Vue JS Document

Definition

VueJS is a progressive JavaScript framework used to develop interactive web interfaces. Focus is more on the view part, which is the front end. It is very easy to integrate with other projects and libraries. The installation of VueJS is fairly simple, and beginners can easily understand and start building their own user interfaces.

VueJS is created by **Evan You**, an ex-employee from **Google**.

Features

Virtual DOM

VueJS makes the use of **virtual DOM**, which is also used by other frameworks such as React, Ember, etc. The changes are not made to the DOM; instead **a replica of the DOM** is created which is present in the form of JavaScript data structures.

Data Binding

VueJS is **two way data binding** feature helps manipulate or assign values to HTML attributes, change the style, assign classes with the help of binding directive called **v-bind** available with VueJS.

Components

Components are one of the important features of VueJS that helps **create custom elements**, which can be reused in HTML.

Event Handling

v-on is the attribute added to the DOM elements to **listen to the events** in VueJS.

Animation/Transition

VueJS provides various ways to apply transition to HTML elements when they are added/ updated or removed from the DOM. We can easily add third party **animation libraries** and also add more interactivity to the interface.

Computed Properties

It helps to listen to the changes made to the UI elements and performs the **necessary calculations**.

Templates

VueJS provides HTML-based **templates that bind the DOM** with the Vue instance data. Vue compiles the templates into virtual DOM Render functions.

Directives

VueJS has built-in directives such as v-if, v-else, v-show, v-on, v-bind, and v-model, which are used to perform various actions on the frontend.

Watchers

Watchers are applied to data that changes. It does not create any new property, but it watches the changes over a reactive property. Have arguments of new and old value.

Routing

Navigation between pages is performed with the help of vue-router.

Lightweight

VueJS script is very **lightweight** and the performance is also **very fast**.

Vue-CLI

VueJS can be installed at the command line using the **vue-cli command line interface.** It helps to **build** and **compile** the project easily using vue-cli.

Comparison with Other Frameworks

VueJS v/s React

- Template vs JSX
  - VueJS uses html, js and css separately. It is very easy for a beginner. React uses jsx approach. Everything is JavaScript for ReactJS. HTML and CSS are all part of JavaScript.
- Popularity
  - VueJS has taken the good parts from Angular and React and has built a powerful library. VueJS is much faster in comparison to React/Angular because of its lightweight library.

VueJS v/s Angular

- Complexity
  - Vuejs is very easy to learn and start with. For Angular, difficult for beginners to get started with Angular. It uses TypeScript for coding which is difficult for people coming from core JavaScript background.
- Performance
  - VueJS file size is much lighter than Angular. A comparison of the framework performance is provided in the following link. http://stefankrause.net/js-frameworks-benchmark4/webdriver-ts/table.html
- Flexibility
  - VueJS can be easily merged with any other big project without any issues. Angular will not be that easy to start working with any other existing project.

VueJS v/s Ember

- Performance
  - VueJS has better performance in comparison to Ember. Ember has added a glimmer rendering engine with the aim of improving the re-render performance, which is a similar concept as VueJS and React using virtual DOM.

VueJS - Environment Setup

There are many ways to install VueJS.

Using the <script> tag directly in HTML file

Simply download and include with a script tag.

There are two versions for use.

- Production version (is minimized)
- Development version. (version is not minimized 30.90KB min+gzip)

Using CDN

```
<script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js">
</script>
```

The link https://unpkg.com/vue will give the latest version of VueJS.

Also CDN JS like https://cdnjs.cloudflare.com/ajax/libs/vue/2.4.0/vue.js

Using NPM

For large scale applications with VueJS, it is recommended to install using the npm package. It comes with Browserify and Webpack along with other necessary tools, Following is the command to install using npm.

```
npm  install vue
```

Using CLI Command Line

VueJS also provides CLI to install the vue and get started. –g use for install globally.

