VUE

Step-By-Step Guide To Mastering Vue.js From Beginner To Advanced

LIONEL LOPEZ

Vue asdasd

Step-By-Step Guide To Mastering Vue.js From Beginner To Advanced

Lionel Lopez

TABLE OF CONTENTS

Things to know before learning Vue JS

Before you getting started with Vue JS. You need following things:

- Basic knowledge of JavaScript:
    - If Statements.
    - Classes.
    - JavaScript Important techniques.
    - Web API's Experience (More info here).
    - Arrays
        - Methods
    - Strings
        - Methods
    - Regular Expression (Not important)

If you know these things and you know how to use these features of JavaScript then you are good to go.

Introduction

This book is about Vue JS, A book written for Vue JS tutorial. VUE JS is based on JavaScript similar to React JS. Let me tell you what is JavaScript? JavaScript is a web based programming language and was created by Brendan Eich, co-founder of Mozilla project. Along with HTML and CSS, JavaScript is the core of World Wide Web.

This book teaches you Vue JS using examples and Quizzes, from basic to advanced.

Chapter 1

Vue JS Introduction

Vue JS is pronounced as view. Vue JS is a progressive framework for building user interfaces. Vue JS is an approachable, versatile and small framework. If you know about HTML, CSS & JavaScript then you are ready to develop Vue JS apps in no time. Vue JS is very simple framework that can be embedded into any web app. Vue JS is very light weight and small framework, Vue JS is 20KB min+gzipped Runtime blazing fast framework.

The whole library is based on view only and is very easy to use with another framework or embed to an existing framework. One the other hand, Vue JS is perfect for building Single Page Applications.

If we compared with other frameworks, like angular or React. Let's talk about ReactJS. Both ReactJS and Vue JS provide same features:

- Utilize a virtual DOM.
- Provide reactive and composable view components.
- Maintain focus in the core library, with concerns such as routing and global state management handled by companion libraries.

Performance

Both React and Vue offer similar execution in most ordinarily observed utilize cases, with Vue more often than not somewhat ahead because of its lighter-weight Virtual DOM usage. In the event that you are keen on numbers, you can look at this outsider benchmark which concentrates on crude rendering/refreshing execution. Note this does not consider complex part structures, so should just be viewed as a source of perspective as opposed to a decision.

Ok we talk about React, now let’s go ahead to compare with angular.

Some of Vue's punctuation will look fundamentally the same as AngularJS (e.g. v-if versus ng-if). This is on account of there were a lot of things that AngularJS got right and these were a motivation for Vue from the get-go in its improvement. There are additionally many torments that accompany AngularJS in any case, where Vue has endeavored to offer a noteworthy change.

Who uses Vue JS?

Here are the list of websites who are using Vue JS:

1. http://www.laravel.com
2. http://www.laracasts.com
3. http://www.gitlab.com
4. http://www.vuejs.org
5. http://www.nu.nl
6. http://www.oschina.net
7. http://www.wapgee.com

In this book we will learn about Vue JS from beginner level to advanced level. This book is written for those who wants to learn Vue JS and you are at right place. Here are the online resources from where you can learn about Vue JS:

1. http://www.vuejs.org
2. http://www.laracasts.com
3. http://www.stackoverflow.com
4. http://wapgee.com/stories/education/programming/vue.js

Chapter 2

Installation and Fundamentals of Vue JS

Before getting started with a framework we should know how to install it before using. Every JavaScript library needs to install in our application. Vue JS is easy to learn library.
In this chapter we will learn following things:

1. How install Vue JS
    1. Using CDN
2. Hello World Example
3. Fundamentals of Vue JS
4. And some useful tricks in Vue Js

Let’s get started

Installation (Using CDN)

To install Vue JS using CDN, we will a fast and flexible CDN to import Vue JS in our application. The easiest to install Vue JS is by using a CDN, you can import Vue JS by:

```
<script src="https://unpkg.com/vue"></script>
```

Now we have successfully installed Vue JS on our application. As a starter, it is not recommend to get started with vue-cli, and to get started with vue-cli you need to get familiar with Node JS. By using above link, we can easily install Vue JS on your system. This is one of the easiest way to install Vue JS.

Hello World Example: To create a hello world example, let's go ahead and create a new HTML and file Install Vue JS. Don't forget to add Vue JS:

```
<html> <head> <title>Vue JS Hello World</title> </head> <body>
    </body> <script src="https://unpkg.com/vue"></script> </html>
```

Now we have successfully added Vue JS to our application. Now go ahead and create an element: `<div id="root" ></div>`

Now let's go ahead and create a new Vue Instance:

```
<script src="https://unpkg.com/vue"></script> <script type="text/javascript" > var App = new Vue({
    el: "#root", data: {
    message: "Hi", }
    }); </script>
```

As you can see the Vue JS App is attached to id "root", you can use a class too for el property. Now change message to "Hello World!"

```
var App = new Vue({
    el: "#root", data: {
    message: "Hello World!", }
});
```

This will change the value of message, this will print something like this:

Hello World!

Now let's go to console by hitting f12 and write this command App.message = "Wow", this will Immediately change value of message and updates in view. Actually Vue JS is real-time, when the value of variable value is changed and immediately updates in view. This will print:

Wow

Vue JS is very easy library to get started and build apps in no time. Now we have successfully created

our Hello World example. Now let’s go ahead and learn about the first topic which is Declarative rendering.

Declarative Rendering:

Do you have ever used variables in a programming language and print them. I hope you did it. Vue JS also allows us to create variables in our applications and print them on view. We call them “data”. In data, we store the variables and their values. Now go ahead and create a variable called name and give it value “Vue JS”. Now we have successfully created a variable. We can use them in our view by using {{variable_name}}, variable_name can be anything.

```
<html> <head> <title>Vue JS Hello World</title> </head> <body> <div id="root" > <h1>{{name}}</h1> </div> </body> <script
src="https://unpkg.com/vue"></script> <script type="text/javascript" > var App = new Vue({
    el: "#root", data: {
    name: "Vue JS"
    }
    }); </script> </html>
```

Vue JS

This will print value of name:

Now go ahead and update value of name in console by writing

```
> App.name = "HTML";
```

This will change value of name immediately and updates in view as well. Because Vue JS is real-time framework.

Now we have learned about declarative rendering and now let’s go ahead and learn about directives

Directives

In Vue JS directives are special attributes for an element with v-prefix. Directives gives special features to elements. There are tons of directives provided by Vue JS. Let's learn Vue JS Directives

v-show: v-show expects a Boolean value, If Boolean value is true then the element on which v-show is bind that element will be shown and if Boolean value is false then that element will be hidden:

```
<html> <head> <title>Vue JS Hello World</title> </head> <body> <div id="root" > <h1 v-show="heading" >{{name}}</h1> </div> </body>
<script src="https://unpkg.com/vue"></script> <script type="text/javascript" > var App = new Vue({
    el: "#root", data: {
    name: "Vue JS", heading: false, }
    }); App.name = "HTML"; </script> </html>
```

We have created another variable called heading and value is false, v-show expects a Boolean and we have passed Boolean data to it. Now let's go ahead and if very thing successfully works, you will be greeted with nothing! Because we set heading to false and for this h1 tag will be hidden from the app.

v-if: v-if works like same as in programming languages. I hope you know about Vue JS. After v-if we can also provide v-else for else. For example if a condition is not met then v-else will work. Look at this example:

```
<html> <head> <title>Vue JS Hello World</title> </head> <body> <div id="root" > <h1 v-if="heading" >{{name}}</h1> <h1 v-else>Above h1 is hidden</h1> </div> </body> <script src="https://unpkg.com/vue"></script> <script type="text/javascript" > var App = new Vue({
    el: "#root", data: {
    name: "Vue JS", heading: false, }
    }); App.name = "HTML"; </script> </html>
```

If v-if condition does not matches v-else will run, in above example we have set heading to false, so v-if condition does not meet and Vue JS will print **v-else** element.

v-on: This directive is used to bind events to a button, form or on input or on anything.

```
vue-on:eventName="method"
```

eventName can be anything. It can be click event, hover event, and double click event or focus or anything. This binds events to a form, button or anything. Let's a form and attach a method on submit:

```
<html> <head> <title>Vue JS Hello World</title> </head> <body> <div id="root" > <form v-on:submit="submitHandle" >
<button>Submit</button> </form> </div> </body> <script src="https://unpkg.com/vue"></script> <script type="text/javascript" > var App =
new Vue({
    el: "#root", methods: {
    submitHandle: function () {
    alert(1); }
    }
    }); </script> </html>
```

If this works successfully, this will greet us with a button. When you click on that button, an event passes called submit. Vue JS provides a directive for form submission. When someone submits the form Vue JS

will call submitHandle function. Go ahead and click on that button, the app will alert 1 and you will redirected. We will need to prevent to directive so Vue JS will not allow browser to refresh on form submit:

```
<form v-on:submit.prevent="submitHandle" > <button>Submit</button> </form>
```

“.prevent” stops redirection after submission of any form.

v-for: We can utilize the v-for directive to render things in a loop. The v-for directive requires a loop syntax to run loops in our web app. We can apply a range in Vue JS by:

```
<html> <head> <title>Vue JS Hello World</title> </head> <body> <div id="root" > <li v-for="i in 11"> {{ i }} X 4 = {{ i * 4 }}.
    </li> </div> </body> <script src="https://unpkg.com/vue"></script> <script type="text/javascript" > var App = new Vue({
    el: "#root", }); </script> </html>
```

This will print a loop between some ranges. For example 1 to 11. Do you think can we run a loop on an Array? Yes we can, we can run a loop on arrays. You will need some data in the form of an Array, then you can run a loop on it. Here is an example how to run a loop on an Array:

```
<html> <head> <title>Vue JS Hello World</title> </head> <body> <div id="root" > <li v-for="currentName in names"> Hello {{currentName.name}}
    </li> </div> </body> <script src="https://unpkg.com/vue"></script> <script type="text/javascript" > var App = new Vue({
    el: "#root", data: {
    names: [
    {name: "John"}, {name: "Gabriel"}, {name: "Albert"}, {name: "Doe"}, ]
    }
    }); </script> </html>
```

This will run a loop on names Array that holds objects that are same. This will print:

- Hello John
- Hello Gabriel
- Hello Albert
- Hello Doe

It means we have successfully run a loop on an Array. We will need a keyword “item in items”, item means current item and items means all items to run a loop.

Index is required sometimes for actions like deleting and updating item in array. You can access index of current item by:

```
<div id="root" > <li v-for="currentName,index in names"> {{index}}: {{currentName.name}}
    </li> </div>
```

Index variable in above code gives index of current loop. So this will print something like this:

- 0: John
- 1: Gabriel
- 2: Albert
- 3: Doe

This means we can easily access index in loops in Vue JS.

v-bind: This keyword binds one or more attributes to an element. Here is an example for **v-bind**

```
<!-- bind an attribute --> <img v-bind:src="imageSrc"> <!-- shorthand --> <img :src="imageSrc"> <!-- with inline string concatenation --> <img :src="'/path/to/images/' + fileName"> <!-- class binding --> <div :class="{ red: isRed }"></div> <div :class="[classA, classB]"></div> <div :class="[classA, { classB: isB, classC: isC }]"> <!-- style binding --> <div :style="{ fontSize: size + 'px' }"></div> <div :style="[styleObjectA, styleObjectB]"></div> <!-- binding an object of attributes --> <div v-bind="{ id: someProp, 'other-attr': otherProp }"></div>
```

We have learned about important directives of Vue JS. You can find here more about directives.

Chapter conclusion: We have learned about very basic fundamentals of Vue JS, installation of CDN based Vue JS. So far we have learned very basic terms in Vue JS. Which helps you to learn very basic idea of Vue JS. Now let's learns Vue JS syntax and data binding.

