

Vue.js

# Знакомимся с фреймворком Vue и его главными концепциями

---

# На этом уроке

1. Познакомимся с фреймворком Vue.js, его ключевыми особенностями и главными концепциями
2. Научимся настраивать webpack сборку для проектов на Vue

## Оглавление

[На этом уроке](#)

[Оглавление](#)

[Теория урока](#)

[\[Опишите тезисно теоретическую составляющую занятия.\]](#)

[\[Заголовок 1\]](#)

[\[Заголовок 2\]](#)

[\[Заголовок 3\]](#)

[Внимание!](#)

[Что такое Vue.js](#)

[Внимание!](#)

[Директивы](#)

[Условия в шаблонах](#)

[Циклы в шаблонах](#)

[v-model](#)

[Практика](#)

[Глоссарий](#)

[Дополнительные материалы](#)

[Используемые источники](#)

[От редактора: оформление, ошибки и опечатки](#)

## Проблемы построения UI

Сложные пользовательские интерфейсы включают в себя работу со списками, формами, управляющими элементами (такими как - кнопки слайдера и так далее). Как правило интерфейсы строятся из отдельных блоков, каждый из которых имеет какой-то интерактивный функционал, но сложность возникает при композиции нескольких элементов на странице или даже организации многостраничного приложения.

Пользовательские интерфейсы не сразу стали сложными. 10-15 лет назад достаточно было иметь статический сайт с формой для отправки заявки или заказа. С тех пор индустрия сильно развилась, а вместе с ней также выросли ожидания пользователей по возможностям, которые должна предоставлять веб-страница.

Часто бывает, что задачи, которые должно решать веб-приложение, основываются на сложной логике. Учитывая это, реализация такого сложного функционала требует применения нескольких парадигм для управления данными.

Какие именно проблемы возникают:

- отображение на странице переменных JS, которые могут меняться по какой-то логике
- действия, которые выполняют пользователи - клики по кнопкам, ввод данных в форме, - все это должно обрабатываться логикой скриптов
- необходимость упростить создание сложных приложений путем их разбиения на логические и функциональные блоки
- обеспечение воспроизводимости поведения - при выполнении одних и тех же действий мы должны получать тот же самый результат - и визуально на странице, и в модели данных
- отработка асинхронных процессов, которые могут выполняться параллельно, например загрузка данных и обработка других действий пользователя.

Учитывая озвученные проблемы, разрабатывать сложное приложение, которое будет обладать большим функционалом, может быть достаточно сложно. Чтобы упростить себе жизнь, можем воспользоваться специальной платформой, которая позволит облегчить разработку отдельных программных компонентов и объединить их в один проект.

По своей сути, мы сейчас почти что дали определение понятию “фреймворк”. В JavaScript существует множество фреймворков различной степени функциональности и, соответственно, популярности. В рамках нашего курса мы с вами рассмотрим фреймворк Vue.js

## Что такое Vue

Vue, как написано на главной странице проекта - это "прогрессивный JavaScript фреймворк". Vue создан, чтобы дать возможность быстро создавать сложные пользовательские интерфейсы.

Прогрессивным он называется, потому что использовать все его возможности совершенно необязательно. Программист может использовать функционал фреймворка по мере усложнения задач, которые решает разрабатываемое приложение.

Например, на начальном этапе разработчик может даже не использовать webpack для сборки кода, а просто подключать фреймворк через самый обычный тег script. И только по мере развития проекта перейти на какой-нибудь сборщик. И так с любым инструментарием, предоставляемым фреймворком. Его возможности прогрессивно раскрываются с ростом потребностей разработчиков - мы увидим это не раз на протяжении курса.

Создатель фреймворка начал работу над проектом в 2015 году. Эван Ю, до этого работал в команде Google которая развивала другой популярный тогда проект AngularJS, поэтому некоторые из решений во Vue напомнят вам Angular если вы с ним работали.

Особенно популярен стал Vue среди разработчиков в 2017 когда вышла вторая мажорная версия фреймворка.

### react vs vue vs jquery vs @angular/core

Enter an npm package...

react

vue

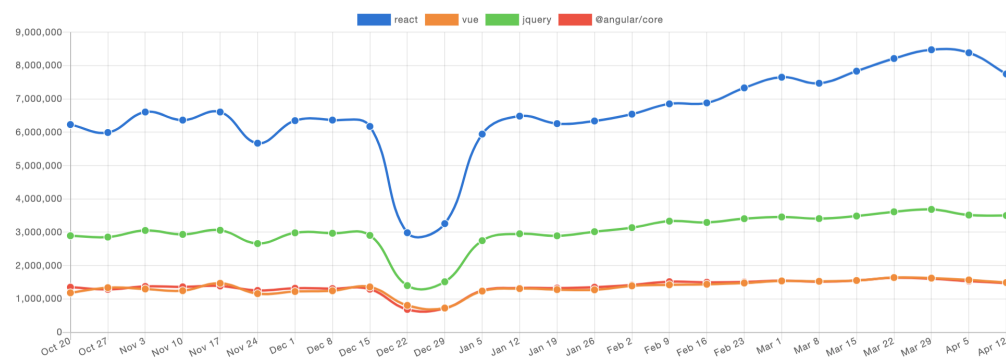
jquery

@angular/core

+ angular

+ ember-source

Downloads in past 6 Months



### Stats

	stars 🌟	forks 🍴	issues 🚩	updated 🔄	created 📅	size 📦
<div>react</div>	147,400	28,506	547	Apr 23, 2020	May 24, 2013	minzipped size 2.6 KB
<div>vue</div>	162,583	24,603	491	Apr 24, 2020	Jul 29, 2013	minzipped size 22.8 KB
<div>jquery</div>	53,216	19,258	84	Apr 25, 2020	Apr 3, 2009	minzipped size 30.4 KB
<div>@angular/core</div>	60,302	16,465	3,415	Apr 25, 2020	Sep 18, 2014	minzipped size 90.1 KB

