

VUE.JS

POSTED BY SEBASTIAN — 5TH JANUAR '17

Vue.js 2 Quickstart Tutorial 2017

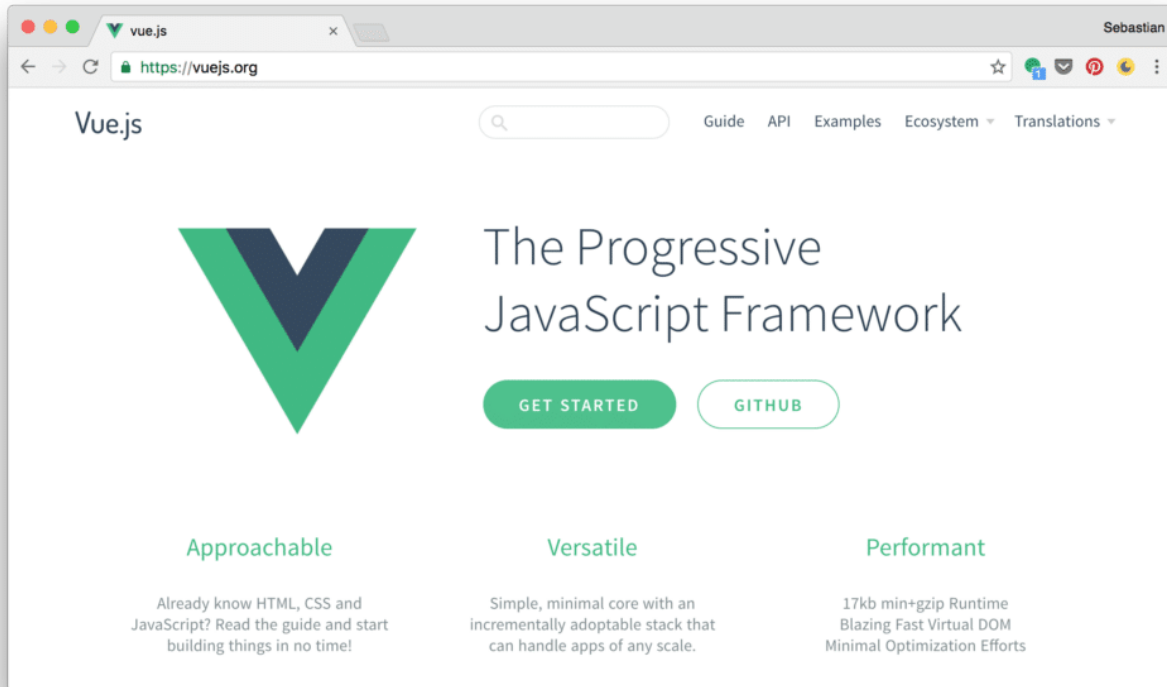
Vue is a progressive JavaScript framework that focuses on building user interfaces. As it only works in the “view layer” it makes no assumption of middleware and backend and therefore can be integrated easily into other projects and libraries. Vue.js offers a lot of

functionality for the view layer and can be used for building powerful single-page webapps. In the following you can find a list of features:

- Reactive Interfaces
- Declarative Rendering
- Data Binding
- Directives
- Template Logic
- Components
- Event Handling
- Computed Properties
- CSS Transitions and Animations
- Filters

The Vue.js 2 core library is very small in size (only 17 kB). This ensures that the overhead which is added to your project by using Vue.js is minimal and your website is loading fast.

The Vue.js website is available at: <https://vuejs.org/>



How to use Vue.js

There are different ways to include Vue.js in your web project:

- Use CDN by including `<script>` tag in HTML file
- Install using Node Package Manager (NPM)
- Install using Bower
- Use Vue-cli to setup your project

In the following we're going to the Vue-cli to setup a new project and install the Vue.js 2 library.

