



The Majesty of Vue.js

Alex
Kyriakidis

Kostas
Maniatis

The Majesty Of Vue.js

**© 2015 - 2016 Alex Kyriakidis, Kostas
Maniatis and Evan You**

Contents

Introduction	i
About Vue.js	ii
Vue.js Overview	ii
What people say about Vue.js	ii
Comparison with Other Frameworks	iv
Angular	iv
React	v
Ember	vii
Polymer	viii
Riot	ix
Welcome	x
About the Book	x
Who is this Book for	x
Get In Touch	x
Homework	xi
Errata	xi
Conventions	xi
Vue.js Fundamentals	1
1. Install Vue.js	2
1.1 Standalone Version	2
1.1.1 Download from vuejs.org	2
1.1.2 Include from CDN	2
1.2 Download using NPM	2
1.3 Download using Bower	3
2. Getting Started	4
2.1 Hello World	4
2.2 Two-way Binding	6
2.3 Comparison with jQuery.	7
2.4 Homework	9

CONTENTS

3. A Flavor of Directives	10
3.1 v-show	10
3.2 v-if	13
3.2.1 Template v-if	14
3.3 v-else	15
3.4 v-if vs. v-show	18
3.5 Homework	19
4. List Rendering	20
4.1 Install & Use Bootstrap	20
4.2 v-for	23
4.2.1 Range v-for	23
4.3 Array Rendering	25
4.3.1 Loop Through an Array	25
4.3.2 Loop Through an Array of Objects	27
4.4 Object v-for	30
4.5 Filtered Results	32
4.6 Ordered Results	38
4.7 Custom Filter	42
4.8 Homework	45
5. Interactivity	46
5.1 Event Handling	46
5.1.1 Handling Events Inline	46
5.1.2 Handling Events using Methods	48
5.1.3 Shorthand for v-on	49
5.2 Event Modifiers	50
5.3 Key Modifiers	54
5.4 Computed Properties	54
5.4.1 Using Computed Properties to Filter an Array	59
5.5 Homework	62
6. Components	63
6.1 What are Components?	63
6.2 Using Components	63
6.3 Templates	65
6.4 Properties	66
6.5 Reusability	69
6.6 Altogether now	72
6.7 Homework	81
7. Class and Style Bindings	83
7.1 Class binding	83
7.1.1 Object Syntax	83

CONTENTS

7.1.2	Array Syntax	87
7.2	Style binding	88
7.2.1	Object Syntax	88
7.2.2	Array Syntax	89
7.3	Bindings in Action	90
7.4	Homework	92

Consuming an API 93

8.	Preface	94
8.1	CRUD	94
8.2	API	94
8.2.1	Download Book's Code	95
8.2.2	API Endpoints	96
9.	Working with real data	98
9.1	Get Data Asynchronous	98
9.2	Refactoring	101
9.3	Update Data	104
9.4	Delete Data	106
10.	Integrating vue-resource	109
10.1	Overview	109
10.2	Migrating	110
10.3	Enhancing Functionality	111
10.3.1	Edit Stories	111
10.3.2	Create New Stories	114
10.3.3	Store & Update Unit	120
10.4	JavaScript File	121
10.5	Source Code	122
10.6	Homework	127
10.6.1	Preface	127
10.6.2	API Endpoints	127
10.6.3	Your Code	128

Introduction

About Vue.js

Vue.js Overview

Vue.js (pronounced /vju:/, like view) is a library for building interactive web interfaces. The goal of Vue.js is to provide the benefits of **reactive data binding** and **composable view components** with an API that is as simple as possible.

Vue.js itself is not a full-blown framework - it is focused on the view layer only. It is therefore very easy to pick up and to integrate with other libraries or existing projects. On the other hand, when used in combination with proper tooling and supporting libraries, Vue.js is also perfectly capable of powering sophisticated Single-Page Applications.

If you are an experienced frontend developer and want to know how Vue.js compares to other libraries/frameworks, check out the [Comparison with Other Frameworks](#) chapter.

If you are interested to learn more information about Vue.js' core take a look at [Vue.js official guide](#)¹.

What people say about Vue.js

“Vue.js is what made me love JavaScript. It’s extremely easy and enjoyable to use. It has a great ecosystem of plugins and tools that extend its basic services. You can quickly include it in any project, small or big, write a few lines of code and you are set. Vue.js is fast, lightweight and is the future of Front end development!”

—*Alex Kyriakidis*

“When I started picking up Javascript I got excited learning a ton of possibilities, but when my friend suggested to learn Vue.js and I followed his advice, things went wild. While reading and watching tutorials I kept thinking all the stuff I’ve done so far and how much easier I could have done them if I had invest time to learn Vue earlier. My opinion is that if you want to do your work fast, nice and easy Vue is the JS Framework you need. “

—*Kostas Maniatis*

¹<http://vuejs.org/guide/overview.html>

“Mark my words: Vue.js will sky-rocket in popularity in 2016. It’s that good.”

— **Jeffrey Way**

“Vue is what I always wanted in a JavaScript framework. It’s a framework that scales with you as a developer. You can sprinkle it onto one page, or build an advanced single page application with Vuex and Vue Router. It’s truly the most polished JavaScript framework I’ve ever seen.”

— **Taylor Otwell**

“Vue.js is the first framework I’ve found that feels just as natural to use in a server-rendered app as it does in a full-blown SPA. Whether I just need a small widget on a single page or I’m building a complex Javascript client, it never feels like not enough or like overkill.”

— **Adam Wathan**

“Vue.js has been able to make a framework that is both simple to use and easy to understand. It’s a breath of fresh air in a world where others are fighting to see who can make the most complex framework.”

— **Eric Barnes**

“The reason I like Vue.js is because I’m a hybrid designer/developer. I’ve looked at React, Angular and a few others but the learning curve and terminology has always put me off. Vue.js is the first JS framework I understand. Also, not only is it easy to pick up for the less confidence JS’ers, such as myself, but I’ve noticed experienced Angular and React developers take note, and liking, Vue.js. This is pretty unprecedented in JS world and it’s that reason I started London Vue.js Meetup.”

— **Jack Barham**

Comparison with Other Frameworks

Angular

There are a few reasons to use Vue over Angular, although they might not apply for everyone:

- Vue.js is much simpler than Angular, both in terms of API and design. You can learn almost everything about it really fast and get productive.
- Vue.js is a more flexible, less opinionated solution. That allows you to structure your app the way you want it to be, instead of being forced to do everything the Angular way. It's only an interface layer so you can use it as a light feature in pages instead of a full blown SPA. It gives you bigger room to mix and match with other libraries, but you are also responsible for making more architectural decisions. For example, Vue.js' core doesn't come with routing or ajax functionalities by default, and usually assumes you are building the application using an external module bundler. This is probably the most important distinction.
- Angular uses two-way binding between scopes. While Vue also supports explicit two-way bindings, it defaults to a one-way, parent-to-child data flow between components. Using one-way binding makes the flow of data easier to reason about in large apps.
- Vue.js has a clearer separation between directives and components. Directives are meant to encapsulate DOM manipulations only, while Components stand for a self-contained unit that has its own view and data logic. In Angular there's a lot of confusion between the two.
- Vue.js has better performance and is much, much easier to optimize, because it doesn't use dirty checking. Angular gets slow when there are a lot of watchers, because every time anything in the scope changes, all these watchers need to be re-evaluated again. Also, the digest cycle may have to run multiple times to "stabilize" if some watcher triggers another update. Angular users often have to resort to esoteric techniques to get around the digest cycle, and in some situations there's simply no way to optimize a scope with a large amount of watchers. Vue.js doesn't suffer from this at all because it uses a transparent dependency-tracking observing system with async queueing - all changes trigger independently unless they have explicit dependency relationships. The only optimization hint you'll ever need is the `track-by` param on `v-for` lists.

Interestingly, there are quite some similarities in how Angular 2 and Vue are addressing these Angular 1 issues.

React

React and Vue.js do share a similarity in that they both provide reactive & composable View components. There are, of course, many differences as well.

First, the internal implementation is fundamentally different. React's rendering leverages the Virtual DOM - an in-memory representation of what the actual DOM should look like. When the state changes, React does a full re-render of the Virtual DOM, diffs it, and then patches the real DOM.

The virtual-DOM approach provides a functional way to describe your view at any point of time, which is really nice. Because it doesn't use observables and re-renders the entire app on every update, the view is by definition guaranteed to be in sync with the data. It also opens up possibilities to isomorphic JavaScript applications.

Instead of a Virtual DOM, Vue.js uses the actual DOM as the template and keeps references to actual nodes for data bindings. This limits Vue.js to environments where DOM is present. However, contrary to the common misconception that Virtual-DOM makes React faster than anything else, Vue.js actually out-performs React when it comes to hot updates, and requires almost no hand-tuned optimization. With React, you need to implement `shouldComponentUpdate` everywhere or use immutable data structures to achieve fully optimized re-renders.

API-wise, one issue with React (or JSX) is that the render function often involves a lot of logic, and ends up looking more like a piece of program (which in fact it is) rather than a visual representation of the interface. For some developers this is a bonus, but for designer/developer hybrids like me, having a template makes it much easier to think visually about the design and CSS. JSX mixed with JavaScript logic breaks that visual model I need to map the code to the design. In contrast, Vue.js pays the cost of a lightweight data-binding DSL so that we have a visually scannable template and with logic encapsulated into directives and filters.

Another issue with React is that because DOM updates are completely delegated to the Virtual DOM, it's a bit tricky when you actually **want** to control the DOM yourself (although theoretically you can, you'd be essentially working against the library when you do that). For applications that needs ad-hoc custom DOM manipulations, especially animations with complex timing requirements, this can become a pretty annoying restriction. On this front, Vue.js allows for more flexibility and there are [multiple FWA/Awwwards winning sites²](#) built with Vue.js.

Some additional notes:

- The React team has very ambitious goals in making React a platform-agnostic UI development paradigm, while Vue is focused on providing a pragmatic solution for the web.
- React, due to its functional nature, plays very well with functional programming patterns. However it also introduces a higher learning barrier for junior developers and beginners. Vue is much easier to pick up and get productive with in this regard.

²<https://github.com/vuejs/vue/wiki/Projects-Using-Vue.js#interactive-experiences>

- For large applications, the React community has been doing a lot of innovation in terms of state management solutions, e.g. Flux/Redux. Vue itself doesn't really address that problem (same for React core), but the state management patterns can be easily adopted for a similar architecture. Vue has its own state management solution called [Vuex](#)³, and it's also possible to [use Redux with Vue](#)⁴.
- The trend in React development is pushing you to put everything in JavaScript, including your CSS. There has been many CSS-in-JS solutions out there but all more or less have its own problems. And most importantly, it deviates from the standard CSS authoring experience and makes it very awkward to leverage existing work in the CSS community. Vue's [single file components](#)⁵ gives you component-encapsulated CSS while still allowing you to use your pre-processors of choice.

³<https://github.com/vuejs/vuex>

⁴<https://github.com/egoist/revue>

⁵http://vuejs.org/guide/application.html#Single_File_Components

Ember

Ember is a full-featured framework that is designed to be highly opinionated. It provides a lot of established conventions, and once you are familiar enough with them, it can make you very productive. However, it also means the learning curve is high and the flexibility suffers. It's a trade-off when you try to pick between an opinionated framework and a library with a loosely coupled set of tools that work together. The latter gives you more freedom but also requires you to make more architectural decisions.

That said, it would probably make a better comparison between Vue.js core and Ember's templating and object model layer:

- Vue provides unobtrusive reactivity on plain JavaScript objects, and fully automatic computed properties. In Ember you need to wrap everything in Ember Objects and manually declare dependencies for computed properties.
- Vue's template syntax harnesses the full power of JavaScript expressions, while Handlebars' expression and helper syntax is quite limited in comparison.
- Performance wise, Vue outperforms Ember by a fair margin, even after the latest Glimmer engine update in Ember 2.0. Vue automatically batches updates, while in Ember you need to manually manage run loops in performance-critical situations.

Polymer

Polymer is yet another Google-sponsored project and in fact was a source of inspiration for Vue.js as well. Vue.js' components can be loosely compared to Polymer's custom elements, and both provide a very similar development style. The biggest difference is that Polymer is built upon the latest Web Components features, and requires non-trivial polyfills to work (with degraded performance) in browsers that don't support those features natively. In contrast, Vue.js works without any dependencies down to IE9.

Also, in Polymer 1.0 the team has really made its data-binding system very limited in order to compensate for the performance. For example, the only expressions supported in Polymer templates are the boolean negation and single method calls. Its computed property implementation is also not very flexible.

Finally, when deploying to production, Polymer elements need to be bundled via a Polymer-specific tool called vulcanizer. In comparison, single file Vue components can leverage everything the Webpack ecosystem has to offer, and thus you can easily use ES6 and any CSS pre-processors you want in your Vue components.

Riot

Riot 2.0 provides a similar component-based development model (which is called a “tag” in Riot), with a minimal and beautifully designed API. I think Riot and Vue share a lot in design philosophies. However, despite being a bit heavier than Riot, Vue does offer some significant advantages over Riot:

- True conditional rendering (Riot renders all if branches and simply show/hide them)
- A far-more powerful router (Riot’s routing API is just way too minimal)
- More mature tooling support (see webpack + vue-loader)
- Transition effect system (Riot has none)
- Better performance. (Riot in fact uses dirty checking rather than a virtual-dom, and thus suffers from the same performance issues with Angular.)

For updated comparisons feel free to check [Vue.js guide](#).

Welcome

About the Book

This book will guide you through the path of the rapidly spreading Javascript Framework called Vue.js!

Some time ago, we started a new project based on Laravel and Vue.js. After thoroughly reading Vue.js guide and a few tutorials, we discovered a lack of resources about Vue.js around the web. During the development of our project, we gained a lot of experience, so we came up with the idea to write this book in order to share our acquired knowledge with the world.

This book is written in an informal, intuitive, and easy-to-follow format, wherein all examples are appropriately detailed enough to provide adequate guidance to whoever.

We'll start from the very basics and through many examples we'll cover the most significant features of Vue.js. By the end of this book you will be able to create fast front end applications and increase the performance of your existing projects with Vue.js integration.

Who is this Book for

Everyone who has spent time to learn modern web development, has seen Bootstrap, Javascript and many Javascript frameworks. This book is for anyone interested in learning a lightweight and simple Javascript framework. No excessive knowledge is required, though it would be good to be familiar with HTML and Javascript. If you don't know what the difference is between a string and an object, maybe you need to do some digging first.

This book is also useful for any reader who already know their way around Vue.js and want to expand their knowledge.

Get In Touch

In case you would like to contact us about the book, send us feedback, or other matters you would like to bring our attention to, don't hesitate to contact us.

Name	Email	Twitter
The Majesty of Vue.js	hello@tmvuejs.com	@tmvuejs
Alex Kyriakidis	alex@tmvuejs.com	@hootlex
Kostas Maniatis	kostas@tmvuejs.com	@kostaskafcas

Homework

The best way to learn code is to write code, so we have prepared one exercise at the end of most chapters for you to solve and actually test yourself on what you have learned. We strongly recommend you to try as much as possible to solve them and though them gain a better understanding of Vue.js. Don't be afraid to test your ideas, a little effort goes a long way! Maybe a few different examples or ways will give you the right idea. Of course we are not merciless, hints and potential solutions will be provided!

You may begin your journey!

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in the book we would be grateful if you could report it to us. By doing so, you can protect other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please submit an issue on our [github repository](#)⁶.

Conventions

The following notational conventions are used throughout the book.

A block of code is set as follows:

JavaScript

```
1 function(x, y){
2     // this is a comment
3 }
```

Code words in text, data are shown as follows: “Use `.container` for a responsive fixed width container.”

New terms and important words are shown in bold.

Tips, notes, and warnings are shown as follows:



This is a Warning

This element indicates a warning or caution.

⁶<https://github.com/hootlex/the-majesty-of-vuejs>



This is a Tip

This element signifies a tip or suggestion.



This is an Information box

Some special information here.



This is a Note

A note about the subject.



This is a Hint

A hint about the subject.

Vue.js Fundamentals

1. Install Vue.js

When it comes to download Vue.js you have a few options to choose from.

1.1 Standalone Version

1.1.1 Download from vuejs.org

To install Vue you can simply download and include it with a script tag. Vue will be registered as a global variable.

You can download two versions of Vue.js:

1. Development Version from <http://vuejs.org/js/vue.js>¹
2. Production Version from <http://vuejs.org/js/vue.min.js>².



Tip: Don't use the minified version during development. You will miss out all the nice warnings for common mistakes.

1.1.2 Include from CDN

You can find Vue.js also on [jsdelivr](#)³ or [cdnjs](#)⁴



It takes some time to sync with the latest version so you have to check frequently for updates.

1.2 Download using NPM

NPM is the recommended installation method when building large scale apps with Vue.js. It pairs nicely with a CommonJS module bundler such as [Webpack](#)⁵ or [Browserify](#)⁶.

¹<http://vuejs.org/js/vue.js>

²<http://vuejs.org/js/vue.min.js>

³<http://cdn.jsdelivr.net/vue/1.0.18/vue.min.js>

⁴<https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.min.js>

⁵<http://webpack.github.io/>

⁶<http://browserify.org/>

```
1 # latest stable
2 $ npm install vue
3 # latest stable + CSP-compliant
4 $ npm install vue@csp
5 # dev build (directly from GitHub):
6 $ npm install vuejs/vue#dev
```

1.3 Download using Bower

```
1 # latest stable
2 $ bower install vue
```



For more installation instructions and updates take a look at the [Vue.js Installation Guide](http://vuejs.org/guide/installation.html)⁷

In most book examples we are including Vue.js from the cdn, although you are free to install it using any method you like.

⁷<http://vuejs.org/guide/installation.html>

2. Getting Started

Let's start with a quick tour of Vue's data binding features. We're going to make a simple application that will allow us to enter a message and have it displayed on the page in real time. It's going to demonstrate the power of Vue's two-way data binding. In order to create our Vue application, we need to do a little bit of setting up, which just involves creating an HTML page.

In the process you will get the idea of the amount of time and effort we save using a javascript Framework like Vue.js instead of a javascript tool (library) like jQuery.

2.1 Hello World

We will create a new file and we will drop some boilerplate code in. You can name it anything you like, this one is called `hello.html`.

```
1 <html>
2 <head>
3   <title>Hello Vue</title>
4 </head>
5 <body>
6   <h1>Greetings your Majesty!</h1>
7 </body>
8 </html>
```

This is a simple HTML file with a greeting message.

Now we will carry on and do the same job using Vue.js. First of all we will include Vue.js and create a new Instance.

```
1 <html>
2 <head>
3   <title>Hello Vue</title>
4 </head>
5 <body>
6   <div id="app">
7     <h1>Greetings your majesty!</h1>
8   </div>
9 </body>
```

```
10 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.min.js">
11
12 </script>
13 <script>
14   new Vue({
15     el: '#app',
16   })
17 </script>
18 </html>
```

For starters, we have included Vue.js from [cdnjs¹](https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.min.js) and inside a `script` tag we have our Vue instance. We use a `div` with an `id` of `#app` which is the element we refer to, so Vue knows where to ‘look’. Try to think of this as a container that Vue works at. Vue won’t recognize anything outside of the targeted element. Use the `el` option to target the element you want.

Now we will assign the message we want to display to a variable inside an object named `data`. Then we’ll pass the data object as an option to Vue constructor.

```
1 var data = {
2   message: 'Greetings your majesty!'
3 };
4 new Vue({
5   el: '#app',
6   data: data
7 })
```

To display our message on the page, we just need to wrap the message in double curly brackets . So whatever is inside our message it will appear automatically in the `h1` tag.

```
1 <div id="app">
2   <h1>{{ message }}</h1>
3 </div>
```

It is as simple as that. Another way to define the message variable is to do it directly inside Vue constructor in `data` object.

¹<https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.10/vue.min.js>

```
1 new Vue({
2   el: '#app',
3   data: {
4     message: 'Greetings your Majesty!'
5   }
6 });
```

Both ways have the exact same result, so you are again free to pick whatever syntax you like.



Info

The double curly brackets are not HTML but scripting code, anything inside mustache tags are called binding expressions. Javascript will evaluate these expressions. The `{{ message }}` brings up the value of the Javascript variable. This piece of code `{{1+2}}` will display the number 3.