На сегодняшний день Vue стоит в ряду самых популярных фреймворков для фронтенд разработки. По многим параметрам - простота освоения, размер кода ядра, число звезд у проекта на GitHub, поддержка web components api, он опережает ближайших конкурентов - Angular и React.

## Vue 3

В сентябре 2020 года вышла третья версия фреймворка, которая принесла существенный прирост в производительности и новые фичи - ее мы рассмотрим в заключительном занятии курса.

Но так как большинство изменений которые принесла 3я версия могут использоваться параллельно с функционалом 2й версии мы сконцентрируемся именно на 2й версии а для рассмотрения 3й версии - у нас будет отдельное занятие в конце курса.

## Подключение Vue на страницу

Первый способ использования Vue, заключается в подключении к странице скрипта Vue в теге `<script>` непосредственно на страницу. То есть если у нас есть HTML-страница, которую надо оснастить неким функционалом, реализованным в какой-то JS-библиотеке, то для этого библиотеку подключают к странице с помощью тега `<script>`.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Vue app</title>
</head>
<body>
  <!-- тут прочая разметка -->
</body>

<!-- вот мы подключаем JS с кодом Vue -->
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
<script>
  // тут код нашего Vue приложения
</script>
</html>
```

Используя приведенный выше подход, просто подключив скрипт Vue к странице, можно создавать приложения, но это плохо масштабируется. То есть, если нужно разработать полноценное приложение среднего или большого размера (и при этом использовать другие инструменты, доступные в vue приложениях) тогда лучше всего будет использовать Vue CLI.

## Установка Vue CLI

Vue CLI это набор инструментов для командной строки, значительно упрощающий жизнь разработчикам, которые хотят запустить новый проект на Vue.js. Он позволяет с помощью одной команды сгенерировать основу (boilerplate) для нового приложения, подключить плагин и даже создать новое приложение с помощью графического интерфейса в браузере (vue ui).

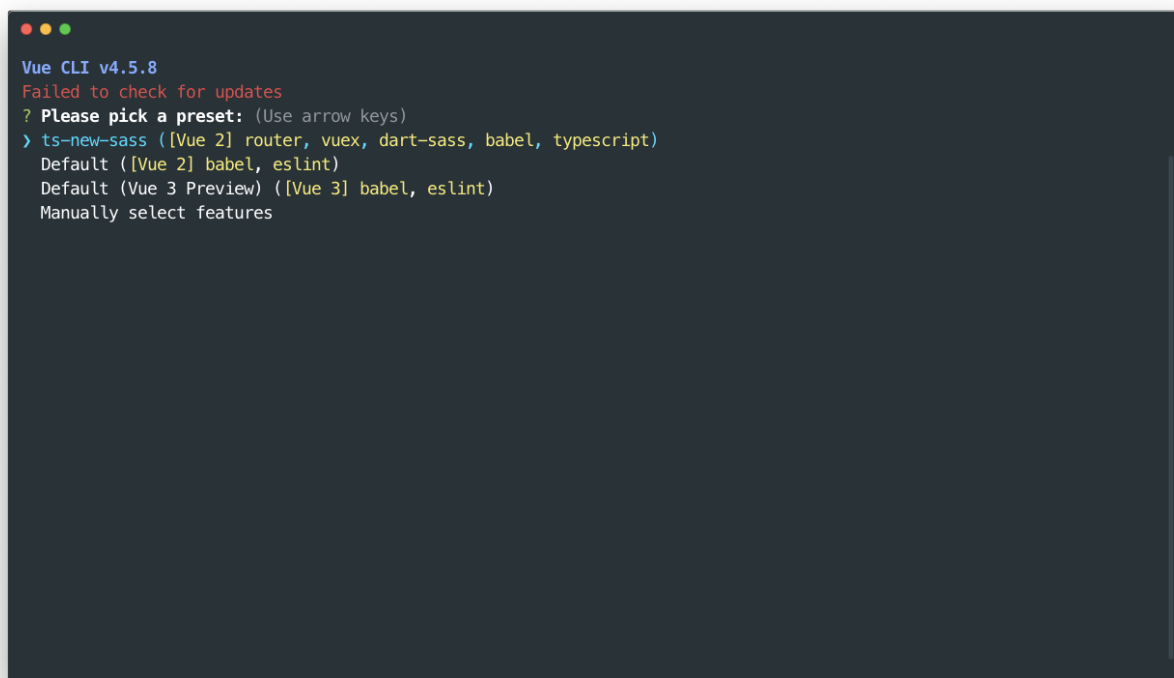
Vue CLI из коробки поддерживает Babel, TypeScript, ESLint, PostCSS, а также unit и end-to-end тесты.

Для установки vue-cli используем команду установки NPM пакета:

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

## Создание нового проекта Vue

Теперь для создания нового проекта во Vue запустим команду **vue create first-project**. Появится вопрос, о том, какие инструменты мы хотим использовать в проекте. Выберем вариант “Manually select features”, чтобы увидеть, какие возможности у нас есть.