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

Following is the command to create the project using Webpack.

```
vue init webpack [projectName]
```

After Project Create get started, use the following command.

```
cd [projectName]
npm install
npm run dev
```

VueJS - Basic

Vue is a JavaScript framework for building user interfaces. Its core part is focused mainly on the view layer.

Below is simple example of VueJS.

```
<html>
   <head>
      <title>VueJs Introduction</title>
      <script type = "text/javascript" src = "js/vue.js"></script>
   </head>
   <body>
      <div id = "app" style = "text-align:center;">
         <h1>{{ message }}</h1>
      </div>
      <script type = "text/javascript">
         var vue_det = new Vue({
            el: '#app,
            data: {
               message: 'Welcome to VueJS'
            }
         });
      </script>
   </body>
</html>
```

Firstly, we are creating an instance of VueJS. This is called the root Vue Instance. There is a parameter called **el**. It takes the id of the DOM element. Ex: - "#app".

VueJS interacts with DOM and changes the value in the DOM {{message}}.

Another Example with data and method

HTML

```
<div id = "vue_det">

   <h1>Firstname : {{firstname}}</h1>

   <h1>Lastname : {{lastname}}</h1>

   <h1>{{mydetails()}}</h1>

</div>
```

Vue Instance

```
var  vm = new Vue({

   el: '#vue_det',

   data: {

      firstname : "Even",

      lastname  : "You",

      address    : "Japan"

   },

   methods: {

      mydetails : function() {

         return "I am "+this.firstname +" "+ this.lastname;

      }

   }

})
```

Data: This type of data can be an object or a function. Vue converts its properties to getters/setters to make it reactive.

Methods: Method block having list of method for calculation or operation all are JS function.

Understanding Vue.js Lifecycle Hooks

Each Vue instance goes through a series of initialization steps when it's created

It needs to set up data observation, compile the template, mount the instance to the DOM, and update the DOM when data changes. Along the way, it also runs functions called lifecycle hooks.

1). Creation

Creation hooks are the very first hooks that run in your component. They allow you to perform actions before your component has even been added to the DOM.

2). beforeCreate

The beforeCreatehook runs at the very initialization of your component. data has not been made reactive, and events have not been set up yet.

```
<script>
  export default {
     beforeCreate() {
        console.log('Nothing gets called before me!')
     }
  }
</script>
```

3). created

You can access all data and events.