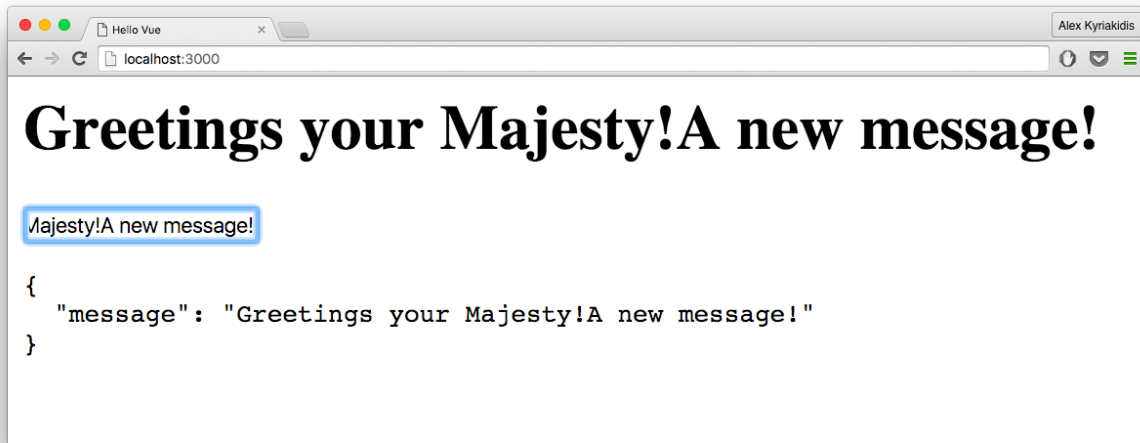
2.2 Two-way Binding

What is cool about Vue is that it makes our lives easier. Say 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, (we will cover more on directives in the next chapter). 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.

```
1 <div id="app">
2   <h1>{{ message }}</h1>
3   <input v-model="message">
4 </div>
```

```
1 new Vue({
2   el: '#app',
3   data: {
4     message: 'Greetings your Majesty!'
5   }
6 });
```

That's it. Now our heading message and user input are binded! By using `v-model` inside the `input` tag we tell Vue which variable should bind with that `input`, in this case `message`.



Two-way data binding

Two-way data binding means that if you change the value of a model in your view, everything will be kept up to date.

2.3 Comparison with jQuery.

Probably all of you have a basic experience with jQuery. If you don't, it's okay, the use of jQuery in this book is minimal. When we use it, its only to demonstrate how things can be done with Vue instead of jQuery and we will make sure everybody gets it.

Anyway, in order to better understand how data-binding is helping us to build apps, take a moment and think how you could do the previous example using jQuery. You would probably create an input element and give it an `id` or a `class` so you could target it and modify it accordingly. After this, you would call a function that changes the desired element to match the input value, whenever the `keyup event` happens. **It's a real bother.**

More or less, your snippet of code would look like this.

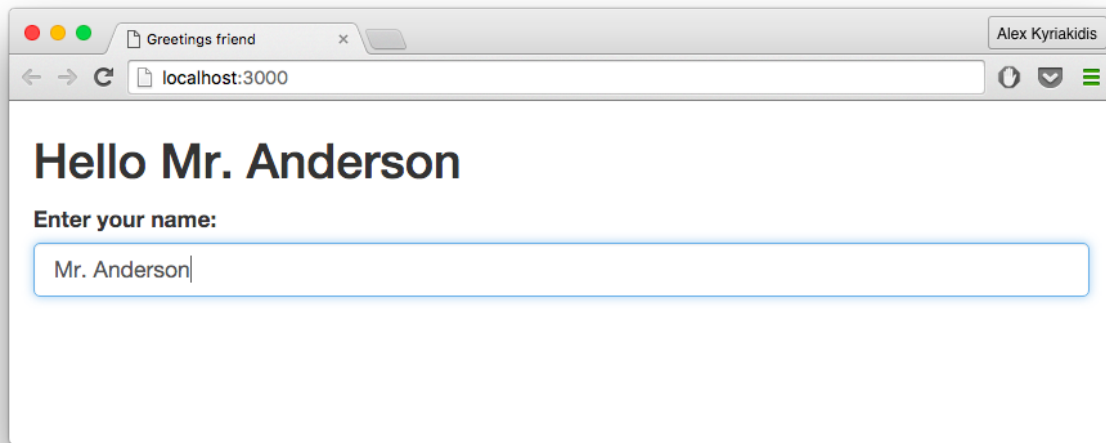
```
1 <html>
2 <head>
3   <title>Hello Vue</title>
4 </head>
5 <body>
6 <div id="app">
7   <h1>Greetings your Majesty!</h1>
8   <input id="message">
9 </div>
```

```
10 </body>
11 <script src="https://code.jquery.com/jquery-2.1.4.min.js"></script>
12 <script type="text/javascript">
13     $('#message').on('keyup', function(){
14         var message = $('#message').val();
15         $('#h1').text(message);
16     })
17 </script>
18 </html>
```

This is a simple example of comparison and as you can see, Vue appears to be much more beautiful, less time consuming, and easier to grasp. Of course, jQuery is a powerful JavaScript library for Document Object Model (DOM) manipulation but everything comes with its ups and downs!

2.4 Homework

A nice and super simple introductory exercise is to create an HTML file with a Hello, `{{name}}` heading. Add an input and bind it to `name` variable. As you can imagine, the heading must change instantly whenever the user types or changes his name. Good luck and have fun!



Example Output

You can find a potential solution to this exercise [here](#)².

²<https://github.com/hootlex/the-majesty-of-vuejs/blob/master/homework/chapter2.html>

3. A Flavor of Directives.

In this chapter we are going through some basic examples of Vue's directives. Well, if you have not used any Framework like Vue.js or AngularJS before, you probably don't know what a directive is. Essentially, a directive is some special token in the markup that tells the library to do something to a DOM element. In Vue.js, the concept of directive is drastically simpler than that in Angular. Some of the directives are:

- **v-show** which is used to conditionally display an element
- **v-if** which can be used instead of **v-show**
- **v-else** which displays an element when **v-if** or **v-show** evaluates to false.

Also, there is **v-for**, which requires a special syntax and its use is for rendering (e.g. render a list of items based on an array). We will elaborate about the use of each later in this book.

Let us begin and take a look at the directives we mentioned.

3.1 v-show

To demonstrate the first directive we are going to build something simple. We will give you some tips that will make your understanding and work much easier! Suppose you find yourself in need to toggle the display of an element, based upon some set of criteria. Maybe a submit button shouldn't display unless you've first typed in a message. How might we accomplish that with Vue?

```
1 <html>
2 <head>
3   <title>Hello Vue</title>
4 </head>
5 <body>
6 <div id="app">
7   <textarea></textarea>
8 </div>
9 </body>
10 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
11 <script>
12   new Vue({
13     el: '#app',
14     data: {
```

```
15         message: 'Our king is dead!'
16     }
17 })
18 </script>
19 </html>
```

Here we have an HTML file with our known `div id="app"` and a `textarea`. Inside the `textarea` we are going to display our message. Of course, it is not yet binded and by this point maybe you have already figured it out. Also you may have noticed that in this example we are no longer using the minified version of Vue.js. As we have mentioned before, the minified version shouldn't be used during development because you will miss out warnings for common mistakes. From now on we are going to use this version in the book but of course you are free to do as you like.

```
1 <html>
2 <head>
3   <title>Hello Vue</title>
4 </head>
5 <body>
6   <div id="app">
7     <textarea v-model="message"></textarea>
8   </div>
9   <pre>
10     {{$data | json}}
11 </pre>
12 </body>
13 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
14 <script>
15   new Vue({
16     el: '#app',
17     data: {
18       message: 'Our king is dead!'
19     }
20   })
21 </script>
22 </html>
```

It is time to bind the value of `textarea` with our `message` variable using `v-model` so it displays our message. Anything we type in is going to change in real time just as we saw in the example from the previous chapter where we were using an input. Additionally here we are using a `pre` tag to spit out the data. What this is going to do, is to take the data from our Vue instance, filter it through `json`, and finally display the data in our browser. We believe, that this gives a much better way to

build and manipulate our data since having everything right in front of you is better than looking constantly at your console.



Info

JSON (JavaScript Object Notation) is a lightweight data-interchange format. You can find more info on JSON [here](http://www.json.org/)¹. The output of `{{ $data | json }}` is binded with Vue data and will get updated on every change.

```
1 <html>
2 <head>
3   <title>Hello Vue</title>
4 </head>
5 <body>
6 <div id="app">
7   <h1>You must send a message for help!</h1>
8   <textarea v-model="message"></textarea>
9   <button v-show="message">
10     Send word to allies for help!
11 </button>
12 <pre>
13   {{{ $data | json }}}
14 </pre>
15 </div>
16 </body>
17 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
18 <script>
19   new Vue({
20     el: '#app',
21     data: {
22       message: 'Our king is dead! Send help!'
23     }
24   })
25 </script>
26 </html>
```

Carrying on, we now have a simple warning in the `h1` tag that will toggle later based on some criteria. Next to it, there is the button which is going to display conditionally, it appears only if there is a message present. If the `textarea` is empty and therefore our data, the button's `display` attribute is automatically set to 'none' and the button disappears.

¹<http://www.json.org/>



Info

An element with `v-show` will always be rendered and remain in the DOM. `v-show` simply toggles the `display` CSS property of the element.

```

1 <h1 v-show="!message">You must send a message for help!</h1>
2 <textarea v-model="message"></textarea>
3 <button v-show="message">
4   Send word to allies for help!
5 </button>

```

What we want to accomplish in this example, is to toggle different elements. In this step, we need to hide the warning inside the `h1` tag, if a message is present, otherwise hide the message by setting its `style` to `display: none`.

3.2 v-if

In this point you might ask ‘What about the `v-if` directive we mentioned earlier?’, so we will build the previous example again, only this time we’ll use `v-if`!

```

1 <html>
2 <head>
3   <title>Hello Vue</title>
4 </head>
5 <body>
6 <div id="app">
7   <h1 v-if="!message">You must send a message for help!</h1>
8   <textarea v-model="message"></textarea>
9   <button v-show="message">
10     Send word to allies for help!
11   </button>
12   <pre>
13     {{$data | json}}
14   </pre>
15 </div>
16 </body>
17 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
18 <script>
19   new Vue({
20     el: '#app',

```

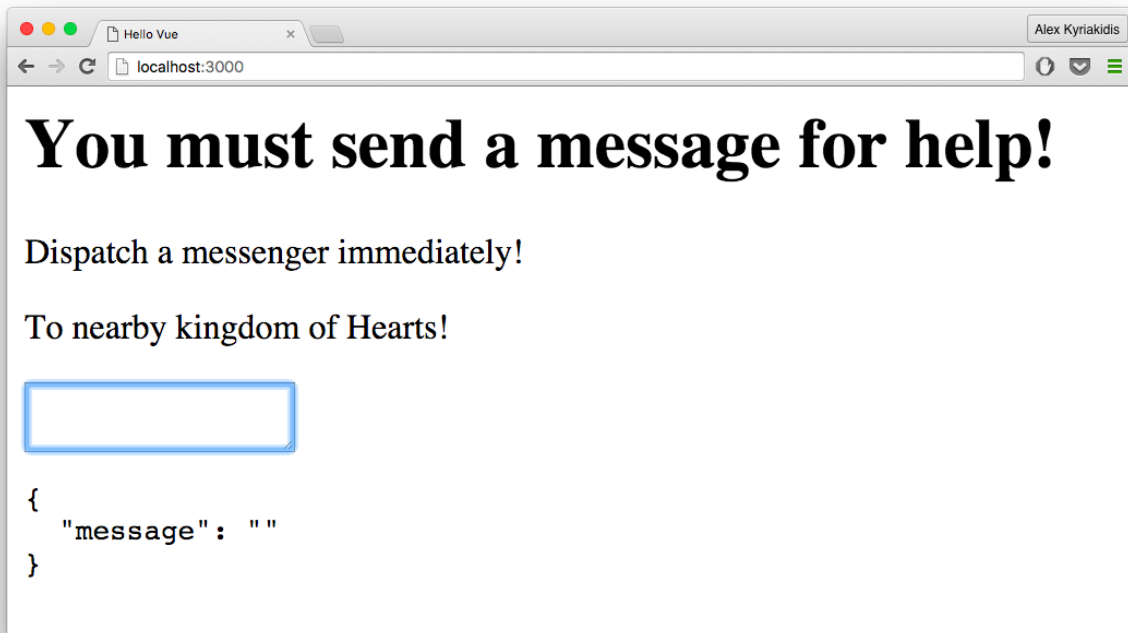
```
21     data: {
22         message: 'Our king is dead! Send help!'
23     }
24 })
25 </script>
26 </html>
```

As shown, the replacement of `v-show` with `v-if` works just as good as we thought. Go ahead and try to make your own experiments to see how this works! The only difference is that an element with `v-if` will not remain in the DOM.

3.2.1 Template v-if

If sometime we find ourselves in a position where we want to toggle the existence of multiple elements at once then we can use `v-if` on a `<template>` element. In occasions where the use of `div` or `span` seems appropriate, the `<template>` element can serve also as an invisible wrapper. Also the `<template>` won't be rendered in the final result.

```
1 <div id="app">
2   <template v-if="!message">
3     <h1>You must send a message for help!</h1>
4     <p>Dispatch a messenger immediately!</p>
5     <p>To nearby kingdom of Hearts!</p>
6   </template>
7   <textarea v-model="message"></textarea>
8   <button v-show="message">
9     Send word to allies for help!
10  </button>
11  <pre>
12    {{$data | json}}
13  </pre>
14 </div>
```



Template v-if

Using the setup from the previous example we have attached the `v-if` directive to the `template` element, toggling the existence of all nested elements.



Warning

The `v-show` directive does not support the `<template>` syntax.

3.3 v-else

When using `v-if` or `v-show` you can use the `v-else` directive to indicate an “else block” as you might have already imagined. Be aware that the `v-else` directive must follow immediately the `v-if` or `v-show` directive - otherwise it will not be recognized.

Using `v-else` with `v-show`.


```

1 <html>
2 <head>
3   <title>Hello Vue</title>
4 </head>
5 <body>
6 <div id="app">
7   <h1 v-show="!message">You must send a message for help!</h1>
8   <h2 v-else>You have sent a message!</h2>
9   <textarea v-model="message"></textarea>
10  <button v-show="message">
11    Send word to allies for help!
12  </button>
13  <pre>
14    {{$data | json}}
15  </pre>
16 </div>
17 </body>
18 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
19
20 <script>
21   new Vue({
22     el: '#app',
23     data: {
24       message: 'Our king is dead! Send help!'
25     }
26   })
27 </script>
28 </html>

```

Using `v-else` with `v-if`.

```

1 <html>
2 <head>
3   <title>Hello Vue</title>
4 </head>
5 <body>
6 <div id="app">
7   <h1 v-if="!message">You must send a message for help!</h1>
8   <h2 v-else>You have send a message!</h2>
9   <textarea v-model="message"></textarea>
10  <button v-show="message">
11    Send word to allies for help!

```

```
12     </button>
13     <pre>
14         {{$data | json}}
15     </pre>
16 </div>
17 </body>
18 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
19
20 <script>
21     new Vue({
22         el: '#app',
23         data: {
24             message: 'Our king is dead! Send help!'
25         }
26     })
27 </script>
28 </html>
```



v-if in action



v-else in action

Just for the sake of the example we have used an `h2` tag with a different warning than before which is displayed conditionally. If there is no message present, we see the `h1` tag. If there is a message, we see the `h2` using this very simple syntax of Vue `v-if` and `v-else`. As you can see above we've used `v-if` as well as `v-show`. Both give us the same result. Simple as a pimple!

3.4 v-if vs. v-show

Even though we have already mentioned a difference between `v-if` and `v-show`, we can deepen a bit more. Some questions may arise out of their use. Is there a big difference between using `v-show` and `v-if`? Is there a situation where performance is affected? Are there problems where you're better off using one or the other? You might experience that the use of `v-show` on a lot of situations causes bigger time of load during page rendering. In comparison, `v-if` is truly conditional according to the guide of Vue.js.

When using `v-if`, if the condition is false on initial render, it will not do anything - partial compilation won't start until the condition becomes true for the first time. Generally speaking, `v-if` has higher toggle costs while `v-show` has higher initial render costs. So prefer `v-show` if you need to toggle something very often, and prefer `v-if` if the condition is unlikely to change at runtime.

So, when to use which really depends on your needs.

3.5 Homework

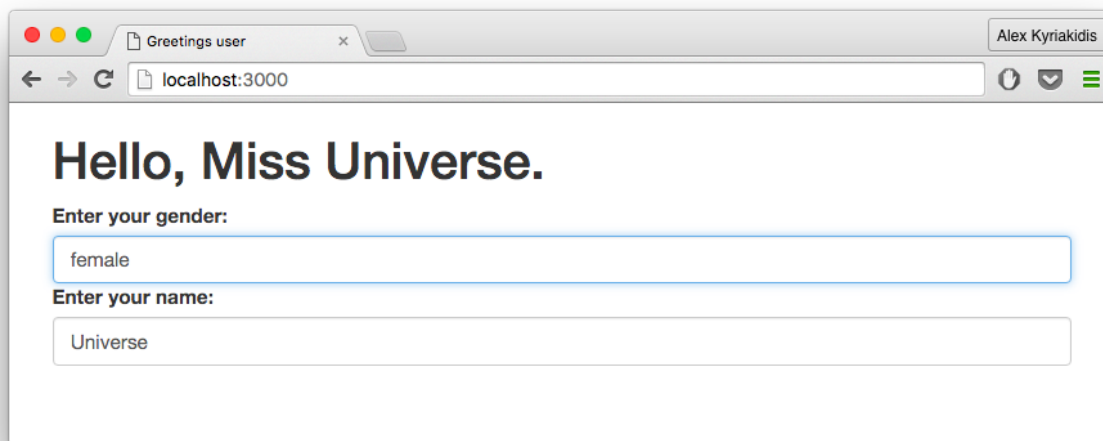
Following the previous homework exercise, you should try to expand it a bit. The user now types in his gender along with his name. If user is a male, then the heading will greet the user with “**Hello Mister {{name}}**”. If user is a female, then “**Hello Miss {{name}}**” should appear instead.

When gender is neither male or female then the user should see the warning heading “**Enter a valid gender, human.**”.



Hint

A logical operator would come handy to determine user title.



Example Output

You can find a potential solution to this exercise [here](#)².

²<https://github.com/hootlex/the-majesty-of-vuejs/blob/master/homework/chapter3.html>

4. List Rendering

In the third chapter of this book, we are going to learn about list rendering. Using Vue's directives we are going to demonstrate how to:

1. Render a list of items based on an array.
2. Repeat a template.
3. Iterate through the properties of an object.
4. Filter an array of items.
5. Order an array of items.
6. Apply a custom filter to a list.

4.1 Install & Use Bootstrap

To make our work easier on the eye, we are going to import Bootstrap.



Info

Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.

Head to <http://getbootstrap.com/>¹ and click the download button. For the time being, we'll just use Bootstrap from the [CDN link](https://www.bootstrapcdn.com/)² but you can install it any way that suits your particular needs. For our example we need only one file, for now: `css/bootstrap.min.css`. When we use this `.css` file in our app, we have access to all the pretty structures and styles. Just include it within the `head` tag of your page and you are good to go.

Bootstrap requires a containing element to wrap site contents and house our grid system. You may choose one of two containers to use in your projects. Note that, due to `padding` and more, neither container is nestable.

