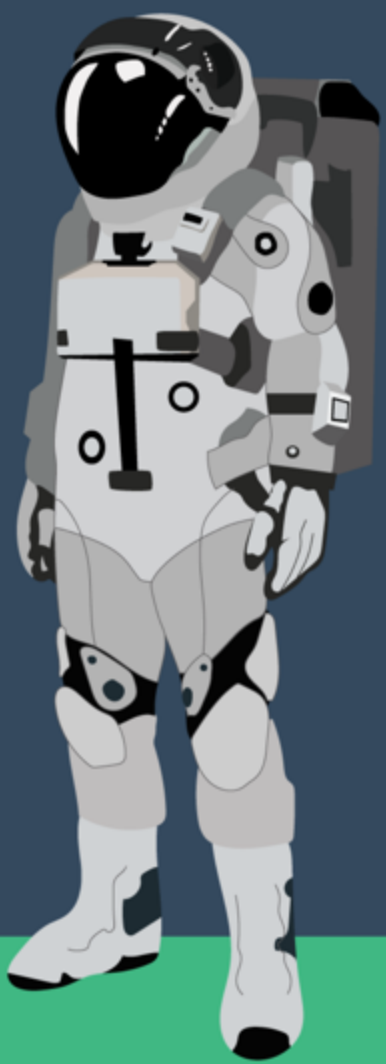
Oleksandr Kocherhin

Mastering Vue.js



Mastering Vue.js

Oleksandr Kocherhin

January, 2018

Contents

Chapter 1

Introduction

1.1 Javascript and Frontend

You chose nice time for learning frontend technologies. They are fresh, evolving really fast and are needed in any company with online projects. Improving your skills in frontend can provide you with a great opportunity to find a good and well-paid job.

1.2 Why Vue?

Building frontend nowadays is as difficult as never before. All business logic was moved from backend side to frontend. This makes architecture really difficult. Earlier it was enough to use jQuery and to do just a bit of javascript on page, but now it's really a tough way to build big and scalable project using something as simple as jQuery. There are a lot of frameworks nowadays which are aimed at achieving the best results with creating project of different sizes in easy and pleasant way. I would say that Vue is one of the best in this competition. It's quite new (so you will be on the cutting edge) but there already exist a lot of production applications. Vue has nice learning curve (but you still have a lot to learn) but also takes best from Angular and React frameworks to create projects with different difficulty level. It may be as easy as writing your own blog and may be something really difficult as ecommerce online shop.

1.3 Who is this book for?

I have about 8 years of frontend experience and want to share my knowledge with you. I really like Vue framework and use it every day to create nice and scalable single page applications. In this book we will start from scratch (but you should still have some knowledge of javascript) and finish with understanding of all features of Vue. We will go step by step increasing complexity in each chapter. In this book you won't find only theory. All examples for each chapter are real. We will also build three applications with different complexity while you will be reading the book. I highly recommend you not only to read the book, but also to try to build all examples and projects step by step with me. In this case you'll receive enough experience to build real applications by yourself later.

1.4 Technologies

This book was written with Vue 2.5.9 and will work with any environment. All projects that we will implement in the book you can find under https://github.com/masteringvuejs/masteringvuejs-projects/tree/master/tracks-admin-page

Chapter 2

Creating first Vue application

In this chapter we will create our first Vue application. But we should install Vue first.

You can install Vue in different ways. The easiest way is to include in your html page Vue script from CDN. You can find CDN link in Vue installation page. `package.json`

You can just put it at the end of the body and you are good to go.

`index.html`

```
<html>
  <head>
  </head>
  <body>
    ...
    <script src="https://cdn.jsdelivr.net/npm/vue
       "></script>
  </body>
</html>
```

There are of course other ways of installation, for example from npm package or by vue-cli but they are more difficult and we will use them later while building more complex apps.

For now it's enough to use just script from CDN.

