Стало невозможным игнорировать шум вокруг JavaScript фреймворков, но не все из них подойдут любому проекту. Если только вы не хотите настраивать системы сборки ради небольшого приложения. Или у вас нет желания писать весь HTML вперемешку с JavaScript. Список можно продолжать вечно.

Немногие знают, что вы можете внедрять Vue в свой проект, тем же способом, что и jQuery, никаких сборщиков. Vue достаточно гибок, чтобы мы могли использовать его прямо в HTML.

Итак, если код вашей страницы выглядит таким образом:

//\*.html  
<main>  
 <div class="thing">  
 <p>Some content here</p>  
 </div>  
</main>  
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>//\*.js  
немного Vue.js кода

Вы могли бы буквально изменить тэг script и все ещё использовать HTML и JS in вместе, как вы и привыкли, меняя всего пару кусочков кода. Вам не нужно будет переписывать HTML в JavaScript, вам не понадобится webpack, и вам не потребуется настраивать огромную систему:

//\*.html  
<main>  
 <div class="thing">  
 <p>Some content here</p>  
 </div>  
</main>  
<script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.5.3/vue.min.js"></script>//\*.js  
немного Vue.js кода

Вы можете изменить тэг и при это совершенно не касаться разметки. Самое лучшее в этом то, что вы моглибы решить что код станет сложнее и запутаннее. Но читая и рассматривая примеры в каждой из частей этой статьи, вы убедитесь, что Vue невероятно прост, прост в понимании, а также легко поддерживаемый и масшатабируемый. Если говорить о размерах, то они достаточно сравнимы, если использовать их через CDN и минимизированными, Vue версия 2.5.3 занимает **86KB**. jQuery 3.2.1 - **87KB**.

Давайте рассмотрим несколько типичных решений jQuery, как мы переписали их на Vue, и зачем нам вообще это нужно.

**Захват ввода данных пользователем**

Достаточно типичная задача решаемая с помощью JavaScript - захват ввода данных пользователем из формы, с неё и начнём. Мы не будем сразу рассматривать полноценную форму, для упрощения понимания примера, но мы придём к этому постепенно.

Вот как бы мы решили эту задачу с помощью jQuery :

//\*.html  
<div id="app">  
 <label for="thing">Name:</label>  
 <input id="thing" type="text" />  
 <p class="formname"></p>  
</div>//\*.js  
// это синоним $(document).ready(function() {  
$(function() {  
 $('#app').keyup(function(e) {  
 var formname = $(this).find('.formname');  
 //Создаём перменную для устранения дубликации кода  
 var n\_input = $(this).find('#thing').val();  
 formname.empty();  
 formname.append(n\_input);  
 });  
});

Vue:

//\*.html  
<div id="app">  
 <label for="name">Name:</label>  
 <input id="name" type="text" v-model="name" /> <!--v-model is doing the magic here-->  
 <p>{{ name }}</p>  
</div>//\*.js  
//Это сущность vuew  
new Vue({  
 //this targets the div id app  
 el: '#app',  
 data: {  
 name: '' //это свойство хранит данные из поля ‘name’  
 }  
})

Я использую конкретно этот пример, потому что он раскрывает несколько сильных сторон Vue. Он [реактивен](https://gist.github.com/staltz/868e7e9bc2a7b8c1f754), что делает его очень способным к реагированию на изменения. Это можно заметить печатая текст, так как мы обновляем то, что печатаем, это происходит мгновенно и без задержек.

Так же видно, что в версии jQuery, DOM контролируется - мы запрашиваем информацию из DOM, прослушиваем его, и отвечаем ему. Это связывает нас, так как дает понимание, что дом уже загружен, и заставляет нас обдумывать пути обхода. Если структура HTML изменится, нам придется адаптировать наш код для соответствия этим изменениям.

В версии Vue, *мы храним состояние*- отслеживаем изменения одного свойства, которое хотим обновить и изменить. Отслеживаем элемент от которого ожидаем изменений, используя то, что называется директивой. Это значит, что она привязана к конкретному HTML элементу, который нас интересует. Структура DOM поможет меняться, HTML теги могут менять свой порядок, и ничего из этого не повлияет на производительность или отлов этих событий. В нашем случае, мы используем атрибут v-model чтобы соединить его с данными, которые мы храним в JavaScript.

Но! Это не так обыденно, как запись чего либо по нажатию клавиши enter, так что давайте продолжим.