- Use `.container` for a responsive fixed width container.
`<div class="container"> ... </div>`
- Use `.container-fluid` for a full width container, spanning the entire width of your viewport.
`<div class="container-fluid"> ... </div>`

¹<http://getbootstrap.com/>

²<https://www.bootstrapcdn.com/>

At this point, we would like to make an example of Vue.js with Bootstrap classes. This is the introductory example concerning classes and many will follow. Of course, not much study or experimentation is required in order make use of combined Vue and Bootstrap.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4 s" rel="stylesheet">
5 <title>Hello Bootstrap</title>
6 </head>
7 <body>
8 <div class="container">
9 <h1>Hello Bootstrap, sit next to Vue.</h1>
10 <pre>
11     {{$data | json}}
12 </pre>
13 </div>
14 </body>
15 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
16 <script type="text/javascript">
17     new Vue({
18         el: '.container',
19         data: {
20
21         }
22     })
23 </script>
24 </html>
```

Shown here is the installed Bootstrap and the basic set up for our stories example.

Notice this time, instead of targeting `app` id, we have targeted the `container` class within the `el` option inside the Vue instance. Going that way, we have gained the styles and structure that comes along with this class and made our app a bit more delightful.



Note

Most of the times we are going to use the `pre` tag in our code to display our data in JSON format.



Tip

In the above example we target the element with class of `.container`. **Be careful** when you are targeting an element by class, when the class is present more than 1 time, Vue.js will mount on the first element **only**.

Using `e1`: you can target any DOM element on the! Try targeting the **body** of your HTML and see how that works!

4.2 v-for

In order to loop through each item in an array, we will use `v-for` Vue's directive.

The `v-for` loop works on arrays/objects and is used to loop through each item in an array. This directive requires a special syntax in the form of `item in array` where `array` is the source data Array and `item` is an alias for the Array element being iterated on.



Warning

If you are coming from the php world you may notice that `v-for` is similar to php's `foreach` function. But be careful if you are used to `foreach($array as $value)`.

Vue's `v-for` is exactly the opposite, `value in array`.

The singular first, the plural next.

4.2.1 Range v-for

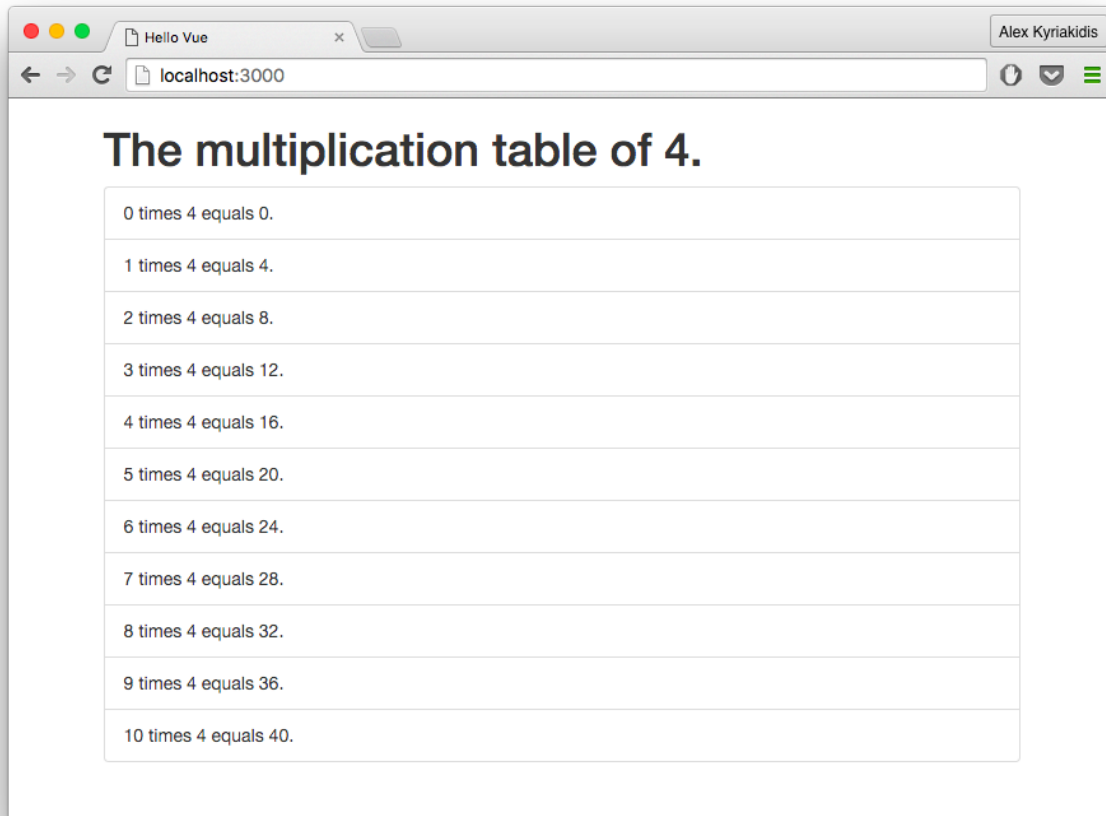
Directive `v-for` can also take an integer. Whenever a number is passed instead of an array/object, the template will be repeated as many times as the number given.

```
1 <html>
2 <head>
3   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.mi\
4 n.css" rel="stylesheet">
5   <title>Hello Vue</title>
6 </head>
7 <body>
8   <div class="container">
9     <h1>The multiplication table of 4.</h1>
10    <ul class="list-group">
11      <li v-for="i in 11" class="list-group-item">
12        {{ i }} times 4 equals {{ i * 4 }}.
13      </li>
14    </ul>
15  </div>
16 </body>
17 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
18 <script type="text/javascript">
19   new Vue({
20     el: '.container'
21   })
```



```
22 </script>  
23 </html>
```

The above code displays the multiplication table of 4.



Multiplication Table of 4



Note

Because we want to display all the multiplication table of 4 (until 40) we repeat the template 11 times since the first value `i` takes is `0`.

4.3 Array Rendering

4.3.1 Loop Through an Array

In the next example we will set up the following array of Stories inside our data object and we will display them all, one by one.

```
1 stories: [  
2   "I crashed my car today!",  
3   "Yesterday, someone stole my bag!",  
4   "Someone ate my chocolate...",  
5 ]
```

What we need to do here, is to render a list. Specifically, an array of strings.

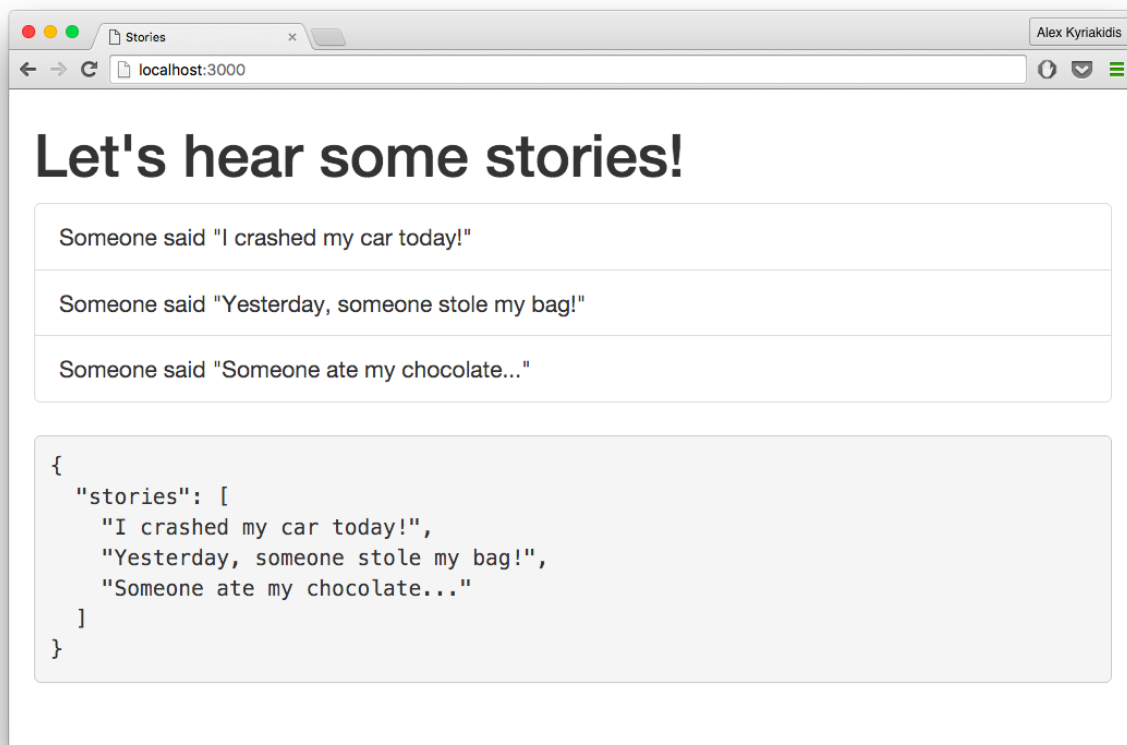
```
1 <html>  
2 <head>  
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\  
4 s" rel="stylesheet">  
5   <title>Stories</title>  
6 </head>  
7 <body>  
8   <div class="container">  
9     <h1>Let's hear some stories!</h1>  
10    <div>  
11      <ul class="list-group">  
12        <li v-for="story in stories" class="list-group-item">  
13          Someone said "{{ story }}"  
14        </li>  
15      </ul>  
16    </div>  
17    <pre>  
18      {{$data | json}}  
19    </pre>  
20  </div>  
21 </body>  
22 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>  
23 <script type="text/javascript">  
24   new Vue({  
25     el: '.container',  
26     data: {
```

```
27     stories: [  
28         "I crashed my car today!",  
29         "Yesterday, someone stole my bag!",  
30         "Someone ate my chocolate...",  
31     ]  
32 }  
33 })  
34 </script>  
35 </html>
```



Info

Both `list-group` and `list-group-item` classes are Bootstrap classes. [Here you can find more information about Bootstrap list styling.](http://getbootstrap.com/css/#type-lists)³



Rendering an array using v-for.

³<http://getbootstrap.com/css/#type-lists>

This is the output of the above code. Using `v-for` we have managed to display our stories in a simple unordered list. It is really that easy!

4.3.2 Loop Through an Array of Objects

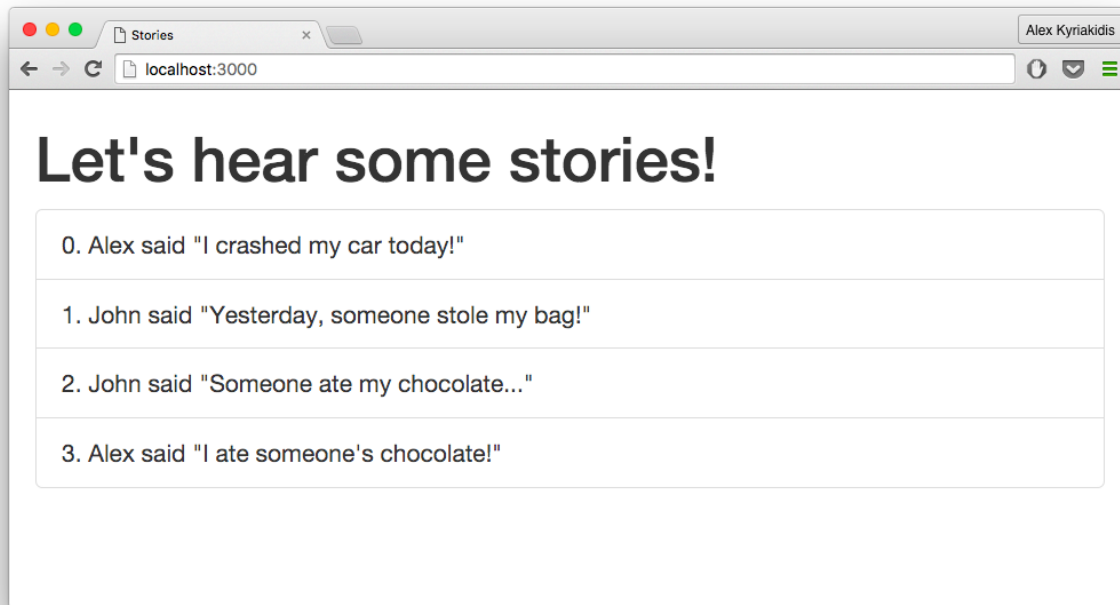
Now, we change the `Stories` array to contain `story` objects. A `story` object has 2 properties: `plot` and `writer`. We will do the same thing we did before but this time instead of echoing `story` immediately, we will echo `story.plot` and `story.writer` respectively.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4 s" rel="stylesheet">
5   <title>Stories</title>
6 </head>
7 <body>
8   <div class="container">
9     <h1>Let's hear some stories!</h1>
10    <div>
11      <ul class="list-group">
12        <li v-for="story in stories"
13          class="list-group-item"
14          >
15          {{ story.writer }} said "{{ story.plot }}"
16        </li>
17      </ul>
18    </div>
19    <pre>
20      {{$data | json}}
21    </pre>
22  </div>
23 </body>
24 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
25 <script type="text/javascript">
26 new Vue({
27   el: '.container',
28   data: {
29     stories: [
30       {
31         plot: "I crashed my car today!",
32         writer: "Alex"
33       },
```

```
34     {
35         plot: "Yesterday, someone stole my bag!",
36         writer: "John"
37     },
38     {
39         plot: "Someone ate my chocolate...",
40         writer: "John"
41     },
42     {
43         plot: "I ate someone's chocolate!",
44         writer: "Alex"
45     },
46 ]
47 }
48 })
49 </script>
50 </html>
```

Additionally, when you need to display the index of the current item, you can use `$index` special variable. Following is an example to show how it works.

```
1 <ul class="list-group">
2   <li v-for="story in stories" class="list-group-item">
3     {{{ $index }}}. {{ story.writer }} said "{{ story.plot }}"
4   </li>
5 </ul>
```



Rendered array with index

The `$index` inside the curly braces, clearly represents the index of the iterated item in the given example.

Another way to access the `index` of the iterated item, is to specify an alias for the `index` of the array as shown below.

```
1 <ul class="list-group">
2   <li v-for="(index, story) in stories"
3     class="list-group-item"
4     >
5     {{index}} {{ story.writer }} said "{{ story.plot }}"
6   </li>
7 </ul>
```

The output of the last code is exactly the same with the previous one.

4.4 Object v-for

You can use `v-for` to iterate through the properties of an Object. We mentioned before that you can bring to display the `index` of the array, but you can also do the same when iterating an object. In addition to `$index`, each scope will have access to another special property, the `$key`.



Info

When iterating an object, `$index` is in range of `0 ... n-1` where `n` is the number of object properties.

We have restructured our data to be a single object with 3 attributes this time: `plot`, `writer` and `upvotes`. As you can see in the example code above, we use `$key` and `$index` to bring inside the list the key-value pairs, as well as the `$index` of each pair.

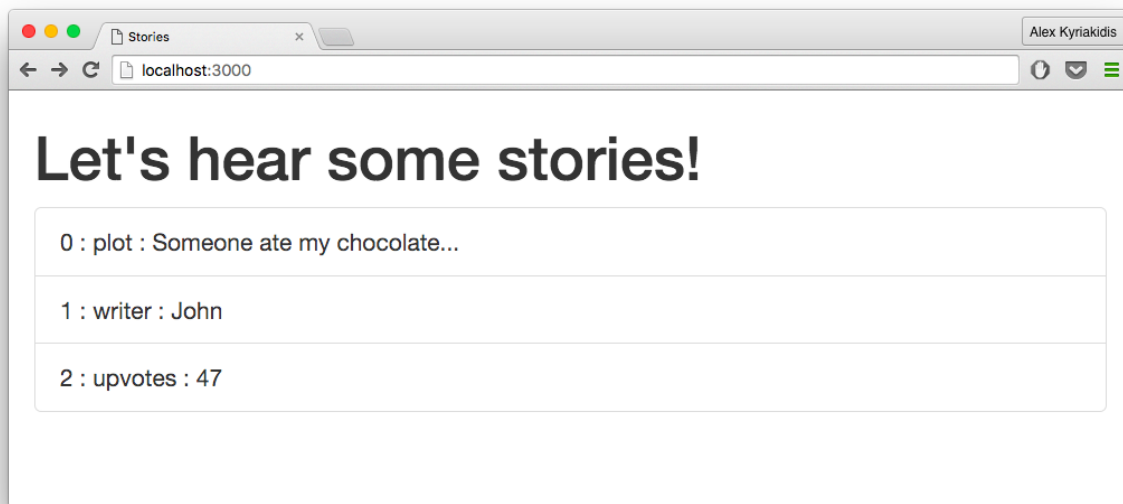
```
1 <div class="container">
2   <h1>Let's hear some stories!</h1>
3   <ul class="list-group">
4     <li v-for="value in story" class="list-group-item">
5       {{ $index }} : {{ $key }} : {{ value }}
6     </li>
7   </ul>
8 </div>

1 new Vue({
2   el: '.container',
3   data: {
4     story: {
5       plot: "Someone ate my chocolate...",
6       writer: 'John',
7       upvotes: 47
8     }
9   }
10 })
```

Alternatively, you can also specify an alias for the key.

```
1 <div class="container">
2   <h1>Let's hear some stories!</h1>
3   <ul class="list-group">
4     <li v-for="(key, value) in story"
5       class="list-group-item"
6     >
7       {{$index}} : {{key}} : {{ value }}
8     </li>
9   </ul>
10 </div>
```

Either way the result will be:



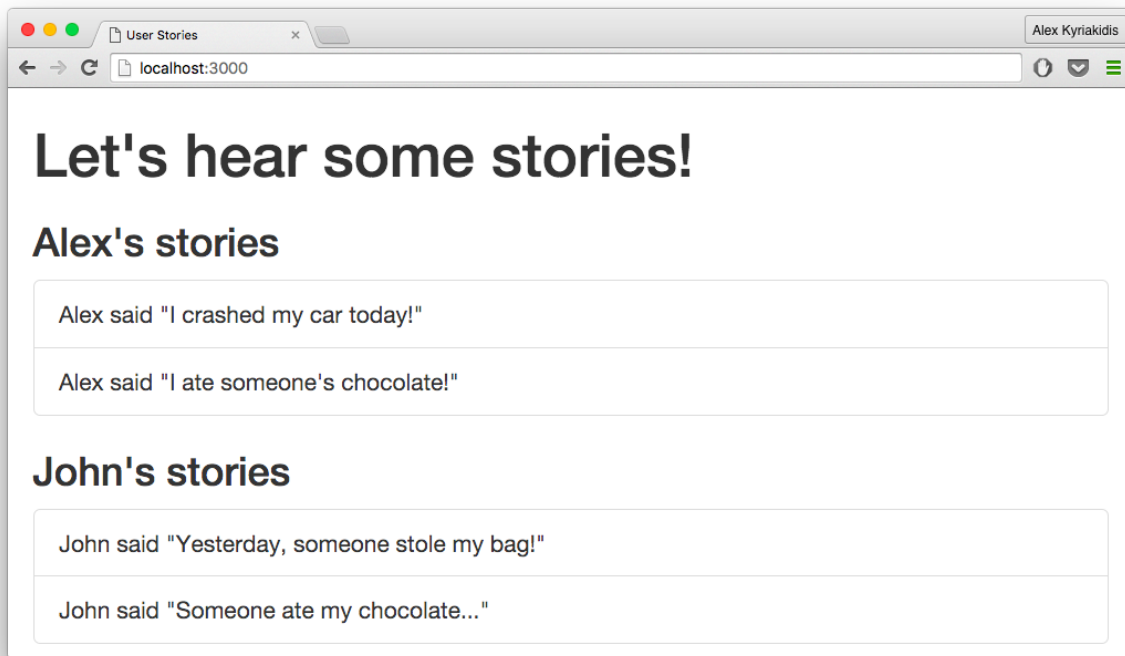
Iterate though object's properties.

4.5 Filtered Results

Sometimes we need to display a filtered version of an array without actually mutating or resetting the original data. In our example we want to display a list with the stories written by Alex and one list with the stories written by John. We can achieve this using the built-in filter, `filterBy`.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4 s" rel="stylesheet">
5   <title>User Stories</title>
6 </head>
7 <body>
8   <div class="container">
9     <h1>Let's hear some stories!</h1>
10    <div>
11      <h3>Alex's stories</h3>
12      <ul class="list-group">
13        <li v-for="story in stories | filterBy 'Alex' in 'writer'"
14          class="list-group-item"
15        >
16          {{ story.writer }} said "{{ story.plot }}"
17        </li>
18      </ul>
19      <h3>John's stories</h3>
20      <ul class="list-group">
21        <li v-for="story in stories | filterBy 'John' in 'writer'"
22          class="list-group-item"
23        >
24          {{ story.writer }} said "{{ story.plot }}"
25        </li>
26      </ul>
27    </div>
28    <pre>
29      {{$data | json}}
30    </pre>
31  </div>
32 </body>
33 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
34 <script type="text/javascript">
35 new Vue({
36   el: '.container',
```

```
37     data: {
38         stories: [
39             {
40                 plot: "I crashed my car today!",
41                 writer: "Alex"
42             },
43             {
44                 plot: "Yesterday, someone stole my bag!",
45                 writer: "John"
46             },
47             {
48                 plot: "Someone ate my chocolate...",
49                 writer: "John"
50             },
51             {
52                 plot: "I ate someone's chocolate!",
53                 writer: "Alex"
54             },
55         ]
56     }
57 })
58 </script>
59 </html>
```



Stories filtered by writer.



Note

As you may noticed, our `li` tag is getting really big, so we have splitted it in more lines.

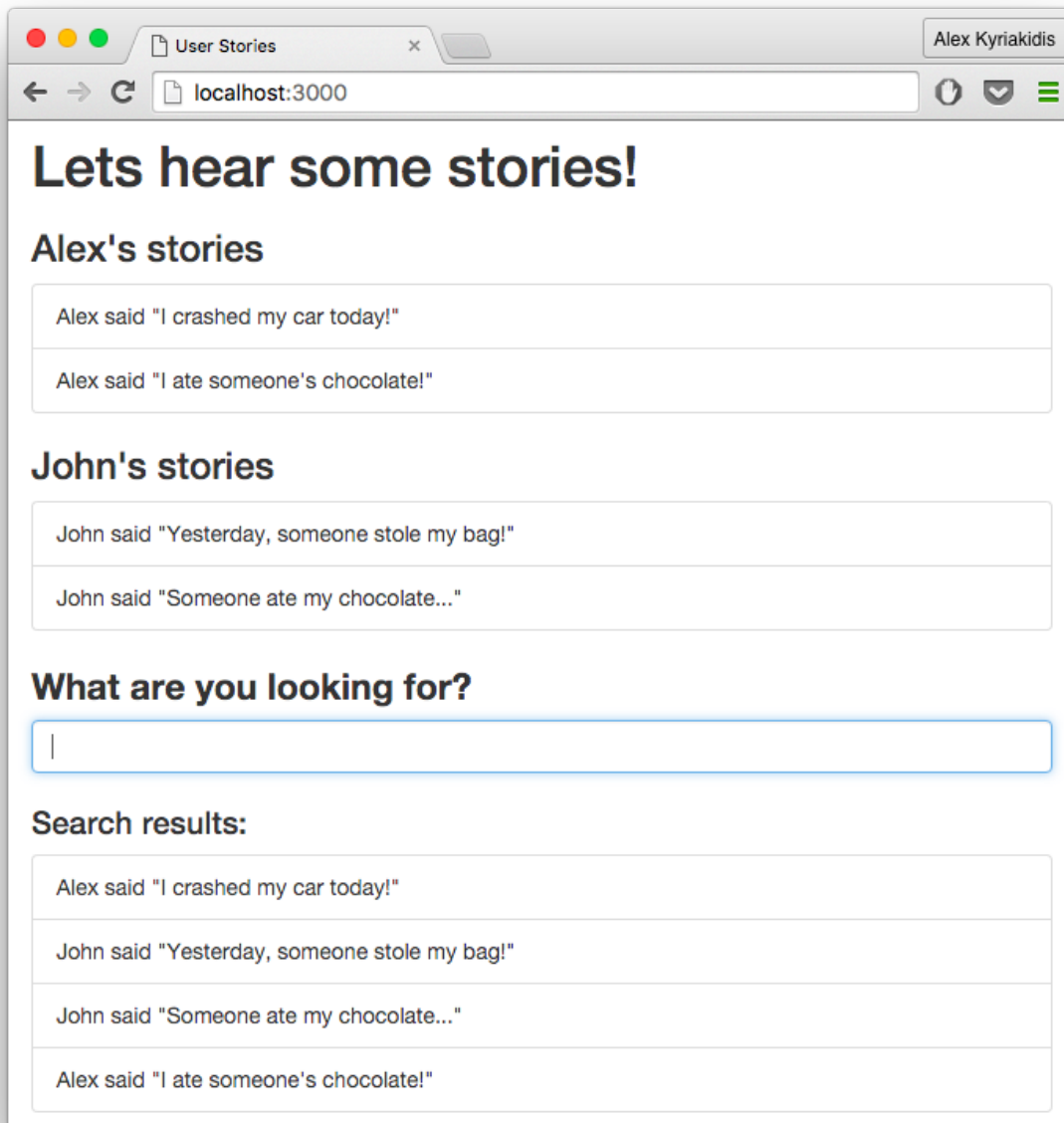
Simple enough, right? Next we will implement a very basic (but awesome) search. When the user types a part of a story, we can guess which story it is and who wrote it, in real time. We'll add a text `input` bound to an empty variable `query` so we can dynamically filter our `Stories` array.

