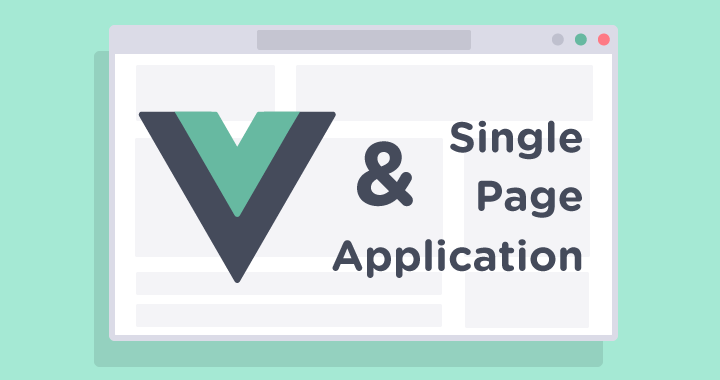
Как написать одностраничное приложение (SPA) с использованием Vue.js

2018.12.19

* [Разработка веб-сайтов](https://habr.com/ru/hub/webdev/),
* [JavaScript](https://habr.com/ru/hub/javascript/),
* [Программирование](https://habr.com/ru/hub/programming/),
* [VueJS](https://habr.com/ru/hub/vuejs/)
* Tutorial
* [Recovery Mode](https://habr.com/ru/info/help/karma/#recovery)

  
  
Одностраничные приложения (SPA) имеют мнжество преимуществ, таких как скорость, по-настоящему хороший UX, и полный контроль HTML-разметки. Становится всё больше и больше сайтов SPA; всё больше инструментов, которые упрощают процесс разработки SPA. Вы, вероятно уже читали о молодом и перспективном фреймворке **Vue.js**. Предлагаю вам глубже погрузиться в Vue и на конкретном примере разобраться с простым SPA.  
  
Мы напишем клиент-серверное приложение простейшего блога. Приложение будет отображать список записей а также полный текст каждой отдельной записи. И само собой, всё это будет происходить без перезагрузки страницы.  
  
Ознакомившись с примером этого приложения, вы научитесь извлекать данные в Vue, создавать роуты и разберётесь с интересной особенностью Vue — однофайловыми компонентами.

Бэкенд

В этом руководстве мы в основном сосредоточимся на фронтенде на **Vue**. Размышлять о написании REST бэкенда мы не будем. Для примера будет использоваться сервис *jsonplaceholder.typicode.com*предостовляющий заглушку в виде REST API.

Фронтенд

Инструменты

Начать работу с Vue очень просто. С использованием правильных инструментов это ещё проще. Рекомендую взглянуть на проект [vue-awesome](https://github.com/vuejs/awesome-vue), который содержит список инструментов, компонентов, библиотек и плагинов на все случаи жизни.

Vue-cli

При создании нового проекта рекомендуется воспользоваться Vue-cli. Так можно создавать проекты с использованием официальных шаблонных проектов Vue, или одного из множества шаблонных проектов с открытым исходным кодом, и, конечно же, вы можете создать свой собственный и использовать его в любом месте.  
  
Итак, для начала установим vue-cli в качестве глобального пакета:

$ npm install -g vue-cli

Затем проинициализируем проект с выбранным шаблоном; для нашего примера более чем достаточно использовать webpack-simple.

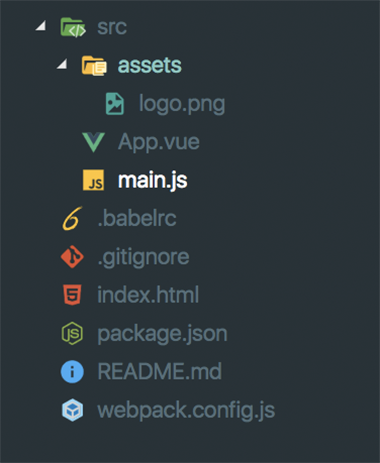
$ vue init webpack-simple vue-spa

Далее перейдём в папку vue-spa и запустим *npm install* в терминале. После установки всех пакетов, мы можем запустить наше приложение в режиме разработки.