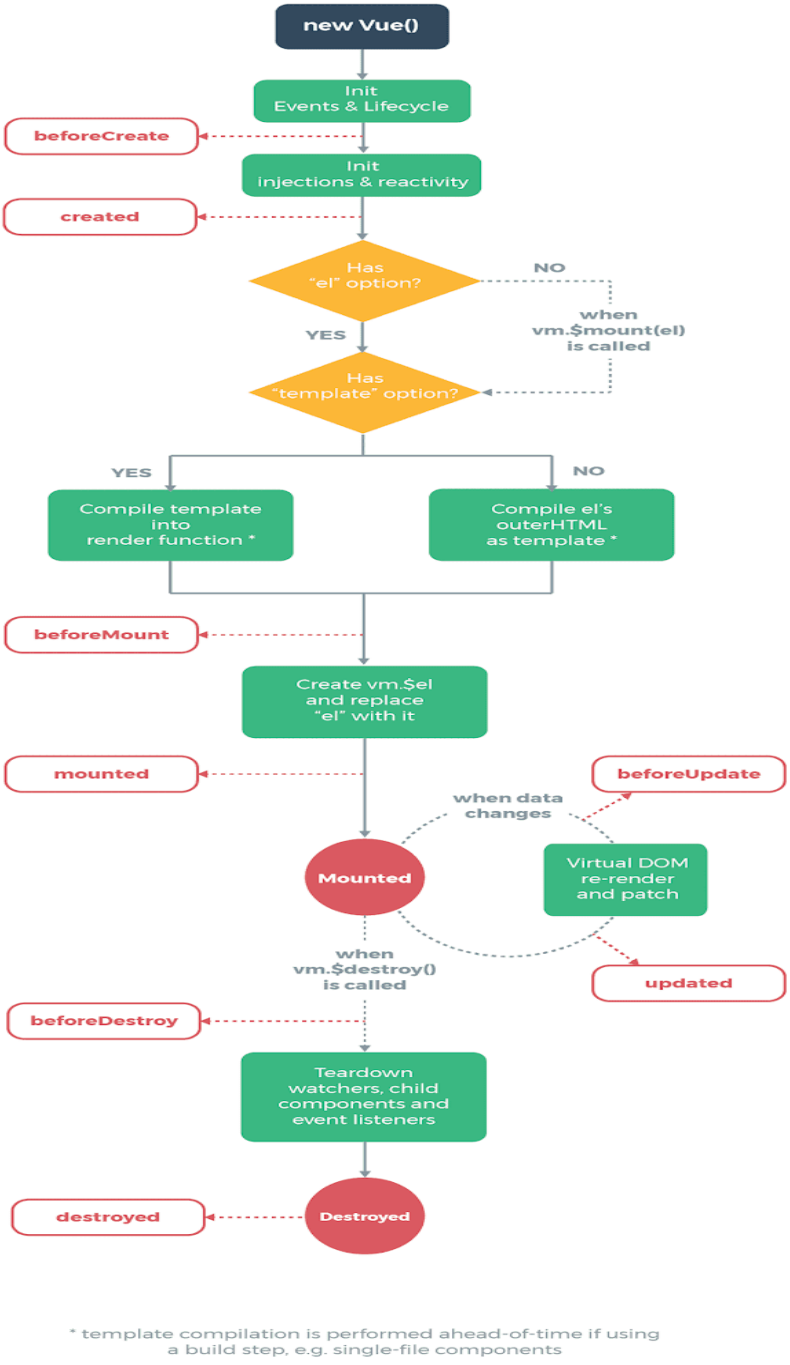
4). Mounting (DOM Insertion)

Mounting hooks are often the most-used hooks, for better or worse. They allow you to access your component immediately before and after the first render. They do not, however, run during server-side rendering.

beforeMount(): The beforeMount() method is invoked after our template has been compiled and our virtual DOM updated by Vue.

mounted(): This hook, you will have full access to the reactive component, templates, and rendered DOM (via. this.$el). Mounted is the most-often used lifecycle hook. The most frequently used patterns are fetching data for your component (use created for this instead,) and modifying the DOM, often to integrate non-Vue libraries.

Below is life cycle of vue.js which is given in vue.js offical site.



VueJS - Template

Vue.js uses an HTML-based template syntax that allows you to declaratively bind the rendered DOM to the underlying Vue instance's data.

```
<p>Using Normal display: {{ rawHtml }}</p>

<p>Using v-html directive: <span v-html="rawHtml"></span></p>
```

Output

```
Using Normal display: <span style="color:red">This should be red.</span>

Using v-html directive: This should be red.
```

Mustaches {{}} cannot be used inside HTML attributes. Instead, use a **v-bind directive**:

```
<div v-bind:id="dynamicId"></div>

<button v-bind:disabled="isButtonDisabled">Button</button>
```

VueJS - Directives

Directives are special attributes with the **v- prefix**. Directive attribute values are expected to be a single JavaScript expression. A directive's job is to reactively apply side effects to the DOM when the value of its expression changes.

We have known list of directives such as v-if, v-show, v-else, v-for, v-bind , v-model, v-on, etc.

```
<p v-if="seen">Now you see me</p>

<a v-bind:href="url"> ... </a>

<a v-on:click="doSomething"> ... </a>
```

Modifitiers

Modifiers are special postfixes denoted by a **dot**, which indicate that a directive should be bound in some special way. For example, the .prevent modifier tells the **v-on** directive to call **event.preventDefault()** on the triggered event:

```
<form v-on:submit.prevent="onSubmit"> ... </form>
```

Short-hands

Vue.js provides special shorthands for two of the most often used directives, **v-bind** and **v-on**:

v-bind Shorthand

```
<!-- full syntax -->
<a v-bind:href="url"> ... </a>
<!-- shorthand -->
<a :href="url"> ... </a>
```

v-on Shorthand

```
<!-- full syntax -->
<a v-on:click="doSomething"> ... </a>
<!-- shorthand -->
<a @click="doSomething"> ... </a>
```

VueJS – Computed Properties and Watchers

Computed Properties automatically invoke when we changed any property used inside Computed block.