```

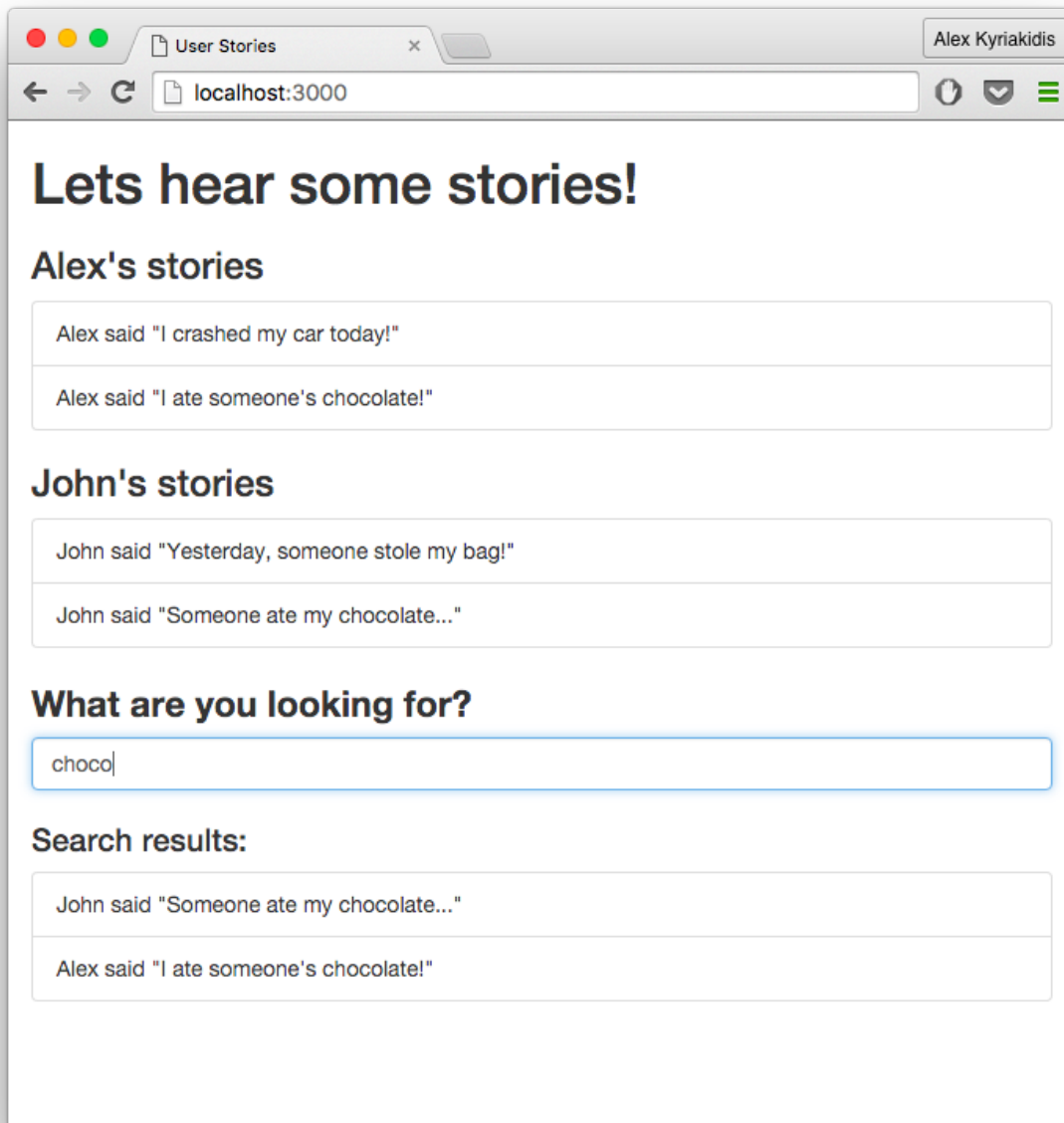
1 <div class="container">
2   <h1>Lets hear some stories!</h1>
3   <div>
4     <h3>Alex's stories</h3>
5     <ul class="list-group">
6       <li v-for="story in stories | filterBy 'Alex' in 'writer'"
7         class="list-group-item"
8       >
9         {{ story.writer }} said "{{ story.plot }}"
10      </li>
11    </ul>
12    <h3>John's stories</h3>

```

```
13     <ul class="list-group">
14         <li v-for="story in stories | filterBy 'John' in 'writer'"
15             class="list-group-item"
16         >
17             {{ story.writer }} said "{{ story.plot }}"
18         </li>
19     </ul>
20     <div class="form-group">
21         <label for="query">
22             What are you looking for?
23         </label>
24         <input v-model="query" class="form-control">
25     </div>
26     <h3>Search results:</h3>
27     <ul class="list-group">
28         <li v-for="story in stories | filterBy query in 'plot'"
29             class="list-group-item"
30         >
31             {{ story.writer }} said "{{ story.plot }}"
32         </li>
33     </ul>
34 </div>
35 </div>
```



Stories filtered by writer with search.



Searching for 'choco'.

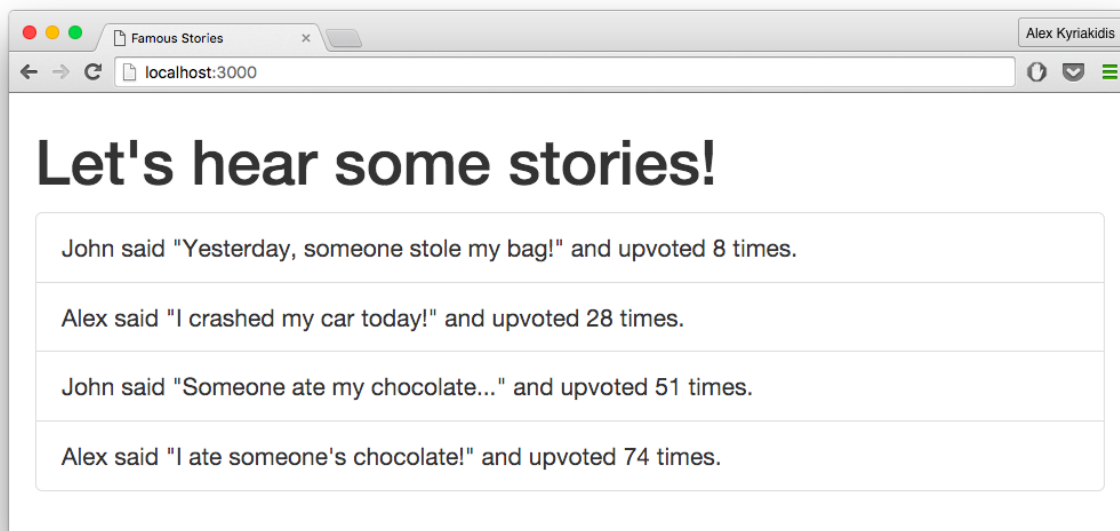
Isn't that awesome??

4.6 Ordered Results

Sometimes we may want to display the items of an Array ordered by some criteria. Luckily there is an `orderBy` built in filter to sort our list in no time! First we will enhance our Stories with a new property called `upvotes`. Then we'll go on and display our array ordered by the count of each story's `upvotes`. *The more famous a story is, the higher it should appear.*

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4 s" rel="stylesheet">
5   <title>Famous Stories</title>
6 </head>
7 <body>
8   <div class="container">
9     <h1>Let's hear some stories!</h1>
10    <ul class="list-group">
11      <li v-for="story in stories | orderBy 'upvotes'"
12        class="list-group-item"
13      >
14        {{ story.writer }} said "{{ story.plot }}"
15        and upvoted {{ story.upvotes }} times.
16      </li>
17    </ul>
18    <pre>
19      {{ $data | json }}
20    </pre>
21  </div>
22 </body>
23 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
24 <script type="text/javascript">
25 new Vue({
26   el: '.container',
27   data: {
28     stories: [
29       {
30         plot: "I crashed my car today!",
31         writer: "Alex",
32         upvotes: 28
33       },
34       {
35         plot: "Yesterday, someone stole my bag!",
```

```
36         writer: "John",
37         upvotes: 8
38     },
39     {
40         plot: "Someone ate my chocolate...",
41         writer: "John",
42         upvotes: 51
43     },
44     {
45         plot: "I ate someone's chocolate!",
46         writer: "Alex",
47         upvotes: 74
48     },
49 ]
50 }
51 })
52 </script>
53 </html>
```



Stories array ordered by upvotes.

Hmmm, the array is ordered but this is not what we expected. We wanted the **famous stories first**. Luckily, again, `orderBy` filter accepts two arguments: the **key** to sort the array, and the **order** which specifies whether the result should be ordered in ascending (`order >= 0`) or descending (`order < 0`) order.

Eventually, for the sake of ordering the array in descending order, our code will look like this:

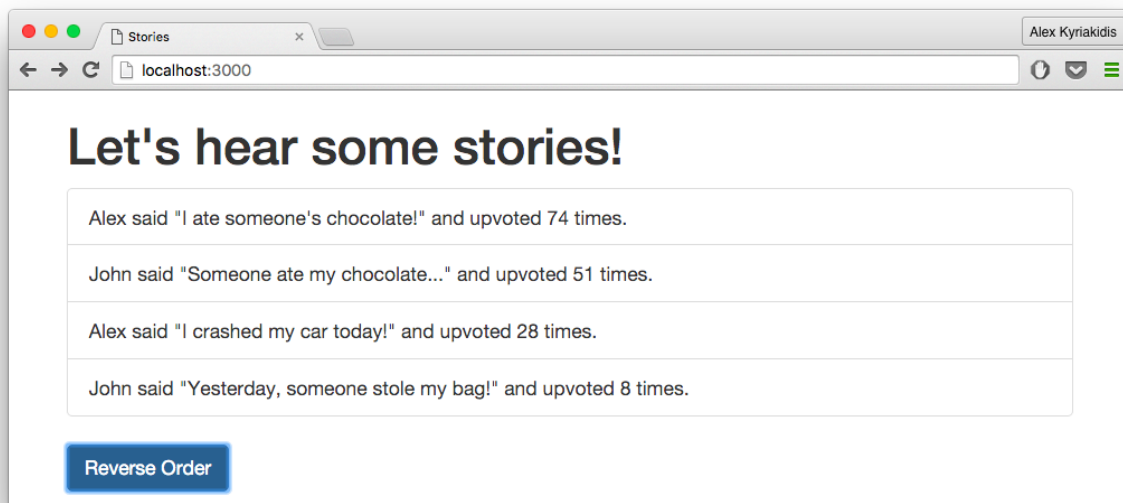
```
1 <ul class="list-group">
2   <li v-for="story in stories | orderBy 'upvotes' -1"
3     class="list-group-item"
4   >
5     {{ story.writer }} said "{{ story.plot }}"
6     and upvoted {{ story.upvotes }} times.
7   </li>
8 </ul>
```

We can easily change the order we sort the array, by dynamically changing the `order` parameter. A `button` is added, which will toggle the value of a new variable between `-1` and `1`, and then the new variable is passed as `order` parameter to `orderBy` filter. *Watch now.*

```
1 new Vue({
2   el: '.container',
3   data: {
4     order: -1,
5     stories: [
6       {
7         plot: "I crashed my car today!",
8         writer: "Alex",
9         upvotes: 28
10      },
11      {
12        plot: "Yesterday, someone stole my bag!",
13        writer: "John",
14        upvotes: 8
15      },
16      {
17        plot: "Someone ate my chocolate...",
18        writer: "John",
19        upvotes: 51
20      },
21      {
22        plot: "I ate someone's chocolate!",
23        writer: "Alex",
24        upvotes: 74
25      },
26    ]
27  }
28 })
```

We initialize `order` variable with the value of `-1` and then we pass it to `orderBy` filter.

```
1 <ul class="list-group">
2   <li v-for="story in stories | orderBy 'upvotes' order"
3     class="list-group-item"
4   >
5     {{ story.writer }} said "{{ story.plot }}"
6     and upvoted {{ story.upvotes }} times.
7   </li>
8 </ul>
9 <button @click="order = order * -1">Reverse Order</button>
```



Array in descending order

Impressing huh? If you are not impressed by now, guess who is! ..**"We are!"**..

4.7 Custom Filter

This is the most cumbersome part of this chapter. Assume we want to display only the famous stories (the ones with upvotes greater than 20). In order to achieve that we have to create a custom filter and apply it to `filterBy`. We are going to create a filter named `famous` which expects two parameters:

- the `array` we want to filter
- and the `bound` which defines the amount of `upvotes` a story **must** have in order to be considered as famous

The `famous` filter returns an array which contains only the objects that satisfy a condition.

If you can't keep up with this example don't worry, you will get it sooner or later, just keep reading..

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4 s" rel="stylesheet">
5   <title>Famous Stories</title>
6 </head>
7 <body>
8   <div class="container">
9     <h1>Let's hear some famous stories!</h1>
10    <ul class="list-group">
11      <li v-for="story in stories | famous"
12        class="list-group-item"
13        >
14          {{ story.writer }} said "{{ story.plot }}"
15          and upvoted {{ story.upvotes }} times.
16        </li>
17    </ul>
18    <pre>
19      {{ $data | json }}
20    </pre>
21  </div>
22 </body>
23 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
24 <script type="text/javascript">
25 Vue.filter('famous', function (stories) {
26   return stories.filter(function(item){
27     return item.upvotes > 20;
28   });
```

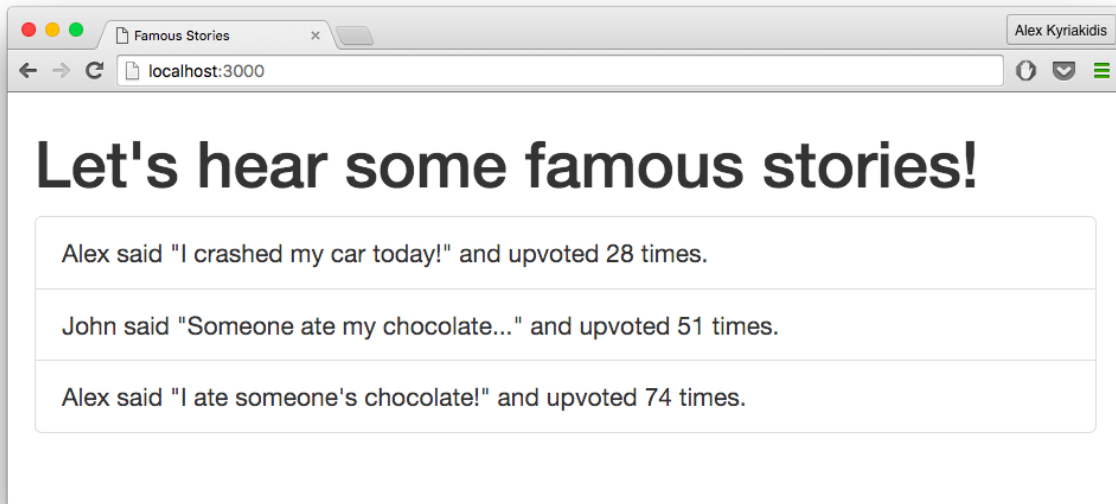
```
29 })
30
31 new Vue({
32   el: '.container',
33   data: {
34     stories: [
35       {
36         plot: "I crashed my car today!",
37         writer: "Alex",
38         upvotes: 28
39       },
40       {
41         plot: "Yesterday, someone stole my bag!",
42         writer: "John",
43         upvotes: 8
44       },
45       {
46         plot: "Someone ate my chocolate...",
47         writer: "John",
48         upvotes: 51
49       },
50       {
51         plot: "I ate someone's chocolate!",
52         writer: "Alex",
53         upvotes: 74
54       },
55     ]
56   }
57 })
58 </script>
59 </html>
```



Info

Our famous filter uses javascript's [filter method](#).⁴ The `filter()` method creates a new array with all elements that pass the test implemented by the provided function.

⁴https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Array/filter



Custom filter 'famous' in action.

4.8 Homework

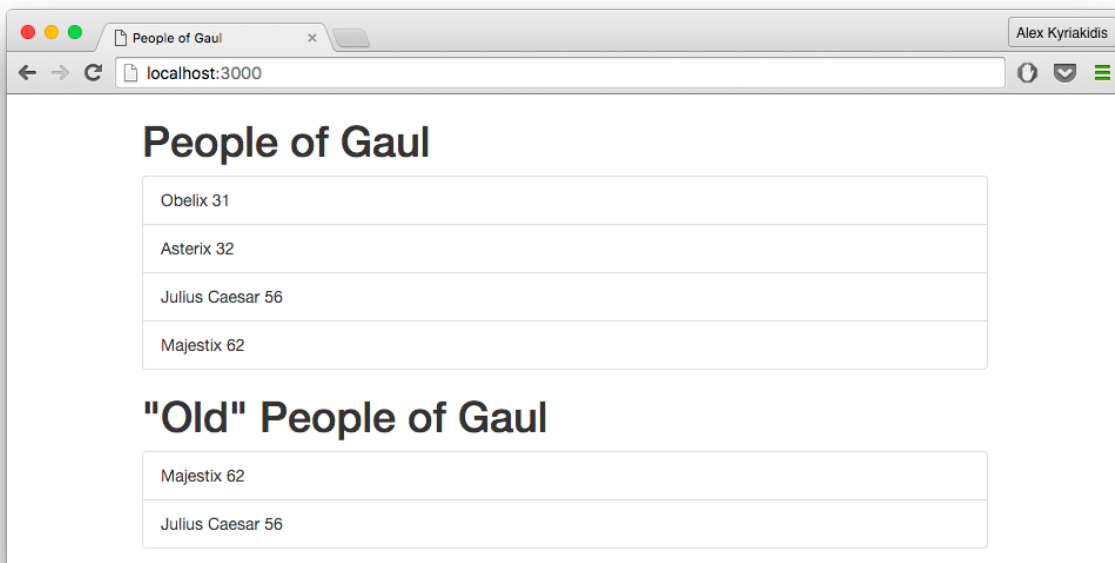
For this chapter's exercise you should do the following. Start by creating an array of people. Each person has a name and an age. Using what you've learned so far, try to render the array in a list and sort it by "age". After that, create a second list below and apply a custom filter called "old" which returns all people older than 55 years old.

Feel free to fill the array with your own data. **Be careful** to add people with age older and younger than 55 to ensure your filter is working properly. Go ahead!



Hint

Built in filter `orderBy` and `Vue.filter` are necessary here.



Example Output

You can find a potential solution to this exercise [here](#)⁵.

⁵<https://github.com/hootlex/the-majesty-of-vuejs/blob/master/homework/chapter4.html>

5. Interactivity

In this chapter, we are going to create and expand previous examples, learn new things concerning ‘methods’, ‘event handling’ and ‘computed properties’. We will develop a few examples using different approaches. It’s time to see how we can implement Vue’s interactivity to get a small app, like a Calculator, running nice and easy.

5.1 Event Handling

HTML events are things that happen to HTML elements. When Vue.js is used in HTML pages, it can **react** to these events.

In HTML, events can represent everything from basic user interactions to things happening in the rendering model.

These are some examples of HTML events:

- A web page has finished loading
- An input field was changed
- A button was clicked
- A form was submitted

The point of event handling is that you can do something whenever an event takes place.

In Vue.js, to **listen** to DOM events you can use the **v-on** directive.

The **v-on** directive attaches an event listener to an element. The type of the event is denoted by the argument, for example **v-on:keyup** listens to the **keyup** event.



Info

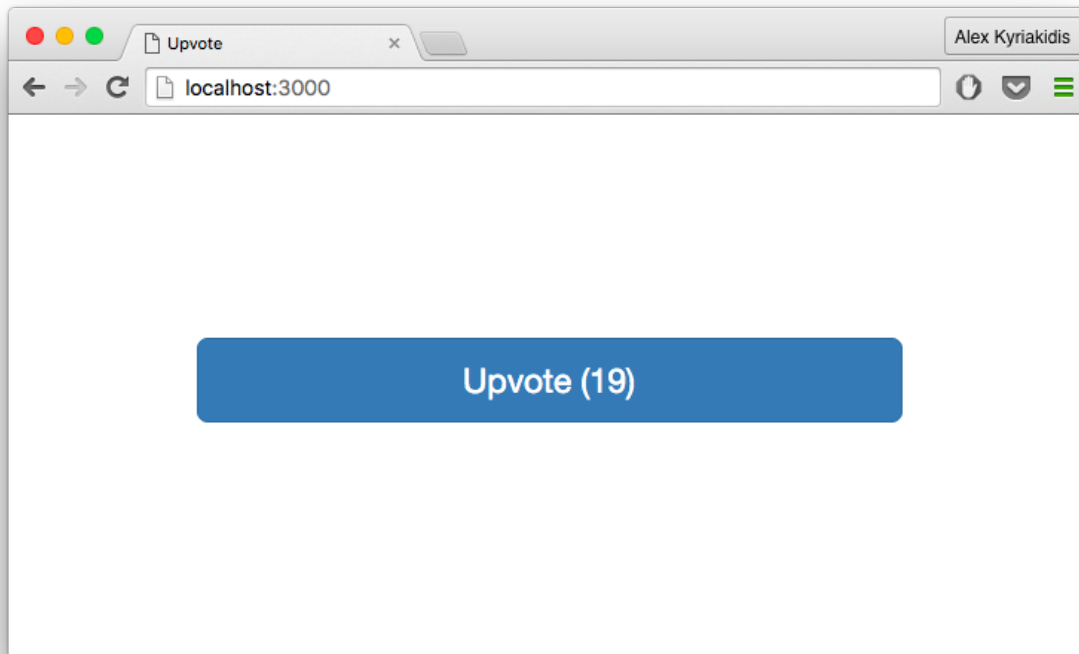
The **keyup** event occurs when the user releases a key. You can find a full list of HTML events [here](http://www.w3schools.com/tags/ref_eventattributes.asp)¹.

5.1.1 Handling Events Inline

Enough with the talking, let’s move on and see event handling in action. Below, there is an ‘Upvote’ button which increases the number of upvotes every time it gets clicked.

¹http://www.w3schools.com/tags/ref_eventattributes.asp

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4 s" rel="stylesheet">
5 <title>Upvote</title>
6 </head>
7 <body>
8   <div class="container">
9     <button v-on:click="upvotes++">
10       Upvote! {{upvotes}}
11     </button>
12   </div>
13 </body>
14 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
15 <script type="text/javascript">
16 new Vue({
17   el: '.container',
18   data: {
19     upvotes: 0
20   }
21 })
22 </script>
23 </html>
```