Using Vue-cli

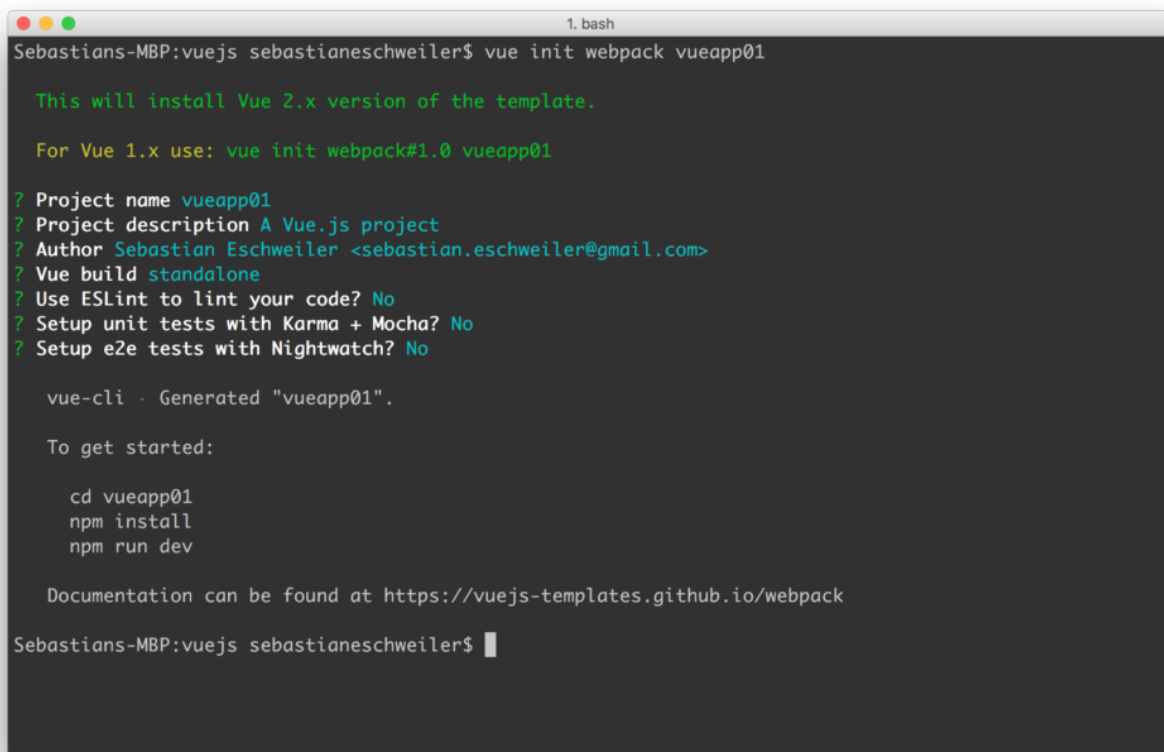
First, we need to install `Vue-cli`. The command line interface is available as an NPM package. Make sure that Node.js and the `npm` command is available on your system and use the following command to install the Vue CLI globally on your local system:

```
$ npm install -g vue-cli
```

Having installed the client successfully the `vue` command becomes available. Now we're able to initiate a project by using this command in the following way:

```
$ vue init webpack vueapp01
```

We're telling `vue` to initiate a new project and use the webpack template. We also give the project the name `vueapp01`. Executing the command brings up a few questions on the command line as you can see in the following screenshot:



```
Sebastians-MBP:vuejs sebastianeschweiler$ vue init webpack vueapp01

This will install Vue 2.x version of the template.

For Vue 1.x use: vue init webpack#1.0 vueapp01

? Project name vueapp01
? Project description A Vue.js project
? Author Sebastian Eschweiler <sebastian.eschweiler@gmail.com>
? Vue build standalone
? Use ESLint to lint your code? No
? Setup unit tests with Karma + Mocha? No
? Setup e2e tests with Nightwatch? No

vue-cli · Generated "vueapp01".

To get started:

  cd vueapp01
  npm install
  npm run dev

Documentation can be found at https://vuejs-templates.github.io/webpack
Sebastians-MBP:vuejs sebastianeschweiler$
```

The project is created in the folder `vueapp01`. Change into that directory with the following command:

```
$ cd vueapp01
```