Chapter 3

Syntax and Data Binding

So far we have learned about basic ideas and infrastructure of Vue JS. We have learned how to install Vue JS, hello world example in Vue JS, basic template, initializing a Vue JS application. In this chapter, we will talk about Vue JS syntax and data binding. We have learned how to bind data in Vue JS in previous chapter, now we will get into deeper in this chapter. In this chapter we will get into deep in Vue JS syntax and Data Binding. How to set a code in a better way so every can understand, and anyone who is a Vue JS developer can easily make changes in your Vue JS code.

What actually Data Binding is?

In data binding we bind values to element, we bind a function to an element like on click, or on mouse over. We can bind values from our Vue App to document elements in HTML with Vue JS. For example I have a function that alerts value from message, it means we bind the value to alert in Vue JS. That message will be get from input. When we try to change text then Vue JS will automatically update the value of message in function used. Vue JS has two data binding. We want to change the message on user input, how this can be easily accomplished? In the example below we use v-model, a directive of Vue,. Then we use two-way data binding to dynamically change the message value when the user changes the message text inside an input. Data is synced on every input event by default by Vue JS.

```
<html> <head> <title>Vue JS Hello World</title> </head> <body> <div id="app"> <h1>{{ message }}</h1> <input v-model="message">
</div> </body> <script src="vue.min.js"></script> <script type="text/javascript"> new Vue({
    el: '#app', data: {
    message: 'Hi this Vue JS!'
    }
    }) </script>
</html>
```

This will automatically update value in HTML as we type in text box.

Vue JS is very easily language to learn, apart from React JS and Angular JS, Vue JS is very easy to learn and implement in any project or web language like Python, Ruby and PHP. Let's learn about Syntax and Data binding in Vue JS.

AS you know we can set up a Vue JS code by:

```
var App = new Vue({
    el: "#root", data: {...}, methods: {...}, });
```

This is very basic Vue JS initialized, we can also initialize our app by: `new Vue({`

```
    el: "#root", data: {...}, methods: {...}, });
```

This is same but we can use our Vue app in above example and in below example we cannot use Vue JS in other JavaScript code because we did not declare a variable for it therefore we cannot use that application in Vue. Now let’s come to data binding and understand how to bind data from Vue JS. Now take this example on:

```
<html> <head><title>Vue JS Hello World</title></head> <body> <div id="root" > <span class="className" ></span> </div> </body> <script
src="vue.min.js"></script> <script type="text/javascript" > var App = new Vue({
```

```
el: "#root", data: {
className: "span-class"
}
}); </script> </html>
```

Actually I want to add class to span tag, which is className and its value is span-class. This will print:

```
<html>
▶ <head>…</head>
▼ <body>
  ▼ <div id="root">
      <span class="className"></span> == $0
    </div>
    <script src="vue.min.js"></script>
  ▶ <script type="text/javascript">…</script>
  </body>
</html>
```

This will print the same className, actually I want to replace the className with span-class. How can you do it? We call it data binding in Vue JS. We can bind anything to element that empowers is v-bind. We can bind the class to span tag by using v-bind:

```
<html> <head><title>Vue JS Hello World</title></head> <body> <div id="root" > <span v-bind:class="className" ></span> </div> </body>
<script src="vue.min.js"></script> <script type="text/javascript" > var App = new Vue({
el: "#root", data: {
className: "span-class"
}
}); </script> </html>
```

So we have bind className to span tag. If everything works successfully, we will be greeted with:

```
<html>
▶ <head>…</head>
▼ <body>
  ▼ <div id="root">
      <span class="span-class"></span> == $0
    </div>
    <script src="vue.min.js"></script>
  ▶ <script type="text/javascript">…</script>
  </body>
</html>
```

So we can see class “span-class” is bind to span tag. Remember we need to use v-bind:name, name is actually change able, we can use class, src, style and class.

Chapter conclusion:

In this chapter we have learned about Vue JS syntax and data binding, how to bind data from Vue JS to view. Now let’s move to Vue JS component.

Chapter 4

Understanding Vue JS Component and their Usage

Components are most important features of Vue JS, you can say components are building parts of your application. Components help us to split our application into smaller parts so we can easily or someone can easily update things. We can bind a different function to components and we can also bind values to components from Vue Js. Components are used to make our application structure very easy to make changes. In many cases, components appear in form of HTML element in VueJS.

We have learned previous that we can bind a new Vue app instance by using:

```
new Vue({
el: '#some-element', // options })
```

This will make a new instance of Vue JS. So how to register a component, we can register a component we can use Vue.component(name,options) to make a new component. So we can make a new component by using:

```
Vue.component("component-name",{
    template: "<div>how are you?</div>", ...options...
});
```

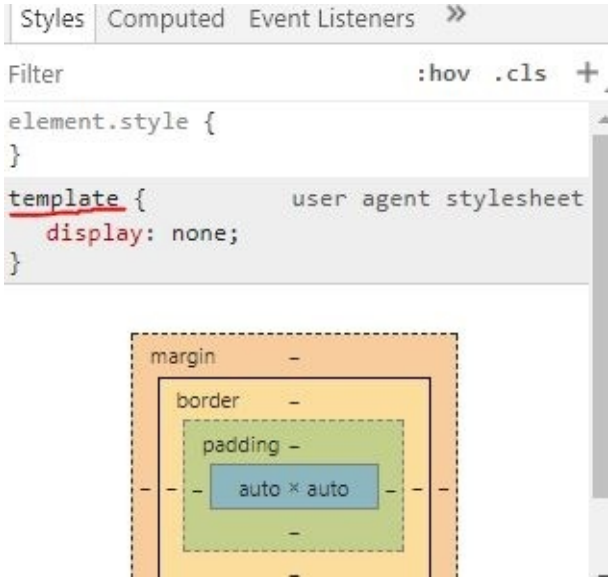
Now we have successfully created a Vue component. Once component is created we can use this component in by:

```
<component-name></component-name>
```

This will print:

how are you?

Now we successfully created a component and used it in our application. It is not easy to write markup in JavaScript. So we can use templates for this, but how we can use templates? I hope you know about <template></template> tag in HTML and template tag is hidden by base CSS.



You can template is hidden by base CSS. So we can use template for component. As you know we can use element id for creating a Vue JS instance, this is same with component. Now go ahead and write this template for component:

```
<template id="hello-world" > <div> <h1>Hello World!</h1> </div> </template>
```

This will be hidden on browser, we have to create component for this:

```
Vue.component("hello",{
    template: "#hello-world"
});
```

We have bind the id of template to component template property so component can use template tag for component rendering. We can use this component in our application by:

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

Whole example for component with template tag:

```
<html> <head> <title>Vue JS Hello World</title> </head> <body> <div id="app"> <hello></hello> </div> <template id="hello-world" > <div>
<h1>Hello World!</h1> </div> </template> </body> <script src="vue.min.js"></script> <script type="text/javascript"> Vue.component("hello",
{
    template: "#hello-world"
    }); new Vue({
    el: '#app', data: {
    message: 'Hi this Vue JS!'
    }
    }) </script>
</html>
```

If this works successfully you will be greeted with:

Hello World!

This is amazing right! . We have learned what is component and how to use components and how to use templates for components. Now let’s go to next section using components by props:

Component props

Do you have ever think how to pass props to components? How to pass values to components in Vue JS. Every component or function in any language must have feature to accept data from outside and use that in component. Vue JS allows us to use that feature too. Let's get this thing done!

First we need to define list of props that will be passed from outside:

```
Vue.component("hello",{
    template: "#hello-world", props: ["message"]
});
```

We have assign above component to accept props called message. We can use that prop in our template like this:

```
<template id="hello-world" > <div> <h1>Hello World!</h1> <p> {{message}}
    </p> </div> </template>
```

Then we have to pass message from outside while calling component:

```
<div id="app"> <hello message="You need to signup to our website" ></hello> </div>
```

If everything works successfully you will be greeted with something like this:

This is all code we have:

```
<html> <head> <title>Vue JS Hello World</title> </head> <body> <div id="app"> <hello message="You need to signup to our website" >
</hello> </div> <template id="hello-world" > <div> <h1>Hello World!</h1> <p> {{message}}
    </p> </div> </template> </body> <script src="vue.min.js"></script> <script type="text/javascript"> Vue.component("hello",{
    template: "#hello-world", props: ["message"]
    }); new Vue({
    el: '#app', data: {
    message: 'Hi this Vue JS!'
    }
    }) </script>
</html>
```

This is will work. Now let's come to methods in components in Vue JS. How to run methods as we do in Vue JS. As we defined props property, so we need to define methods property as well. And then write functions or methods:

```
Vue.component("hello",{
    template: "#hello-world", methods: {
    alert: function () {
```

```
    alert(1); }
    }
});
```

We have defined a function that will run on button click, we will need to add a button and add click event to button:

```
<template id="hello-world" > <div> <h1>Hello World!</h1> <button v-on:click="alert" >Click Me</button> </div> </template>
```

When someone clicks on above button, alert function or method will run. The code for methods:

```
<html> <head> <title> Vue JS Hello World </title> </head> <body> <div id = "app" > <hello message = "You need to signup to our website" >
</hello> </div> <template id = "hello-world" > <div> <h1>Hello World!</h1> <button v-on:click = "alert" > Click Me </button> </div>
</template> </body> <script src = "vue.min.js" ></script> <script type = "text/javascript"> Vue.component("hello",{
    template: "#hello-world", methods: {
    alert: function () {
    alert(1); }
    }
});
    new Vue({
    el: '#app', data: {
    message: 'Hi this Vue JS!'
    }
    }) </script>
</html>
```

If this code compiles successfully we will be greeted with a button, after clicking on that button we will be see an alert with 1 value.

When someone clicks on that button you will be greeted with alert:

So we have working example of component methods. In this chapter we have learned how to add props and methods feature to Vue JS components and we can bind different data to it. Now let's deeper into it!

We have created a method in our component, when someone clicks on that button, the method will run. For example you have a method in your Vue JS instance and you want to pass it to further components, how can you do it? Let's get that thing done!

Oh wait! Before you proceed need to learn these differences:

```
<hello :message="message" ></hello>
```

And:

```
<hello message="message" ></hello>
```

In above the value of message will be processed from a variable in data. If we use colon, then Vue JS will take message as a variable, if we don't use colon, then it will be a value. Right! I hope you understand it now .

Now let's create a method in Vue JS instance and then we can pass it to further.

```
new Vue ({
    el : '#app', data : {
    message: 'Hi this Vue JS!'
    }, Methods : {
    alert : function () {
    alert(1); }
    }
})
```

Now tell component to expect a parameter or props:

```
Vue.component("hello",{
    template: "#hello-world", props: ['alert']
});
```

Now add a button to component and assign v-on to it: `<template id="hello-world" > <div> <h1> Hello World ! </h1>`

`<button v-on:click = "alert" > Click Me </button> </div> </template>`

Now pass method while using component:

```
<hello :alert = "alert" > </hello>
```

While combining these terms our final executable code will look like this:

```
<html> <head> <title> Vue JS Hello World </title> </head> <body> <div id = "app"> <hello :alert = "alert" ></hello> </div> <template id =
"hello-world" > <div> <h1> Hello World ! </h1> <button v-on:click = "alert" > Click Me </button> </div> </template> </body> <script src =
"vue.min.js"> </script> <script type="text/javascript"> Vue.component("hello",{
    Template : "#hello-world", Props : ['alert']
    }); new Vue({
    el : '#app', data : {
    message: 'Hi this Vue JS!'
    }, Methods : {
    Alert : function () {
    alert(1); }
    }
    }) </script> </html>
```

If everything compiles successfully we will be greeted with something like this:

After clicking on that button we will something like this:

We have successfully created a Vue JS application that supports methods to pass from one component to another. We can use this feature to easily pass functions or methods to other parts of our Vue JS application.