Upvotes counter

As you can see above, we have a basic setup and this time we use the class **container** in our view model. There is an **upvotes** variable within our data. In this case, we bind an event listener for **click**, with the statement that is right next to it. Inside the quotes we're simply increasing the count of upvotes by one, each time the button is pressed, using the increment operator (**upvotes++**).

Shown above is a very simple inline JavaScript statement.

5.1.2 Handling Events using Methods

Now we are going to do the exact same thing as before, using a method instead. A method in Vue.js is a block of code designed to perform a particular task. To execute a method you have to define it and then invoke it.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4 s" rel="stylesheet">
5 <title>Upvote</title>
6 </head>
7 <body>
8   <div class="container">
9     <button v-on:click="upvote">
10       Upvote! {{upvotes}}
11     </button>
12   </div>
13 </body>
14 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
15 <script type="text/javascript">
16 new Vue({
17   el: '.container',
18   data: {
19     upvotes: 0
20   },
21   // define methods under the **`methods`** object
22   methods: {
23     upvote: function(){
24       // **`this`** inside methods points to the Vue instance
25       this.upvotes++;
26     }
27   }
28 })
29 </script>
30 </html>
```

We are binding a click event listener to a method named `upvote`. It works just as before, but cleaner and easier to understand when reading your code.



Warning

Event handlers are restricted to execute **one statement only**.

5.1.3 Shorthand for `v-on`

When you find yourself using `v-on` all the time in a project, you will find out that your HTML will quickly become dirty. Thankfully, there is a shorthand for `v-on`, the `@` symbol. The `@` replaces the

entire `v-on:` and when using it, the code looks *a lot cleaner*, but everyone has their own practices and this is totally optional.

Using the shorthand the button of our previous example will be:

Listening to 'click' using `v-on:`:

```
1 <button v-on:click="upvote">
2   Upvote! {{upvotes}}
3 </button>
```

Listening to 'click' using `@` shorthand

```
1 <button @click="upvote">
2   Upvote! {{upvotes}}
3 </button>
```

5.2 Event Modifiers

Now we will move on and create a Calculator app. To do so, we gonna use a form with two inputs and one dropdown to select the desired operation.

Even though the following code seems fine, our calculator does not work as expected.

```
1 <html>
2 <head>
3   <title>Calculator</title>
4   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.\
5 css" rel="stylesheet">
6 </head>
7 <body>
8   <div class="container">
9     <h1>Type 2 numbers and choose operation.</h1>
10    <form class="form-inline">
11      <!-- Notice here the special attribute 'number'
12       is passed in order to parse inputs as numbers.-->
13      <input v-model="a" number class="form-control">
14      <select v-model="operator" class="form-control">
15        <option selected>+</option>
16        <option>-</option>
17        <option>*</option>
18        <option>/</option>
```

```
19     </select>
20     <!-- Notice here the special attribute 'number'
21     is passed in order to parse inputs as numbers.-->
22     <input v-model="b" number class="form-control">
23     <button type="submit" @click="calculate"
24     class="btn btn-primary">
25         Calculate
26     </button>
27 </form>
28 <h2>Result: {{a}} {{operator}} {{b}} = {{c}}</h2>
29 <pre>
30     {{$data | json}}
31 </pre>
32 </div>
33 </body>
34 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
35 <script type="text/javascript">
36     new Vue({
37         el: '.container',
38         data: {
39             a: 1,
40             b: 2,
41             c: null,
42             operator: " ",
43         },
44         methods: {
45             calculate: function(){
46                 switch (this.operator) {
47                     case "+":
48                         this.c = this.a + this.b
49                         break;
50                     case "-":
51                         this.c = this.a - this.b
52                         break;
53                     case "*":
54                         this.c = this.a * this.b
55                         break;
56                     case "/":
57                         this.c = this.a / this.b
58                         break;
59                 }
60             }
```

```
61 },
62 });
63 </script>
64 </html>
```

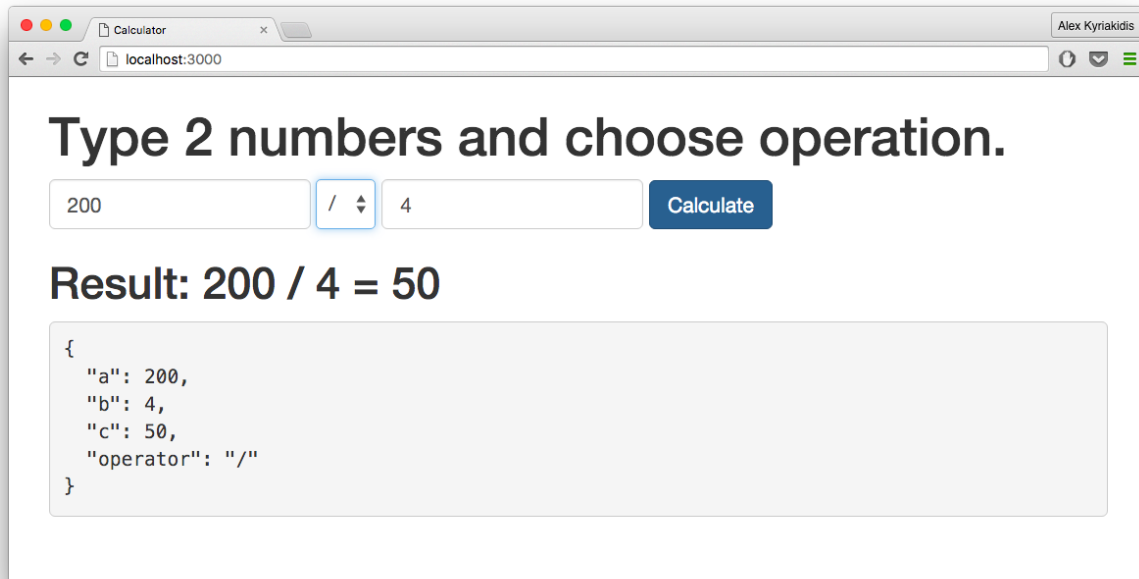
If you try and run this code yourself, you will find out that when the “calculate” button is clicked, instead of calculating, it reloads the page.

This makes sense because when you click “calculate”, in the background, you are submitting the form and thus the page reloads.

To prevent the submission of the form, we have to cancel the default action of the `onsubmit` event. It is a very common need to call `event.preventDefault()` inside our event handling method. In our case the event handling method is called `calculate`.

So, our method will become:

```
1 calculate: function(){
2     event.preventDefault();
3     switch (this.operator) {
4         case "+":
5             this.c = this.a + this.b
6             break;
7         case "-":
8             this.c = this.a - this.b
9             break;
10        case "*":
11            this.c = this.a * this.b
12            break;
13        case "/":
14            this.c = this.a / this.b
15            break;
16    }
17 }
```



Using Event Modifiers to build a calculator.

Although we can do this easily inside methods, it would be better if the methods can be purely ignorant about data logic rather than having to deal with DOM event details.

Vue.js provides two event modifiers for **v-on** to prevent the event default behavior:

1. **.prevent**
2. **.stop**

So, using one of them, our submit button will change **from**:

```
1 <button type="submit" @click="calculate">Calculate</button>
```

to:

```
1 <button type="submit" @click.prevent="calculate">Calculate</button>
2 <!-- or -->
3 <button type="submit" @click.stop="calculate">Calculate</button>
```

And we can now safely remove `event.preventDefault()` from our `calculate` method.

5.3 Key Modifiers

If you hit enter when you are focused in one of the inputs, you will notice that the page reloads again instead of calculating. This happens because we have prevented the behavior of the submit button but not of the inputs.

To fix this, we have to use ‘Key Modifiers’.

```
1 <input v-model="a" @keyup.enter="calculate">
2 <input v-model="b" @keyup.enter="calculate">
```



Tip

When you have a form with a lot of inputs/buttons/etc and you need to prevent their default submit behavior you can modify the `submit` event of the form. Example: `<form @submit.prevent="calculate">`

Finally, the calculator is up and running.

5.4 Computed Properties

Vue.js inline expressions are very convenient, but for more complicated logic, you should use computed properties. Practically, computed properties are variables which their value depends on other factors.

Computed properties work like functions that you can use as properties. But there is a significant difference, every time a dependency of a computed property changes, the value of the computed property re-evaluates.

In Vue.js, you define computed properties within the `computed` object inside your `Vue` instance.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4 s" rel="stylesheet">
5 <title>Hello Vue</title>
6 </head>
7 <body>
8 <div class="container">
9     a={{ a }}, b={{ b }}
10 <pre>
```

```
11     {{$data | json}}
12   </pre>
13 </div>
14 </body>
15 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
16 <script type="text/javascript">
17 new Vue({
18   el: '.container',
19   data: {
20     a: 1,
21   },
22   computed: {
23     // a computed getter
24     b: function () {
25       // **`this`** points to the Vue instance
26       return this.a + 1
27     }
28   }
29 });
30 </script>
31 </html>
```

This is a basic example demonstrating the use of computed properties. We've set two variables, the first, **a**, is set to 1 and the second, **b**, will be set by the returned result of the function inside the computed object. In this example the value of **b** will be set to 2.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4 s" rel="stylesheet">
5 <title>Hello Vue</title>
6 </head>
7 <body>
8 <div class="container">
9   a={{ a }}, b={{ b }}
10 <input v-model="a">
11 <pre>
12     {{$data | json}}
13 </pre>
14 </div>
15 </body>
16 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
```



```
17 <script type="text/javascript">
18 new Vue({
19   el: '.container',
20   data: {
21     a: 1,
22   },
23   computed: {
24     // a computed getter
25     b: function () {
26       // **`this`** points to the vm instance
27       return this.a + 1
28     }
29   }
30 });
31 </script>
32 </html>
```

Above there is the same example as before with one difference, there is an input binded to the **a** variable. The desired outcome is to change the value of the binded attribute and update instantly the result of **b**. But what you will notice here, is that it does not work as we would expect.

If you run this code and enter an input for variable **a** the number 5, you expect that **b** will be set to 6. Sure, but it doesn't, **b** is set to 51.

Why is this happening? Well, as you might have already thought of, **b** takes the given value from the input ("a") as a string, and appends the number 1 at the end of it.

One solution to solve this problem is to use the `parseFloat()` function that parses a string and returns a floating point number.

```
1 new Vue({
2   el: '.container',
3   data: {
4     a: 1,
5   },
6   computed: {
7     b: function () {
8       return parseFloat(this.a) + 1
9     }
10  }
11 });
```

Another option that comes to mind, is to use the `<input type="number">` that is used for input fields that should contain a numeric value.

But there is a more neat way. With Vue.js, whenever you want your user inputs to be automatically persisted as numbers, you can add the special attribute `number` to these inputs.

```

1 <body>
2 <div class="container">
3   a={{ a }}, b={{ b }}
4   <input v-model="a" number>
5   <pre>
6     {{$data | json}}
7   </pre>
8 </div>
9 </body>

```

The `number` attribute is going to give us the desired result without any further effort.

To demonstrate a wider picture of computed properties, we are going to make use of them and build the calculator we have showed before again, but this time using computed properties instead of methods.

Lets start with a simple example, where a computed property `c` contains the sum of `a` plus `b`.

```

1 <html>
2 <head>
3   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.mi\
4 n.css" rel="stylesheet">
5   <title>Hello Vue</title>
6 </head>
7 <body>
8   <div class="container">
9     <h1>Enter 2 numbers to calculate their sum.</h1>
10    <form class="form-inline">
11      <input v-model="a" number class="form-control">
12      +
13      <input v-model="b" number class="form-control">
14    </form>
15    <h2>Result: {{a}} + {{b}} = {{c}}</h2>
16    <pre> {{$data | json}} </pre>
17  </div>
18 </body>
19 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
20 <script type="text/javascript">
21   new Vue({
22     el: '.container',

```

```
23     data: {
24       a: 1,
25       b: 2
26     },
27     computed: {
28       c: function () {
29         return this.a + this.b
30       }
31     }
32   });
33 </script>
34 </html>
```

The initial code is ready, and at this point the user can type in 2 numbers and get the sum of these two. A calculator that can do the four basic operations is the goal, so let's continue building!

Since the HTML code will be the same with the [calculator we build in the previous section of this chapter](#) (except now we don't need a button), I am gonna show you here only the Javascript codeblock.

```
1   new Vue({
2     el: '.container',
3     data: {
4       a: 1,
5       b: 2,
6       operator: " ",
7     },
8     computed: {
9       c: function () {
10        switch (this.operator) {
11          case "+":
12            return this.a + this.b
13            break;
14          case "-":
15            return this.a - this.b
16            break;
17          case "*":
18            return this.a * this.b
19            break;
20          case "/":
21            return this.a / this.b
22            break;
23        }
24      }
25    }
26  });
```

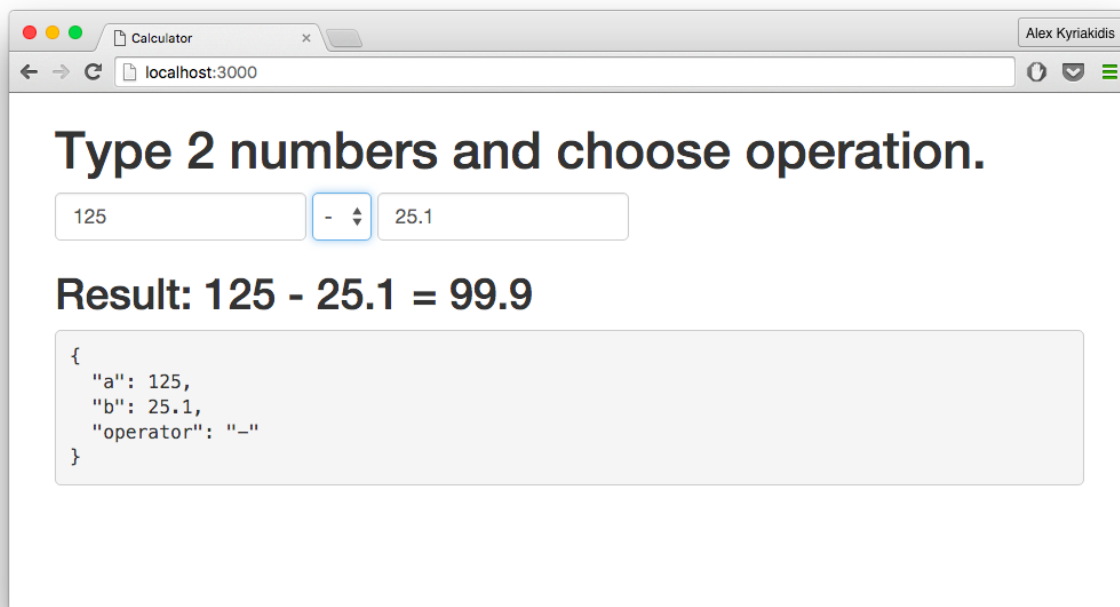
```
24         }  
25     },  
26 });
```

The calculator is ready to be put to use. We just had to move whatever was inside `calculate` method to the computed property `c` and we are done! Whenever you change the value of `a` or `b` the result updates in real time! We don't need no buttons, no events, nor anything. **How awesome is that??**



Info

Note here that a normal approach would be to have an `if` statement to avoid error for the division. The best part about this is that there is already a prediction for this kind of flaws. If the user types `1/0` the result automatically becomes infinity! If the user types a text the displayed result is “not a number”.



Calculator built with computed properties.

5.4.1 Using Computed Properties to Filter an Array

A *computed property* can also be used to filter an array. Using a computed property to perform array filtering gives you in-depth control and more flexibility, since it's full JavaScript, and allows you to

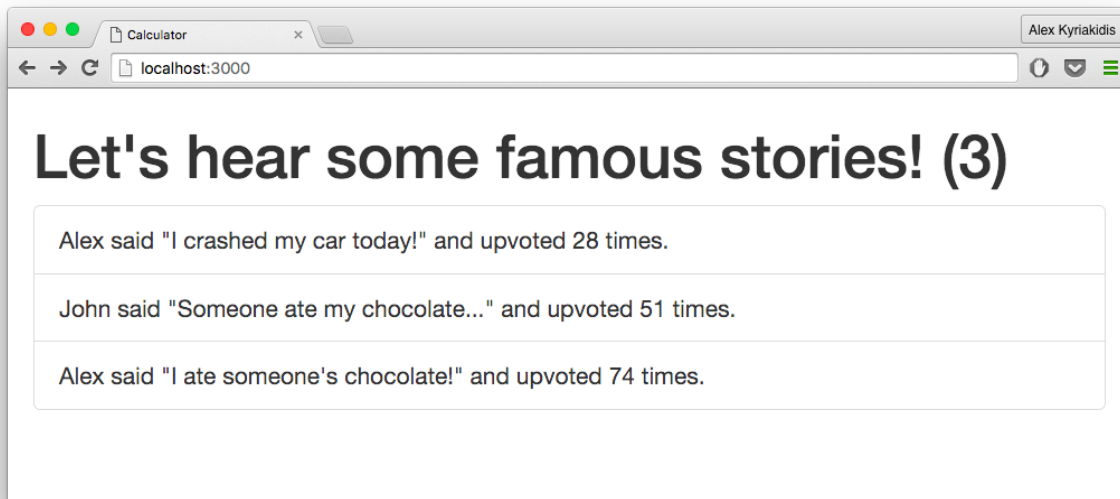
access the filtered result elsewhere. For example you can get the length of a filtered array elsewhere in your code.

To see how it's done, we will filter the **famous** stories as we did in the [Custom Filter example](#). This time we will create a computed property that returns the filtered Array.