```
Vue CLI v4.5.8
Failed to check for updates
? Please pick a preset: (Use arrow keys)
> ts-new-sass ([Vue 2] router, vuex, dart-sass, babel, typescript)
  Default ([Vue 2] babel, eslint)
  Default (Vue 3 Preview) ([Vue 3] babel, eslint)
  Manually select features
```

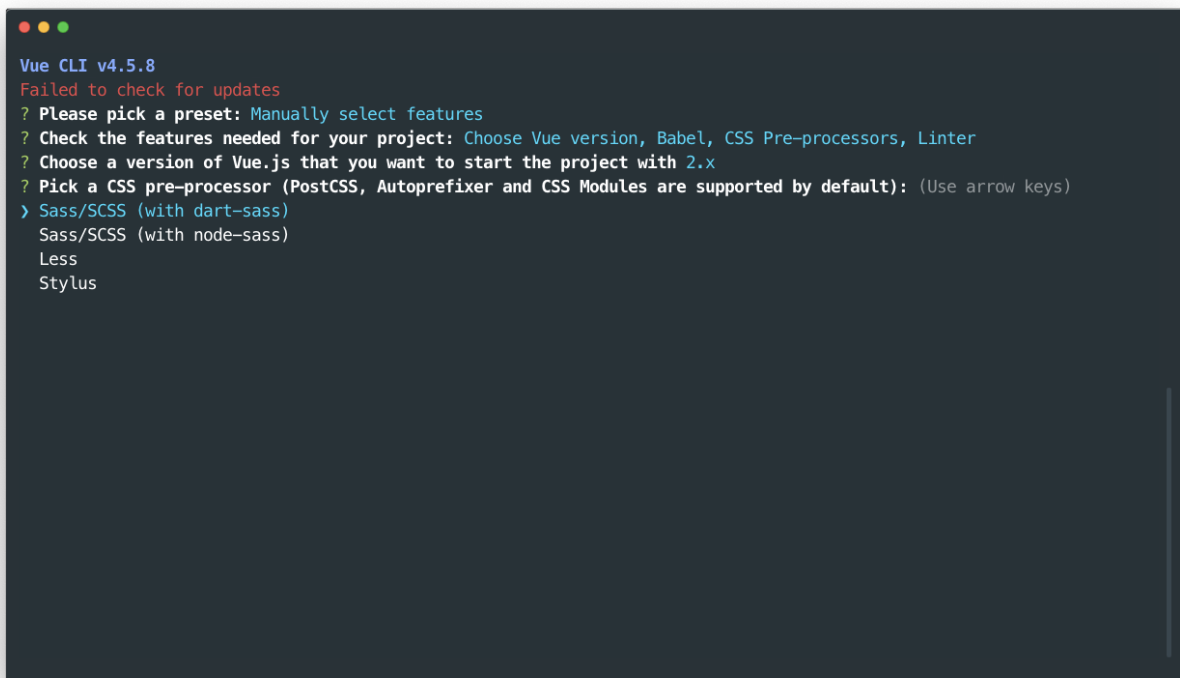
Выберем часть опций - в частности: **Babel, Linter и CSS pre-processors**.

```
Vue CLI v4.5.8
Failed to check for updates
? Please pick a preset: Manually select features
? Check the features needed for your project:
  ● Choose Vue version
  ● Babel
  ○ TypeScript
  ○ Progressive Web App (PWA) Support
  ● Router
  ● Vuex
  > ● CSS Pre-processors
  ● Linter / Formatter
  ○ Unit Testing
  ○ E2E Testing
```

Так как нас сегодня уже доступна 3я версия Vue, следующий экран предлагает выбрать, что именно установить. Выберем пока версию 2.x

```
Vue CLI v4.5.8
Failed to check for updates
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel, Router, Vuex, CSS Pre-processors, Linter
? Choose a version of Vue.js that you want to start the project with (Use arrow keys)
> 2.x
  3.x (Preview)
```

Затем из вариантов CSS pre-processor'ов выбираем **Sass/SCSS**. (pre-processor'ы CSS нужны для того, чтобы сделать код стилей более кратким и удобочитаемым. Этот код во время сборки приложения конвертируется в обычный код CSS. Делается это именно с помощью препроцессоров).



```
Vue CLI v4.5.8
Failed to check for updates
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel, CSS Pre-processors, Linter
? Choose a version of Vue.js that you want to start the project with 2.x
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by default): (Use arrow keys)
> Sass/SCSS (with dart-sass)
  Sass/SCSS (with node-sass)
  Less
  Stylus
```

Затем, в шаге настройки линтера/форматтера выбираем **ESLint with error prevention only**. После этого выбираем опцию **Lint on save** и хранение конфигураций а отдельных файлах.