We have learned about Vue JS syntax, installation and components. Let's have a quiz, create a chat application using Vue JS components.

Let's get that thing done, go ahead and create a fresh application and include Vue JS on to it. Here is a fresh application example for you.

```
<html> <head> <title> Vue JS Hello World </title> </head> <body> <div id = " app ">
    </div> </body> <script src = "vue.min.js"> </script> <script type = "text/javascript"> new Vue({
    el: '#app', }) </script> </html>
```

We have a fresh application and now go ahead and fresh application, following components we will need to create a chat application:

- Messages List: for listing messages.
    - Sub Components
        - Remove message button
        - Remove Message
- Form: for sending messages.

Now let's go ahead and create a fresh Vue JS instance:

```
var ChatApp = new Vue({
    el: "#root", data: {
    appName: "Welcome to chat App", messages: [
    {text: "Hi are you?",time: new Date()}, {text: "I'm fine",time: new Date()}, {text: "I'm fine",time: new Date()}
    ], }, methods: {
    save: function (message) {
    if (message) this.messages.push({text: message,time: new Date()}); }, removemessage: function (index) {
    this.messages.splice(index,1); }
```

```
    }
});
```

Now we have a fresh Vue JS instance, you can see we have data and methods:

Data:

- appName
    - Name of the chat app.
- messages
    - Contains all messages that are sent.

Methods:

- Save:
    - This will create a new object of message that pushed to array of messages.
- removeMessage:
    - This will remove a specific message.

Components:

Messages List:

```
var messagesList = Vue.component("messages",{
    template: "#messages-list", props: ['chat',"removemessage"], methods: {
    getRelativeTime: function (timeStamp) {
    var now = new Date(), secondsPast = (now.getTime() - timeStamp.getTime()) / 1000; if(secondsPast < 60){
    return parseInt(secondsPast) + ' Seconds Ago'; }
    if(secondsPast < 3600){
    return parseInt(secondsPast/60) + ' Minutes ago'; }
    if(secondsPast <= 86400){
    return parseInt(secondsPast/3600) + 'Hours Ago'; }
    if(secondsPast > 86400) {
    day = timeStamp.getDate(); month = timeStamp.toDateString().match(/ [a-zA-Z]*/)[0].replace(" ",""); year = timeStamp.getFullYear() ==
now.getFullYear() ? "" : " "+timeStamp.getFullYear(); return day + " " + month + year; }
    }
    }, });
```

This contains messages list component that will accept following props:

1. Chat: List of chat from Vue Instance.
2. removeMessage: function to be passed from outside to remove messages.

And methods:

1. getRelativeTime: This will get relative time, how much time has been passed after a specific

message is created.

Now let’s move to sub components:

```
var removemessage = Vue.component("removemessage",{
    template: "#remove_message", props: ["removemessagefunction"], });
var messageItem = Vue.component("messageitem",{
    template: "#message-item", props: ["message"]
});
```

Both are small components first one is removemessage that will accept a prop as function to delete a targeted message. Second is messageitem that will accept a message as string to print the message text.

Now we have JS is ready and create components now:

Messages List:

```
<template id = " messages-list " > <div> <ul class= "list-group " > <li class = " list-group-item " v-for = "m,index in chat" > <messageitem :message = "m" ></messageitem> <removemessage :removemessagefunction="removemessage"></removemessage> <span style = "position: relative;left: -50px;" class = " pull-right " > Date: {{getRelativeTime(m.time)}}
    </span> </li> <div v-if = " chat.length <= 0 " > <div class =" alert alert-info " > <p> NO Messages </p> </div> </div> </ul> </div>
</template>
```

Remove Message:

```
<template id = " remove_message " > <span> <a v-on:click=" removemessagefunction() " class = " close " >X</a>
</span> </template>
```

Message Item:

```
<template id = " message-item " > <span> {{ message.text }} </span> </template>
```

Chat Form:

```
<template id = " chat-form" > <div> <form v-on:submit.prevent = " save(message),empty()" > <label>Your Message</label> <input v-model="message" type = " text " class = "form-control" /> <div v-if = "hasErrors" class = "alert alert-danger" > <p> Please enter a message.
    </p> </div> <button class = "btn btn-info" >Send</button> <span class = " pull-right badge badge-info " > {{totalmessages}}
    </span> </form> </div> </template>
```

Now we all components ready now, run these our application:

```
<div id="root" > <div> <h1>{{appName}}</h1>
<messages :removemessage="removemessage" :chat="messages" ></messages> <chatform :totalmessages="messages.length" :save="save"
></chatform> </div> </div>
```

Now copy these all components as HTML and JS to their proper location, our final chat application will look like this:

Index.html

```
<!DOCTYPE html> <html> <head> <title> Title </title> <link rel="stylesheet" type="text/css"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"> <style type="text/css"> body {
    margin: 20px; }
    </style> </head> <body> <div id="root" > <div> <h1>{{appName}}</h1> <messages :removemessage="removemessage"
:chat="messages" ></messages> <chatform :totalmessages="messages.length" :save="save" ></chatform> </div> </div> <!-- templates -->
<template id="messages-list" > <div> <ul class="list-group" > <li class="list-group-item" v-for="m,index in chat" > <messageitem :message="m"
></messageitem> <removemessage :removemessagefunction="removemessage"></removemessage> <span style="position: relative;left:
-50px;" class="pull-right" > Date: {{getRelativeTime(m.time)}}
```

```
</span> </li> <div v-if="chat.length <= 0" > <div class="alert alert-info" > <p> NO Messages </p> </div> </div> </ul> </div> </template>
<template id="remove_message" > <span> <a v-on:click="removemessagefunction()" class="close">X</a> </span> </template> <template
id="message-item" > <span>{{message.text}}</span> </template> <template id="chat-form" > <div> <form v-
on:submit.prevent="save(message),empty()" > <label>Your Message</label> <input v-model="message" type="text" class="form-control" />
<div v-if="hasErrors" class="alert alert-danger" > <p> Please enter a message.
</p> </div> <button class="btn btn-info" >Send</button> <span class="pull-right badge badge-info" > {{totalmessages}}
</span> </form> </div> </template> <!-- templates --> <script type="text/javascript" src="vue.js" ></script> <script type="text/javascript"
src="app.js" ></script> </body> </html>
```

App.js

```
var removemessage = Vue.component("removemessage",{
    template: "#remove_message", props: ["removemessagefunction"], });
var messageItem = Vue.component("messageitem",{
    template: "#message-item", props: ["message"]
});
var messagesList = Vue.component("messages",{
    template: "#messages-list", props: ['chat',"removemessage"], methods: {
    getRelativeTime: function (timeStamp) {
    var now = new Date(), secondsPast = (now.getTime() - timeStamp.getTime()) / 1000; if(secondsPast < 60){
    return parseInt(secondsPast) + ' Seconds Ago'; }
    if(secondsPast < 3600){
    return parseInt(secondsPast/60) + ' Minutes ago'; }
    if(secondsPast <= 86400){
    return parseInt(secondsPast/3600) + 'Hours Ago'; }
    if(secondsPast > 86400) {
    day = timeStamp.getDate(); month = timeStamp.toDateString().match(/ [a-zA-Z]*/)[0].replace(" ",""); year = timeStamp.getFullYear() ==
now.getFullYear() ? "" : " "+timeStamp.getFullYear(); return day + " " + month + year; }
    }
    }, });
var chatForm = Vue.component("chatform",{
    template: "#chat-form", props: ["save","totalmessages"], data: function () {
    return {
    message: "", hasErrors: false, }
    }, methods: {
    empty: function () {
    if (this.message) {
    this.message = ""; this.hasErrors = false; }else {
    this.hasErrors = true; }
    }
    }
});
var ChatApp = new Vue({
    el: "#root", data: {
    appName: "Welcome to chat App", messages: [
    {text: "Hi are you?",time: new Date()}, {text: "I'm fine",time: new Date()}, {text: "I'm fine",time: new Date()}
    ], }, methods: {
    save: function (message) {
    if (message) this.messages.push({text: message,time: new Date()}); }, removemessage: function (index) {
```

```
    this.messages.splice(index,1); }
    }
});
```

If everything compiles successfully you will be greeted with:

Now we have working chat application and it is working without any errors.

Chapter conclusion:

We have learned and created applications in this chapter, we have learned about components and other things. We have learned how to pass different values to components and other values. Now let's move to VueX.

Chapter 5

Guide to VueX, its patterns, and usage

What is VueX?

Just like Redux, VueX is a state management library as well as a pattern for state management. VueX serves as base store for all components in Vue JS applications. VueX is also integrated with Vue JS extension.

Installation

Let’s get started by installation of VueX in our app. For a beginner Vue JS developer I recommend to getting started with CDN based installation, let’s install VueX by CDN based, you can install VueX along with Vue JS by:

```
<html> <head> <title> Hello </title> </head> <body> <div id = "root" > </div> </body> <script src = "https://unpkg.com/vue" > </script>
<script src = "https://unpkg.com/vuex" > </script> <script> //
    </script> </html>
```

Now we have successfully installed VueJS and VueX in our app.

You can install by NPM by:

```
npm install vuex --save
```

You can install by YARN by:

```
yarn add vuex
```

These both commands will install VueX in different systems like NPM or YARN. We will learn VueX by simple HTML.

Guide to State Management Pattern

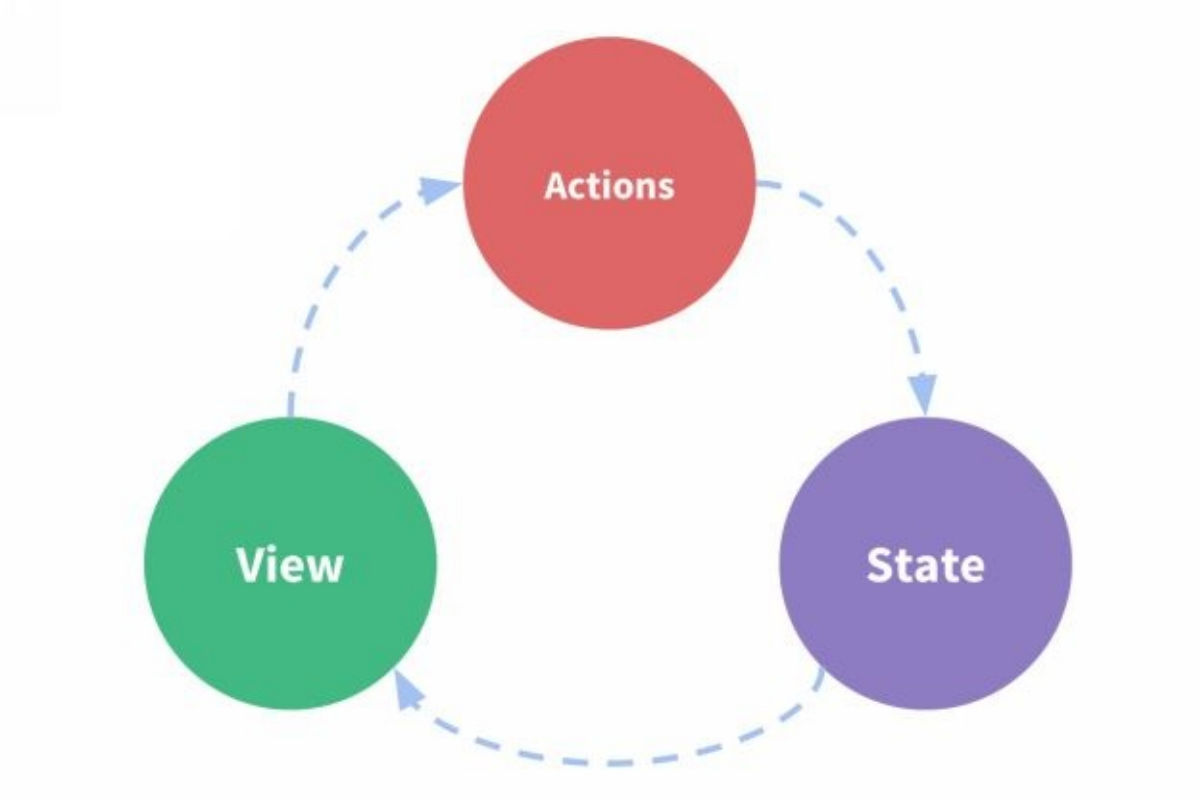
To show state management pattern works, let's start with a simple Vue Counter app:

```
new Vue({
    // state data () {
    return {
    count: 0
    }
    }, // view template: `
    <div>{{ count }}</div> `, // actions methods: {
    increment () {
    this.count++
    }
    }
});
```

This small Vue JS app contains following parts:

- state: The source of truth.
- view: View for Vue JS app.
- actions: Contains all methods to interact when something happens in view, like an event or something else.