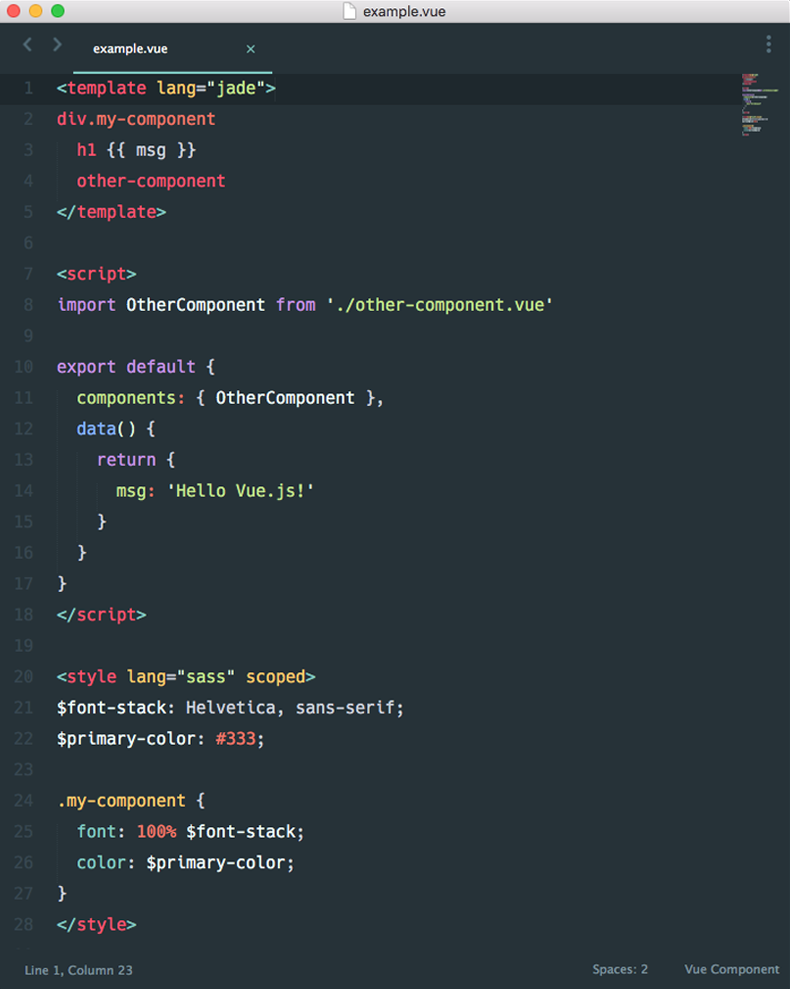
$ npm run dev

Эта команда автоматически запустит наш проект на локальном dev-сервере webpack. В браузере появится наше простейшее приложение Vue. Конечно оно выглядит совсем не так, как бы нам хотелось, и годится лишь в качестве отправной точки для начала чего-то большего. Чтобы продолжить работу, предлагаю сначала ознакомиться со структурой нашего шаблона.

Шаблон webpack-simple

Внутри шаблон webpack-simple имеет следующую структуру:  
  
  
  
Файл *index.html* содержит простую разметку HTML с единственным элементом “app” в body. Он будет заменён на DOM, сгенерированный vue. По этой причине тэг *body* не рекомендуется использовать в качестве корневого элемента.  
  
В папке *src* лежит файл main.js, который содержит точку входа webpack. Компоненты Vue импортируются там же. Ещё там описан корневой экземпляр Vue, который пока что имеет два свойства. Свойство ‘el’ обеспечивает экземпляру Vue связь с указанным DOM элементом. Ещё одно — функция отрисовки, генерирующая DOM из *App.vue*. В общем, это все, что нам нужно знать о структуре шаблона webpack-simple, не так много, не так ли? Основная часть нашего приложения будет запрограммирована в App.vue. Расширение .vue определяет файл как однофайловый компонент vue. Это одна из особенностей Vue с которой мы сейчас познакомимся поближе.

