<head>` документа.
2. Добавить класс `animated` в элемент, для которого вы хотите добавить анимацию. Вы также можете включить класс `infinite` для бесконечного цикла.

[12] https://daneden.github.io/animate.css/

3. В итоге добавить один из доступных классов, например, `bounce`, `rollIn`, `fadeIn` и т.д. Вы можете найти список всех доступных классов здесь[13].

Давайте начнём с импорта с CDNJS в заголовок нашего файла `index.html`.

Вместо того, чтобы полагаться на свойство `name` компонента `transition`, в этот раз мы будем использовать пользовательский класс для `v-enter-active`.

src/App.vue

```
<template>
  <div>
  <!-- ... -->
  <transition enter-active-class="animated rollIn">
    <router-view></router-view>
  </transition>
  </div>
</template>
```

Мы можем также сделать анимацию, когда компонент удаляется из DOM, добавив пользовательский `leave-active-class` следующим образом: **`leave-active-class="animated rollOut"`**.

19.12 Навигационные хуки

Маршрутизатор Vue предоставляет удобный механизм для фильтрации переходов. Для фильтрации перехода вы можете использовать **`router.beforeEach()`**, который срабатывает перед каждый переходом, и **`router.afterEach()`**, который срабатывает после, однако **`afterEach()`** не может повлиять на навигацию.

`router.beforeEach()` может быть удобным в сценарии организации авторизации. Например, если пользователь не имеет доступа к странице вашего приложения, вам следует перенаправить его на страницу входа.

Давайте посмотрим на небольшом примере, как мы можем этого достичь.

[13] https://github.com/daneden/animate.css

src/main.js

```
// создать объект-заглушку пользователя
let User = {
  isAdmin: false
}

router.beforeEach((to, from, next) => {
  if (to.path !== '/login' && !User.isAdmin) {
    // если не страница входа и не администратор, перенаправлять на страницу входа
    next('/login')
  } else {
    // если авторизирован, обработать
    next()
  }
})
```

Здесь мы применили правило, при котором маршрутизатор даст пользователям перейти только на страницу `login`. Будьте уверены, что всегда вызываете функцию **`next()`**, иначе хук никогда не будет разрешён.

Примеры коды

Вы можете найти примеры кода этой главы на GitHub[14].

[14] https://github.com/hootlex/the-majesty-of-vuejs-2/tree/ru/codes/chapter19

19.13 Домашнее задание

В этой главе мы проанализировали много чего, и прошло много времени, с тех пор как мы давали вам домашнее задание!

Приложение, которое мы предлагаем вам написать, — миниатюрный Покедекс (Pokédex).

Домашняя страница будет показывать список категорий покемонов (Pokémon), например *Fire, Water и т.д.*. Отсюда пользователь сможет перейти в категорию, посмотреть его покемонов и добавить новых.

Ваши маршруты могут выглядеть примерно так:

Маршрут	Описание
/	Список категорий.
/category/:name	Показать категорию покемонов.
/category/:name/pokemons/new	Добавить нового покемона в категорию.

Каждый переход должен быть зарегистрирован в консоли. Например, когда пользователь решает перейти в категорию *Fire*, должно быть зарегистрировано сообщение, информирующее пользователя, то он собирается посетить `/category/Fire`.

Мы создали объект с покедексом, чтобы помочь вам начать. Вы можете найти его здесь[15].

На заметку

Маршрут **`/category/:name/pokemons/new`** является дочерним для **`/category/:name`**.

При посещении **`/category/:name/pokemons/new`** пользователь должен увидеть форму для добавления нового покемона вместе со списком покемонов этой категории.

Первая подсказка

Чтобы иметь доступ к покемону отдельной категории, изучите использование JavaScript-метода find[16].

Вторая подсказка

Для регистрации сообщений в консоли перед каждый переходом используйте **`router.beforeEach()`**.

[15] https://github.com/hootlex/the-majesty-of-vuejs-2/blob/ru/homework/Chapter19/pokedex.js
[16] https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array/find

Добро пожаловать в Паллет-Таун!