```
1 new Vue({
2   el: '.container',
3   data: {
4     stories: [
5       {
6         plot: "I crashed my car today!",
7         writer: "Alex",
8         upvotes: 28
9       },
10      {
11        plot: "Yesterday, someone stole my bag!",
12        writer: "John",
13        upvotes: 8
14      },
15      {
16        plot: "Someone ate my chocolate...",
17        writer: "John",
18        upvotes: 51
19      },
20      {
21        plot: "I ate someone's chocolate!",
22        writer: "Alex",
23        upvotes: 74
24      },
25    ]
26  },
27  computed: {
28    famous: function() {
29      return this.stories.filter(function(item){
30        return item.upvotes > 25;
31      });
32    }
33  }
34 })
```

In our HTML code, instead of **stories** array, we will render the **famous** computed property.

```
1 <body>
2   <div class="container">
3     <h1>Let's hear some famous stories! ({{famous.length}})</h1>
4     <ul class="list-group">
5       <li v-for="story in famous"
6         class="list-group-item"
7         >
8           {{ story.writer }} said "{{ story.plot }}"
9           and upvoted {{ story.upvotes }} times.
10      </li>
11    </ul>
12  </div>
13 </body>
```



Filter array using a computed property

That's it. We have filtered our array using a computed property. Did you notice how easily we managed to display the *number of famous stories* next to our heading message using `{{ famous.length }}`?



Info

Although using a **computed property** to perform array filtering gives you more flexibility, **array filters** can be more convenient for common use cases.

5.5 Homework

Now that you have a basic understanding of Vue's event handling, methods, computed properties etc, you should try something a bit more challenging. Start by creating an array of "Mayor" candidates. Each candidate has a "name" and a number of "votes". Use a button to increase the count of votes for each candidate. Use a computed property to determine who is the current "Mayor", and display his name.

Finally when key 'c' is pressed the elections start from the beginning, and all votes become 0.



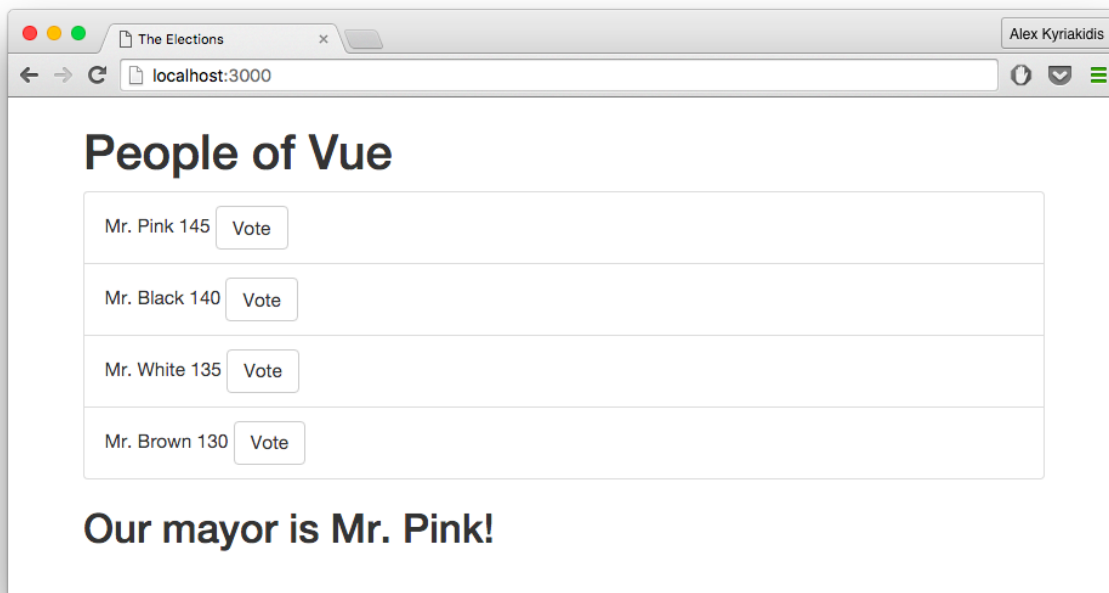
Hint

Javascript's `sort()` and `map()` methods could prove very useful and Key modifiers will get you there.



Hint 2

To listen globally for events you should target the `body` element.



Example Output

You can find a potential solution to this exercise [here](#)².

²<https://github.com/hootlex/the-majesty-of-vuejs/blob/master/homework/chapter5.html>

6. Components

6.1 What are Components?

Components are one of the most powerful features of Vue.js. They help you extend basic HTML elements to encapsulate reusable code. At a high level, Components are custom elements that Vue.js' compiler would attach specified behavior to. In some cases, they may also appear as a native HTML element extended with the special `is` attribute.

It is a really clever and powerful way to extend HTML to do new things. In this chapter we are going to start out with an extremely simple example and next we are going to see how Components can help us improve the code we have created, in some of the previous chapters.

6.2 Using Components

We are going to start with a simple Component. In order to use a component we have to register it first.

One way to register a component is to use the `Vue.component` method and pass in the `tag` and the `constructor`. Think of the `tag` as the name of the Component and the `constructor` as the options. In our occasion, we'll name the Component `story` and we'll define the property `story` (again). The option `template` (how we would like our story to be displayed), is inside the `constructor` where other options will be added as well.

Our story component will be registered like this

```
1 Vue.component('story', {  
2   template: '<h1>My horse is amazing!</h1>'  
3 });
```

Now that we have registered the component we will make use of it. We will add the custom element `<story>` inside the HTML to display the story.


```
1 <html>
2 <head>
3   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.mi\
4 n.css" rel="stylesheet">
5   <title>Hello Vue</title>
6 </head>
7 <body>
8   <div class="container">
9     <story></story>
10  </div>
11 </body>
12 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
13 <script type="text/javascript">
14 Vue.component('story', {
15   template: '<h1>My horse is amazing!</h1>'
16 });
17
18 new Vue({
19   el: '.container'
20 })
21 </script>
22 </html>
```

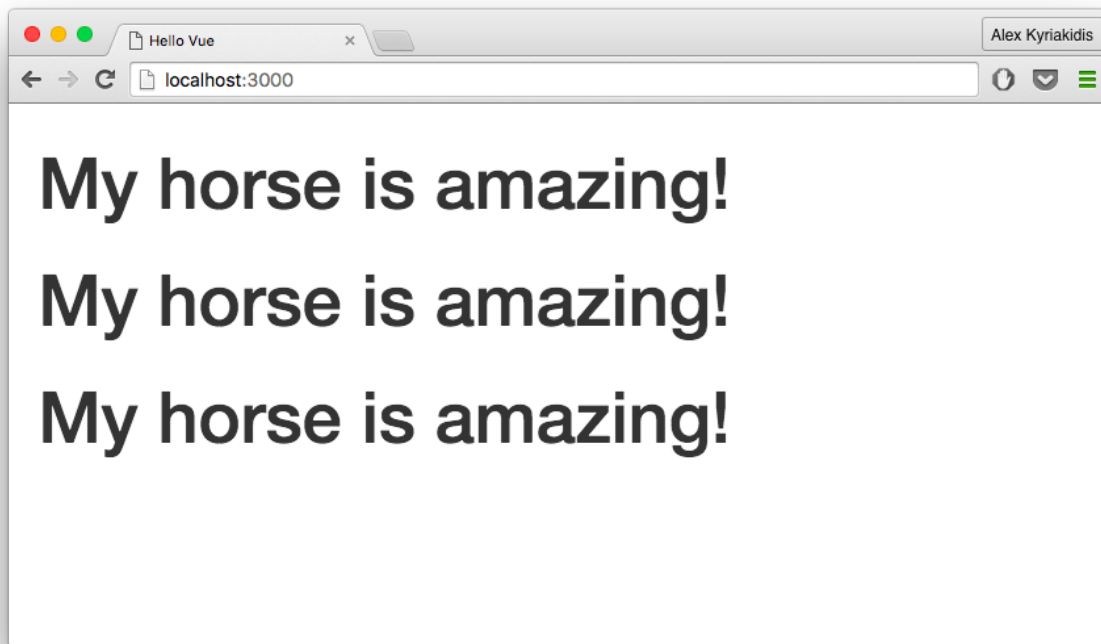


Note

Note here that you can give your custom component any name you want, but it is generally recommended that you should use a unique name to avoid having collisions with actual tags that might get introduced at some point in the future, and saving you time from having to change large amounts of code.

As we mentioned at the beginning of the chapter, components are reusable which means you can append as many `<story>` elements as you want. The following HTML snippet will display our story 3 times.

```
1 <body>
2   <div class="container">
3     <story></story>
4     <story></story>
5     <story></story>
6   </div>
7 </body>
```



Displaying story component

6.3 Templates

There is more than one way of using a template for our component. The inline template we've used before can get "dirty" very fast.

Another way to declare a template is to create a `script` tag with type set to `text/template` and set an `id` of `story-template`. To use this template we need to reference a selector in the `template` option of our component to this script.

```
1 <script type="text/template" id="story-template">
2   <h1>My horse is amazing!</h1>
3 </script>
4 <script type="text/javascript">
5   Vue.component('story', {
6     template: "#story-template"
7   });
8 </script>
```



Info

"text/template" is not a script that the browser can understand and so the browser will simply ignore it. This allows you to put anything in there, which can then be extracted and generate HTML snippets.

My favorite way to define a **template** (and the one I am gonna use in this book examples) is to create a **template** HTML tag and give it an **id**. Then we can reference a selector as we did before. Using this technique the above component will look like this:

```
1 <template id="story-template">
2   <h1>My horse is amazing!</h1>
3 </template>
4 <script type="text/javascript">
5   Vue.component('story', {
6     template: "#story-template"
7   });
8 </script>
```

6.4 Properties

Lets see now how we can use multiple instances of our **story** component to display a list of stories. We have to update the **template** to not display always the same story, but the plot of any story we want.

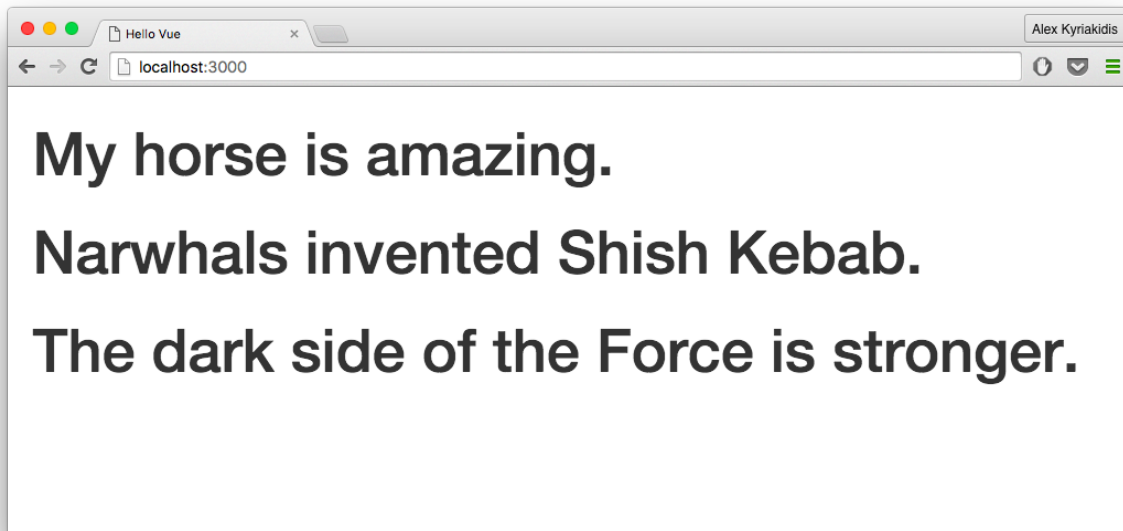
```
1 <template id="story-template">
2   <h1>{{ plot }}</h1>
3 </template>
```

We also have to update our component to use this property. To do so we will add the new property, 'plot', to **props** attribute of the component.

```
1 Vue.component('story', {
2   props: ['plot'],
3   template: "#story-template"
4 });
```

Now we can pass a `plot` and a plain string to it, every time we use the `<story>` element.

```
1 <body>
2   <div class="container">
3     <story plot="My horse is amazing."></story>
4     <story plot="Narwhals invented Shish Kebab."></story>
5     <story plot="The dark side of the Force is stronger."></story>
6   </div>
7 </body>
```



Display different 'stories'.



Warning

HTML attributes are case-insensitive. When using camelCased prop names as attributes, you need to use their kebab-case equivalents.

As you may have imagined, a component can have more than one property. For example, if we want to display the writer along with the plot for every story, we have to pass the `writer` too.

```
1 <story plot="My horse is amazing." writer="Mr. Weebl"></story>
```

If you have a lot of properties and your elements are becoming dirty you can pass an object and display its properties.

We will refactor our example one more time to wrap it up.

```
1 <html>
2 <head>
3   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.mi\
4 n.css" rel="stylesheet">
5   <title>Awesome Stories</title>
6 </head>
7 <body>
8   <div class="container">
9     <story v-bind:story="{plot: 'My horse is amazing.', writer: 'Mr. Weebl'}\
10 ">
11     </story>
12     <story v-bind:story="{plot: 'Narwhals invented Shish Kebab.', writer: 'M\
13 r. Weebl'}">
14     >
15     </story>
16     <story v-bind:story="{plot: 'The dark side of the Force is stronger.', w\
17 riter: 'Darth Vader'}">
18     >
19     </story>
20     <template id="story-template">
21       <h1>{{ story.writer }} said "{{ story.plot }}"</h1>
22     </template>
23   </div>
24 </body>
25 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
26 <script type="text/javascript">
27 Vue.component('story', {
28   props: ['story'],
29   template: "#story-template"
30 });
31
32 new Vue({
33   el: '.container'
34 })
35 </script>
36 </html>
```



Info

`v-bind` is used to dynamically bind one or more attributes, or a component prop to an expression.

Since `story` property is not a string but a javascript object instead of `story="..."` we use `v-bind:story="..."` to bind `story` property with the passed object.

The shorthand for `v-bind` is `:`, so from now on we are gonna use it like this: `:story="..."`.

6.5 Reusability

Let's take a look again at our [Filtered Results](#) example. Assume this time we take the `stories` variable data from an external API through an http call. The API developers decide to rename `plot` story property to `body`. So now, we have to go through our code and make the necessary changes.



Info

Later in this book we will cover how we can use `Vue` to make web requests.

```

1 <div class="container">
2   <h1>Lets hear some stories!</h1>
3   <div>
4     <h3>Alex's stories</h3>
5     <ul class="list-group">
6       <li v-for="story in stories | filterBy 'Alex' in 'writer'"
7         class="list-group-item"
8       >
9             {{ story.writer }} said "{{ story.plot }}"
10        {{ story.writer }} said "{{ story.body }}"
11      </li>
12    </ul>
13    <h3>John's stories</h3>
14    <ul class="list-group">
15      <li v-for="story in stories | filterBy 'John' in 'writer'"
16        class="list-group-item"
17      >
18            {{ story.writer }} said "{{ story.plot }}"
19        {{ story.writer }} said "{{ story.body }}"
20      </li>
21    </ul>

```

```
22     <div class="form-group">
23       <label for="query">
24         What are you looking for?
25       </label>
26       <input v-model="query" class="form-control">
27     </div>
28     <h3>Search results:</h3>
29     <ul class="list-group">
30       <li v-for="story in stories | filterBy query in 'body'"
31         class="list-group-item"
32       >
33         ----- {{ story.writer }} said "{{ story.plot }}"
34         {{ story.writer }} said "{{ story.body }}"
35       </li>
36     </ul>
37   </div>
38 </div>
```



Note

In this particular example syntax highlighting is turned off.

As you may have noticed, we had to do the exact same change 3 times and I don't know about you, but I hate repeating myself. If it doesn't seem like a big deal for you, imagine that you may use the above code block in 100 places, what would you do then? Fortunately, 'Vue' has a solution for that kind of situations, and this solution has a name, **Component**.



Tip

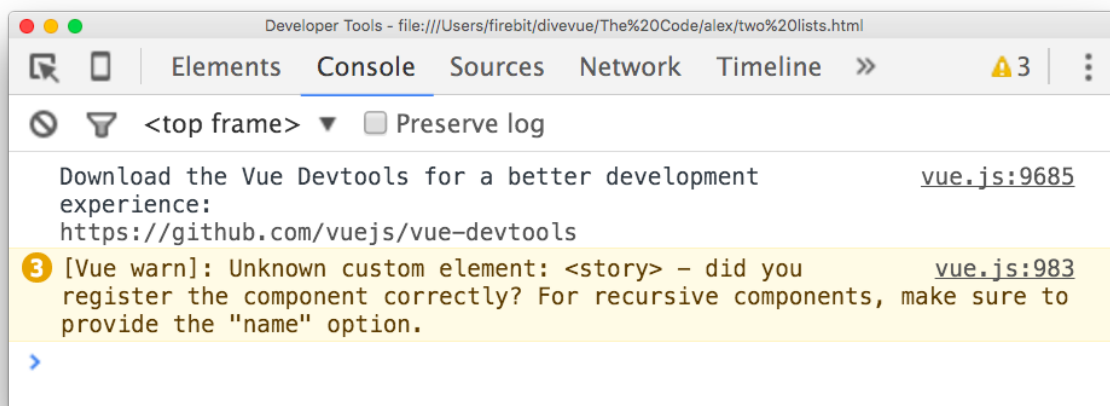
Whenever you find yourself repeating a piece of functionality, the most efficient way to deal with it is to create a dedicated Component.

Luckily we have created a **story** Component in the previous example, which displays the writer and the body for a specified story. We can use the custom element `<story>` inside our **HTML** and pass each story as we did before with `:story` tag but this time we will do it inside `v-for` directive.

So our code will be:

```
1 <div class="container">
2   <h1>Lets hear some stories!</h1>
3   <div>
4     <h3>Alex's stories</h3>
5     <ul class="list-group">
6       <story v-for="story in stories | filterBy 'Alex' in 'writer'"
7         :story="story"></story>
8     </ul>
9     <h3>John's stories</h3>
10    <ul class="list-group">
11      <story v-for="story in stories | filterBy 'John' in 'writer'"
12        :story="story"></story>
13    </ul>
14    <div class="form-group">
15      <label for="query">What are you looking for?</label>
16      <input v-model="query" class="form-control">
17    </div>
18    <h3>Search results:</h3>
19    <ul class="list-group">
20      <story v-for="story in stories | filterBy query in 'body'"
21        :story="story"></story>
22    </ul>
23  </div>
24 </div>
```

If you try to run this code you will get the following warning.



Vue warning

Vue warn: Unknown custom element: <story> - did you register the component correctly? For recursive components, make sure to provide the “name” option.

To fix this we need to register the Component again. This time we have to make some changes to the component's template. We will change `plot` attribute to `body` and `<h1>` tag to `` to suit our needs.

So, the story's template will be:

```
1 <template id="story-template">
2   <li class="list-group-item">
3     {{ story.writer }} said "{{ story.plot }}"
4   </li>
5 </template>
```

But the component will be the same.