Однофайловые компоненты

  
  
Каждый файл \*.vue состоит из блоков трёх типов: <template>, <script> и опционально <style>. В результате, мы можем разделить проект на связанные компоненты. Внутри компонента его шаблон, логика и стили неотъемлемо связаны, и их совмещение фактически делает компонент более целостным и легко поддерживаемым. Теперь мы готовы приступить к созданию блога на Vue.

Пишем приложение

Давайте посмотрим, что собственно мы собираемся реализовать. У нас будет заголовок с названием нашего блога в верхней части страницы. С левой стороны у нас будет фиксированная боковая панель, в которой мы будем отображать заголовки наших записей, это будет что-то вроде оглавления. Остальная часть страницы будет занята динамическим блоком, в котором будет отображаться сам текст записи.  
  


Шаг 1

Прежде всего, удалим все лишние строки из App.vue. И перепишем шаблон в соответствии с нашими требованиями.

<template>

<div id="app">

<header>

<h1>Vue.js SPA</h1>

</header>

<main>

<aside class="sidebar">

</aside>

<div class="content">

</div>

</main>

</div>

</template>

Во-вторых, создадим экземпляр Vue со свойством data, которое мы разместим в массиве с нашими сообщениями. На данный момент он пуст, но вскоре мы поместим в него данные, полученные с сервера внутрь массива.  
  
После первого обращения, вы больше не сможете добавлять реактивные свойства к корневому объекту данных. Поэтому, прежде чем создавать экземпляр Vue, рекомендуется объявить все реактивные свойства на уровне корня.

<script>

export default {

data () {

return {

posts: []

}

}

}

</script>

Кроме того, можно добавить немного стилей, чтобы приложение выглядело лучше.  
Код приложения хостится на [github.com](https://github.com/DChuchin/vue-spa-tutorial/). Достаточно клонировать репозиторий и переключать ветку по номеру шага чтобы проследить разработку приложения шаг за шагом например:

$ git checkout step-1

В настоящий момент нам абсолютно нечего отображать в нашей навигационной панели, поэтому давайте получим данные с сервера. Для этого я выбрал Axios — простой в использовании HTTP-клиент. Вы также можете использовать любой удобный для вас способ, например, Vue-ресурс или собственную выборку или даже jQuery Ajax.

Шаг 2

Установим Axios

$ npm install --save-dev axios

Затем импортируем его в компонент App и определим метод getAllPosts() который будет делать запрос к серверу и присваивать его свойству posts. Вызываем метод в хуке created(), который будет вызываться после создания экземпляра Vue и после установки настроек обращения к данным.

import axios from 'axios'

export default {

data () {

return {

posts: null,

endpoint: 'https://jsonplaceholder.typicode.com/posts/',

}

},

created() {

this.getAllPosts();

},

methods: {

getAllPosts() {

axios.get(this.endpoint)

.then(response => {

this.posts = response.data;

})

.catch(error => {

console.log('-----error-------');

console.log(error);

})

}

}

}

А теперь отобразим все заголовки записей в боковой панели.

<aside class="sidebar">

<div v-for="post in posts">

{{ post.title }}

</div>

</aside>

До сих пор мы отображали только заголовки записей, но пока мы ещё не можем видеть сами записи. Теперь необходимо отобразить полный пост в разделе контента в соответствии с выбранным названием на боковой панели. В то же время хотелось бы, чтобы каждая запись была доступна по своему уникальному адресу.