Категория покемонов "Водный"

Сквиртл
Level: 03

Бластойз
Level: 45

Старью
Level: 09

Гярадос
Level: 26

Тентакруэль
Level: 29

Обратно к категориям | Добавить покемона

Пример вывода

Добро пожаловать в Паллет-Таун!

Категория покемонов "Электрический"

Пикачу
Level: 74

Электабазз
Level: 34

Запдос
Level: 88

Обратно к категориям | Добавить покемона

Добавить нового покемона!

Магнетон

Отмена | Сохранить

Пример вывода

Вы можете найти потенциальное решение этого упражнения здесь[17]

[17] https://github.com/hootlex/the-majesty-of-vuejs-2/tree/ru/homework/Chapter19

20. Заключение

Кажется, наше путешествие подходит к концу. Надеемся, что вам понравилось читать эту книгу так же, как нам нравилось писать её. Мы также надеемся, что вам понравилось (хотя бы немного) наше чувство юмора.

Хотим поблагодарить каждого из вас за прочтение всей книги и, конечно же, за поддержку наших усилий сделать эту книгу реальностью. Огромное спасибо всем участникам[1], кто уделил немного своего времени, чтобы сообщить о некоторых грамматических, синтаксических ошибках в тексте либо коде.

В случае, если вы не знали, мы запустили Vue.js Feed[2], где вы можете найти **ежедневно** публикуемые новости, учебные материалы, код и плагины, связанные со Vue.js. Мы будем рады, если вы захотите стать частью этого, потому что любим вас всех. Всех и каждого :)

Не оставайтесь в стороне!

Если вы хотите спросить о чём-то или просто сказать «привет», не стесняйтесь связываться с нами через Twitter[3]. Мы также будем информировать вас об обновлениях книги.

[1] https://github.com/hootlex/the-majesty-of-vuejs-2/graphs/contributors
[2] https://vuejsfeed.com/
[3] https://twitter.com/tmvuejs

21. Дальнейшее обучение

Мы собрали в одном месте некоторые учебные материалы и книги, которые вам будет полезно прочесть. Также собрали несколько проектов с открытым исходным кодом, чтобы вы могли посмотреть Vue в действии. Некоторые из ресурсов не связаны непосредственно со Vue.js, но они относятся к смежным темам.

Если вы хотели бы изучить более продвинутые темы по Vue, а также как создавать реальные приложения, то вам, возможно, следует посетить Vue School[1] от Алекса Кириакидиса.

21.1 Руководства

- Введение в Vue.js (серия)[2] — это серия статей охватывает множество всего, что связано со Vue, например, отрисовку, директивы, события, компоненты, Vuex и анимаций.
- Аутентификация приложения на Vue JS с Firebase UI[3] — Настройка Firebase UI для обычного веб-приложения невероятно проста. Это также просто и во Vue.js, но существует несколько особенностей для понимания.
- Введение во Vue и Weex для нативных мобильных приложений[4] — Это руководство показывает, как использовать фреймворк, как понять его концепции, начиная от привязки данных и шаблонов. Затем рассказывается о Weex-платформе для создания нативных мобильных приложений с помощью Vue!
- Как построить реактивный движок в JavaScript.[5] — Это руководство о геттерах/сеттерах для наблюдения и реагирования на изменения в режиме реального времени.
- Простое руководство по созданию компонентов Vue.js с открытым исходным кодом[6] — Здесь собраны некоторые советы для тех, кто уже имеет либо думает о создании своих собственных решений для Vue.js с открытым исходным кодом.

21.2 Видеоуроки

- Объяснение основ производительности фронтенда[7] — В этом обсуждении Эван Ю рассказывает о техниках, используемых в основных фронтенд-фреймворках: dirty checking, сравнении с виртуальным DOM и отслеживании зависимостей.