```

Vue CLI v4.5.8
Failed to check for updates
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel, Router, Vuex, CSS Pre-processors, Linter
? Choose a version of Vue.js that you want to start the project with 2.x
? Use history mode for router? (Requires proper server setup for index fallback in production) Yes
? Pick a CSS pre-processor (PostCSS, Autoprefixer and CSS Modules are supported by default): Sass/SCSS (with dart-sass)
? Pick a linter / formatter config: (Use arrow keys)
> ESLint with error prevention only
  ESLint + Airbnb config
  ESLint + Standard config
  ESLint + Prettier

```

Получив от нас первоначальные инструкции, vue-cli начнет инициализацию проекта, установку необходимых модулей и генерацию структуры проекта из папок и файлов. В конце должно появиться сообщение об успешной установке проекта, а также инструкция дальнейших действий.

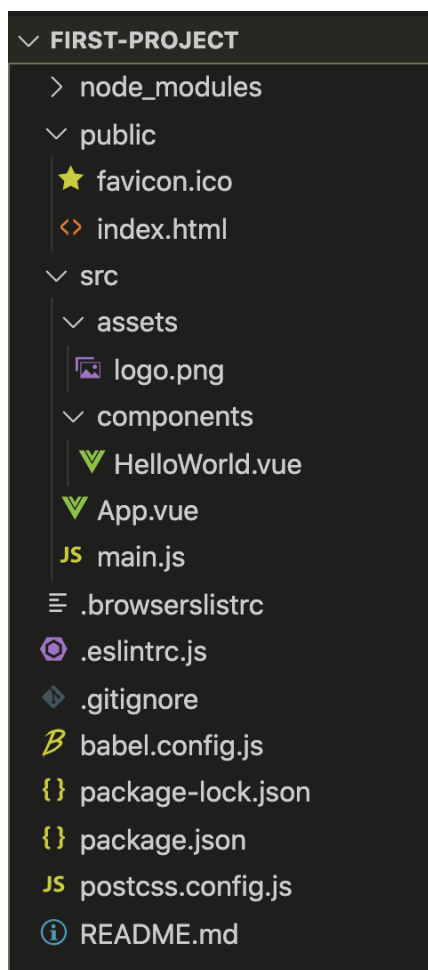
```

Vue CLI v4.5.8
Failed to check for updates
✨ Creating project in /Users/QJ26DT/Desktop/vue/my-app.
📁 Initializing git repository...
⚙ Installing CLI plugins. This might take a while...