This can be explained by this image provided Vue JS:



The benefits of using VueX are following:

1. Multiple views are depend on same piece of state.
2. Actions from different views may need to mutate the same piece of state.

For issue one, passing props can be dull for profoundly settled parts, and essentially doesn't work for sibling component. For issue two, we regularly end up depending on arrangements, for example, going after direct parent/youngster occurrence references or attempting to transform and synchronize numerous duplicates of the state by means of occasions. Both of these examples are fragile and rapidly prompt unmaintainable code.

So why don't we separate the mutual state out of the components, and oversee it in a worldwide singleton? With this, our part tree turns into a major "view", and any component can get to the state or trigger activities, regardless of where they are in the tree!

What's more, by characterizing and isolating the ideas associated with state administration and upholding certain principles, we likewise give our code more structure and viability.

This is the fundamental thought behind Vuex, motivated by Flux, Redux and The Elm Architecture. Dissimilar to alternate examples, Vuex is likewise a library usage custom fitted particularly for Vue.js to exploit its granular reactivity framework for productive updates.

Getting Started

At the focal point of each Vuex application is the store. A "store" is essentially a compartment that holds your application state. There are two things that make a Vuex store unique in relation to a plain worldwide protest:

1. Vuex stores are responsive. At the point when Vue parts recover state from it, they will responsively and effectively refresh if the store's state changes.

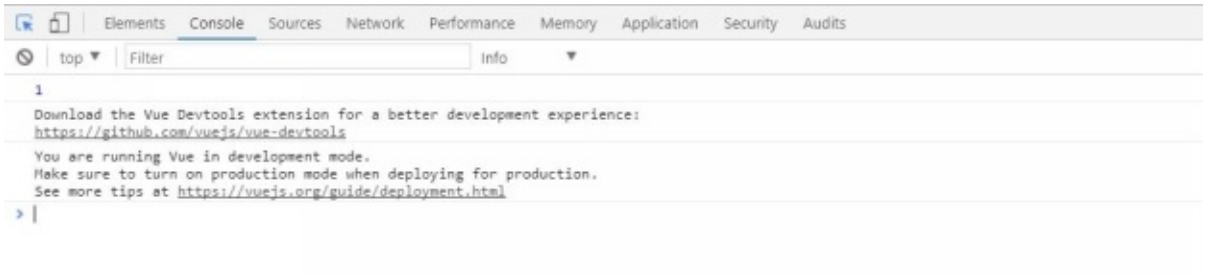
2. You can't straightforwardly change the store's state. The best way to change a store's state is by unequivocally submitting transformations. This guarantees each state change leaves a track-capable record, and empowers tooling that causes us better comprehend our applications.

A simplest Store

After installation of VueX, let’s get started by an example for store.

```
<html> <head> <title>Hello</title> </head> <body> <div id="root" > </div> </body> <script src="https://unpkg.com/vue" ></script> <script src="https://unpkg.com/vuex" ></script> <script> const store = new Vuex.Store({
    state: {
    count: 0
    }, mutations: {
    increment (state) {
    state.count++
    }
    }
    }); store.commit('increment'); console.log(store.state.count); // -> 1
    </script> </html>
```

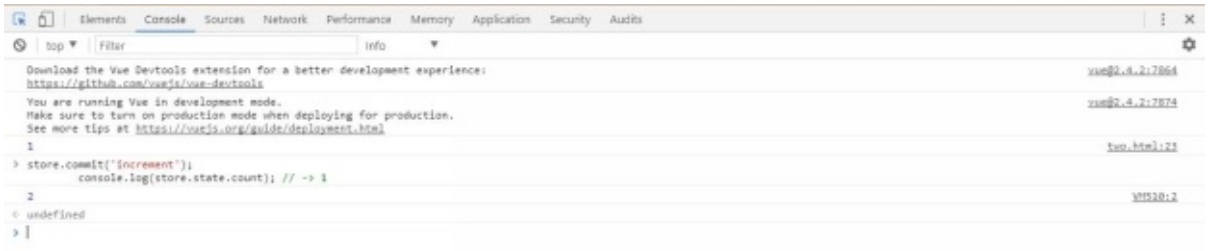
Mutations are actually like methods, in the above example we have incremented the current value of counter. We have changed the state of your application. Now open that file in browser and see console. You will see something like this:



You can see 1 is printed in console. Now write this code and press enter:

```
store.commit('increment'); console.log(store.state.count);
```

This will print 2 on your console:



Now we have working example of VueX. Let's get more deeper!

State

Vuex utilizes a solitary state tree - that is, this single question contains all your application level state and fills in as the "single wellspring of truth". This likewise implies for the most part you will have just a single store for every application. A solitary state tree makes it direct to find a particular bit of state, and enables us to effortlessly take previews of the current application state for troubleshooting purposes.

Getters

Getters are type of functions that are used to return values from state object. Let suppose we have a store for all songs list. Then we can use a function to get list of all songs:

```
const store = new Vuex.Store({
    state: {
    songs: [
    { name: " Despacito"} , { name: " Will You !"} , { name: " You are not alone ! "} , ]
    }, getters: {
    getAllSongs: state => {
    return state.songs ; }
    }
});
```

A function in getters called getAllSongs() will return all songs in state management. We use this function by: `store.getters.getAllSongs() ;`

This will return all list of songs presented in songs array.

Mutations

You can call mutation also as methods. Mutations are also methods. From above example, create a mutation that will add a new song to list:

```
const store = new Vuex.Store({
    state: {
    songs: [
    {name: " Despacito " } , {name: " Will You ! " } , {name: " You are not alone ! " } , ]
    }, getters: {
    getAllSongs: state => {
    return state.songs; }
```

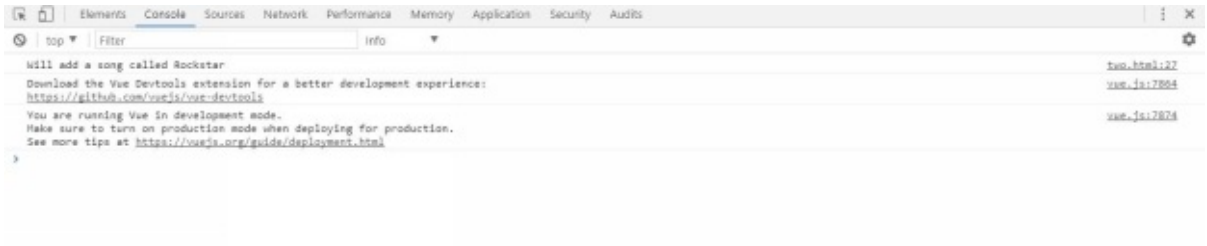
```
    }, mutations: {
    addSong: function () {
    console.log("Will add a song") ; }
    }
});
```

We can the addSong by: `store.commit("addSong") ;`
This will add print a log in your console.

Ok , You did it, How can we pass parameters to parameters. We can pass parameters by using payloads, we call payloads as parameters. Go ahead and create parameters. We can change our mutation to something like this:

```
mutations: {
    addSong: function (state,name) {
    console.log(` Will add a song called ${name} ` ) ; }
}
```

State parameter is used by VueX and second parameter is the name of the song you want to add. You can add the song by: `store.commit("addSong","Rockstar");`
This will print something like this:



Well, copy this code and paste it your file to see how it works:

```
<html> <head> <title> Hello </title> </head> <body> <div id = "root" > </div> </body> <script src = "vue.js" type = " text/javascript " >
</script> <script src = "vuex.js" type = " text/javascript " ></script> <script> const store = new Vuex.Store({
    state: {
    songs: [
    {name: " Despacito " } , {name: " Will You ! " } , {name: " You are not alone ! " } , ]
    }, getters: {
    getAllSongs: state => {
    return state.songs; }
    }, mutations: {
    addSong: function (state,name) {
    console.log(` Will add a song called ${name} `); }
    }
    }); store.commit("addSong"," Rockstar "); </script> </html>
```

If you copy everything correctly you will be greeted with nothing on screen , press f12 and see console to see list of logs generated by code.

Actions

Both actions and mutation are same things, but there are following differences;

- Instead of committing a mutation, actions commits mutations.
- Actions can contains arbitrary asynchronous operations.

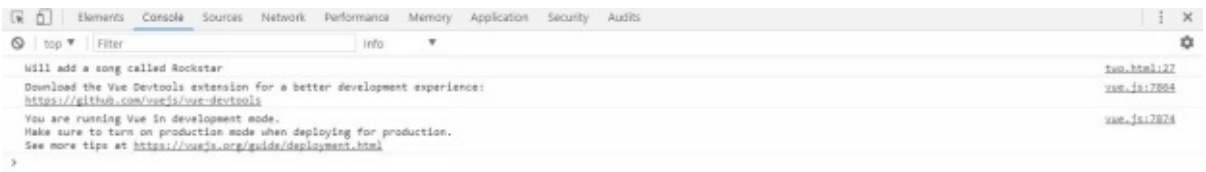
Let’s go ahead and register an action in our VueX:

```
const store = new Vuex.Store({
    state: {
    songs: [
    {name: " Despacito " } , {name: " Will You! " } , {name: " You are not alone ! " } , ]
    }, getters : {
    getAllSongs : state => {
    return state.songs; }
    }, mutations : {
    addSong : function (state,name) {
    console.log(`Will add a song called ${name}`); }
    }, actions : {
    addSong : function (context) {
    context.commit(" addSong "," Rock Star " ) ; }
    }
});
```

We have created an action that runs a mutation, actually it runs addSong mutation to add a song. How can you run an action? We can run an action by dispatch command.

```
store.dispatch("addSong");
```

This will run the action addSong. Now copy this whole code and paste it in your html file while including Vuex and Vue. Then check your console to see logs on your console:



This will print the same log, because our concept is same but approach is different.

Chapter conclusion: We have learned about VueX and now let’s move to chapter testing.

Chapter 6

Unit Testing

What actually is unit testing in Vue JS?

Let me explain you what is Vue JS again: “Vue.js is a JavaScript structure which is helpful for building the front-end of web applications, especially while making complex elements. For each venture, it's important to experience everything in our applications and watch that everything fills in obviously. Notwithstanding, for extensive ventures, it rapidly ends up noticeably dreary to check each component after each new refresh. Thus, we can make computerized tests which can run constantly and give us consolation that our code works. In this instructional exercise, we will make a couple of basic unit tests for VueJS which demonstrate to begin. ”

Now let’s move to unit testing in Vue JS, Unit testing allows to ensure Vue JS working. For example with unit testing you can test Vue JS units to confirm if it is working fine. Unit testing also allows us to ensure that behavior of every component is working fine and is consistent.

Setup and Tooling

Anything good with a module-based form framework will work, yet in the event that you're searching for a particular proposal, attempt the Karma test sprinter. It has a considerable measure of group plugins, including support for Webpack and Browserify. For a point by point setup, please allude to each venture's particular documentation, however, these illustration Karma arrangements for Webpack and Browserify may enable you to begin.