Let's add also a script where we will write our js to our index.html page.

index.html

```
<body>
  ...
  <script src="https://cdn.jsdelivr.net/npm/vue
     "></script>
  <script src="code.js"></script>
</body>
```

Now we should add file `code.js` in the same folder as `index.html`. To start using Vue we always create an instance of Vue.

code.js

```
new Vue()
```

Inside our constructor we should pass properties for Vue as an object. And the most important property in object is `el`. It indicates which part of our DOM this Vue instance should manage.

code.js

```
new Vue({
  el: '#root'
})
```

We set el as an element with id `root` and now we should add such DOM element in our html page.

index.html

```
<body>
  <div id='root'></div>
  ...
</body>
```

It's time to render some data on page via Vue. And for this we should provide data field inside object. Let's create property `message` inside our `data`.

code.js

```
new Vue({
  el: '#root',
  data: {
    message: 'Hello Vue!'
  }
})
```

Now we want to render this property in our DOM. We can use special Vue syntax with double curly braces for this.

index.html

```
<div id='root'>
  {{message}}
</div>
```

Since Vue controls root element now, it sees the special Vue syntax, reads our message property from object and renders it instead of double curly braces.

We can see now in browser that our message `Hello Vue!` was successfully rendered. It means that everything works and we have just built our first Vue app in several minutes.

Chapter 3

Methods in Vue

While creating our first Vue application we used double curly braces in template. This is the way to say to Vue, what variable we want to render.

But it's also possible to use function that will return some value, which we can use in our template. And we can create any function that we want inside `methods` field.

`code.js`

```
new Vue({
  el: '#root',
  methods: {
    getMessage () {
      return 'Hello Method!'
    }
  }
})
```

Now we can call this function inside our template and our string is rendered.

`index.html`

```
<div id='root'>
  {{getMessage()}}
</div>
```

And we can even use our variables from `data` inside any of our functions. They are all accessible inside `this` property.

code.js

```
new Vue({
  el: '#root',
  data: {
    message: 'Hello Vue!'
  },
  methods: {
    getMessage () {
      return this.message
    }
  }
})
```

Here we are using `this.message` inside our method `getMessage`. And everything is working as before.

Very often we need to attach some event to the button and change some other part of the DOM. In Vue we have a lot of standard directives to use. If you worked with Angular 1 before you know what the directives are for sure. In Vue they are working exactly in the same way.

Basically, directives are just pieces of javascript code, sometimes with template which is tied to element on which we are using our directive.

And there is a directive `v-on` which helps us to bind events to elements. The most popular event is of course `click`. Let's add button and change our variable inside Vue.

index.html

```
<button v-on:click='onSubmit'>Submit</button>
```

And we should add `onSubmit` method now.

code.js

```
methods: {
  getMessage () {
    return this.message
  },
  onSubmit () {
    this.message = 'Submitted'
  }
}
```

Now when we click on our submit button the message on the screen is changed automatically to `Submitted`.

There is a small trick here. As writing `v-on:click` takes quite long and we are using it a lot there is an alias for this. We can use `@` instead of `v-on:`

index.html

```
<button @click='onSubmit'>Submit</button>
```

As you can see, everything works as before.

Chapter 4

Getting event inside method

Quite often we need to get access to event when it happens. And in Vue it's quite easy to do it. Let's add `mousemove` event on our block with message.

`index.html`

```
<div @mousemove='hoverMessage'>
  {{getMessage()}}
</div>
```

Now we can add this method. As a first parameter we will get event that happens. Let's check it in console.

`code.js`

```
methods: {
  ...
  hoverMessage (event) {
    console.log('hoverMessage', event)
  }
}
```

Now on mousemove we are getting our mouseevent in console. Let's render X and Y from our event on the screen to see the position of cursor.

First we should create x and y with default values.

`code.js`

```
data: {
  ...
  x: 0,
  y: 0
}
```

Now let's assign them when our mousemove happens.

code.js

```
hoverMessage (event) {
  this.x = event.clientX
  this.y = event.clientY
}
```

Now let's render them on the screen.

index.html

```
<div>
  Position: {{x}} | {{y}}
</div>
```

As you can see in browser, when we hover on our `Hello Vue` the message position of our cursor changes and we render new positions on the screen.