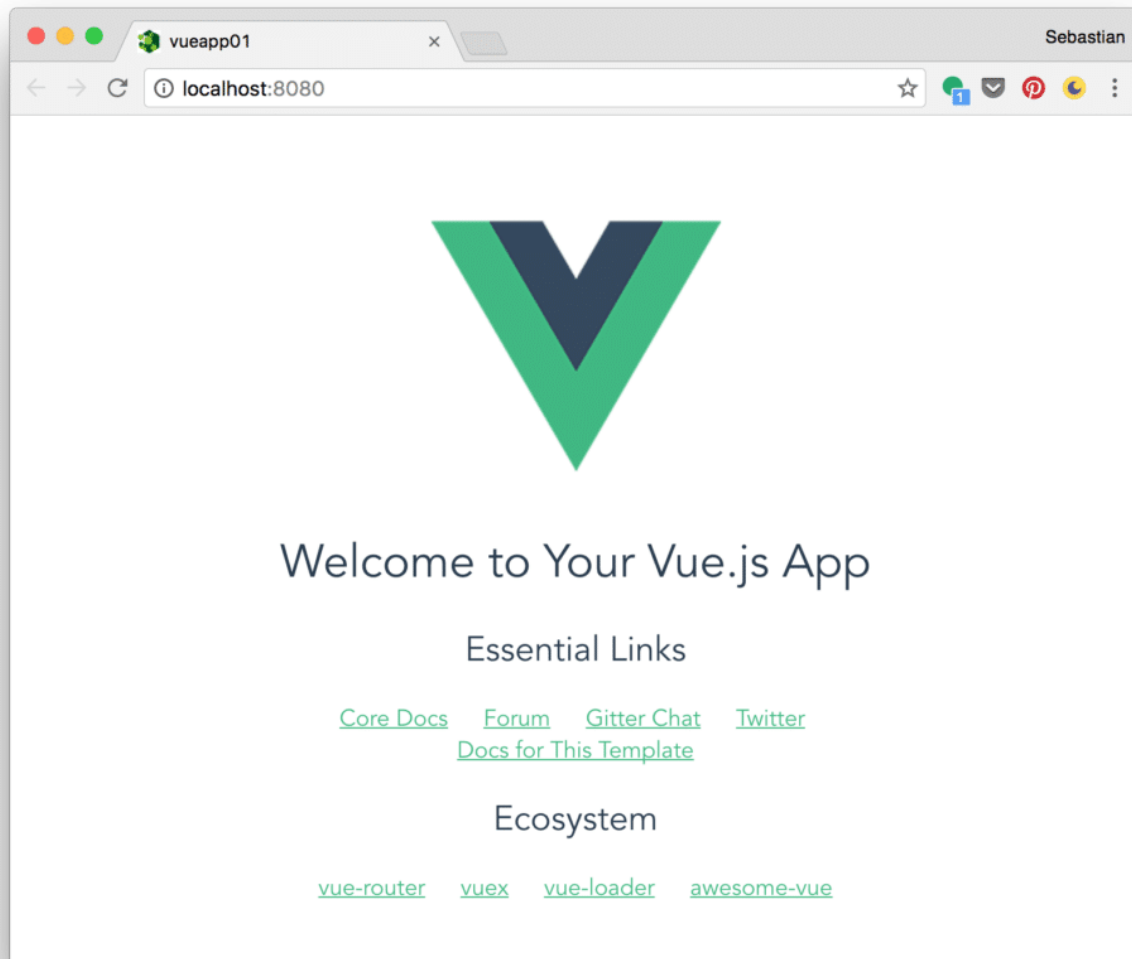
Start installing the dependencies by using *npm* again:

```
$ npm install
```

After having completed the installation of packages you can start the web server in development mode by using *npm* in the following way:

```
$ npm run dev
```

This will start the server on port 8080 and the application output is displayed in the browser automatically:

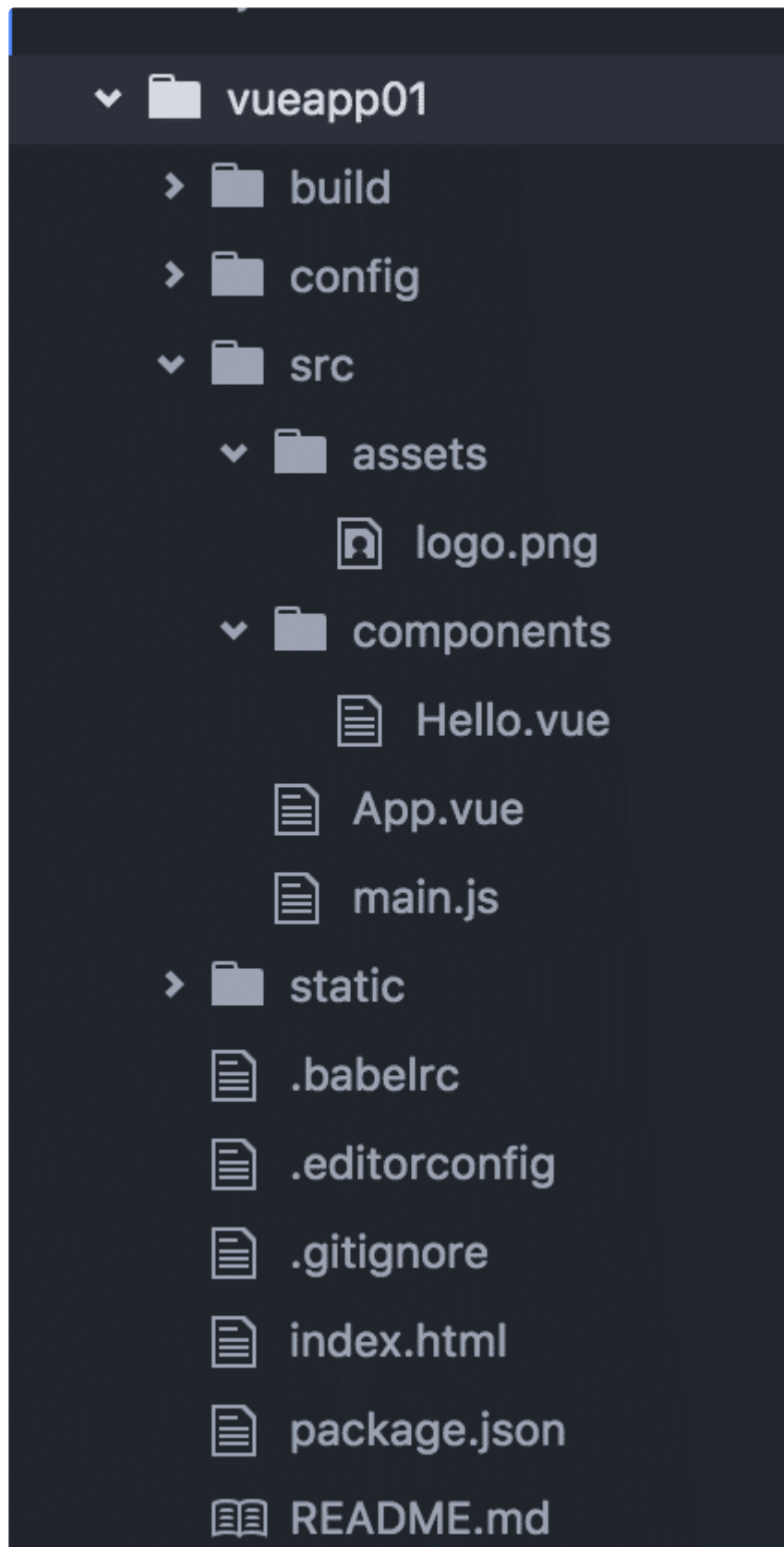


Later, if you want to build for production you can use the following command instead. In this case a `dist` folder is created containing the files needed for productive deployment.

```
$ npm run build
```

Project Structure

Let's take a look at the initial project structure which is available in folder *vueapp01*:



In the project root folder you can find files and folders. Let's examine the most important ones. The *package.json* file contains all the dependencies of your project. By using the

command `npm install` before we have made sure that the dependencies listed in *package.json* are installed into the *node_modules* folder of the project.