```
1 Vue.component('story', {
2   props: ['story'],
3   template: '#story-template'
4 });
```

If you run the above code you will see for yourself that everything works same as before but this time with the use of a custom component.

Pretty neat huh?

6.6 Altogether now

Using our newly acquired knowledge we should be able to build something a bit more complex. Taking the structure example from before, we are going to create a voting system for our **stories**, and add a favorite feature. The way to accomplish these is through methods, directives, and of course, components.

Lets start with the stories setup.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4 s" rel="stylesheet">
5   <title>Hello Vue</title>
6 </head>
7 <body>
8 <div id="app">
9   <div class="container">
10     <h1>Let's hear some stories!</h1>
11     <ul class="list-group">
12       <story v-for="story in stories" :story="story"></story>
13     </ul>
14     <pre>{{ $data | json }}</pre>
15   </div>
16 </div>
17 <template id="story-template">
18   <li class="list-group-item">
19     {{ story.writer }} said "{{ story.plot }}"
20   </li>
21 </template>
22 </body>
23 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
24 <script type="text/javascript">
25 Vue.component('story', {
26   template: "#story-template",
27   props: ['story'],
28 });
29
30 new Vue({
31   el: '#app',
32   data: {
33     stories: [
34       {
35         plot: 'My horse is amazing.',
36         writer: 'Mr. Weebl',
37       },
38       {
39         plot: 'Narwhals invented Shish Kebab.',
40         writer: 'Mr. Weebl',
41       },
42       {
```

```

43         plot: 'The dark side of the Force is stronger.',
44         writer: 'Darth Vader',
45     },
46     {
47         plot: 'One does not simply walk into Mordor',
48         writer: 'Boromir',
49     },
50 ]
51 }
52 })
53 </script>
54 </html>

```

The next step is to give the user a way to vote once, the story he desires to. To apply this limit (1 vote per story) we will display the ‘Upvote’ button only if user has not already voted. So, every story must have a **voted** property that becomes true when **upvote** function executes.

```

1 <template id="story-template">
2   <li class="list-group-item">
3     {{ story.writer }} said "{{ story.plot }}" .
4     Story upvotes {{ story.upvotes }} .
5     <button v-show="!story.voted" @click="upvote"
6         class="btn btn-default"
7     >
8       Upvote
9     </button>
10  </li>
11 </template>

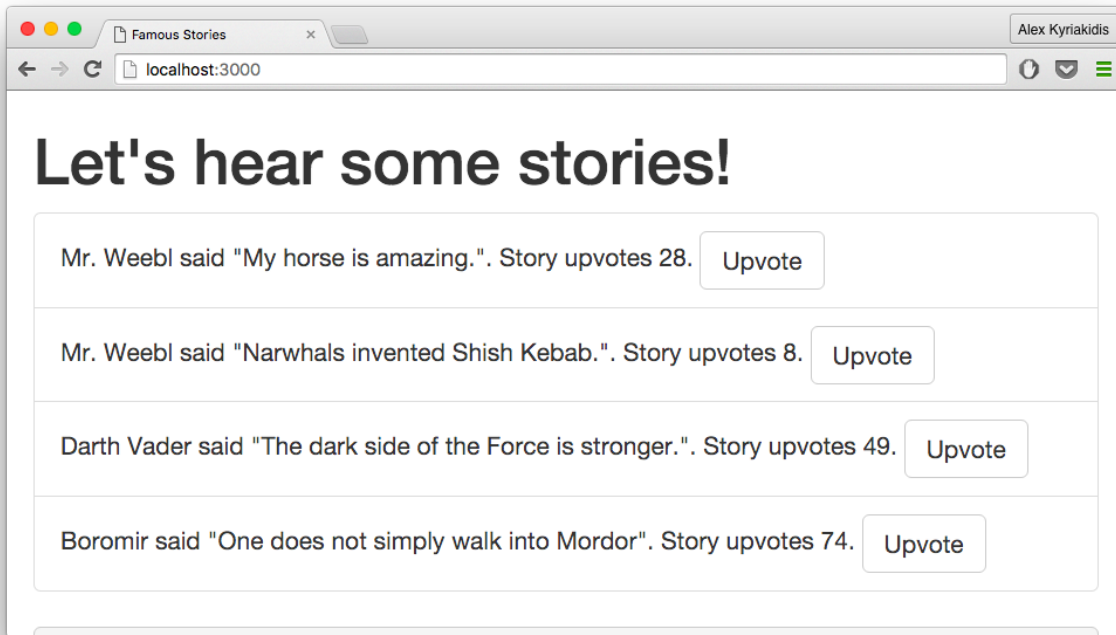
```

```

1 Vue.component('story', {
2   template: "#story-template",
3   props: ['story'],
4   methods: {
5     upvote: function(){
6       this.story.upvotes += 1;
7       this.story.voted = true;
8     },
9   }
10 });
11
12 new Vue({

```

```
13     el: '#app',
14     data: {
15         stories: [
16             {
17                 plot: 'My horse is amazing.',
18                 writer: 'Mr. Weebl',
19                 upvotes: 28,
20                 voted: false,
21             },
22             {
23                 plot: 'Narwhals invented Shish Kebab.',
24                 writer: 'Mr. Weebl',
25                 upvotes: 8,
26                 voted: false,
27             },
28             {
29                 plot: 'The dark side of the Force is stronger.',
30                 writer: 'Darth Vader',
31                 upvotes: 49,
32                 voted: false,
33             },
34             {
35                 plot: 'One does not simply walk into Mordor',
36                 writer: 'Boromir',
37                 upvotes: 74,
38                 voted: false,
39             },
40         ]
41     }
42 })
```



Ready to vote!

We have implemented, with the use of methods, the voting system. I think it looks good, so we can continue with the ‘favorite story’ part. We want the user to be able to choose only one story to be his favorite. The first thing that comes to my mind is to add one new empty object (favorite) and whenever the user chooses one story to be his favorite, update **favorite** variable. This way we will be able to check if a story is equal to the user’s favorite story. Let’s do this.

```

1 <template id="story-template">
2   <li class="list-group-item">
3     {{ story.writer }} said "{{ story.plot }}".
4     Story upvotes {{ story.upvotes }}.
5     <button v-show="!story.voted" @click="upvote"
6     class="btn btn-default">
7       Upvote
8     </button>
9     <button v-show="!isFavorite" @click="setFavorite"
10    class="btn btn-primary">
11       Favorite
12    </button>
13    <span v-show="isFavorite"
14    class="glyphicon glyphicon-star pull-right" aria-hidden="true">
15    </span>

```

```
16     </li>
17 </template>

1  Vue.component('story', {
2    template: "#story-template",
3    props: ['story'],
4    methods: {
5      upvote: function(){
6        this.story.upvotes += 1;
7        this.story.voted = true;
8      },
9      setFavorite: function(){
10       this.favorite = this.story;
11     },
12   },
13   computed: {
14     isFavorite: function(){
15       return this.story == this.favorite;
16     },
17   }
18 });

19
20 new Vue({
21   el: '#app',
22   data: {
23     stories: [
24       ...
25     ],
26     favorite: {}
27   }
28 })
```

If you try to run the above code, you will notice that it does not work as it should be. Whenever you try to favorite a story, the variable **favorite** inside **\$data** remains null and we get none response.

It seems that our **story** component is unable to update **favorite** object, so we are going to pass it on each story and add **favorite** to component's properties.

```

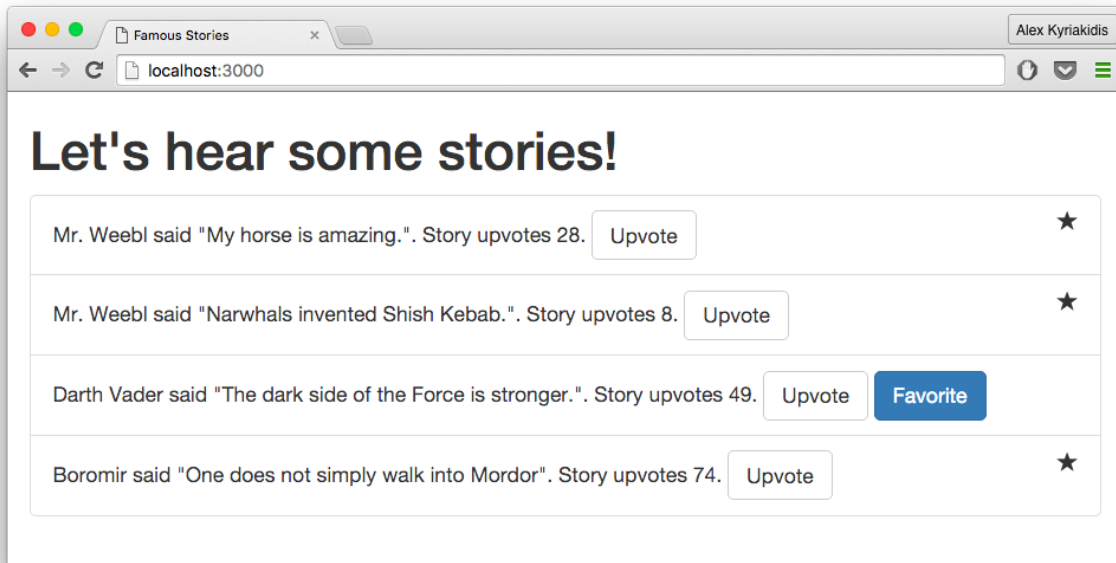
1 <ul class="list-group">
2   <story v-for="story in stories"
3     :story="story"
4     :favorite="favorite">
5   </story>
6 </ul>

```

```

1 Vue.component('story', {
2   ...
3   props: ['story', 'favorite'],
4   ...
5 });

```



setFavorite method malfunctioning

Hmmm, **favorite** still doesn't get updated when **setFavorite** is executed. The button disappears as expected and a star icon appears but variable **favorite** is still null. This results in the user being able to favorite all stories.

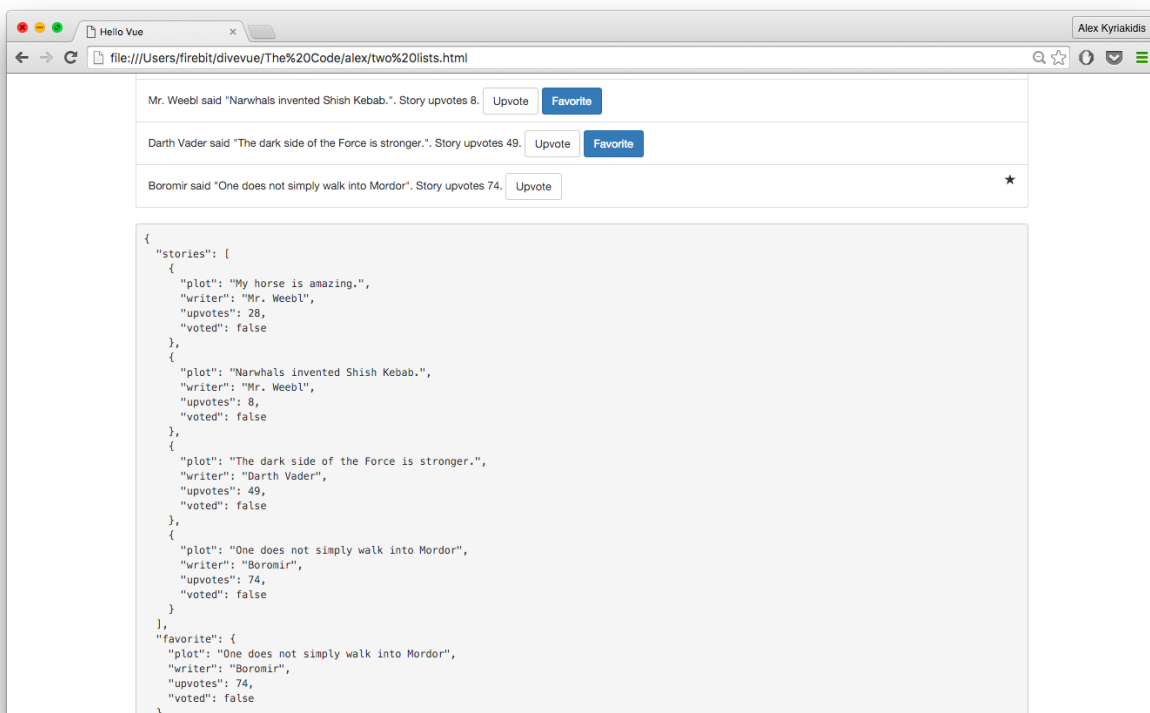
The problem with this approach is that we don't keep things *synced*. By default, all props form a one-way-down binding between the child property and the parent one. When the parent property updates, it will flow down to the child, but not the other way around.

This may be confusing but stick with me. In Vue, you can enforce a **two-way binding** with *.sync binding type modifier*. So, we will pass the variable **favorite** to each story like this **:favorite.sync="favorite"**.

```

1 <div id="app">
2   <div class="container">
3     <h1>Let's hear some stories!</h1>
4     <ul class="list-group">
5       <story v-for="story in stories"
6         :story="story"
7         :favorite.sync="favorite">
8     </story>
9   </ul>
10  <pre>
11    {{ $data | json }}
12  </pre>
13 </div>
14 </div>

```



Favorite only one story

Now, the desired result is achieved and the user is able to choose only one story to be his favorite while he can vote as many stories as he wants. With the use of `.sync` we have synced the property 'favorite' and made the binding two-way with the `favorite` object.

Before the `.sync` is a one-way down binding, after is a two-way binding, keeping them

synchronized.

6.7 Homework

This is the most difficult exercise so far, so make sure to put in use everything you have learned in this book. Create an array of 4 horse-drawn chariots. Each chariot has a “name” and a number of “horses” (from 1 to 4). Create a component named “chariot”. The “chariot” component should display the name of the chariot and the number of the horses it has. It also must have an action button. The button’s text depends on the currently selected chariot.

More specifically button’s text should be:

- ‘Pick Chariot’, before the user has chosen any chariot
- ‘Dismiss Horses’, when the chariot has less horses than the selected chariot
- ‘Hire Horses’, when the chariot has more horses than the selected chariot
- ‘Riding!’, when the chariot is the selected chariot (this button has to be disabled)

The user should be able to pick a chariot and then choose between any chariot he wants to.

Example Scenario: User has chosen a chariot with 2 horses and its button says ‘Riding!’. A chariot with 3 horses has one more horse, so its button says ‘Hire Horses’. A chariot with 1 horse has one less horse than user’s chariot, so it’s button says ‘Dismiss Horses’. I think you got the idea..



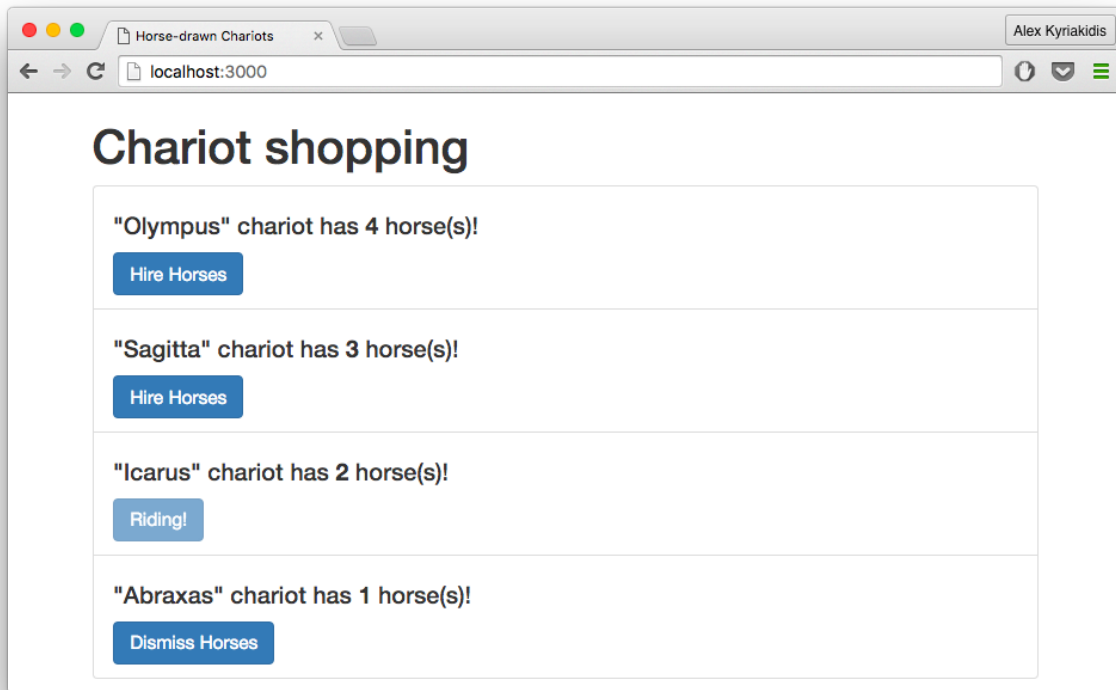
Hint

You need to use two-way binding between child’s `selectedChariot` property and parent’s one.



Hint

To disable a button use `disabled="true"` attribute. You have to figure out how to apply it conditionally.



Example Output

You can find a potential solution to this exercise [here](#)¹.

¹<https://github.com/hootlex/the-majesty-of-vuejs/blob/master/homework/chapter6.html>

7. Class and Style Bindings

7.1 Class binding

7.1.1 Object Syntax

A common need for data binding is manipulating an element's class and its styles. For such cases, you can use `v-bind:class`. This can be used to apply classes conditionally, toggle them and/or apply many of them using one binded object et al.

The `v-bind:class` directive takes an object with the following format as an argument

```
1 {  
2   'classA': true,  
3   'classB': false,  
4   'classC': true  
5 }
```

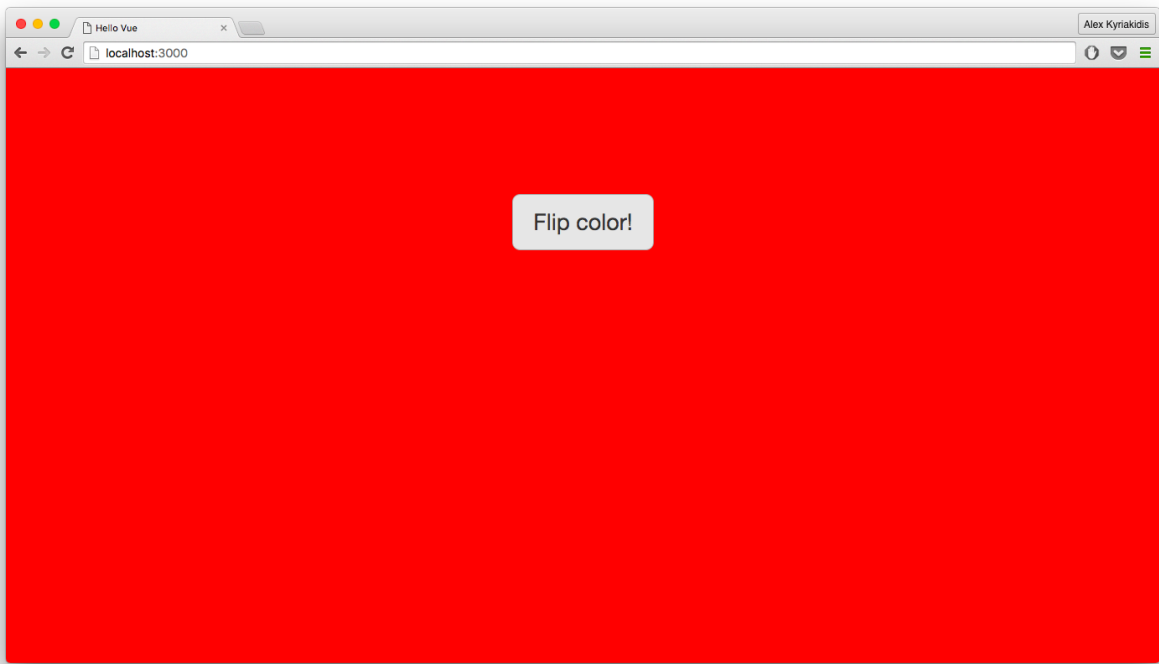
and applies all classes with `true` value to the element. For example, the following element will have `classA` and `classC` classes.

```
1 <div v-bind:class="elClasses"></div>
```

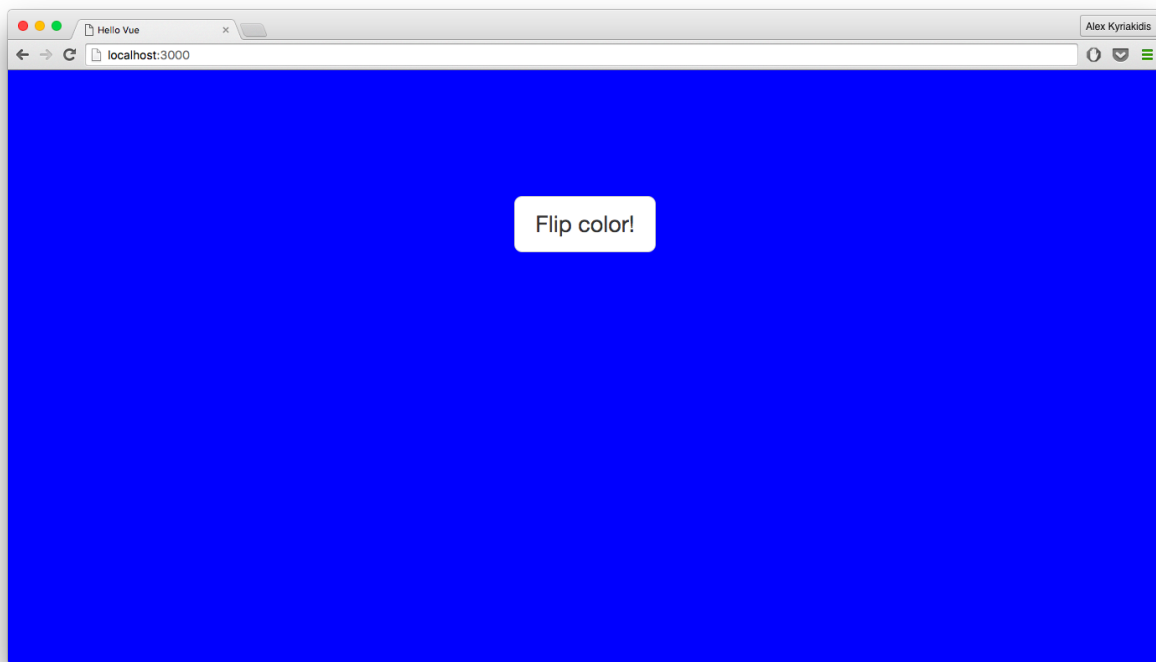
```
1 data: {  
2   elClasses:  
3     {  
4       'classA': true,  
5       'classB': false,  
6       'classC': true  
7     }  
8 }
```

To demonstrate how `v-bind` is used with class attributes, we are going to make an example of class toggling. Using `v-bind:class` directive, we are going to dynamically toggle the class of `body` element.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4 s" rel="stylesheet">
5 <title>Hello Vue</title>
6 </head>
7 <body class="text-center"
8 v-bind:class="{ 'body-red' : color, 'body-blue' : !color }"
9 >
10   <button v-on:click="flipColor" class="btn">
11     Flip color!
12   </button>
13 </body>
14 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
15 <script type="text/javascript">
16 new Vue({
17   el: 'body',
18   data: {
19     color: true
20   },
21   methods: {
22     flipColor: function() {
23       this.color = !this.color;
24     }
25   }
26 });
27 </script>
28 <style type="text/css">
29   .body-red {
30     background-color: #ff0000;
31   }
32   .body-blue {
33     background-color: #0000ff;
34   }
35 </style>
36 </html>
```



Changed background color



Changed background color

We have applied a class to the **body** for our convenience and now **body** is referenced within our **el** property. What this code actually does, is “flipping” the background color with a hit of the button. Pressing it invokes the **flipColor** function that reverses the value of “color” originally set to **true**. Then the **v-bind:class** is going to toggle the class name to ‘body-red’ or ‘body-blue’ conditionally depending on the truthfulness of “color” value. That given, the style is going to apply on each class and give us the desired result according to which class is active.



Info

The **v-bind:class** directive can co-exist with the plain class attribute.

So in our example, **body** always has the **text-center** class and conditionally one of **body-red** or **body-blue**.



Warning

Although you can use mustache interpolations such as `class="{{ className }}"` to bind the class, it is not recommended to mix that style with **v-bind:class**. Use one or the other!

7.1.2 Array Syntax

We can also apply a list of classes to an element using an array of classnames.