Simple Assertions For code structure test we don’t need to do anything, we just need to export raw things:

```
<template> <span> {{ message }} </span> </template> <script> export default {
    data () {
    return {
    message: 'hello ! '
    }
    }, created () {
    this.message = ' bye ! '
    }
}
</script>
```

When you want to test that component, you have include that exported object with Vue to make common assertions.

```
import Vue from 'vue'
import TheComponent from 'your/drive/path/to/TheComponent.vue'
describe('MyComponent', () => {
    it('has a created hook', () => {
    expect(typeof TheComponent.created).toBe('function') }) it('sets the actual default data', () => {
    expect(typeof TheComponent.data).toBe('function') const defaultData = TheComponent.data() expect(defaultData.message).toBe('Hello
how are you!') }) it('without errors sets the message when created', () => {
    const vm = new Vue(TheComponent).$mount() expect(vm.message).toBe('Goodbye') }) it('renders the actual message', () => {
    const Ctor = Vue.extend(TheComponent) const vm = new Ctor().$mount() expect(vm.$el.textContent).toBe('hello, Bye!') }) })
```

Writing Testable components

A considerable measure of component's render yield are fundamentally controlled by the props they get. Actually, if a part's render yield exclusively relies upon its props, it turns out to be very clear to test, like attesting the arrival estimation of an unadulterated capacity with various contentions. For example:

```
<template> <p> {{ msg }} </p> </template>

<script> export default {
    props : [ 'msg' ]
}
</script>
```

You can affirm its render yield with various props utilizing the propsData alternative: import Vue from 'vue'

```
import MyComponent from './MyComponent.vue'
// helper function that mounts and returns the rendered text function getRenderedText (Component, propsData) {
const Ctor = Vue.extend(Component) const vm = new Ctor({ propsData: propsData }).$mount() return vm.$el.textContent }
describe('MyComponent', () => {
it('renders correctly with different props', () => {
    expect(getRenderedText(MyComponent, {
    msg: 'Hello'
    })).toBe('Hello') expect(getRenderedText(MyComponent, {
    msg: 'Bye'
    })).toBe('Bye') })
})
```

Unit testing With Karma and Mocha

With this testing we will test our application by using, first you will need to install Vue JS by CLI, and this means you need to install Node JS based Vue JS: `$ vue init webpack my-project`

This will install vue-cli and create a project. Then we need to make some changes in test/unit/karma.config.js. We will need to specify the names of the plugins we want to use.

```
var webpackConfig = require('path/to/webpacktest.conf');
module.exports = function (config) {
config.set({
    frameworks: ['mocha', 'sinon-chai'], files: ['./index.js'], preprocessors: {
    './index.js': ['webpack', 'sourcemap']
    }, webpackMiddleware: {
    noInfo: true }, browsers: ['Chrome'], webpack: webpackConfig, reporters: ['spec', 'coverage'], plugins: [
    'karma-chrome-launcher', 'karma-mocha', 'karma-sinon-chai', 'karma-webpack', 'karma-sourcemap-loader', 'karma-spec-reporter', 'karma-coverage'
    ], coverageReporter: {
    reporters: [
    { type: 'lcov', subdir: '.' }, { type: 'text-summary' }
    ], dir: './coverage', }
})
}
```

First component unit test

Let’s create a simple component to test it: <template> {{propValue}} </template>

```
<script> export default {
props: ['propValue']
}
</script>
```

In spec add a new one “test/unit/spec”. This will check if the component text is same as defined:

```
import Vue from 'vue'; import TestIT from 'src/components/TestIT';
describe('TestIT.vue', () => {
it(`should render property Value as it's text content`, () => {
    const Constructor = Vue.extend(TestIT);
    const comp = new Constructor({
    propsData: {
    propValue: 'Test Text'
    }
    }).$mount();
    expect(comp.$el.textContent) .to.equal('Test Text'); }); });
```

As vue.js updates for asyn updates we will need to trigger a function that will check on regular basis on component updates:

```
<template> <p>{{dataProp}}</p> </template>
<script> export default {
    data: function () {
    return {
    dataProp: 'Data Text'
    }; }
}
</script> import Vue from 'vue'; import TestMe2 from 'src/components/TestMe2';
describe('TestMe2.vue', () => {
...
it(`This updates when dataText is changed.`, done => {
    const Constructor = Vue.extend(TestMe2);
    const comp = new Constructor().$mount();
    comp.dataProp = 'New Text';
    Vue.nextTick(() => {
    expect(comp.$el.textContent) .to.equal('New Text'); done(); }); }); });
```

This is what we have in Vue JS unit testing, you can learn more about Vue JS unit testing through following resources:

https://alligator.io/vuejs/unit-testing-karma-mocha/

https://vuejs.org/v2/guide/unit-testing.html

https://scotch.io/tutorials/how-to-write-a-unit-test-for-vuejs

Chapter Conclusion: Well we have learned about Vue JS unit testing and we have learned how to test components in Vue JS. In the next chapter we will create some examples,

Let’s move to next chapter!

Chapter 7

Vue JS CLI and Examples

We have learned all basic of Vue JS, in this chapter we will learn how to get started with Vue JS CLI version and in the last of this chapter we will do some exercises. I recommend everyone to use Vue JS CLI, you can also say NODE JS based Vue JS. Vue JS by CLI is most recommended way to use Vue JS. Well you can install Vue JS by using: You need to install globally first

```
npm install --global vue-cli
```

This will install vue js globally, then you will need to install initialize a Vue JS app: `vue init webpack my-project`

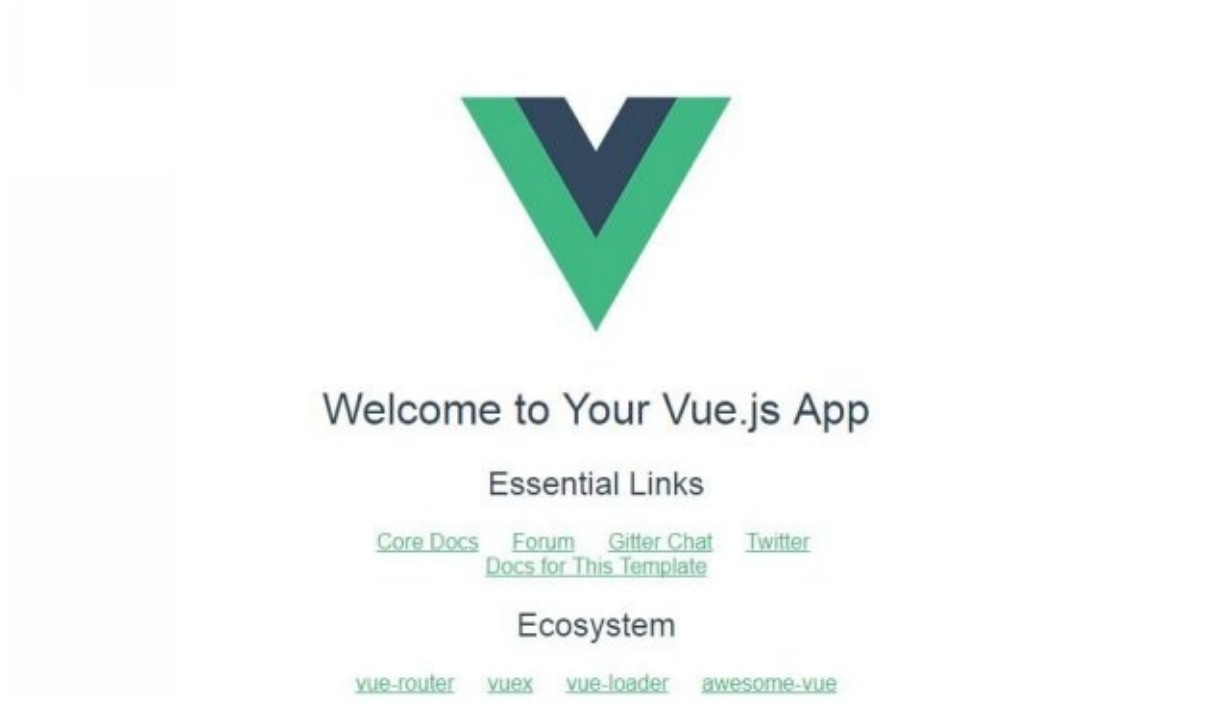
This will create a Vue JS app without Node Modules. We will need to continue to the folder:

```
cd my-project
```

Then we will install all dependencies including Vue JS resources: `npm install`

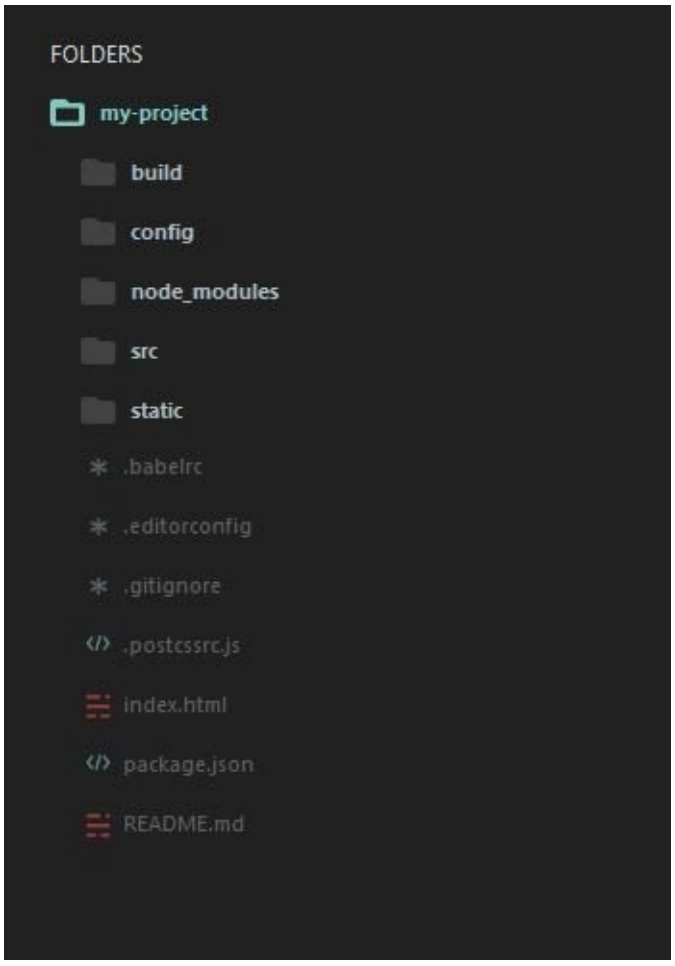
This will install all dependencies including Vue JS resources, we will need to start our app: `npm run dev`

This will open a server on http://localhost:8080/, if everything compiles successfully, you will be greeted



with:

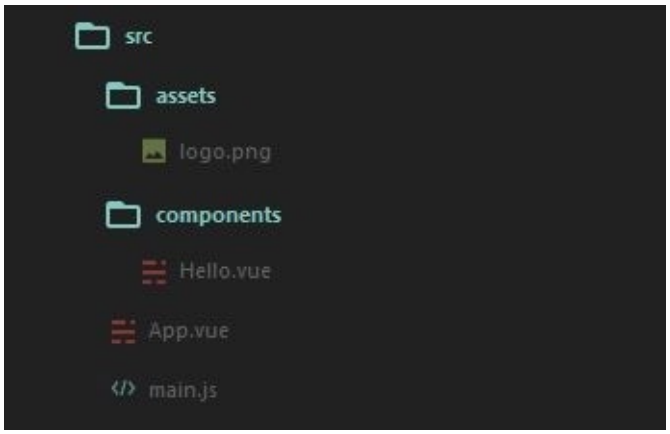
If you see this screen, this means you have successfully installed Vue JS CLi(Node JS based Vue JS) on your system . If you open the directory listing for your Vue JS, you will something like this: (Head to

next page) 

You will see something like this if you have installed everything correctly. Now let me explain folder structure:

1. build: This folder contains build files.
2. config: This folder contains configuration for app.
3. node_modules: This folder contains other modules along with Vue JS.
4. Src: This folder contains Vue JS App.
5. Package.json: This file contains app info and dependencies list.