**Сохранение введённого пользователем по единственному событию**

Что интересного в работе с Vue так это о, что нам не стоит зацикливаться на специфических событиях DOM во время хранения и выдачи даных. По сути, мы же знаем, что мы хотим захватить; мы придаем этому форму связывая его с событием. Мы можем наблюдать это в дополненном примере, где информация собирается при нажатии по клавише:

//\*.html  
<div id="app">  
 <label for="thing">Name:</label>  
 <input id="thing" type="text" />  
 <p class="formname"></p>  
</div>//\*.js  
// Это синоним $(document).ready(function() {  
$(function() {  
 //Мы даём тегу div аттрибут id - app и находим остальные элементы для сокращения поисков  
 $('#app').change(function(e) {  
 var n\_input = $(this).find('#thing').val();  
 $(this).find('.formname').append(n\_input);  
 });  
});

//\*.html  
<div id="app">  
 <label for="name">Name:</label>  
 <input id="name" type="text" v-model.lazy="name" />  
 <p>{{ name }}</p>  
</div>//\*.js  
new Vue({  
 el: '#app',  
 data: {  
 name: ''  
 }  
});

В этой версии , jQuery немного упрощена, так как нам не требуется захватывать данные при каждом нажатии, но мы всё ещё ловим их из DOM и отвечаем на каждое изменение. Наш код на jQuery всегда можно будет описать примерно так:

“Найди мне этот элемент, посмотри чем он занят, притормози эти изменения, сделай что нибудь с этими изменениями.”

Для сравнения: В Vue, мы под контролем того, что изменяется, и DOM отвечает на эти изменения базируясь на наших командах. Мы прикрепляемся конкретно к тому, что хотим обновить. В нашем случае, у нас имеется небольшая абстракция называемая модификатором: v-model.lazy. Теперь Vue знает, что ему не стоит хранить это до события изменений. Выглядит просто.

Продолжение во второй части статьи - <https://goo.gl/kQrkz6>

# Тогл(переключение между двумя состояниями) CSS классов

Далее мы рассмотрим тогл CSS классов, так как это самая распространённая задача для jQuery.

jQuery:

//\*.html  
<div id="app">  
 <button aria-pressed="false">Затогли меня</button>  
 <p class="toggle">Иногда мне нужно быть иначе стилизованным</p>  
</div>//\*.css  
.red {  
 color: red;  
}//\*.js  
$(function() {  
 $('button').click(function(e) {  
 $('.toggle').toggleClass('red');  
 $(this).attr('aria-pressed', ($(this).attr('aria-pressed') == "false" ? true : false));  
 });  
});

Vue:

//\*.html  
<div id="app">  
 <button @click="active = !active" :aria-pressed="active ? 'true' : 'false'">Затогли меня</button>  
 <p :class="{ red: active }">Иногда мне нужно быть иначе стилизованным</p>  
</div>//\*.js  
new Vue({  
 el: '#app',  
 data: {  
 active: false  
 }  
})

И опять в версии на jQuery мы храним состояние в DOM. У элемента есть класс, и jQuery принимает решение в зависимости от наличия класса, которое она проверяет отправляя запросы DOM. В версии Vue , мы храним условие, оно будет изменяться в зависимости от состояния. Мы не запрашиваем информацию из DOM, мы храним её самостоятельно.

Мы храним свойство active внутри data, кнопка переключает условие, и класс .red меняет свое состояние базируясь на условии.

Скорее всего вы уже заметили, что опасение о том, что с Vue.js придётся писать больше кода полностью развеиваются.

# Скрытие и раскрытие

Типичной задачей для jQuery является скрытие и раскрытие чего-либо. jQuery отлично справлялась с тем, чтобы эта задача была как можно более простой. Давайте сравним реализацию с Vue.

jQuery:

//\*.html  
<div id="app">  
 <button type="button" id="toggle" aria-expanded="false">  
 Toggle Panel  
 </button>  
 <p class="hello">hello</p>  
</div>//\*.js  
$(function() {  
 $('#toggle').on('click', function() {  
 $('.hello').toggle();  
 $(this).attr('aria-expanded', ($(this).attr('aria-expanded') == "false" ? true : false));  
 });  
});

Vue:

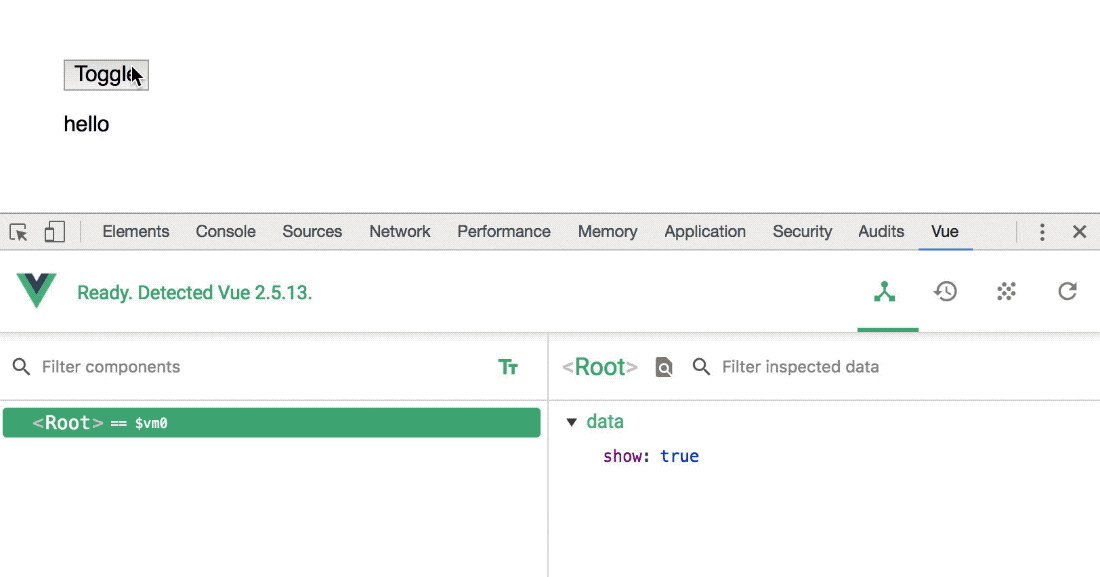
//\*.html  
<div id="app">  
 <button @click="show = !show" :aria-expanded="show ? 'true' : 'false'">  
 Toggle Panel  
 </button>  
 <p v-if="show">hello</p>  
</div>//\*.js  
new Vue({  
 el: '#app',  
 data: {  
 show: true  
 }  
})

И jQuery и Vue отлично справляются с упрощением задачи, но есть несколько причин, чтобы выбрать Vue для тогла. У Vue есть инструмент под названием [Vue devtools](https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd?hl=en). он совсем непохож на Chrome devtools, он даёт нам понять какие процессы Vue выполняются.

В обоих версиях jQuery и Vue мы можем наблюдать как элемент “прячется” и появляется. Но что если что-то пойдёт не по плану? Чтобы начать процесс дебагинга с jQuery нам понадобилось бы добавить console.log-и или расставить точки остановки, чтобы разобраться с проблемой.

Нет ничего плохого в том, чтобы использовать console.log-и, но с помощью Vue devtools мы можем наблюдать за выполнением кода в реальном времени. В гифке ниже можно заметить, что после нажатий на кнопку Vue devtools обновляет состояние true/false соответственно. Если DOM когда-либо отработает не так как мы этого ожидаем, мы все еще сможем проследить изменения в Vue в реальном времени. Это значительно облегчает процесс дебагинга приложения.

https://miro.medium.com/freeze/max/60/1*YGruZXccWPjtXIKmIiim9Q.gif?q=20



Так же v-if легко изменяется и под другие условия. Я могу использовать v-show вместо v-if если то, состояние чего я изменяю, будет скрываться и показываться часто: v-if полностью извлечёт элемент, в то время как v-show просто изменит свойство visibility. Это различие является очень важным, так как изменять visibility требует меньше производительных мощностей, чем полностью изымать/вставлять элемент DOM. Я могу показывать или прятать что-либо базируясь на множестве условий. Это место как раз вызывает путаницу в jQuery, стучаться к DOM в нескольких местах и координировать их соответственно. Ниже показан пример координирования изменений базируясь на присутствии ввода информации пользователем:

//\*.html  
<div id="app">  
 <label for="textarea">Какой у тебя любимый вид шаурмы?</label>  
 <textarea id="textarea" v-model="tacos"></textarea>  
 <br>  
 <button v-show="tacos">Дай нам знать!</button>  
</div>//\*.js  
new Vue({  
 el: '#app',  
 data() {  
 return {  
 tacos: ''  
 }  
 }  
})

jQuery:

//\*.html  
<div id="app">  
 <label for="textarea">Какой у тебя любимый вид шаурмы?</label>  
 <textarea id="textarea"></textarea>  
 <br>  
 <button v-show="tacos">Дай нам знать!</button>  
</div>//\*.js  
$(function() {  
 var button = $('.button');  
 var textarea = $('#textarea'); button.hide();  
 textarea.keyup(function() {  
 if (textarea.val().length > 0) {  
 button.show();  
 } else {  
 button.hide();  
 }  
 })  
});

# Контролирование форм

Традиционной задачей для jQuery является отправка данных из формы с помощью AJAX, поэтому нам следует рассмотреть и это. На самом деле Vue не имеет встроенных инструментов по типу AJAX; В основном при написании Vue кода используют что то типа Axios (JavaScript библиотека для создания HTTP запросов).

Этот пример немного сложнее остальных.