Computed Properties

HTML

```
      <div id = "computed_props">

         FirstName : <input type = "text" v-model = "firstname" /> <br/><br/>

         LastName : <input type = "text" v-model = "lastname"/> <br/><br/>
```

```
        <h1>My name is {{firstname}} {{lastname}}</h1>
        <h1>Using computed method : {{getfullname}}</h1>
    </div>
```

Vue Code

```
var vm = new Vue({
   el: '#computed_props',
   data: {
      firstname :"",
      lastname :""
      },
   computed :{
      getfullname : function(){
         return this.firstname +" "+ this.lastname;
      }
   }
})
```

When the properties firstname or lastname is changed. we don't have to do anything, with computed it gets called by itself automatically and update fullname.

Watchers

Computed properties are more appropriate in most cases, But there are some times when a custom watcher is necessary. That's why Vue provides a more generic way to react to data changes through the **watch** option. This is most useful when you want to perform asynchronous or expensive operations in response to changing data.

```
<div id = "computed_props">
        Kilometers : <input type = "text" v-model = "kilometers">
        Meters : <input type = "text" v-model = "meters">
    </div>
```

```
<script type = "text/javascript">
   var vm = new Vue({
      el: '#computed_props',
      data: {
         kilometers : 0,
         meters:0
      },
      watch : {
         kilometers:function(val) {
            this.kilometers = val;
            this.meters = val * 1000;
         },
         meters : function (val) {
            this.kilometers = val/ 1000;
            this.meters = val;
         }
      }
   });
</script>
```

Let's enter some values in the kilometers textbox and see it changing in the meters textbox and vice-versa.

VueJS – Binding

Vue provides special enhancements when v-bind is used with **class** and **style.**

There are different way assign classes and style, as per below.

```
1). <div v-bind:class="{ active: true }"></div>
2). <div v-bind:class="classObject"></div>
3). <div v-bind:class="[activeClass, errorClass]"></div>
```

```
4). <div v-bind:style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>

5). <div v-bind:style="styleObject"></div>

6). <div v-bind:class = "[isActive ? infoclass : '', haserror ? errorclass :
'']">{{title}}</div>
```

VueJS – Form Input Binding

You can use the **v-model** directive to create two-way data bindings.

Textbox binding

```
<input v-model="message" placeholder="edit me">

<p>Message is: {{ message }}</p>
```

Multiple checkboxes, bound to the same Array:

```
<div id='example-3'>
  <input type="checkbox" id="jack" value="Jack" v-model="checkedNames">
  <label for="jack">Jack</label>
  <input type="checkbox" id="john" value="John" v-model="checkedNames">
  <label for="john">John</label>
  <input type="checkbox" id="mike" value="Mike" v-model="checkedNames">
  <label for="mike">Mike</label>
  <br>
  <span>Checked names: {{ checkedNames }}</span>
</div>

new Vue({
  el: '#example-3',
  data: {
    checkedNames: []
  }
})
```

For Radio button

```
<input type="radio" id="one" value="One" v-model="picked">
<label for="one">One</label>
<br>
<input type="radio" id="two" value="Two" v-model="picked">
<label for="two">Two</label>
<br>
<span>Picked: {{ picked }}</span>
```

Select

```
<select v-model="selected">
  <option disabled value="">Please select one</option>
  <option>A</option>
  <option>B</option>
  <option>C</option>
</select>
<span>Selected: {{ selected }}</span>
```