Chapter 5

Listening to keyboard events

Quite often we need to control keyboard events in javascript. For example we can use space for play/pause of video or escape for going out of edit mode.

And the is a nice directive `v-on:keyup` to listen to the events from keyboard. Let's try it through a real example. We can create input and handle keyboard events of what the user is typing.

First we should add input in our html page.

index.html

```
<input type='text' />
```

Now let's add a keyup event for it.

index.html

```
<input type='text' v—on:keyup='onKeyUp' />
```

`v-on:keyup` here is a method from Vue. We can also use shorthand for `v-on`.

index.html

```
<input type='text' @keyup='onKeyUp' />
```

We should add `onKeyUp` method in Vue instance now.

code.js

```
onKeyUp (event) {
  console.log('onKeyUp', event)
}
```

While typing in browser we see our event in console. We can read a key from it and do whatever we want with it.

But let's say that we want to listen only to a single key, for example, to escape key. In Vue there is an easy way to do it. We can just set in keyup what key we want to listen to. This is called a modifier.

index.html

```
<input type='text' @keyup.space='onKeyUp' />
```

As we can see now, our `onKeyUp` method is being fired only when we are hitting space. We can add any amount of keys with modifiers.

index.html

```
<input type='text' @keyup.space.esc='onKeyUp' />
```

In browser now we will see console.log only by hitting escape or space.

Chapter 6

Basic styling in Vue

When we want to style elements in Vue, we are doing it in the same way as for normal html and css.

Let's add a div with class `test` and apply some styles for this element in head section.

index.html

```
<head>
  <style>
    .test {
      background: green;
    }
  </style>
</head>
<body>
  <div class='test'>Test</div>
  ...
</body>
```

As we can see styles are applied. But what if we want to apply styles based on some condition in Vue? We can use `v-bind` syntax for this.

index.html

```
<div v-bind:class="{test: isActive}">Test</div>
```

So here we used `v-bind` directive with class and passed a string with object inside. A key of the object is the class that we want to apply and value is a boolean from Vue instance.

To make this working we should add `isActive` variable in Vue.

code.js

```
data: {
  isActive: true
}
```

In browser our Test container is rendered with green background now. If we change `isActive` to false, our class won't be applied.

It's also possible to store classes as string in Vue and then render inside `v-bind`.

Let's add `myTestClass` variable inside data.

code.js

```
data: {
  myTestClass: 'test'
}
```

And to pass this class inside our div we need to bind it to this variable.

index.html

```
<div v-bind:class='myTestClass'>Test</div>
```

As writing `v-bind` each time is not that efficient, there is also an alias for this. We can simply use colon.

index.html

```
<div :class='myTestClass'>Test</div>
```

Sometimes we want to apply several classes together. In Vue we can use array notation for this.

Let's add a new variable with class name inside data object.

code.js

```
data: {
  myTestClass: 'test',
  errorClass: 'error'
}
```

Then let's apply both classes in our template.

index.html

```
<div :class='[myTestClass, errorClass]'>Test</div
    >
```

In browser we see, that our container has not only green background but also red color.

It's also possible to apply styles inline, without classes. We can write a bunch of styles as an object and apply them with style directive to our element.

First let's create an object with styles.

code.js

```
data: {
  myTestStyles: {
    background: 'green'
  }
}
```

Don't forget that you need to write all styles as strings. So wrap them in single quotes.

We can apply it to our container now.

index.html

```
<div :style='myTestStyles'>Test</div>
```

We can also apply several blocks together with array notation.

Let's add one more block with styles.

code.js

```
data: {
  myTestStyles: {
    background: 'green'
  },
  errorStyles: {
    color: 'red'
  }
}
```

Then apply it, too

index.html

```
<div :style='[myTestStyles, errorStyles]'>
  Test
</div>
```

As you can see, we got the same result as before. But the main difference is that with style all styles are applied inline and with class as normal classes.