Now let’s move to folder src and you will see something like this:

(Next Page) 

There are following folders in this folder:

1. Assets
2. Components

Assets folder contains assets for our application. Images, logos, CSS and JS files for our application. Components folder contains all components for our application. I recommend to use Vue CLI for future because you can easily make your application build in HTML version.

Main.js file is entry point of our Vue CLI app. In main.js we import all Vue JS components and render them. We can also define our Vue JS application Router configuration with router main.js will look like something like this:

```
// The Vue build version to load with the `import` command // (runtime-only or standalone) has been set in webpack.base.conf with an alias.
import Vue from 'vue'; import { App } from './app'; import router from './router'; import store from './store';
/* eslint-disable no-new */
new Vue({
el: '#app', store, router, template: '<App/>', components: { App }
});
```

Now go ahead and open a component file in src/App.vue, You will this code in App.vue:

```
<template> <div id = "app"> <img src = "./assets/logo.png" > <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>
```

One thing to remember, there are three parts in components:

1. Template
2. Script
3. Style

Template contains view for the current component, what type of view you want to render for the current component.

The second part which is script, this contains script code, JavaScript code for current component or for current view. This code is based on Vue JS.

The last one which is style, this contains styles. Not for current but for whole page. If we add an attribute scoped in style tag, the CSS in style tag will be only affect able on current component.

Now we have created a Vue JS CLI application and we have started our application. Now let's move to an example in Vue JS.

Examples In this examples section we will create some examples by using components. Examples are important to learn programming, in section we will consider to work on examples.

Todo Example In the first example we will create a todo app. Before Continuing, Install Vue JS and Bootstrap, You can easily link to them by: `<link rel = "stylesheet" type = "text/css" href = "bootstrap.css"> <script type = "text/javascript" src = "vue.js" > </script>`

Bootstrap CSS is a framework to design awesome web pages, to make our Todo App nice, I have used bootstrap and Vue JS is Vue Core Library.

Vue App

Let's go ahead create todo app. First we will need a form to create a to do.

<form v-on:submit = 'submit' > <label> What do you want to do ? </label> <textarea required = "required" v-model = 'input' class = "form-control" ></textarea> <button v-if = 'editingIndex != null' class = "btn btn-info" > Update </button> <button v-else class = "btn btn-info" > Create </button> </form>

Now form is created and now Vue APP

```
var App = new Vue({
el: "#todo_app", data: {
    editingIndex: null, total_todos: 0, input: "", todos: [], },
methods: {
    getObject: function (text,completed) {
    }, submit: function () {
    }, edit: function (index) {
    }, mark: function (index) {
    }, deleteTodo: function (index) {
    }
}
});
```

In the data, There is input, todoeditingIndex, total_todos and todos. Input is the model that value is used to create to do. It serves as the text for to do. total_todos contain a total number of todos. editingIndex identifies the s index of a todo that is being edited. todos contains all todo items that saved.

Now come to methods:
getObject(): This method get a an object for todo, that contains text of todo and completed.
submit() This method runs whenever form is submitted. This saves or updates a todo item.
edit() This will update editingIndex,
mark() This will mark selected todo as completed
deleteTodo() This will delete todo item.

Listing Todos To list todos, we will need `v-for` Vue js feature to loop through the array. We can list todos by: Task: {{todo.text}}   Completed: Yes <p v-else > Completed: No <button v-on:click = 'mark(index)' v-if = 'todo.completed' class = "btn btn-info btn-xs" >Mark as Incomplete</button> <button v-on:click='mark(index)' v-else class = "btn btn-info btn-xs" > Mark as Complete</button> <button v-on:click = 'edit(index)' class = "btn btn-success btn-xs" >Edit </button> <button v-on:click = 'deleteTodo(index)' class = "btn btn-danger btn-xs" > delete </button>

App.JS

```
var App = new Vue({
el: "#todo_app", data: {
    editingIndex: null, total_todos: 0, input: "", todos: [], },
methods: {
```

```
getObject: function (text,completed) {
return function () {
return {
text: text, completed: completed }; }
}, submit: function () {
if (this.editingIndex === null) {
var objc = this.getObject(this.input,false); var old = this.todos; var n = [...old,objc()]; this.todos = []; this.todos = n; this.input = "";
this.total_todos = this.todos.length; }else {
this.todos[this.editingIndex].text = this.input; this.editingIndex = null; this.input = ""; }
}, edit: function (index) {
this.editingIndex = index; this.input = this.todos[index].text; }, mark: function (index) {
var object = this.todos[index]; if (object.completed) {
this.todos[index].completed = false; }else {
this.todos[index].completed = true; }
this.total_todos = this.todos.length; }, deleteTodo: function (index) {
this.todos.splice(index,1); this.total_todos = this.todos.length; }
}
});
```

Index.html

```
<div id= "todo_app" class = "container" > <div class = "navbar navbar-default" > <div class = "navbar-header" > <a class = "navbar-brand">
Todo App </a> </div> <div class = "navbar-collapse collapse" > <ul class = "navbar-right navbar-nav nav" > <li> <a href = "#">
{{total_todos}} Todos</a> </li> </ul> </div> </div> <div> <div class = "row" > <!-- form --> <form class = "col-sm-4 col-sm-offset-4" v-
on:submit = 'submit' > <h3> Create Todo </h3> <div class = "form-group"> <label> What do you want to do ? </label> <textarea required =
"required" v-model = 'input' class = "form-control" ></textarea> </div>
<button v-if = 'editingIndex != null' class = "btn btn-info" > Update </button> <button v-else class = "btn btn-info" > Create </button> <br
/> <br /> <legend> </legend> </form> <!-- form --> <div class = "cols-m-12" > <div class = "col-sm-4 col-sm-offset-4" > <div v-if =
'total_todos == 0' > <div class = "alert alert-info" > No Todos </div> </div> <div class = "list-group" > <div class = "list-group-item" v-for =
"todo,index in todos" > <p> <strong> Task: </strong> {{ todo.text }}
</p> <p v-if = 'todo.completed' ><strong> Completed :</strong> Yes</p> <p v-else > <strong> Completed: </strong> No</p> <button v-
on:click ='mark(index)' v-if = 'todo.completed' class = "btn btn-info btn-xs" >Mark as Incomplete</button> <button v-on:click = 'mark(index)'
v-else class = "btn btn-info btn-xs" >Mark as Complete</button> <button v-on:click = 'edit(index)' class = "btn btn-success btn-xs" > Edit
</button> <button v-on:click = 'deleteTodo(index)' class = " btn btn-danger btn-xs" >delete</button> </div> </div> </div> </div> </div> </div>
</div>
```

You can check the example here (http://www.wapgee.com/story/235/creating-todo-app-in-vue-js-with-bootstrap). This is fully working example. Now let's move to next example.

Example two In this example, we will create an example in CLI version. So in this example we will create a chat application by using simple components system.

Let's build our chat application, now you need to install bootstrap in your application, bootstrap is also build for Vue JS, and so you can also install bootstrap in Vue JS: `npm install --save bootstrap-vue`

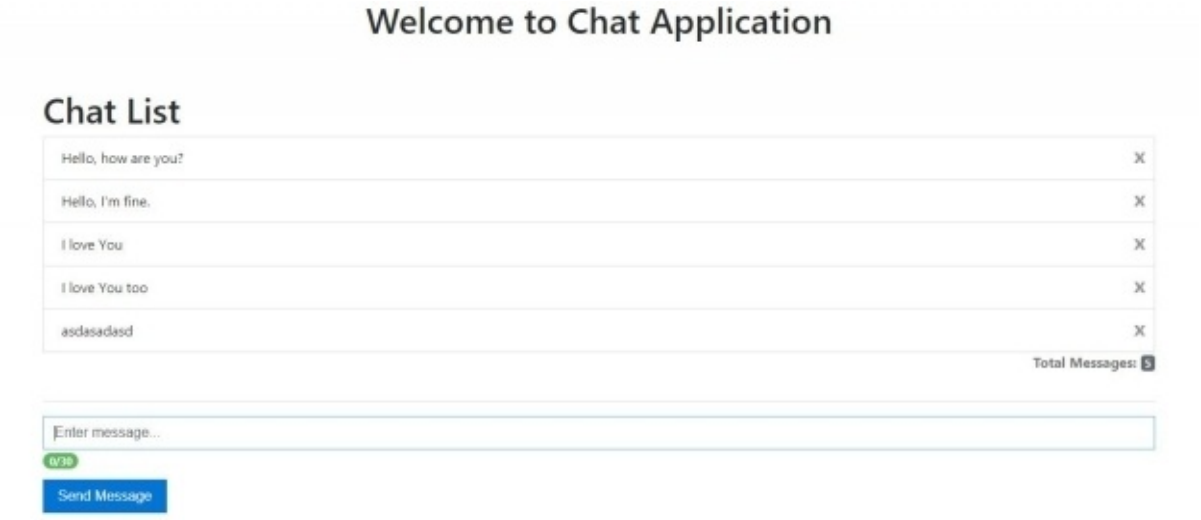
This will install Bootstrap in your Vue JS. As we talked earlier about main.js, which is our entry point for our application, we can include bootstrap on main.js which is our entry point for our application. Your

main.js file should look like this:

```
import Vue from 'vue'
import App from './App'
import BootstrapVue from 'bootstrap-vue';
Vue.use(BootstrapVue)
import 'bootstrap/dist/css/bootstrap.css'
import 'bootstrap-vue/dist/bootstrap-vue.css'

Vue.config.productionTip = false
/* eslint-disable no-new */
new Vue({
el: '#app', render: h => h(App) })
```

In this file we have used Bootstrap in our Vue Application. Now let's go ahead to our chat application. We will create an example something like this:



Before getting start with our application we will need following components in our Vue JS Appp:

1. Chat list. This will contain list of all messages.
2. Chat form. This will contain form for sending messages.

Now go ahead and create our first component which App, App.vue. This file will be in src folder. With this content: <template>   <h1>Welcome to Chat Application</h1>   <chatlist :chats="messages" ></chatlist> <hr></hr> <chatform :isEmptyText="isMessageEmpty" :save="save"></chatform>   </template>

```
<script> import chatlist from "./components/chat/chatlist"
import chatform from './components/chat/Form'
export default {
name: 'app', components: {
    chatlist, chatform },
data: function () {
    return {
    isMessageEmpty: false, messages: [
    {text: "Hello, how are you?"}, {text: "Hello, I'm fine."}, {text: "I love You"}, {text: "I love You too"}, ]
```

```
    }
},

methods: {
    save: function (message) {
    if (message) {
    this.messages.push({text: message}); }else {
    this.isMessageEmpty = true; }
    }
}
}
</script>
<style scope > .app-header {
    text-align: center;; }
#app-container {
    padding: 50px; }
</style>
```

This is our main root for our chat application that contains methods/function and data for our app. You can see methods in methods property:

1. Save

This will save a chat instance. Now let's move to data object. There are only two items there, one is isMessageisempty and second is messages. isMessageisempty verifies if text field is empty and messages contains all messages that are stored by visitor.