VueJS – Condition Rending

There are few directives use for condition rending like **v-if, v-else, v-else-if, v-show**

```
<div v-if="type === 'A'">
  A
</div>
<div v-else-if="type === 'B'">
  B
</div>
<div v-else-if="type === 'C'">
  C
</div>
<div v-else>
  Not A/B/C
</div>
<h1 v-show="ok">Hello!</h1>
```

VueJS – List Rendering

We can use the **v-for** directive to render a list of items based on an array.

```
<ul id="example-1">
  <li v-for="item in items">
    {{ item.message }}
  </li>
</ul>

var example1 = new Vue({
  el: '#example-1',
  data: {
    items: [
      { message: 'Foo' },
      { message: 'Bar' }
    ]
  }
})
```

There are few property we can used in for like. **value, key, index**

```
<div v-for="(value, key, index) in object">

  {{ index }}. {{ key }}: {{ value }}

</div>
```

Vue wraps an observed array's mutation methods so they will also trigger view updates. The lists of methods are:

- push()
- pop()
- shift()
- unshift()
- splice()
- sort()
- reverse()

There are also non-mutating methods, e.g**. filter(), concat() and slice(),** which do not mutate the original array but **always return a new array**.

```
</div> example1.items = example1.items.filter(function (item) {

  return item.message.match(/Vue/)

})
```

VueJS – Event Handing

v-on is the attribute added to the DOM elements to listen to the events in VueJS.

```
<div id="example-1">

  <button v-on:click="counter += 1">Add 1</button>

  <p>The button above has been clicked {{ counter }} times.</p>

</div>

var example1 = new Vue({
  el: '#example-1',
  data: {
    counter: 0
  }
})
```

Event Modifiers

Vue provides **event modifiers** for **v-on**. Recall that modifiers are directive postfixes denoted by a dot.

- .stop
- .prevent
- .capture
- .self
- .once
- .passive

```
<!-- the click event's propagation will be stopped -->

<a v-on:click.stop="doThis"></a>



<!-- the submit event will no longer reload the page -->

<form v-on:submit.prevent="onSubmit"></form>



<!-- modifiers can be chained -->

<a v-on:click.stop.prevent="doThat"></a>



<!-- just the modifier -->
```

```
<form v-on:submit.prevent></form>



<!-- use capture mode when adding the event listener -->

<!-- i.e. an event targeting an inner element is handled here before being handled
by that element -->

<div v-on:click.capture="doThis">...</div>



<!-- only trigger handler if event.target is the element itself -->

<!-- i.e. not from a child element -->

<div v-on:click.self="doThat">...</div>
```

Key Modifiers

Here's the full list of key modifier aliases:

- .enter
- .tab
- .delete (captures both "Delete" and "Backspace" keys)
- .esc
- .space
- .up
- .down
- .left
- .right

```
<input v-on:keyup.enter="submit">

<input v-on:keyup.13="submit">
```

VueJS – Components Basics

Vue Components are one of the important features of VueJS that creates custom elements, which can be reused in HTML.

```
<body id = "component_test">
      <div>
         <testcomponent></testcomponent>
      </div>
      <div>
         <testcomponent></testcomponent>
      </div>
   </body>
```

Globally Component

We have Created **TestComponent** which is use multiple times in body.

```
Vue.component('testcomponent',{
   template : '<div><h1>This is coming from component</h1></div>'
});
var vm = new Vue({
   el: '#component_test'
});
```

To create a component, following is the syntax.

```
Vue.component('nameofthecomponent',{ option });
```

You might have components for a header, sidebar, and content area, each typically containing other components for navigation links, blog posts, etc.

To use these components in templates, they must be registered so that Vue knows about them. There are **two types** of component registration: global and local.

So far, we've only registered components **globally**, using **Vue.component.**

Local Component

```
var ComponentA = { /* ... */ }

var ComponentB = { /* ... */ }
```

Then define the components you'd like to use in a **components** option:

```
new Vue({
  el: '#app'
  components: {
    'component-a': ComponentA,
    'component-b': ComponentB
  }
})
```

Note that **locally registered components are** *not* **also available in subcomponents**.