And there is one more hidden benefit of Vue. When you apply some styles that require vendor prefixes (for example transform) Vue will automatically apply all prefixes.

Chapter 7

Conditions in Vue

Very often we need to manipulate our DOM to hide and show some elements on the screen. We can do it in several ways with Vue.

Let's first add `Submit` button, `isVisible` variable and `onSubmit` method.

`index.html`

```
<button @click='onSubmit'>Submit</button>
```

`code.js`

```
data: {
  isVisible: false
},
methods: {
  onSubmit () {
    this.isVisible = true
  }
}
```

Now when we click on button, `isVisible` variable will be set to true.

Now let's add block with text, that we want to show only in case `isVisible` variable equals true.

`index.html`

```
<div v-show='isVisible'>Text</div>
```

Here we used special `v-show` directive which applies `display:none` on element, based on `isVisible` value. When we inspect our element we see that it is always rendered, but has `display:none` styles by default. When we

click on button, `display:none` is being removed.

Sometimes we want to remove element from page completely, not just hide it. We can use `v-if` directive for this.

`index.html`

```
<div v-if='isVisible'>Text</div>
```

It works in the same ways as `v-show`, but removes DOM element from page completely.

There is also `v-else` directive that you can use together with `v-if`.

Let's create new div after first div.

`index.html`

```
<div v-else>Else</div>
```

As you can see, else block is rendered by default now. It looks quite useful, but you should be careful with it. If you move this element, or put one more element between them, it won't work. Thats why I always use `v-if` block with negation but not `v-else`.

`index.html`

```
<div v-if='!isVisible'>Else</div>
```

In this chapter we tried the most popular ways of working with conditions in Vue.

Chapter 8

Loops in Vue

Basically all frontend is about rendering lists. And we can do it easily in Vue using `v-for` directive.

But we need to create our `data` inside Vue first. Let's assume that we have a list of users and want to render them on the screen.

`code.js`

```
data: {
  users: [
    {id: 1, name: 'Alex', isActive: true},
    {id: 2, name: 'John', isActive: true},
    {id: 3, name: 'Jack', isActive: false}
  ]
}
```

Now let's render them in template using `v-for` directive.

`index.html`

```
<ul>
  <li v-for='user in users'>
    {{user.name}}
  </li>
</ul>
```

As you can see, we are passing construction `user in users`. `Users` here is our variable from `data`. `User` is just a reference to each record inside our array. We can name it as we want. It just helps us to get access to each element of the array. We can use it inside `v-for` to render information of each element.

As you can see it's really easy to render lists with Vue inside a template.

We can also render more markup inside each element if we want.

index.html

```
<div v-for='user in users'>
  <h1>{{user.name}}</h1>
  <div>
    isActive: {{user.isActive}}
  </div>
</div>
```

Sometimes we need to get index of each element in the array. To do this we can change a bit our `v-for` directive.

index.html

```
<div v-for='(user, index) in users'>
  <div>
    Index: {{index}}
  </div>
</div>
```

As you can see, we wrote `user, index` like they are parameters in a function. In such way we have access to index of each element inside `v-for`.

Sometimes our data can be more difficult. Let's say each of our users has todos property, that we want to render. We can just add them to a first user to test.

code.js

```
{
  id: 1,
  name: 'Alex',
  isActive: true,
  todos: [
    'Learn Vue',
    'Create first Vue app'
  ]
}
```

Now we can put `v-for` inside `v-for` to render todos for each user.

index.html

```
<div v-for='user in users'>
  <h1>{{user.name}}</h1>
  <div v-for='todo in user.todos'>
    {{todo}}
  </div>
</div>
```

In browser we can see the list of users with their todos under each name.

There is also a possibility in `v-for` to be applied not on an array but on a number. Than `v-for` will render all elements from 1 to this number.

index.html

```
<div v-for='number in 10'>
  {{number}}
</div>
```

Now on the screen we can see the list of numbers from 1 to 10.

In this section we checked the most popular ways of using `v-for` to render lists.