(⌘) : fetchMetadata: sill install loadAllDepsIntoIdealTree

```

В результате наших действий, у нас создалась папка с именем будущего проекта (first-project) содержащую в себе стандартную структуру файлов.



## Разбор структуры проекта Vue

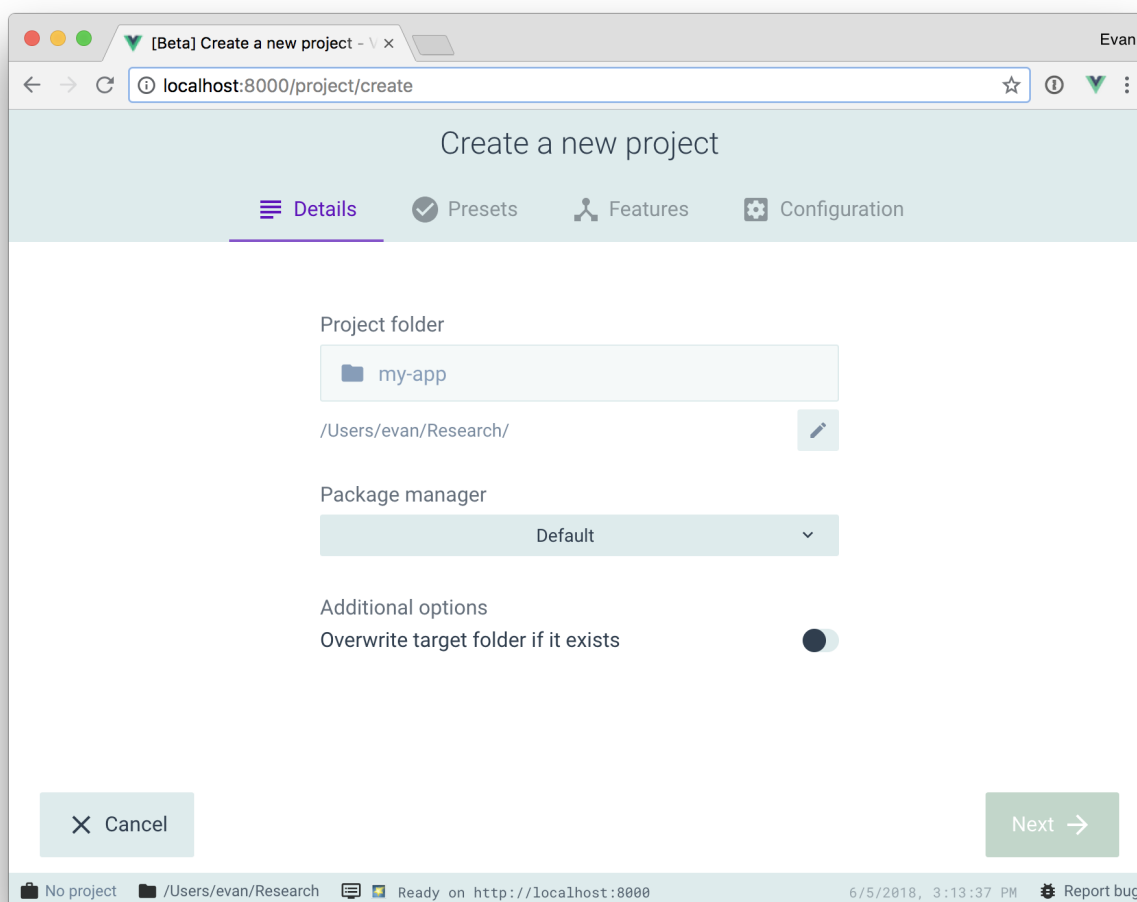
Рассмотрим файлы в созданном проекте, и что находится в каждой из папок:

- `node_modules` - в этой папке лежат коды и исполняемые файлы установленных внешних (не наших) модулей, ее не стоит открывать и менять. Также ее не стоит коммитить в репозиторий (поэтому она по умолчанию была добавлена в файл `.gitignore`)
- `public` - здесь лежат файлы, которые используются в проекте, но при сборке будут скопированы без изменений. Исключением является только файл `index.html` - во время компиляции в него добавятся тэги `<script>`, подключающие трансформированные файлы `.js`.
- `src` - папка со исходным кодом проекта. Здесь будет лежать вся логика нашего приложения
- `src/assets` - здесь обычно лежат изображения и `css` файлы со стилями, которые используются в приложении
- `src/components` - папка, содержащая отдельные Vue компоненты в виде файлов с расширением `vue`
- `src/main.js` - входной файл проекта во Vue, мы будем его изменять, когда захотим что-нибудь подключить к нашему приложению, например - плагин.

- `src/App.vue` - корневой компонент приложения, все остальные компоненты будут добавляться внутри него
- `src/components/HelloWorld.vue` - пример обычного компонента, который мы подключаем

Если вы из тех, кто не очень любит работать в командной строке, тогда вы вполне можете воспользоваться инструментом с аналогичными возможностями, имеющим графический интерфейс Vue UI - больше про него можно прочитать тут -

<https://cli.vuejs.org/ru/guide/creating-a-project.html#%D0%B8%D1%81%D0%BF%D0%BE%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-gui>



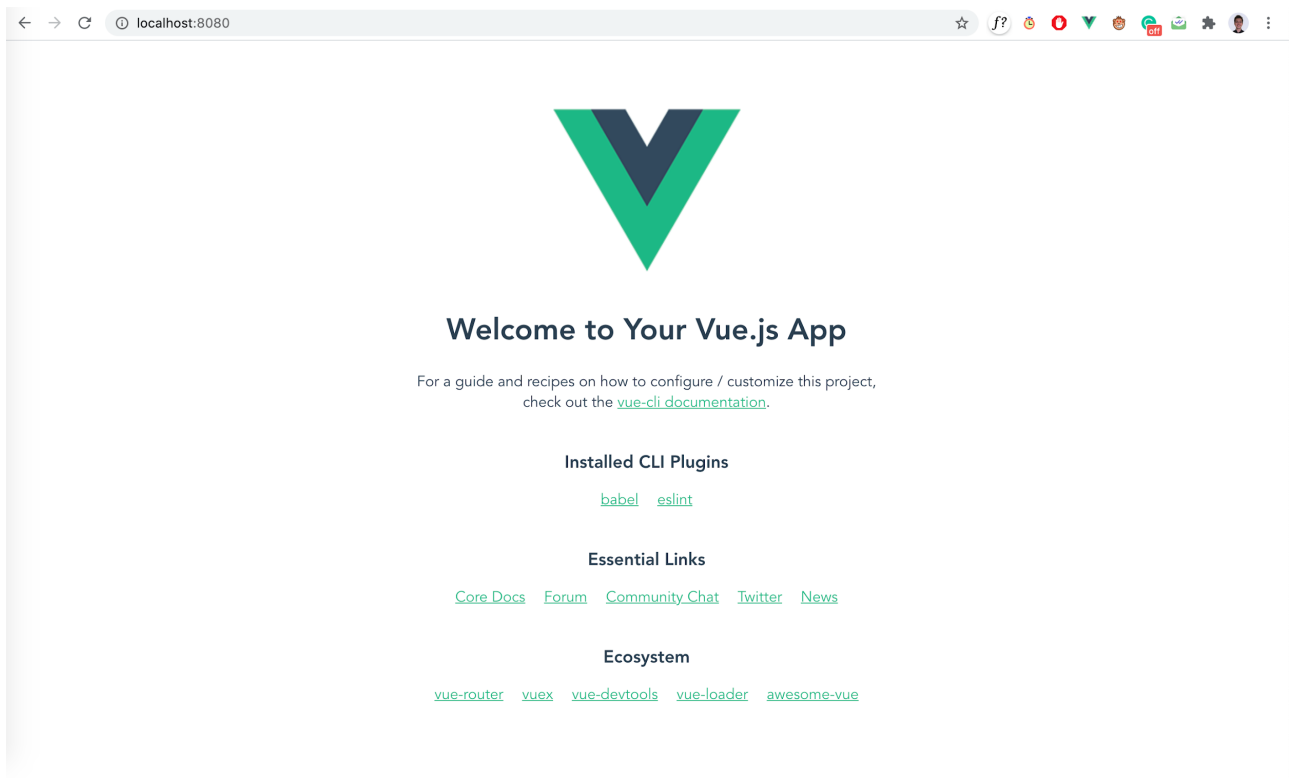
Для запуска приложения перейдем в его папку (в терминале **cd first-project**) и запустим его с помощью команды **npm run serve**. Спустя несколько секунд после успешной сборки мы должны увидеть похожее сообщение:

```
DONE Compiled successfully in 3111ms 10:17:51 AM

App running at:
- Local: http://localhost:8080/
- Network: http://192.168.2.2:8080/

Note that the development build is not optimized.
To create a production build, run npm run build.
```

Перейдем в браузер и убедимся, что приложение запущено и работает. Для этого в адресной строке впишем <http://localhost:8080>. В браузере должны отображаться страница с текстом Welcome to Your Vue.js App.



Сейчас приложение запущено в режиме live-reload, то есть, если мы изменим его код - изменения отобразятся на странице.

Разберем файлы приложения более детально:

- `main.js` - здесь создается инстанс приложения Vue, который привязывается к html элементу с идентификатором **app**. Мы можем найти этот div элемент в файле `index.html`. Функциональный параметр **render** использует единственный подгруженный компонент - корневой компонент `App.vue`.

Обратите внимание на свойство `el` - с помощью него мы говорим Vue, в каком месте ему искать шаблон в DOM дереве. Vue автоматически найдет элемент и превратит все его содержимое в шаблон. С этого момента внутри шаблона нам станут доступны данные нашего приложения

- `App.vue` - В шаблоне находится корневой элемент `div` и два элемента внутри, один из которых - компонент `HelloWorld`. Его подключение производится в js-части с помощью кода

```
components: {  
  HelloWorld,  
}
```

Такая запись является ничем иным, как как сокращенным объявлением свойства ES6. Мы подключаем компонент в приложение и будем использовать его по имени свойства. Если нам необходимо использовать в шаблоне компонент под другим именем, то мы можем сделать это:

```
components: {  
  "MySecondComponent": HelloWorld  
}
```

- `HelloWorld.vue` - Шаблонная часть здесь гораздо больше, а часть с кодом js содержит неизвестное нам ранее свойство **props**, которое используется для передачи данных дочерним компонентам (`HelloWorld` является дочерним компонентом для компонента `App`).. Эти данные затем можно использовать в коде шаблона (`{{ msg }}`). Более подробно о передаче данных между компонентами мы поговорим с вами на следующих уроках.

Вернемся в код App.vue и изменим передаваемое значение пропса **msg** "Welcome to Your Vue.js App" на "Здесь могла быть ваша реклама". После изменения файла изменения должны отобразиться в браузере, так как приложение запущено в режиме разработки с опцией живой перезагрузки.

```
<template>
  <div id="app">
    
    <HelloWorld msg="Здесь могла быть ваша реклама"/>
  </div>
</template>
```

Посмотрим еще раз более детально на файл App.vue - он, как и любой файл с расширением .vue это по умолчанию - Single File Vue Component или сокращенно SFC. Он имеет состоит из трех основных частей:

- Шаблон - разметка HTML внутри тегов <template></template>
- JS логика - код внутри тегов <script></script>
- Стили - css/sass код внутри тегов <style></style>

```
<template>
  ... отображение данных, шаблон
</template>

<script>
  ... данные, логика их изменения
</script>
```

В коде компонента мы отделяем шаблон от логики, в действительности же, под капотом, Vue создает реактивную связь между данными и их представлением.

Благодаря этому, нам больше нет необходимости работать с DOM напрямую, Vue позаботится о максимально производительном обновлении DOM дерева в зависимости от изменений наших данных. Давайте посмотрим на пример простого компонента:

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

