VueJS

VueJS is a progressive JavaScript framework used to develop interactive web interfaces.

DOM is an interface that allows the script to update the content, style, and structure of the document. Virtual DOM is just like a blueprint of a machine, can do the changes in the blueprint but those changes will not directly apply to the machine.

Any time a change needs to be made to the DOM, a new JavaScript object is created and the changes are made. Later, both the JavaScript objects are compared and the final changes are updated in the real DOM.

VueJS and React both use virtual DOM, which makes it faster. React is popular than VueJS. The job opportunity with React is more than VueJS. There is a big name behind React i.e. Facebook which makes it more popular. VueJS is much faster in comparison to React/Angular because of its lightweight library. VueJS does not have all the built-in features as Angular and needs to depend on third party libraries to work on it.

Using the <script> tag directly in HTML file

```
<html>
  <head>
    <script type = "text/javascript" src = "vue.min.js"></script>
  </head>
  <body></body>
</html>
```

Installation

npm install vue

npm install --global vue-cli

Create the project using Webpack.

vue init webpack myproject

To get started, use the following command.

cd myproject

npm install

npm run dev

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

To get the value of the message in the DOM, we are creating an instance of vuejs as follows –

```
var vue_det = new Vue({
  el: '#intro',
  data: {
    message: 'My first VueJS Task'
  }
})
```

In the above code snippet, we are calling Vue instance, which takes the id of the DOM element i.e. e1:'#intro', it is the id of the div. There is data with the message which is assigned the value 'My first VueJS Task'.

VueJS - Instances

To start with VueJS, we need to create the instance of Vue, which is called the root Vue Instance.

Syntax

```
var app = new Vue({
  // options
})
```

Let us look at an example to understand what needs to be part of the Vue constructor.

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "vue_det">
```

```
        <h1>Firstname : {{firstname}}</h1>
        <h1>Lastname : {{lastname}}</h1>
        <h1>{{mydetails()}}</h1>
      </div>
      <script type = "text/javascript" src = "js/vue_instance.js"></script>
    </body>
</html>
```

vue_instance.js

```
var  vm = new Vue({
  el: '#vue_det',
  data: {
    firstname : "Ria",
    lastname  : "Singh",
    address    : "Mumbai"
  },
  methods: {
    mydetails : function() {
      return "I am "+this.firstname +" "+ this.lastname;
    }
  }
})
```

For Vue, there is a parameter called el. It takes the id of the DOM element. In the above example, we have the id #vue_det. It is the id of the div element, which is present in .html.

```
<div id = "vue_det"></div>
```

Now, whatever we are going to do will affect the div element and nothing outside it.

VueJS - Template

How to get an output in the form of HTML template on the screen.

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "vue_det">
      <h1>Firstname : {{firstname}}</h1>
      <h1>Lastname : {{lastname}}</h1>
      <div>{{htmlcontent}}</div>
    </div>
    <script type = "text/javascript" src = "js/vue_template.js"></script>
  </body>
</html>
```

vue_template.js

```
var vm = new Vue({
  el: '#vue_det',
  data: {
    firstname : "Ria",
```

```
    lastname  : "Singh",
    htmlcontent : "<div><h1>Vue Js Template</h1></div>"
  }
})
```

If we see the html content is displayed the same way we have given in the variable htmlcontent, this is not what we want, we want it to be displayed in a proper HTML content on the browser.

For this, we will have to use v-html directive. The moment we assign v-html directive to the html element, VueJS knows that it has to output it as HTML content. Let's add v-html directive in the .html file and see the difference.

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "vue_det">
      <h1>Firstname : {{firstname}}</h1>
```

```
      <h1>Lastname : {{lastname}}</h1>
      <div v-html = "htmlcontent"></div>
    </div>
    <script type = "text/javascript" src = "js/vue_template.js"></script>
  </body>
</html>
```

Now, we don't need the double curly brackets to show the HTML content, instead we have used v-html = "htmlcontent" where htmlcontent is defined inside the js file as follows −

```
var vm = new Vue({
  el: '#vue_det',
  data: {
    firstname : "Ria",
    lastname  : "Singh",
    htmlcontent : "<div><h1>Vue Js Template</h1></div>"
  }
})
```

The output in the browser is as follows −

Computed properties using an example.

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <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>
    <script type = "text/javascript" src = "js/vue_computedprops.js"></script>
  </body>
</html>
```

vue_computeprops.js

```
var vm = new Vue({
  el: '#computed_props',
  data: {
    firstname :"",
    lastname :"",
    birthyear : ""
  },
```

```
  computed :{
    getfullname : function(){
      return this.firstname +" "+ this.lastname;
    }
  }
})
```

VueJS - Events

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

Click Event

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "databinding">
      <button v-on:click = "displaynumbers">Click ME</button>
      <h2> Add Number 100 + 200 = {{total}}</h2>
    </div>
    <script type = "text/javascript">
      var vm = new Vue({
        el: '#databinding',
        data: {
          num1: 100,
          num2 : 200,
```

```
        total : ''
      },
      methods : {
        displaynumbers : function(event) {
          console.log(event);
          return this.total =  this.num1+ this.num2;
        }
      },
    });
  </script>
 </body>
</html>
```

The following code is used to assign a click event for the DOM element.