Шаг 3

Для этого воспользуемся официальной Vue библиотекой vue-router. Как уже понятно из названия, библиотека позволяет настраивать роутинг для нашего приложения.  
Установим библиотеку:

$ npm install --save-dev vue-router

Для настройки роутинга вернёмся к файлу main.js. Здесь мы определим настройки роутинга и добавим их в наш экземпляр Vue.

import Vue from 'vue'

import Router from 'vue-router'

import App from './App.vue'

import Post from './components/Post.vue'

import Hello from './components/Hello.vue'

Vue.use(Router)

const router = new Router({

routes: [

{

path: '/',

name:'home',

component: Hello,

},

{

path: '/post/:id',

name:'post',

component: Post,

props: true,

},

]

})

new Vue({

el: '#app',

render: h => h(App),

router

})

В настройках роутинга мы указали, какой компонент вызывает отрисовку по соответствующему пути. Поскольку только компонент Post.vue будет отвечать за отрисовку каждого поста, нам не потребуется определять путь к каждому посту, достаточно определить динамический путь.

path: '/post/:id'

Этот путь содержит динамический сегмент :id который указывает на конкретный пост. При этом у нас есть доступ к этому сегменту в компоненте Post через *this.$route.params.id*. Однако использование $route в нашем компоненте закрепит жесткую связь с роутом, что в свою очередь ограничивает гибкость компонента, поскольку он может использоваться только на определенных URL-адресах. Вместо этого мы можем использовать опцию *props* и установить её в *true*. После этого $route.params станет связан с опцией props компонента Post.  
Теперь, когда мы создали роутер, мы можем вернуться к нашему приложению и добавить еще несколько строк в шаблон.

<main>

<aside class="sidebar">

<router-link

v-for="post in posts"

active-class="is-active"

class="link"

:to="{ name: 'post', params: { id: post.id } }">

{{post.id}}. {{post.title}}

</router-link>

</aside>

<div class="content">

<router-view></router-view>

</div>

</main>

Здесь мы имеем два компонента *vue-router*: <router-link> и <router-view>. Первый — это компонент для включения навигации пользователя в приложении с поддержкой роутинга. Второй компонент — это функциональный компонент, который отрисовывает согласованный компонент для данного пути.  
  
Остался заключительный шаг. Нам нужно отобразить содержимое записи поста.

Шаг 4

Перейдём к файлу Post.vue, в котором добавим простой шаблон:

<template lang="html">

<div class="post" v-if="post">

<h1 class="post\_\_title">{{ post.title }}</h1>

<p class="post\_\_body">{{ post.body }}</p>

<p class="post\_\_id">{{ post.id }}</p>

</div>

</template>

Затем нам нужно установить параметры экземпляра Vue для этого компонента. Здесь все также как в настройках отображения всех постов. Объявим опцию *props* с меняющимся *id*, которая будет получать номер нашего поста. Далее, объявим объект данных, как уже делали в App.vue:

import axios from 'axios';

export default {

props: ['id'],

data() {

return {

post: null,

endpoint: 'https://jsonplaceholder.typicode.com/posts/',

}

}

}

Затем опишем метод *getPost()*, который будет получать только одну запись поста по идентификатору и вызовем его в хуке *created()*.

methods: {

getPost(id) {

axios(this.endpoint + id)

.then(response => {

this.post = response.data

})

.catch( error => {

console.log(error)

})

}

},

created() {

this.getPost(this.id);

},

Почти готово. Если мы запустим приложение сейчас, мы увидим, что, хотя URL-адрес меняется, мы видим единственный пост, который был отрисован первым. Дело в том, что для отрисовки разных постов у нас есть один и тот же компонент, и Vue не нужно его пересоздавать из-за лишней траты ресурсов, а это также означает, что хуки жизненного цикла компонента не будут вызваны.  
Чтобы это исправить, нам просто нужно установить watcher для объекта *$route*.

watch: {

'$route'() {

this.getPost(this.id);

}

}

Теперь всё работает так, как и должно. Чтобы получить версию для продакшина достаточно выполнить команду *npm run build* в консоли. 

Подведём итоги

Мы написали простое одностраничное приложение с использованием Vue за четыре шага. Мы узнали, как легко начать свой проект с vue-cli. Мы рассмотрели концепцию однофайловых компонентов Vue, которые делают ваш проект более гибким и масштабируемым. Мы узнали, как извлекать данные из внешнего API с помощью Axios. И мы увидели, как настроить роутинг с помощью vue-router. Разумеется, это базовые знания, но я надеюсь, что это поможет вам начать использовать Vue.js используя его расширенные возможности.