If we are using Babel and Webpack, that might look more like:

```
import ComponentA from './ComponentA.vue'

export default {
  components: {
    ComponentA
  }
}
```

Passing Data to Child Components with Props

Props are custom attributes you can register on a component. When a value is passed to a prop attribute, it becomes a property on that component instance.

```
Vue.component('blog-post', {
  props: ['title'],
  template: '<h3>{{ title }}</h3>'
})
```

Here **title** is property which is used in component, once a prop is registered, you can pass data to it as a custom attribute.

```
<blog-post title="My journey with Vue"></blog-post>

<blog-post title="Blogging with Vue"></blog-post>
```

```
<blog-post title="Why Vue is so fun"></blog-post>
```

Note:- In **Prop** we can pass Number, Boolean, Array, Object etc. We can set type of Property which is passing in component.

Dynamic Components

Sometimes, we required to dynamically switch between components. Below is example here.

VueJS Components Registration

Component Name

You have two options when defining component names:

With kebab-case

```
Vue.component('my-component-name', { /* ... */ })
```

With PascalCase

```
Vue.component('MyComponentName', { /* ... */ })
```

Slots

To allow a parent component to pass DOM elements into a child component, provide a <slot></slot> element inside the child component. You can then populate the elements in that slot.
Slots basically used for creating Generic components (button, Card, Modal), layout (App, Header, Footer), Recursive component (Tree, Menu).

For Example JS Fiddler Here
We have create one **component** app-child. And we add slot tag and fixed content which always display. But slot content is dynamically what we give it will display.

```
<div id="app">
  <h2>We can use slots to populate content</h2>
  <app-child>
    <h3>This is slot number one</h3>
  </app-child>
  <app-child>
    <h3>This is slot number two</h3>
    <small>I can put more info in, too!</small>
  </app-child>
</div>


<script type="text/x-template" id="childarea">
  <div class="child">
    <slot></slot>
    <p>It's a veritable slot machine!</p>
  </div>
</script>
```

```
const Child = {
  template: '#childarea'
};
new Vue({
  el: '#app',
  components: {
    appChild: Child
  }
});
```

Output will be

```
This is slot number one

It's a veritable slot machine!


This is slot number two

I can put more info in, too!

It's a veritable slot machine!
```

You can also have named slots. If you were to have two slots in a component, you could differentiate between them by adding a name attribute <slot name="header"></slot>

Vus JS – Animation and Transition

VueJS provides various ways to apply transition to the HTML elements when they are added/updated in the DOM.

Using Transition tag we do animation in our application. Here example of Transition.

There are various list of classes are available in VueJs.

v-enter: Starting state for enter. Added before element is inserted, removed one frame after element is inserted.
v-enter-active: Active state for enter. Applied during the entire entering phase. Added before element is inserted, removed when transition/animation finishes.
v-enter-to: Ending state for enter. Added one frame after element is inserted (at the same time v-enter is removed), removed when transition/animation finishes.
v-leave: Starting state for leave. Added immediately when a leaving transition is triggered, removed after one frame.
v-leave-active: Active state for leave. Applied during the entire leaving phase. Added immediately when leave transition is triggered, removed when the transition/animation finishes.

v-leave-to: Ending state for leave. Added one frame after a leaving transition is triggered (at the same time v-leave is removed), removed when the transition/animation finishes.

Custom Transition Classes

You can also specify custom transition classes by providing the following attributes:
enter-class
enter-active-class
enter-to-class (2.1.8+)
leave-class
leave-active-class
leave-to-class (2.1.8+)
There are few Examples here... Example 1 Example 2

Explicit Transition Durations

We can also specify separate values for enter and leave durations:

```
<transition :duration="{ enter: 500, leave: 800 }">...</transition>
```

List Move Transition

The <transition-group> component has another trick up its sleeve. It can not only animate entering and leaving, but also changes in position. The only new concept you need to know to use this feature is the addition of the v-move class.

Here is example of Transition Group