The file *index.html* contains the following HTML code:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>vueapp01</title>
  </head>
  <body>
    <div id="app"></div>
    <!-- built files will be auto injected -->
  </body>
</html>
```

This file is the starting point of your application. Note that within the *body* section a `<div>` element is available which has the *id* property set to string *app*. This element is used as a placeholder for the output which is generated by Vue.js.

Next take a look at file *main.js* in folder *src*. That's the place where the Vue application is initialized:

```
import Vue from 'vue'
import App from './App'

new Vue({
  el: '#app',
  template: '<App/>',
```



```
    components: { App }  
  })
```

On top of the file you can find two import statements:

- `import Vue from 'vue':` *Vue* is the main class of the framework
- `import App from './App':` *App* is the root component of our application

By using the *new* keyword a new instance of the main framework class *Vue* is created. The constructor takes an object as a parameter which contains three properties:

- `el`: By assigning the string *#app* to this property we're defining that the output of the Vue application should be rendered to the `<div id="app"></div>` element in *index.html*.
- `template`: The template contains the HTML code which is used to generate the output of the Vue.js application.
- `components`: List of Vue.js components which are used in the template.

The template only consists of one element: `<App/>`. Of course this is not a standard HTML element. This is the element which is assigned to App component. In order to be able to use `<App/>` in the template the App component is also listed in the object which is assigned to the *components* property.

So let's see what's inside the App component implementation in file *App.vue*:

```
<template>  
  <div id="app">  
      
    <hello></hello>  
  </div>
```

```
</template>

<script>
import Hello from './components/Hello'

export default {
  name: 'app',
  components: {
    Hello
  }
}
</script>

<style>
#app {
  font-family: 'Avenir', Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>
```

As in every Vue.js 2 single-file component the App implementation is split up into three parts:

- `<template></template>`: Component's template code
- `<script></script>`: Component's script code

- `<style></style>`: Component' CSS code

Let's focus on the the first two sections *template* and *script*. The script section is making a default export of an object declaring the component named *app*. Again, the *components* property is used to declare that another component (Hello) is required by App. This subcomponent is used in the template code of app and implemented in file *hello.vue* in folder *components*. In order to be able to use the *Hello* component in *App* it's also needed to include the corresponding import statement on top of the script section.

The implementation of component *Hello* looks like the following:

```
<template>
  <div class="hello">
    <h1>{{ msg }}</h1>
    <h2>Essential Links</h2>
    <ul>
      <li><a href="https://vuejs.org" target="_blank">Core Docs</a></li>
      <li><a href="https://forum.vuejs.org" target="_blank">Forum</a></li>
      <li><a href="https://gitter.im/vuejs/vue" target="_blank">Gitter</li>
      <li><a href="https://twitter.com/vuejs" target="_blank">Twitter</li>
      <br>
      <li><a href="http://vuejs-templates.github.io/webpack/" target="_blank">Webpack</li>
    </ul>
    <h2>Ecosystem</h2>
    <ul>
      <li><a href="http://router.vuejs.org/" target="_blank">vue-router</li>
      <li><a href="http://vuex.vuejs.org/" target="_blank">vuex</a></li>
      <li><a href="http://vue-loader.vuejs.org/" target="_blank">vue-loader</li>
      <li><a href="https://github.com/vuejs/awesome-vue" target="_blank">Awesome Vue</li>
    </ul>
  </div>
</template>
```

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

```
<script>
export default {
  name: 'hello',
  data () {
    return {
      msg: 'Welcome to Your Vue.js App'
    }
  }
}
</script>
```

```
<!-- Add "scoped" attribute to limit CSS to this component only -->
```

```
<style scoped>
```

```
h1, h2 {
  font-weight: normal;
}
```

```
ul {
  list-style-type: none;
  padding: 0;
}
```

```
li {
  display: inline-block;
  margin: 0 10px;
```

```
}
```

```
a {
```

```
  color: #42b983;
```

```
}
```

```
</style>
```



The component configuration object is exported as default. This time the component configuration object contains a *data* method. This method returns an object which represents the component's model. Properties defined in the model object can be used in the component's template by using the interpolation syntax. In the example from above the model object has only one property: *msg*. The string which is assigned to this property is included in the component's template by using:

```
<h1>{{ msg }}</h1>
```

The interpolation syntax required double curly braces to include model data in the template.

Using Standard Directives

Let's adapt the *Hello* component implementation to learn more about the usage of Vue.js standard directives.

v-for

The *v-for* directive makes it possible to render an element multiple times based on source data. You can use this directive to iterate over an array and at the array data to the output. First add an array to the object which is returned by the *data* method:

```
users: [  
  {firstname: 'Sebastian', lastname: 'Eschweiler'},  
  {firstname: 'Bill', lastname: 'Smith'},  
  {firstname: 'John', lastname: 'Porter'}  
],
```

Then use the *v-for* directive to include a list in the output printing out the *firstname* and *lastname* value of each array element:

```
<div>  
  <ul>  
    <li v-for="user in users">  
      {{user.firstname}} {{user.lastname}}  
    </li>  
  </ul>  
</div>
```

v-model

The *v-model* directive creates a two-way binding on an input element or a component. Make sure to define a property in your data object which should be used as the binding target:

```
input_val: ''
```

Then use the directive to bind the value of an input element to that property:

```
<div>  
  <input type="text" v-model="input_val">
```

```
</div>
```

With that binding established we're getting two effects:

- everytime the user enters a value in the input field the value of *input_val* is updated accordingly
- If we change the value of *input_val* in our program the value which is displayed in the input element is updated as well

v-text

By using the v-text directive the text content of an element is set. We can use it as an alternative to the `{{ ... }}` syntax if the complete text content should be set. E.g. we can use this directive to output the *input_val* value to the user:

Input Value: ``

Summary

The complete code of the adapted *Hello* component implementation should now look like the following:

```
<template>
  <div class="hello">
    <h1>{{ msg }}</h1>
    <hr />
    <div>
      <ul>
        <li v-for="user in users">
```


```
      {{user.firstname}} {{user.lastname}}
    </li>
  </ul>
</div>
<hr />
<div>
  <input type="text" v-model="input_val">
</div>
<div>
  Input Value: <span v-text="input_val"></span>
</div>
<hr />
<div>
  <button class="btn btn-primary" v-on:click="counter++">You've cli
</div>
</div>
</template>

<script>
export default {
  name: 'hello',
  data () {
    return {
      msg: 'Welcome to Your Vue.js App',
      users: [
        {firstname: 'Sebastian', lastname: 'Eschweiler'},
        {firstname: 'Bill', lastname: 'Smith'},
        {firstname: 'John', lastname: 'Porter'}
      ]
    }
  }
}
```



```
    ],  
    input_val: '',  
    counter: 0  
  }  
}  
}  
</script>
```

```
<!-- Add "scoped" attribute to limit CSS to this component only -->  
<style scoped>  
h1, h2 {  
  font-weight: normal;  
}  
ul {  
  list-style-position: inside;  
}  
a {  
  color: #42b983;  
}  
</style>
```



The result can be seen in the following screenshot:

ONLINE COURSE: Vue.js 2 - The Complete Guide



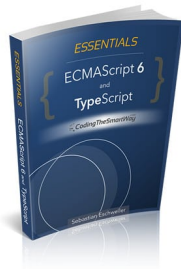
Check out the great **Vue.js online course** by **Maximilian Schwarzmüller** with thousands of students already enrolled:

Vue JS 2 – The Complete Guide

- Build amazing Vue.js Applications – all the Way from Small and Simple Ones up to Large Enterprise-level Ones

- Leverage Vue.js in both Multi- and Single-Page-Applications (MPAs and SPAs)
- Understand the Theory behind Vue.js and use it in Real Projects

[Go To Course](#)



Subscribe & Get Free eBook: Essentials - ECMAScript 6 and TypeScript

Subscribe to our newsletter to receive updates and tips about web development & get instant access to the free eBook. You can unsubscribe at any time from it.

Sign Up

We take your privacy seriously. See our [data privacy statement here](#).

[FRONTEND](#)

[QUICKSTART](#)

[TUTORIAL](#)

[VUE.JS](#)

[VUE.JS 2](#)

[WEB DEVELOPMENT](#)

 SHARE

 FACEBOOK

 GOOGLE+

 TWITTER

 LINKEDIN

PREVIOUS POST

ANGULAR

Top Online Courses For Frontend Developers In 2017

NEXT POST

VUE.JS

Vue.js 2 and Firebase



@s_eschweiler
Sebastian

Using and writing about best practices and latest technologies in web design & development is my passion.