План действий:

1. Кнопка будет серой до начала ввода в поле формы, потом она получит класс “active” и станет синей;
2. Когда мы подтверждаем отправку информации из формы, мы приостановим загрузку страницы.
3. Когда данные будут отправлены, мы отобразим ответ сервера на странице.

jQuery:

//\*.html  
<div id="app">  
 <form action="/">  
 <div>  
 <label for="name">Name:</label><br>  
 <input id="name" type="text" name="name" required/>  
 </div>  
 <div>  
 <label for="email">Email:</label><br>  
 <input id="email" type="email" name="email" required/>  
 </div>  
 <div>  
 <label for="caps">HOW DO I TURN OFF CAPS LOCK:</label><br>  
 <textarea id="caps" name="caps" required></textarea>  
 </div>  
 <button class="submit" type="submit">Submit</button>  
 <div>  
 <h3>Response from server:</h3>  
 <pre class="response"></pre>  
 </div>  
 </form>  
</div>//\*.js  
$(function() {  
 var button = $("button");  
 var name = $("input[name=name]");name.keyup(function() {  
 if (name.val().length > 0) {  
 button.addClass('active');  
 } else {  
 button.removeClass('active');  
 }  
 });$("form").submit(function(event) {  
 event.preventDefault();//получение данных из формы  
 var formData = {  
 name: $("input[name=name]").val(),  
 email: $("input[name=email]").val(),  
 caps: $("input[name=caps]").val()  
 };// отправка данных, обработка ответа  
 $.ajax({  
 type: "POST",  
 url: "//jsonplaceholder.typicode.com/posts",  
 data: formData,  
 dataType: "json",  
 encode: true  
 }).done(function(data) {  
 $(".response")  
 .empty()  
 .append(JSON.stringify(data, null, 2));  
 });  
 });  
});

Код на строчках с 2 по 10 отвечает за смену класса кнопки, как мы уже делали ранее. Мы отправляем параметр названый “event” в форму, и потом пишем event.preventDefault() предотвращая обновление страницы. Далее мы собираем данные из полей формы, отправляем их, и посылаем ответ от сервера в метод .done() AJAX запроса.

Vue:

//\*.html  
<div id="app">  
 <form @submit.prevent="submitForm">  
 <div>  
 <label for="name">Name:</label><br>  
 <input id="name" type="text" v-model="name" required/>  
 </div>  
 <div>  
 <label for="email">Email:</label><br>  
 <input id="email" type="email" v-model="email" required/>  
 </div>  
 <div>  
 <label for="caps">HOW DO I TURN OFF CAPS LOCK:</label><br>  
 <textarea id="caps" v-model="caps" required></textarea>  
 </div>  
 <button :class="[name ? activeClass : '']" type="submit">Submit</button>  
 <div>  
 <h3>Response from server:</h3>  
 <pre>{{ response }}</pre>  
 </div>  
 </form>  
</div>//\*.js  
new Vue({  
 el: '#app',  
 data() {  
 return {  
 name: '',  
 email: '',  
 caps: '',  
 response: '',  
 activeClass: 'active'  
 }  
 },  
 methods: {  
 submitForm() {  
 axios.post('//jsonplaceholder.typicode.com/posts', {  
 name: this.name,  
 email: this.email,  
 caps: this.caps  
 }).then(response => {  
 this.response = JSON.stringify(response, null, 2)  
 })  
 }  
 }  
})

В версии Vue мы решаем какие поля формы заполнять, и “прикрепляем” их с помощью v-model. Мы проверяем наличие имени чтобы тоглить класс. Вместо отправки параметра event и event.preventDefault(),всечто нам потребуется написать - это @submit.prevent на нашем элементе формы, он сделает все за нас. Для отправки запроса мы используем Axios и сохраняем ответ сервера внутри сущности Vue.

Для версии готового продукта нам следует добавить [валидацию](https://vuejs.org/v2/cookbook/form-validation.html), отлов о, и [написать тесты](https://eddyerburgh.me/unit-test-vue-components-beginners), но в этом маленьком примере можно заметить каким аккуратным и чистым может быть код Vue.

# Вывод

Вполне нормально использовать jQuery! Смысл этой статьи показать что Vue предоставляет отличную абстракцию для небольших сайтов без излишеств. Vue сопоставима по размеру, и достаточно проста для изменения кусочков готового функционала без переписывания структуры HTML в JavaScript и адаптации систем построения проектов.

Vue является очень гибкой библиотекой, поэтому очень легко подстроится под проекты на всех этапах, начиная с самого начала до разрастания до большого проекта.

Источник - [https://www.smashingmagazine.com/2018/02/jquery-vue-javascript](https://www.smashingmagazine.com/2018/02/jquery-vue-javascript/)