```
<button v-on:click = "displaynumbers">Click ME</button>
```

There is a shorthand for v-on, which means we can also call the event as follows −

```
<button @click = "displaynumbers">Click ME</button>
```

Conditional Rendering

In conditional rendering, we will discuss about using if, if-else, if-else-if, show, etc.

v-if

```
<html>
 <head>
  <title>VueJs Instance</title>
  <script type = "text/javascript" src = "js/vue.js"></script>
 </head>
```

```
  <body>
    <div id = "databinding">
      <button v-on:click = "showdata" v-bind:style = "styleobj">Click Me</button>
      <span style = "font-size:25px;"><b>{{show}}</b></span>
      <h1 v-if = "show">This is h1 tag</h1>
      <h2>This is h2 tag</h2>
    </div>
    <script type = "text/javascript">
      var vm = new Vue({
        el: '#databinding',
        data: {
          show: true,
          styleobj: {
            backgroundColor: '#2196F3!important',
            cursor: 'pointer',
            padding: '8px 16px',
            verticalAlign: 'middle',
          }
        },
        methods : {
          showdata : function() {
            this.show = !this.show;
          }
        },
      });
    </script>
  </body>
</html>
```

We have assigned if to the h1 tag as shown in the following code snippet.

```
<button v-on:click = "showdata" v-bind:style = "styleobj">Click Me</button>

<h1 v-if = "show">This is h1 tag</h1>
```

Now what it will do is, it will check the value of the variable show and if its true the h1 tag will be displayed. Click the button and view in the browser, as the value of the show variable changes to false, the h1 tag is not displayed in the browser. It is displayed only when the show variable is true.

```
 <h2 v-else>This is h2 tag</h2>
```

List Rendering

v-for

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
  </head>
  <body>
    <div id = "databinding">
      <input type = "text" v-on:keyup.enter = "showinputvalue"
        v-bind:style = "styleobj" placeholder = "Enter Fruits Names"/>
      <h1 v-if = "items.length>0">Display Fruits Name</h1>
      <ul>
        <li v-for = "a in items">{{a}}</li>
      </ul>
    </div>
    <script type = "text/javascript">
      var vm = new Vue({
```

```
        el: '#databinding',
        data: {
          items:[],
          styleobj: {
            width: "30%",
            padding: "12px 20px",
            margin: "8px 0",
            boxSizing: "border-box"
          }
        },
        methods : {
          showinputvalue : function(event) {
            this.items.push(event.target.value);
          }
        },
      });
    </script>
  </body>
</html>
```

VueJS - Routing

VueJS does not have a built-in router feauture. We need to follow some additional steps to install it.

npm install vue-router

```
<html>
  <head>
    <title>VueJs Instance</title>
    <script type = "text/javascript" src = "js/vue.js"></script>
```

```
    <script type = "text/javascript" src = "js/vue-router.js"></script>
  </head>
  <body>
    <div id = "app">
      <h1>Routing Example</h1>
      <p>
        <router-link to = "/route1">Router Link 1</router-link>
        <router-link to = "/route2">Router Link 2</router-link>
      </p>
      <!-- route outlet -->
      <!-- component matched by the route will render here -->
      <router-view></router-view>
    </div>
    <script type = "text/javascript">
      const Route1 = { template: '<div style = "border-radius:20px;background-
color:cyan;width:200px;height:50px;margin:10px;font-size:25px;padding:10px;">This is
router 1</div>' }
      const Route2 = { template: '<div style = "border-radius:20px;background-
color:green;width:200px;height:50px;margin:10px;font-size:25px;padding:10px;">This is
router 2</div>' }
      const routes = [
        { path: '/route1', component: Route1 },
        { path: '/route2', component: Route2 }
      ];
      const router = new VueRouter({
        routes // short for `routes: routes`
      });
      var vm = new Vue({
        el: '#app',
        router
```

```
    });
  </script>
 </body>
</html>
```

VueJS - Reactive Interface

VueJS provides options to add reactivity to properties, which are added dynamically. Consider that we have already created vue instance and need to add the watch property. It can be done as follows –

Example

```
<html>
 <head>
  <title>VueJs Instance</title>
  <script type = "text/javascript" src = "js/vue.js"></script>
 </head>
 <body>
  <div id = "app">
   <p style = "font-size:25px;">Counter: {{ counter }}</p>
   <button @click = "counter++" style = "font-size:25px;">Click Me</button>
  </div>
  <script type = "text/javascript">
   var vm = new Vue({
    el: '#app',
    data: {
     counter: 1
    }
   });
   vm.$watch('counter', function(nval, oval) {
    alert('Counter is incremented :' + oval + ' to ' + nval + '!');
```

```
      });
      setTimeout(
        function(){
          vm.counter = 20;
        },2000
      );
    </script>
  </body>
</html>
```

There is a property counter defined as 1 in data object. The counter is incremented when we click the button.

Vue instance is already created. To add watch to it, we need to do it as follows −

```
vm.$watch('counter', function(nval, oval) {
  alert('Counter is incremented :' + oval + ' to ' + nval + '!');
});
```

We need to use $watch to add watch outside the vue instance. There is an alert added, which shows the value change for the counter property. There is also a timer function added, i.e. setTimeout, which sets the counter value to 20.

```
setTimeout(
  function(){
    vm.counter = 20;
  },2000
);
```