Now let's move to our child components, first one is chat list. The chat list contains all chat list sent by user. So our content will look like alike: (Remember this file will be in src/components/chat/)

```
<template> <div> <h1>Chat List</h1> <b-list-group> <b-list-group-item key="index" v-for="m,index in chats" > <messagetext :text="m.text" >
</messagetext> <a class="close" v-on:click="removeMsg(index)">x</a> </b-list-group-item> <div v-if="chats.length==0"> <div class="alert
alert-info"> No Messages </div> </div> <div v-else class="total"> <p><strong>Total Messages: <span><b-badge>{{chats.length}}</b-badge>
</span></strong></p> </div> </b-list-group> </div> </template>
<script> import messagetext from './MessageText'
    export default {
    name: "chatlist", props: ['chats'], components: {
    messagetext }, methods:{
    removeMsg:function (index) {
    this.chats.splice(index,1); }
    }
    }
</script>
<style> .close{
    position: absolute; top: 8px; right: 10px; }
    .total{
    text-align: right; }
    p{
```

```
color: #777; }

</style>
```

In the template we have listed all chats sent by user. This component will expect following props:

1. Chatlist

The chatlist will contain all chat list which is sent from App.vue.

In this component we have imported a component called messagetext, a messagetext is a child component, let's go ahead and create that component:
The component should be in src/components/chat/

```
<template> <span> {{text}}
    {{date}}
    </span> </template>
<script> export default {
    name: "messagetext", props: ['text', 'date']
    }
</script>
<style> </style>
```

This component expects props which are text and date. Text for the message and date is the time when the message is created. This means the created date.

Now go back to the message list component. You will find a method called removeMsg that will remove the message from the app. By using splice method.

Now let's move to Form, this is form for our chat application:

```
<template> <div> <div> <b-form-input v-on:input="alert(1)" v-model="text" type="text" placeholder="Enter message..."
    ></b-form-input> <div> <b-badge pill variant="success">{{text.length}}/30</b-badge> </div> <button v-on:click="validateInput"
class="btn btn-primary" >Send Message</button> </div> <div v-if="isEmptyText" > <b-alert variant="danger" show> Please enter a message
</b-alert> </div> </div> </template>
<script> export default {
    name: "chatform", props: ['save','isEmptyText'], data: function () {
    return {
    text: ""
    }
    }, methods: {
    validateInput: function () {
    this.save(this.text); if (!this.isEmptyText) {
    this.text = ""; }
    }
    }
```

```
    }
</script>
<style> .form-control{
    border-radius: 0; }
    .btn{border-radius: 0; margin-top: 10px;}
</style>
```

This component is built for chat form. The chat form will be needed to create messages. Here are following methods in this form:

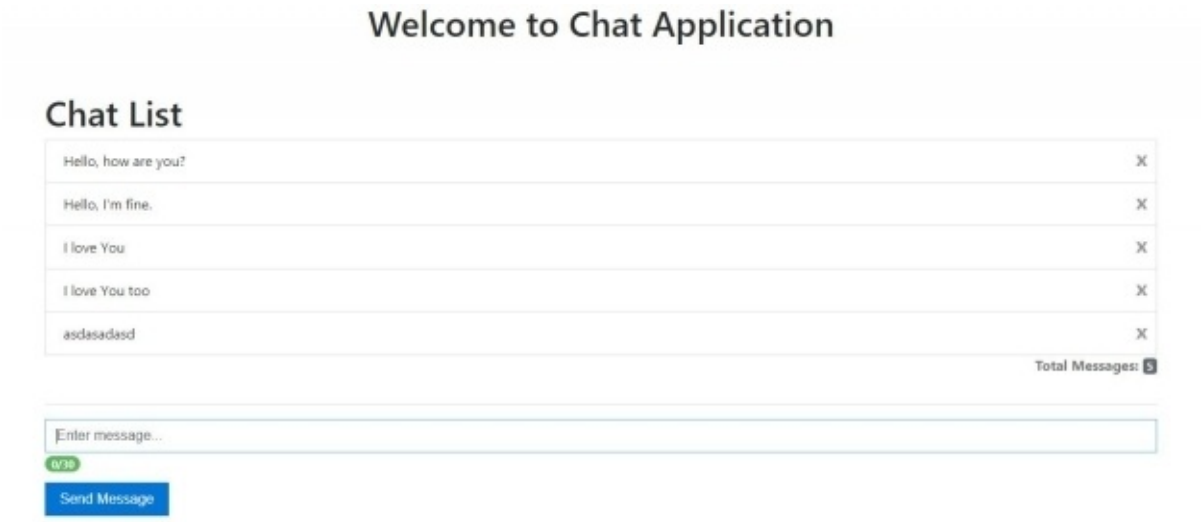
1. validateInput

This method checks if save works then set the text to null so it will not destroy the user experience. The text box needs to be empty when someone sends a message. Go back and read the code and each line. Nothing is harder. We have everything which we have learned earlier in this book.

Now run this app by running:
npm run dev

This will start your application server on http://localhost:8080

On startup you will see something like this:



We have used bootstrap components for our application. You will need to install bootstrap vue to make our application a nice.

Now let's move to a small example called building a like system in Vue JS, this is our third example in which we will create an example. This is very small example to show how to create like system in Vue JS:
Let's include these libraries first:
Bootstrap

```
<link href = "https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel = "stylesheet" type = "text/css" >
```

VueJS

```
<script src = "https://unpkg.com/vue@2.1.3/dist/vue.min.js" > </script>
```

Create the basic app and call like component with prop likes(The initial count):

```
<div id="app" > <h1>{{message}}</h1> <like likes="0" ></like> <like likes="0" ></like> <like likes="0" ></like> <like likes="0" ></like> <like likes="0" ></like> <like likes="0" ></like> <like likes="0" ></like> </div>
var app = new Vue({
    el: "#app", data: {
    message: "Like System"
    }
});
```

Like button Component in the same page(Find More here):

```
<div style = "display: none;" >
<template id = "like-template" > <div> <button class = "btn btn-info" v-on:click = "count()" > Like {{ likes }}</button> </div> </template>
</div>
Vue.component("like",{
template : "#like-template", props : ["likes"], data: function () {
    return {
    likes: 0
    }; },
methods: {
    count: function () {
    var a = parseInt(this.likes) + 1; this.likes = a; }
}
});
```

In the template. `v-on:click` event is attached to button. When someone clicks to button, `count()` function will run and it will add 1 to current likes. Here is the complete demo:
Final Markup:

```
<div id="app" > <h1>{{message}}</h1> <like likes="0" ></like> <like likes="0" ></like> <like likes="0" ></like>
<like likes="0" ></like> <like likes="0" ></like> <like likes="0" ></like> <like likes="0" ></like>
</div> <div style="display: none;" >
<template id="like-template" > <div> <button class="btn btn-info" v-on:click="count()" >Like {{ likes }}</button> </div> </template> </div>
Vue.component("like",{
template : "#like-template", props : ["likes"], data: function () {
    return {
    likes: 0
    }; },
methods: {
    count: function () {
```

```
    var a = parseInt(this.likes) + 1; this.likes = a; }
}
});
var app = new Vue({
el: "#app", data: {
    message: "Like System"
}
});
```

You can complete demo and article from here.

Chapter conclusion

In this chapter we have learned how to install Vue JS CLI version and created three examples. In this chapter we learn how to create Vue JS App in both CLI and Vue JS CDN based.

Here are some questions about Vue JS.

1. What is difference between Vue JS and React JS?
2. What is difference between Vue JS CLI and Vue JS CDN based?

What is difference between Vue JS and React JS?

Answer: Both are View based libraries, usage is different. We need to include to react and react-dom to make React, but in vue js we need to include only Vue JS to make it work. Both are somehow different somehow same. But Vue JS is easy to learn and implement in any web application. If you are in hurry to start an application, you will need to

What is difference between Vue JS CLI and Vue JS CDN based?

Usage is different and but working is same. Vue JS CLI is used in Node JS. Vue JS in CDN based is used to include Vue JS easily and you can use Vue JS in any application by just using CDN. You can find VUE JS in below link:

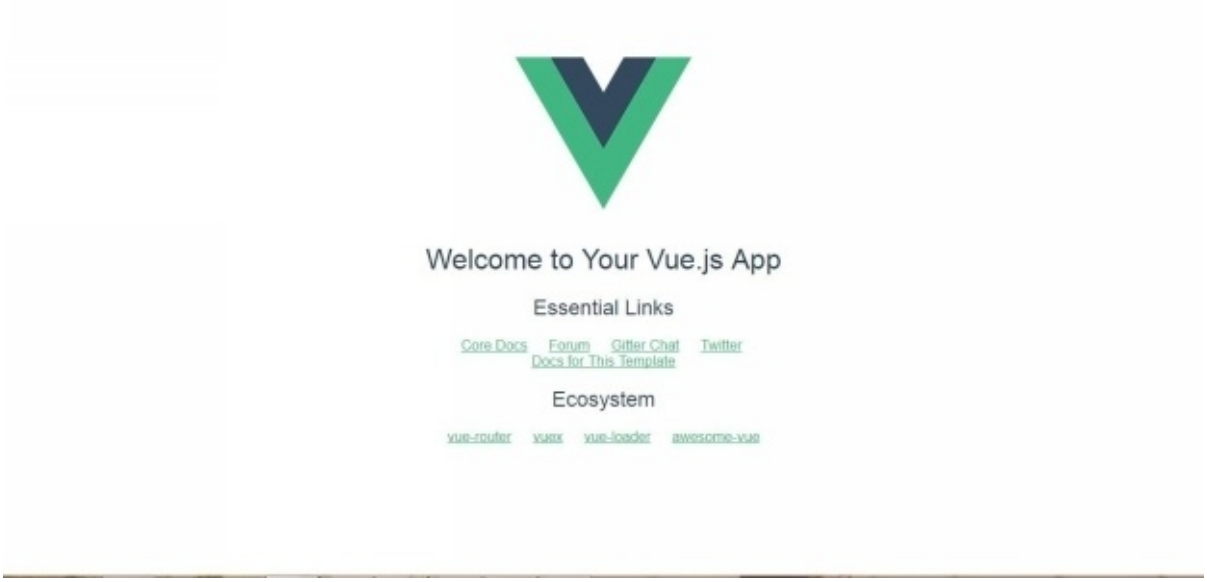
```
<script src = "https://unpkg.com/vue@2.1.3/dist/vue.min.js" > </script>
```

Now lets move to chapter 8

Chapter 8

Guide to Awesome Vue and Vue Automation

So far we have learned everything about Vue JS and you should know how to build Vue JS apps in mean time. So far, we have covered many important tasks in Vue JS. In Vue JS we can build larger apps to solve our world problems. Vue JS is backed by little team of great JavaScript developers. And Vue JS competing with large framework like ReactJS and Angular JS. In this chapter we will learn work with Awesome Vue. Actually the awesome Vue is not a library, this contains list of all libraries for Vue JS you can also resources for all Vue JS. In this chapter we will implement three (3) resources or libraries in our Vue JS application. Now let's go ahead and create a fresh Vue JS CLI application. You first screen should look like this:



After installation of Vue JS CLI, your app should look like above one. It's this means you have installed your Vue JS CLI. In the first example we will install bootstrap four 4 and you can find bootstrap here. Bootstrap is very popular library. Bootstrap is also available for Vue JS, we can install bootstrap in our Vue JS app by:

```
npm install --save bootstrap-vue
```

Well this will install Vue bootstrap in our Vue JS application and we can import Bootstrap by:

```
import BootstrapVue from 'bootstralp-vue'; Vue.use(BootstrapVue);
```

This will import bootstrap and will set bootstrap in action by use function. Let's start with a button, how to create a button? It is simple. Now go ahead to App.vue, add a button to it:

So far we have imported Bootstrap and We need to import bootstrap into our application, we can import bootstrap by: `import 'bootstrap/dist/css/bootstrap.css'`

```
import 'bootstrap-vue/dist/bootstrap-vue.css'
```

This will import bootstrap-vue and bootstrap in to our application, now go ahead and update App.vue file and add the a button to your application, you can add a button to your application by:

```
<b-button variant="info"> Button Info </b-button>
```

So your App.vue should look like this:

```
<template> <div id="app"> <h1>Vue</h1> <b-button variant="info"> Button Info </b-button> </div> </template>
<script> import Hello from './components/Hello'

export default {
name: 'app', components: {
    Hello }
}
</script>
<style> #app {
padding: 40px; text-align: center; }
</style>
```

This will greet us with something like this:

This means we have added a button to our application. Now let's go ahead add some other examples. You can find all examples here on this page: https://bootstrap-vue.js.org/docs/components/alert/

All components starts from alert so you can check on sidebar to see all components. Now go ahead and add an alert after the button,

```
<b-alert variant = "danger" dismissible show = "true" > Oops! Something went wrong on our side.
</b-alert>
```

Variant shows the type of alert. Danger, Info, Success and warning. Your app.vue should look like this:

```
<template> <div id="app"> <h1>Vue</h1> <b-button variant="info"> Button Info </b-button> <br /> <br /> <b-alert variant="danger"
dismissible show="true" > Oops! Something went wrong on our side.
    </b-alert> </div> </template>
<script> import Hello from './components/Hello'

export default {
name: 'app', components: {
    Hello }
}
</script>
<style> #app {
padding: 40px; }
</style>
```

Your app should look like this:

We have added alert to our application. Now let’s move and add another package to your Vue Application. We will add font awesome to our Vue Application.