[1] https://vueschool.io
[2] https://css-tricks.com/intro-to-vue-1-rendering-directives-events/
[3] https://medium.com/dailyjs/authenticating-a-vue-js-application-with-firebase-ui-8870a3a5cff8
[4] https://code.tutsplus.com/tutorials/introducing-vue-and-weex-for-native-mobile-apps--cms-28782
[5] https://monterail.com/blog/2016/how-to-build-a-reactive-engine-in-javascript-part-1-observable-objects/
[6] https://monterail.com/blog/2016/simple-guide-to-authoring-open-source-vue-js-components/
[7] https://vuejsfeed.com/blog/demystifying-frontend-framework-performance-video

- Реактивность в JavaScript-фреймворках[8] — Как фреймворки определяют изменения в состоянии и как они эффективно распространяют изменения по системе? Эван отвечает на эти вопросы, основываясь на своём опыте создания Vue.js.
- Изучение Vue 2: шаг за шагом (серия)[9] — Серия уроков по изучению Vue.

21.3 Книги

- Понимание ECMAScript 6[10] — В ES6 существует много новых концепций для изучения и понимания. Начните изучение с этой книги!
- Создание API, который вы не будете ненавидеть[11] — Сейчас стало довольно стандартно создавать приложение с отделённой логикой фронтенда и бекенда. Фронтенд сделан на JavaScript в браузере с помощью потрясающих фреймворков вроде Vue.js, а бэкенд обычно построен на каком-нибудь серверном языке, предоставляя JSON.
- SVG-анимации[12] — SVG является чрезвычайно мощным из-за уменьшения HTTP-запросов и чёткостью на любом экране. В этой книге вы изучите всё о SVG, включая как сделать SVG кроссбраузерным, обратно совместимым, оптимизированным и адаптивным.