```
{
  name: 'MyComponent',
```

```
data: () => ({
  message: 'Привет, Vue!'
})
}
```

Свойство **data** - это функция, которая возвращает объект с начальными данными компонента (стейт, если вы работали с React), которые Vue автоматически делает реактивными. Что это значит? Мы поговорим об этом подробнее позже, но на данный момент достаточно упомянуть, что Vue создает список зависимостей от каждого свойства объекта **data**. С этого момента он автоматически будет перерисовывать те части шаблона, которые ссылаются на какое-то свойство в этом объекте. При этом, перерисовка будет выполняться “по-умному”, с помощью виртуального DOM.

Помимо data у компонента есть ещё ряд свойств, например:

- name - для указания имени компонента;
- components - для декларации дочерних компонентов используемых в шаблоне;
- methods - для описания функций которые могут работать с данными компонента - как методы JS класса
- computed, watch - для специальных реактивных свойств;
- create, beforeDestroy - события жизненного цикла компонента;

На следующих занятиях мы подробно разберем каждый из них.

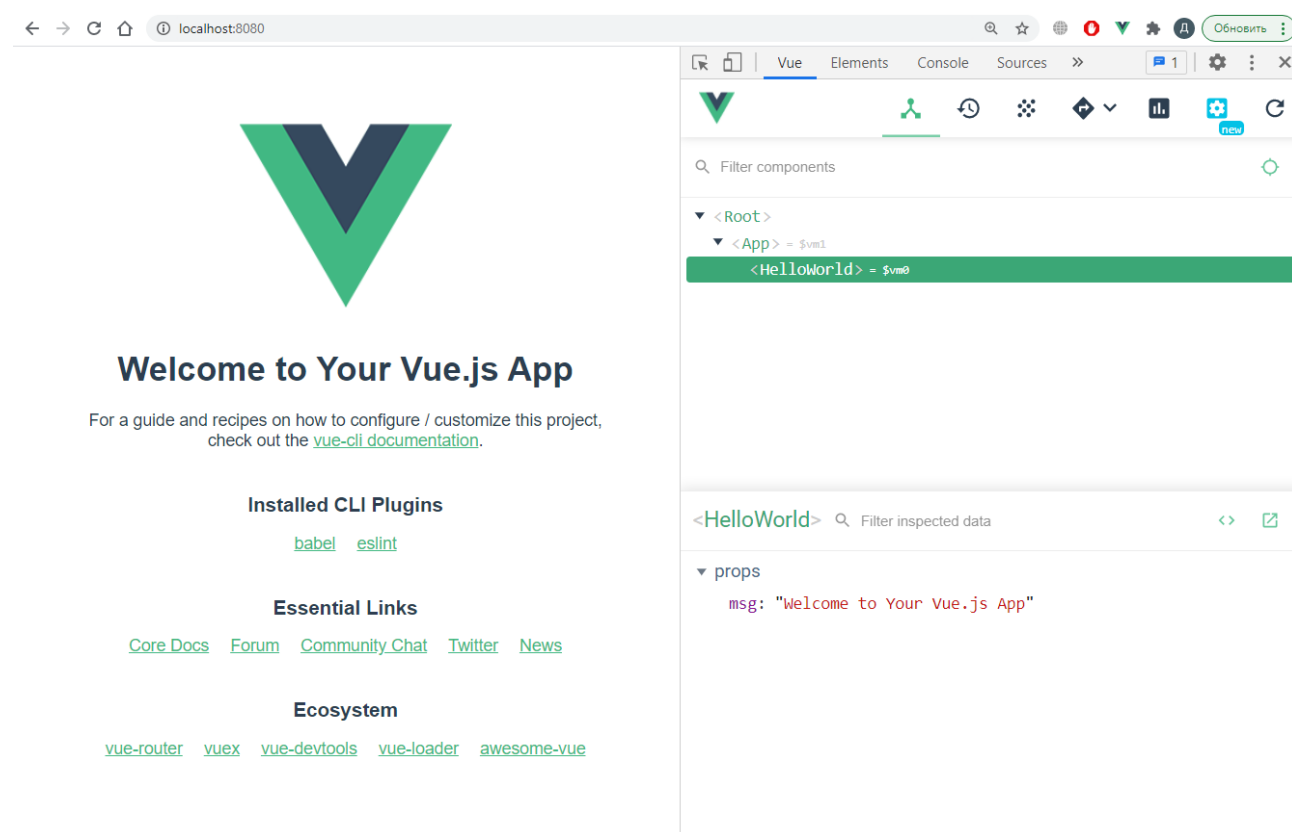
Пока важно понять что совокупность свойств объекта компонента Vue ещё называется Options API, то есть по сути список опций с помощью которых мы можем описать наш компонент.

## Vue DevTools

Прежде чем мы с вами перейдем к дальнейшему ознакомлению с возможностями Vue, стоит остановиться на инструментах разработчика Vue приложения. Когда мы с вами работаем над каким-то проектом, мы постоянно пользуемся панелью разработчика в браузере. Там мы можем отслеживать текущее состояние DOM, заниматься отладкой JS кода, просматривать и изменять данные приложения (такие как куки, локальное хранилище и прочие). Однако, когда мы работаем с фреймворком, у нас появляются дополнительные возможности разработки.

Мы уже с вами познакомились с таким понятием как компонент. К сожалению, стандартная панель разработчика в браузере не умеет отображать иерархию компонентов в приложении, так как она это делает с DOM элементами. Также хотелось бы уметь просматривать данные, которыми владеет тот или иной компонент. На помощь нам придет специальное расширение - [Vue DevTools](#).

Vue DevTools - это официальное расширение для браузера от команды Vue. С его помощью можно отслеживать текущее состояние Vue приложения - просмотреть отрендеренные компоненты, узнать их текущее состояние, понять какими данными они владеют, какие props в них передаются, какие события они генерируют. Все это, а также многое другое. Мы с вами в течение курса будем активно работать с Vue DevTools. Как же оно выглядит?





Как можно видеть из скриншоты выше, основной экран Vue DevTools представляет из себя инспектор компонентов. Он похож на инспектор элементов DOM обычной панели разработчика, за тем лишь отличием, что в инспекторе Vue предоставляется информация об иерархии компонентов, а не html элементов. Если выбрать какой-то компонент из иерархии, то снизу инспектора будет выведена краткая (но исчерпывающая) информация о состоянии выбранного компонента. В нашем случае мы выбрали в инспекторе компонент `<HelloWorld />` и смогли увидеть, что через props ему передали свойство `msg` со значением "Welcome to Your Vue.js App". Здесь же будут выводиться данные из блока `data` и других источников, таких как, например, `computed` свойства. Но о них мы поговорим мы будущих занятиях!

## Директивы

Еще одной важной концепцией Vue являются **директивы**. Для описания шаблонов мы используем html с дополнительными конструкциями, которые помогают связывать представление с данными. Мы уже говорили об `{{ интерполяциях }}`, пришло время поговорить еще об одном способе связывать наши данные с шаблоном.

Директива - это специальный атрибут html элемента или компонента с приставкой **v-**, который "привязывает" изменение этого элемента к какому-то js выражению.

Рассмотрим на первом занятии одну из популярных встроенных директив Vue.

### Директива `v-on`

Прикрепляет к элементу подписчик события. Тип события указывается в параметре после тире, например:

```
v-on:click
v-on:input
v-on:focus
v-on:mouseover
v-on:keyup
```

Выражение может быть именем метода, inline-выражением или вовсе отсутствовать, если указан один или несколько модификаторов.

Давайте посмотрим на варианты использования `v-on`:

```
<!-- 1: inline-выражение -->
<button v-on:click="c = a + b"></button>
```

```
<!-- 2: вызов метода обработчика с параметрами -->
<button v-on:click="doThat('hello', $event)"></button>

<!-- 3: вызов метода обработчика без параметров -->
<button v-on:click="doThis"></button>
```

У обычного элемента можно подписаться только на нативные события DOM. У элемента компонента можно подписаться на пользовательские события, вызываемые этим дочерним компонентом - это мы разберем на 3м уроке.

**v-on:event** также можно записать в сокращенной форме **@event**

### Практическое задание: калькулятор

Так в качестве модельного здания, чтобы изучить базовые возможности Vue и недавно изученной директивы v-on, мы попробуем реализовать простейший калькулятор. Мы все знаем как выглядит калькулятор физическом мире, в нашем же случае будем делать на странице как-то у нас есть возможность сделать несколько полей для ввода в частности 2 для указания 2 цифр которые мы будем складывать вычитать или перемножать.

Начнем с формы в которой эти элементы будут реализованы:

```
<template>
  <div>
    <input v-model="operand1"/>
    <input v-model="operand2"/>
  </div>
</template>

<script>
export default {
  name: 'Calculator',
  data() {
    return {
      operand1: 0,
      operand2: 0,
    }
  }
}
```

Добавим кнопки для вычисления. Пока это просто элементы на странице с соответствующим знаком математической операции.

```
<template>
  <div>
    <div class="display">
      <input v-model="operand1"/>
      <input v-model="operand2"/>
    </div>
    <div class="keyboard">
```

```

        <button>+</button>
        <button>-</button>
        <button>/</button>
        <button>*</button>
    </div>
</div>
</template>

<script>
export default {
  name: 'Calculator',
  data() {
    return {
      operand1: 0,
      operand2: 0,
    }
  }
}
</script>

```

Добавим еще одно значение - в наш data объект - поле result в котором будет храниться результат выполнения математического действий. Сначала добавим свойство в data, и потом добавим вывод его в шаблон:

```

<div class="display">
  <input v-model="operand1"/>
  <input v-model="operand2"/>
  = {{result}}
</div>

...

data() {
  return {
    operand1: 0,
    operand2: 0,
    result: 0
  }
},

```

А теперь реализуем 4 функции вычисления. Для этого воспользуемся директивой `v-on`. И будем подписываться на событие `click` по кнопке.

```
<button v-on:click="">+</button>
<button v-on:click="">-</button>
```

И тд.

Далее вспомним, что в выражениях в шаблоне мы можем обращаться к переменным из функции `data` и напишем выражения которые будут вычислять `result` в зависимости от того на какую кнопку мы кликнем.

```
<template>
  ...
  <div class="keyboard">
    <button v-on:click="result = operand1 + operand2">+</button>
    <button v-on:click="result = operand1 - operand2">-</button>
    <button @click="result = operand1 / operand2">/</button>
    <button @click="result = operand1 * operand2">*</button>
  </div>
</template>
```

## Шаблон компонента

В шаблоне компонента мы увидим обычную html разметку, указание других компонентов и вывод данных. Разберемся с самым простым - выводом данных. Сделать мы это можем с помощью `{{ }}` - это особый синтаксис в шаблоне Vue, с помощью которого мы можем сделать вывод данных из JavaScript кода компонента в шаблон. Внутри скобок мы описываем какое-то одно JavaScript выражение, которое Vue выполнит при отрисовке шаблона.

В этом выражении мы имеем прямой доступ к нашим данным. Поэтому `{{ message }}` меняется на содержимое свойства **message**.

**Выражения также поддерживают выполнение обычного JS кода внутри, например посмотрим на код в этом примере:**

```
<template>
  <div>
    <div>
```

```

Имя: {{name}}
Фамилия: {{lastname}}
<br>
<!-- сложение строк -->
Полное имя: {{name + ' ' + lastname}}
</div>
<div>
  <!-- арифметические операции -->
  2 + 2 = {{2 + 2}}
</div>
<div>
  {{message}}
  <!-- работа с массивами -->
  {{message.length}}
  {{message.split().reverse().join()}}
</div>
</div>
</template>

```

## ПРАКТИКА:

- Установить vue-cli
- Создать новое приложение vue@2.x с использованием vue-cli
- Повторить код калькулятора
- Также в качестве домашнего задания мы предлагаем расширить возможности нашего калькулятора который мы написали на уроке. Добавьте кнопки и возможность выполнения следующих действий:
  - Возведение в степень
  - Целочисленное деление

## Глоссарий

- SFC - Single File Vue Component,
- CLI - command-line interface
- Скаффолдинг, бутстрап проекта - создание каркаса, заготовки нового приложения
- Реактивность - отслеживание изменений свойств объекта с последующим изменением зависимостей данных свойств.

# Дополнительные материалы

1. Документация Vue CLI: [ссылка](#)
2. Статья на хабре “[Всё, что нужно для начала работы с Vue.js](#)”
3. Статья на medium “[Single-page application vs. multiple-page application](#)”

# Используемые источники

1. Официальный сайт Vue.js - [ссылка](#)
2. Документация Vue CLI - [ссылка](#)