What is Font Awesome?

Font awesome is an icon library. This library contains more than 300 icons with outline support. Font Awesome is very popular library for creating websites with Icon support. In this section, we will install font awesome for Vue JS and we will use it in our Vue JS Application. Now go ahead and install Vue Font Awesome by:

```
npm install vue-awesome
```

This will install font awesome for Vue JS in your application. After installation go to main.js and import the icons:

```
import 'vue-awesome/icons'
import Icon from 'vue-awesome/components/Icon'
Vue.component('icon', Icon)
```

This will assign a component icon and we can use the icon by: `<icon name="flag"></icon>`

Now go to App.vue and add an icon in template: `<p> Go and find a <icon name="flag"></icon> In your home </p>`

This will add an icon to our application. If everything compiles successfully you will be greeted with something like this:

Your App.vue should look like this:

```
<template> <div id="app" > <h1>Vue</h1> <b-button variant = "info" > Button Info </b-button> <br /> <br /> <p> Go and find a <icon name = "flag" > </icon> in your home </p> <br /> <br /> <b-alert variant = "danger" dismissible show = "true" > Oops! Something went wrong on our side.
    </b-alert> </div> </template>
<script> import Hello from './components/Hello'

export default {
name: 'app', components: {
    Hello }
}
</script>
<style> #app {
padding: 40px; }
</style>
```

Now let's move to other options for the Icon component:

Scale:

```
<icon name="language" scale="3"></icon>
```

This will print: 

Spin:

```
<icon name="refresh" spin scale="3" ></icon>
```

This will create a moving Icon:

This is what we have and we used icons in our Vue JS app. Now let’s move to Vue JS automation.

Vue JS automation

Automatic testing is important in every App. App needs to test itself while application is running. For automatic Testing we will use TestCafe for this task. To get started with Test Café you will need to install Test Café by following command:

```
npm install -g testcafe
```

This will install test café in your application.

Creating a Test

TestCafe enables you to compose tests utilizing TypeScript or JavaScript (with its cutting edge highlights like async/anticipate). By utilizing TypeScript to compose your TestCafe tests, you get the upsides of specifically dialects, for example, rich coding help, effortless adaptability, registration you-write code confirmation, and considerably more.

To create a test you need to create a JS file anywhere.

Import Test:

```
import { Selector } from 'testcafe';
```

Declare a fixture:

```
fixture `Getting Started`
```

In this instructional exercise, you will make a test for the http://devexpress.github.io/testcafe/illustration test page. Determine this page as a begin page for the installation by utilizing the page work.

```
fixture `Getting Started`
    .page `http://devexpress.github.io/testcafe/example`;
```

Create a test function

```
import { Selector } from 'testcafe';
fixture `Getting Started`
    .page `http://devexpress.github.io/testcafe/example`;
test('My first test', async t => {
    // Test code });
```

Running the test

Well you can run a test by: `testcafe chrome test1.js`

This will open automatically a browser and start executing a test.

Running a test in VueJS

In this section we will learn how to run a test in Vue JS. End-to-end testing is a standout amongst the most significant devices in your testing armory, enabling you to mimic what your client would do as they travel through your application and decide whether your application is reacting accurately to that. Shockingly, it's likewise a standout amongst the most troublesome and tedious test techniques also, and the typical apparatuses to do as such require an OK measure of arrangement and setup, additionally convoluting the procedure. Gratefully, there are some generally straightforward arrangements. Here we'll exhibit one of them, TestCafe, and demonstrate to you proper methodologies to do end-to-end testing with your Vue.js application. (All things considered, you can utilize these techniques with any structure or site.)

Installation

Not at all like conventional arrangements that for the most part include a close unmanageable measure of conditions, for example, Selenium/WebDriver + Browser Drivers + Client Libraries, the total of TestCafe is hub based and installable in one bundle. It's likewise a zero-design instrument. The required alternatives are passed through the summon line. All things considered, it's best utilized through NPM contents.

To start utilizing it in your Vue application, introduce testcafe by means of Yarn or NPM. You may likewise consider utilizing the vuepack layout for vue-cli in case you're beginning another venture.

```
# Yarn $ yarn add testcafe -D

# NPM
$ npm install testcafe --save-dev
```

Setup:

It is accepted from here that your application has an improvement server that can be keep running with npm run dev. webpack-dev-server works fine.

Add another content to your package.json contents question that begins testcafe.

```
"scripts": {
"dev": "...", "test": "testcafe all tests/*.test.js --app \"npm run dev\" --app-init-delay 10000 -S -s screenshots", }
```

Your First Test We should accept for a minute that the sum of your application is just a passage component inside <body> that contains the words "Hello World!". Here's the way we would guarantee that is without a doubt the case.

```
// A ficture must be declared.
fixture(`Index page`) // load url for development server .page('http://localhost:8080');
// new test test('Body > Paragraph contains "Hello World!"', async testController => {
// select a content const paragraphSelector = await new Selector('body > p');
// Assert that the inner text of the paragraph is "Hello World!"
await testController.expect(paragraphSelector.innerText).eql('Hello World!'); });
```

Controlling User Input

Presently to do any half-conventional end-to-end testing, you should have the capacity to mimic client activities and information. Like any great testing suite, TestCafe gives the vital strategies to deal with this utilization case.

```
test('Typing in an input', async testController => {
// select input const inputSelector = await new Selector('body > input[type="text"]');
await testController .typeText(inputSelector, 'World!') .click(inputSelector, { caretPos: 0 }) .keyPress('H e l l o space')
.expect(inputSelector.value).eql('Hello World!'); });
```

Now we have implemented real time test to our Vue JS.

Here is a question from you.

What other libraries did you know out there?

Answer: In case you did not about the libraries, here are following JavaScript libraries:

1. Jasmine
2. Qunit
3. Mocha
4. Buster.js
5. YUI Test

We have completed our 8 chapter and we have learned how to use packages in our Vue JS application. In the next chapter we will learn how to use Vue JS with other frameworks. In the next chapter we will implement Vue JS with React JS.

Chapter 9

Integrating with external framework

We have learned all basic things of Vue JS and learned how to make automatic testing in Vue JS. In this chapter we will use Vue JS with React JS. In this chapter we will include Vue JS with React JS. Let me tell you about React JS:
Sometimes React or ReactJS anyway it is React s. ReactJS is a JavaScript library to create User Interfaces. React was first created by Jordan Walke software engineer on Facebook. React was deployed on Facebook news feed on 2011 and later on 2012, it was deployed on Instagram. It allows developers to create large scale applications. React is currently deployed on NetFlix, Facebook, Instagram, Airbnb, Wallmart, and Imgur. Initially, ReactJS was not open source. In May 2013, it was open sourced. Its main goal is to allow developers to create Web Apps that are fast, simple and scalable. Here are some reasons to choose ReactJS

- One-way data flow.
- Virtual Dom.
- JSX.
- JavaScript Expressions.
- Routing.

Let’s create an example in both Reeact and Vue JS. In the example we will create buttons for Vue JS and React JS that will give an alert with value 1.

First you need to include React, ReactDOM and Vue JS. You can find React, ReactDOM and Vue JS in following links:
https://unpkg.com/vue@2.4.2/dist/vue.js
https://cdnjs.cloudflare.com/ajax/libs/react/15.3.1/react.min.js
https://cdnjs.cloudflare.com/ajax/libs/react/15.3.1/react-dom.min.js
https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.24/browser.js

Now include these libraries and create an app:

```
<html> <head> <title>Welcome to my App</title> </head> <body> <p> Welcome to App </p> <div id="react-app" >
    </div> <div id="vue-app"> Hello my name is {{name}}
    </div> </body> <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.24/browser.js" ></script> <script
```

```
type="text/javascript" src="react.js" ></script> <script type="text/javascript" src="reactdom.js" ></script>
    <script type="text/javascript" src="vue.js" ></script> <script type="text/javascript"> // vue new Vue({
    el: "#vue-app", data: {
    name: "John Doe"
    }
    }) </script> <script type="text/babel"> var Hello = React.createClass({
    render: function() {
    return (
    <div> <h1>Hello World</h1> <p>This is some text</p> </div> ) }
    });
    ReactDOM.render(
    <Hello name="World" />, document.getElementById('react-app') ); </script> </html>
```

This is your application and we have included React, ReactDOM and Vue JS in our application. If this compiles successfully it means we have integrated with external frameworks. Now add a buttons to get alert on both libraries. After adding buttons we our code will look like this:

```
<html> <head> <title>Welcome to my App</title> </head> <body> <p> Welcome to App </p> <div id="react-app" >
    </div> <div id="vue-app"> Hello my name is {{name}}

    <button v-on:click="alert" >Click Me</button> </div> </body> <script type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.24/browser.js" ></script> <script type="text/javascript" src="react.js" ></script>
<script type="text/javascript" src="reactdom.js" ></script>
    <script type="text/javascript" src="vue.js" ></script> <script type="text/javascript"> // vue new Vue({
    el: "#vue-app", data: {
    name: "John Doe"
    }, methods: {
    alert: function () {
    alert(1); }
    }
    }) </script> <script type="text/babel"> var Hello = React.createClass({
    alert: function () {
    alert(1); }, render: function() {
    return (
    <div> <h1>Hello World</h1> <p>This is some text</p> <button onClick={this.alert} >Click Me</button> </div> ) }
    });
    ReactDOM.render(
    <Hello name="World" />, document.getElementById('react-app') ); </script> </html>
```

You can see in both applications I have appointed functions to buttons and in functions I have made an alert with value 1. If everything compiles successfully you will be greeted with something like this:
PTO

After clicking on both buttons you will be greeted with alert.

As we have ended our chapter, here are some questions.

1. How to implement jQuery with these two libraries?
2. How to implement Backbone in this project?

How to implement jQuery with these two libraries?

jQuery is a quick, little, and highlight rich JavaScript library. It makes things like HTML archive traversal and control, occasion taking care of, activity, and Ajax considerably more straightforward with a simple to-utilize API that works over a large number of programs. As we included different libraries you can include jQuery by: <script type="text/javascript" src="jquery.js" ></script>

Then you can write jQuery code now.

How to implement Backbone in this project?

As we did for jQuery, we need to include Backbone, let me tell you what is backbone? Backbone.js gives structure to web applications by providing models with key-value binding and custom events, collections with a rich API of enumerable functions, views with declarative event handling, and connects it all to your existing API over a RESTful JSON interface.

We can include backbone to our application then we can get started with our application.