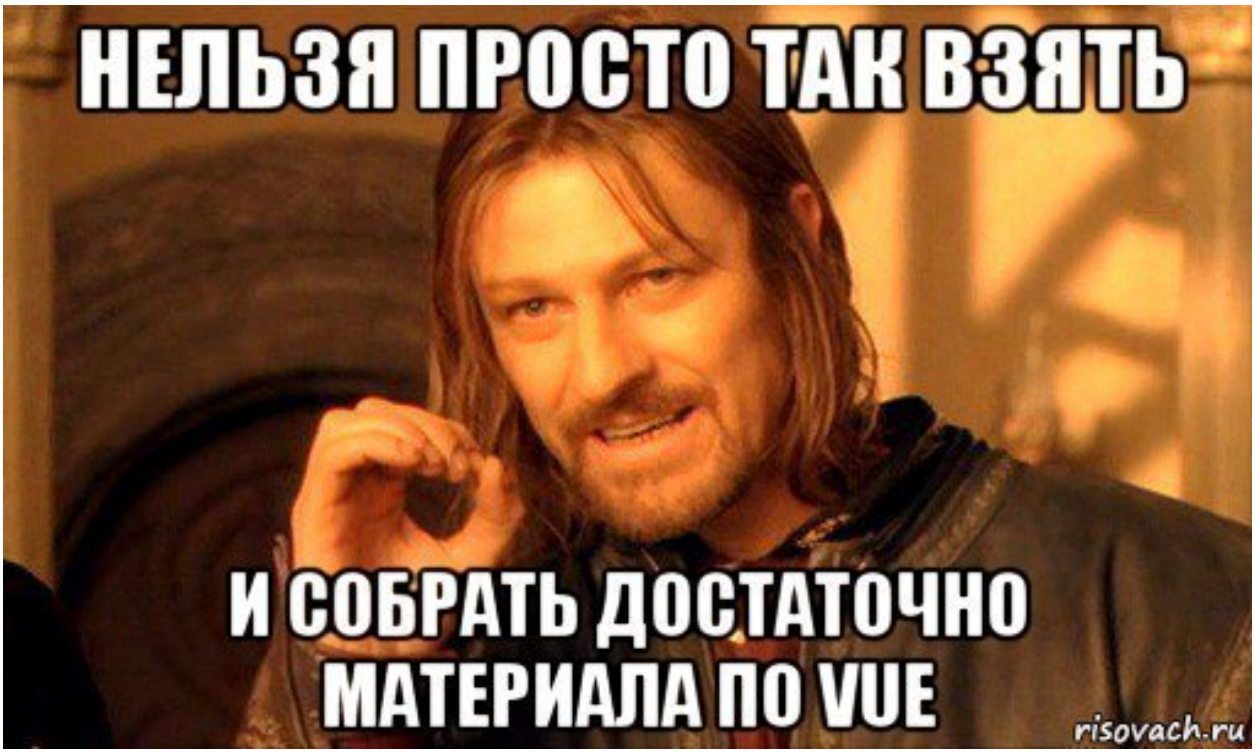
21.4 Проекты с открытым исходным кодом

- Vuedo[13] — Блог-платформа, созданная на Laravel и Vue.js.
- Airflix[14] — Дружественный к AirPlay веб-интерфейс для потоковой передачи ваших видео и телевизионных передач с домашнего сервера.
- Koel[15] — Личный музыкальный сервер для потоковой передачи, который работает.
- Mini e-shop[16] — Небольшой онлайн-магазин, созданный с Vue.js. У него достаточно возможностей для начала создания вашего!
- The Movie Database App[17] — Похожее на IMDb приложение.
- Vue WordPress PWA[18] — SPA с подходом «Offline first» с использованием Vue.js, WordPress REST API и прогрессивных веб-приложений (Progressive Web Apps).
- Hypersurface[19] — Hypersurface — место для людей, где они могут задавать вопросы, отвечать, общаться, делать заметки по вопросам, затрагивающим личные мнения. Данные по этим вопросам отображаются в режиме реального времени, позволяя любому повлиять и узнать текущую точку зрения.

[8] https://vuejsfeed.com/blog/watch-evan-you-reactivity-in-frontend-javascript-frameworks-in-dotjs-conference
[9] https://laracasts.com/series/learn-vue-2-step-by-step
[10] https://leanpub.com/understandinges6
[11] https://leanpub.com/build-apis-you-wont-hate
[12] https://shop.oreilly.com/product/0636920045335.do
[13] https://github.com/Vuedo/vuedo
[14] https://github.com/wells/airflix
[15] https://github.com/phanan/koel
[16] https://github.com/BosNaufal/vue-mini-shop
[17] https://github.com/dmtrbrl/tmdb-app
[18] https://github.com/bstavroulakis/vue-wordpress-pwa
[19] https://github.com/aswdesign/hypersurface

21.5 Awesome Vue

Awesome Vue.js[20] — это специально отобранный список всего того удивительного, что связано со Vue.js. Вы можете найти здесь кучу материалов по Vue. Сейчас команда Vue работает над проектом Curated Vue[21], который включает только подтверждённые пакеты и предоставляет возможность искать по конкретным ресурсам.



Конец…

[20] https://github.com/vuejs/awesome-vue
[21] https://curated.vuejs.org/

Полезные русскоязычные ресурсы по Vue.js

Общее

- Русская версия официального сайта с документацией и не только[22] (если вам нравится русская документация, буду рад, если поставите лайк-звёздочку репозиторию на GitHub[23])
- Русскоговорящее сообщество Vue.js в Telegram[24]

Статьи

- Переводы статей по Vue.js на русский язык от команды devSchacht[25] (рекомендую подписаться на их Medium-канал!)
- Статьи по Vue.js на «Хабре»[26]
- Переводы статей по Vue.js на WebForMySelf[27]

Лучшие практики

- Паттерны Vue[28]
- Руководство по написанию компонентов во Vue.js[29]

[22] https://ru.vuejs.org/
[23] https://github.com/translation-gang/ru.vuejs.org
[24] https://t.me/vuejs_ru
[25] https://medium.com/devschacht/vue/home
[26] https://habr.com/hub/vuejs/
[27] https://webformyself.com/category/javascriptajax-2/vuejs/
[28] https://learn-vuejs.github.io/vue-patterns/ru/
[29] https://github.com/pablohpsilva/vuejs-component-style-guide/blob/master/README-RU.md

Видео

Митапы

- Moscow Vue.js Meetup #1[30]
- Moscow Vue.js Meetup #2[31]

Доклады

- Андрей Грачёв “Vue.js или как наконец отказаться от React”[32]

Аудио

- Подкаст «Weekly Vue»[33]

[30] https://www.youtube.com/watch?v=h9NQs0SEVoA
[31] https://www.youtube.com/watch?v=SiPKxngecQ0
[32] https://www.youtube.com/watch?v=x1ySbc-ocRg
[33] https://soundcloud.com/dmitry-zlygin/