```
1 <div v-bind:class="['classA', 'classB', anotherClass]"></div>
```

Applying conditionally a class, can also be achieved with the use of inline `if` inside the array.

```
1 <div v-bind:class="['classA', condition ? 'classB' : '']"></div>
```



Info

Inline `if` is commonly referred to as the **ternary operator**, **conditional operator**, or **ternary if**.

The conditional (ternary) operator is the only JavaScript operator that takes three operands.

The syntax of **ternary operator** is **condition ? expression1 : expression2**. If condition is true, the operator returns the value of expression1, otherwise, it returns the value of expression2.

Using inline `if`, the flipping colors example will look like:

```
1 <body class="text-center body"
2 v-bind:class="[ color ? 'body-red' : 'body-blue']">
3   <button v-on:click="flipColor"
4     class="btn">
5     Flip color!
6   </button>
7 </body>
```

```
1 new Vue({
2   el: 'body',
3   data: {
4     color: true
5   },
6   methods: {
7     flipColor: function() {
8       this.color = !this.color;
9     }
10  }
11 });
```




Tip

To actually use a class name instead of a variable inside classes array, use single quotes.
`v-bind:class="[variable, 'classname']"`

7.2 Style binding

7.2.1 Object Syntax

The Object syntax for `v-bind:style` is pretty straightforward; it looks almost like CSS, except it's a JavaScript object. We are going to use the shorthand Vue.js provides for the previously used directive, `v-bind(:)`.

```
1 <!-- shorthand -->
2 <div :style="niceStyle"></div>
```

```
1 data: {
2   niceStyle:
3   {
4     color: 'blue',
5     fontSize: '20px'
6   }
7 }
```

We can also declare the style properties inside an object `:style="..."` inline.

```
1 <div :style="{color: 'blue', fontSize: '20px'}"></div>
```

We can even **reference variables** inside style object:

```
1 <!-- Variable 'niceStyle' is the same we used in the previous example -->
2 <div :style="{color: niceStyle.color, fontSize: niceStyle.fontSize}">
3 </div>
```



Style object binding

It is often a good idea to use a style object and bind it, so the template is cleaner.

7.2.2 Array Syntax

Using inline array syntax for `v-bind:style`, we are able to apply multiple style objects to the same element, meaning here that every list item is going to have the `color` and `fontSize` of `niceStyle` and the font weight of `badStyle`.

```
1 <!-- shorthand -->
2 <div :style="[niceStyle, badStyle]"></div>
```

```
1 data: {
2   niceStyle:
3     {
4       color: 'blue',
5       fontSize: '20px'
6     }
7   badStyle:
8     {
9       fontweight: 'bold'
10    }
11 }
```



Info for Intermediates

When you use a CSS property that requires vendor prefixes in `v-bind:style`, for example `transform`, Vue.js will automatically detect and add appropriate prefixes to the applied styles.

You can find more information about vendor prefixes [here](#)¹.

7.3 Bindings in Action

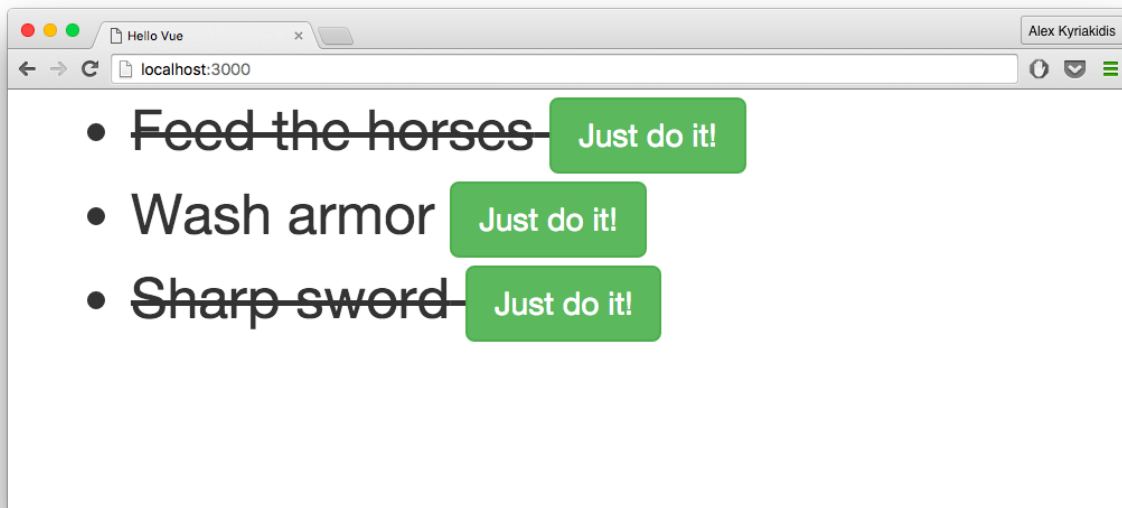
```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4 s" rel="stylesheet">
5 <title>Hello Vue</title>
6 </head>
7 <body class="container-fluid">
8   <ul>
9     <li :class="{ 'completed' : task.done}"
10       :style="styleObject"
11       v-for="task in tasks">
12       {{task.body}}
13       <button @click="completeTask(task)" class="btn">
14         Just do it!
15       </button>
16     </li>
17   </ul>
18 </body>
19 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.js"></script>
20 <script type="text/javascript">
21 new Vue({
22   el: 'body',
23   data: {
24     tasks: [
25       {body: "Feed the horses", done: true},
26       {body: "Wash armor", done: true},
27       {body: "Sharp sword", done: false},
28     ],
29     styleObject: {
30       fontSize: '25px'
31     }

```

¹https://developer.mozilla.org/en-US/docs/Glossary/Vendor_Prefix

```
32     },
33     methods: {
34         completeTask: function(task) {
35             task.done = !task.done;
36         }
37     },
38 });
39 </script>
40 <style type="text/css">
41     .completed {
42         text-decoration: line-through;
43     }
44 </style>
45 </html>
```

The above example has an array of objects called “tasks” and a styleObject which contains only one property. With the use of `v-for`, a list of tasks is rendered and each task has a “done” property with a boolean value. Depending on the value of “done”, a class is applied conditionally as before. If a task has been completed, then `css` style applies and then task has a text-decoration of line-through. Each task is accompanied by a button listening for the “click” event which triggers a method, altering the completion status of the task. The `style` attribute is bound to `styleObject` resulting in the change of ‘fontsize’ of all tasks. As you can see, the `completedTasks` method takes in the parameter `task`.



Styling completed tasks

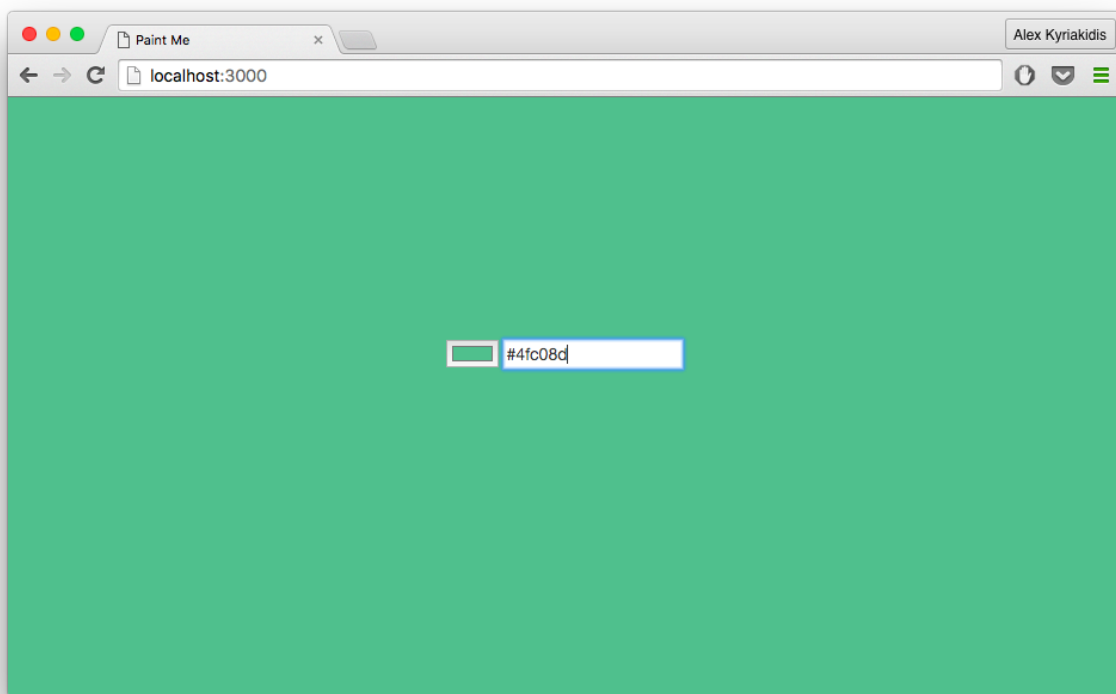
7.4 Homework

A more fun but maybe tricky exercise for this chapter. Create an input where the user can choose a color. When a color is chosen, apply it to the `body`. That's it, **let's paint!!** :)



Hint

You can use `input type="color"` for your ease.



Example Output

You can find a potential solution to this exercise [here](https://github.com/hootlex/the-majesty-of-vuejs/blob/master/homework/chapter7.html)².

²<https://github.com/hootlex/the-majesty-of-vuejs/blob/master/homework/chapter7.html>

Consuming an API

8. Preface

In this chapter, we are going to go a little deeper and demonstrate how we can use Vue.js to consume an API.

Following the *story* examples of previous chapters, we will now use some real data, coming from an external source.

In order to use real data, we need to make use of a database. Assuming that you already know how to create a database, it won't be covered in this book. To work along with the book's examples, we got you covered, we have already created one to be put to use.

8.1 CRUD

Presume we have a database, we need to perform CRUD operations (Create, Read, Update, Delete). More particularly, we want to

- **Create** new stories in the database
- **Read** existing stories
- **Update** existing story's details (such as 'upvotes')
- **Delete** stories that we don't like

Since Vue.js is a Front-end JavaScript framework, it cannot connect to a database directly. To access a database we need a layer between Vue.js and the database, this layer is the API (Application Program Interface).

8.2 API

Because this book is about Vue.js and not about designing APIs, we will provide you a demo API built with [Laravel](https://laravel.com/)¹. Laravel is one of most powerful PHP frameworks along with Symfony2, Nette, CodeIgniter, and Yii2. You are free to create your API using *any language or framework* you like. I use Laravel because it is simple, it has a great community, and it is awesome! :)

Therefore, we strongly recommend to use the *demo API* that we have built exclusively for the book's examples.

¹<https://laravel.com/>

8.2.1 Download Book's Code

To use our API you have to download the book's code and start a server. To do so, follow the instructions below.

1. Open your terminal and create a directory (we will create '~/themajestyofvuejs')

```
$ mkdir ~/themajestyofvuejs
```

2. Download the source code from github

```
$ cd ~/themajestyofvuejs
$ git clone https://github.com/hootlex/the-majesty-of-vuejs .
```

Alternatively, you can visit the repository on [github](https://github.com/hootlex/the-majesty-of-vuejs)² and download the zip file. Then, extract its contents under the created directory.

3. Navigate to the current chapter under 'apis' of the newly created directory.

```
$ cd ~/themajestyofvuejs/apis/stories
```

4. Run the installation script

```
$ sh setup.sh
```

5. You now have a database filled with dummy data as well as a fully functional server **running on 'http://localhost:3000'!**

If you want to customize the server(host, port, etc), you can make the setup manually. Below is the source code of our script.

Installation Script: setup.sh

```
# navigate to chapter directory
$ cd ~/themajestyofvuejs/apis/stories

# install dependencies
$ composer install

# Create the database
$ touch database/database.sqlite;

# Migrate & Seed
$ php artisan migrate;
```

²<https://github.com/hootlex/the-majesty-of-vuejs>


```
$ php artisan db:seed;

# Start server
$ php artisan serve --port=3000 --host localhost;
```

Great! You now have a *fully functional API* and a database filled with nice stories.



Note

If you are using Vagrant you have to run the server on **host '0.0.0.0'**. Then, you will be able to access your server on Vagrant's box ip.

If, for example, Vagrant's box ip is `192.168.10.10` and you run

```
$ php artisan serve --port=3000 --host 0.0.0.0;
```

you can browse your website on `192.168.10.10:3000`.

If you have downloaded our demo API, you can continue to the next section.

If you chose to create you own API, you have to create a database table to store the stories. The following columns must be present.

Column Name	Type
<code>id</code>	Integer, Auto Increment
<code>plot</code>	String
<code>writer</code>	String
<code>upvotes</code>	Integer, Unsigned

Don't forget to seed some fake data to follow up with the next examples.

8.2.2 API Endpoints

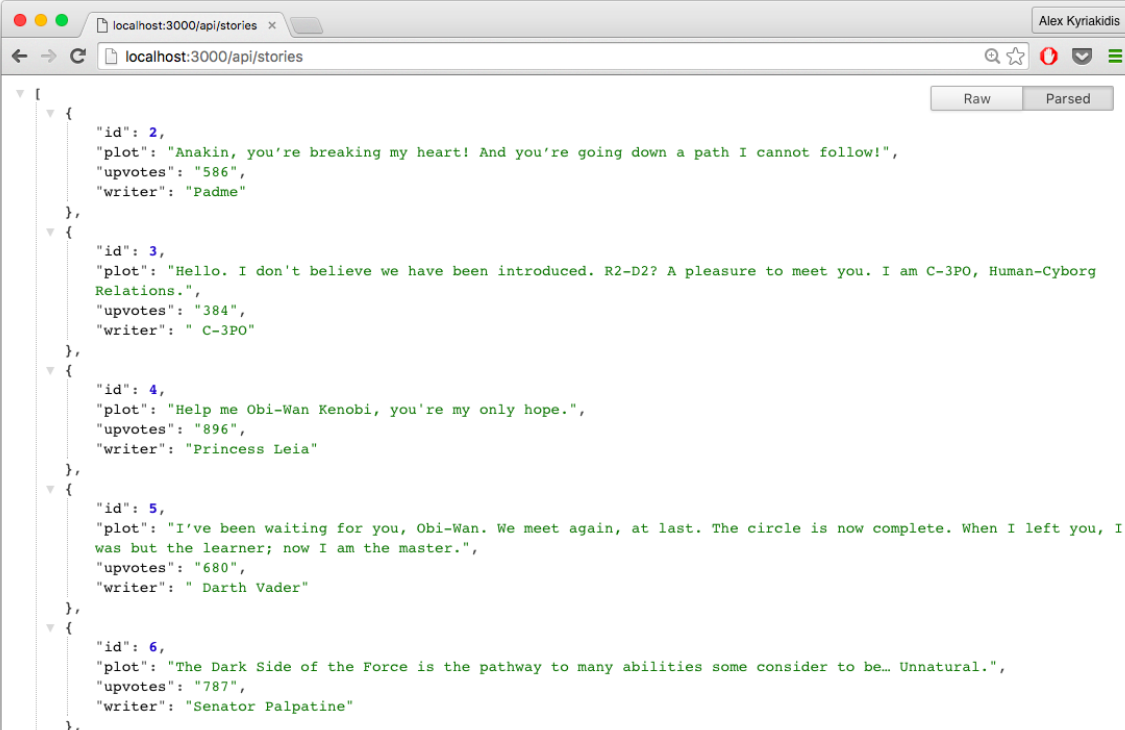
An endpoint is simply a URL. When you go to `http://example.com/foo/bar` then that is an endpoint and you simply need to call it `/foo/bar` because the domain will be the same for all the endpoints.

To manage the **Story** resource we need 5 endpoints. Each endpoint corresponds to a specific action.

HTTP Method	URI	Action
GET/HEAD	<code>api/stories</code>	<i>Fetches</i> all stories
GET/HEAD	<code>api/stories/{id}</code>	<i>Fetches</i> specified story
POST	<code>api/stories</code>	<i>Creates</i> a new story
PUT/PATCH	<code>api/stories/{id}</code>	<i>Updates</i> an existing story
DELETE	<code>api/stories/{id}</code>	<i>Deletes</i> specified story

As indicated in the above table, to get a listing with all the ‘stories’ we have to make an HTTP GET or HEAD request to `api/stories`. To update an existing story we have to make an HTTP PUT or PATCH request to `api/stories/{storyID}` providing the data we want to override, and replacing `{storyID}` with the id of the story we want to update. The same logic applies to all endpoints, I think you get the idea.

Assuming your server is running on `http://localhost:3000`, you can view a listing of all stories in JSON format by visiting `http://localhost:3000/api/stories` on your web browser.



```
{
  "id": 2,
  "plot": "Anakin, you're breaking my heart! And you're going down a path I cannot follow!",
  "upvotes": "586",
  "writer": "Padme"
},
{
  "id": 3,
  "plot": "Hello. I don't believe we have been introduced. R2-D2? A pleasure to meet you. I am C-3PO, Human-Cyborg Relations.",
  "upvotes": "384",
  "writer": "C-3PO"
},
{
  "id": 4,
  "plot": "Help me Obi-Wan Kenobi, you're my only hope.",
  "upvotes": "896",
  "writer": "Princess Leia"
},
{
  "id": 5,
  "plot": "I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.",
  "upvotes": "680",
  "writer": "Darth Vader"
},
{
  "id": 6,
  "plot": "The Dark Side of the Force is the pathway to many abilities some consider to be... Unnatural.",
  "upvotes": "787",
  "writer": "Senator Palpatine"
}
```

JSON response



Tip

Reading raw JSON data on browser can be a pain . It is always easier to read a **well formatted JSON**. Chrome has some great extensions that could format raw JSON data into tree view format that can be easily read.

I use [JSONFormatter³](#) because it supports syntax highlighting and displays JSON in tree view where the nodes on the tree can be collapsed or expanded by clicking the triangle icon on the left of each node. It also provides a button for switching to original (raw) data.

You can choose whichever extension you like but you should **definitely use one!**

³<https://chrome.google.com/webstore/detail/json-formatter/bejindccaagfpajjmafapmmgkxhgoa>

9. Working with real data

It is time to actually put to use our database and perform the operations we have mentioned (CRUD). We will utilize the [last example from the Components chapter](#), but this time of course our data will be coming from an external source. To exchange data with the server we need to perform asynchronous HTTP (Ajax) requests.



Info

AJAX is a technique that allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes.

9.1 Get Data Asynchronous

Take a moment to have a look at the [last example from the Components chapter](#). As you can see we hardcode `stories` array inside the `data` object of `Vue` instance.

Stories array hardcoded

```
1 new Vue({
2   data: {
3     stories: [
4       {
5         plot: 'My horse is amazing.',
6         writer: 'Mr. Weebl',
7       },
8       {
9         plot: 'Narwhals invented Shish Kebab.',
10        writer: 'Mr. Weebl',
11      },
12      ...
13    ]
14  }
15 })
```

This time, we want to fetch the existing stories from the server.

To do so, we'll perform a HTTP GET request using jQuery at first. Later on this chapter, we will migrate to [vue-resource](#)¹ to see the differences between the two of them.

To make the AJAX call we are going to use `$.get()`, a jQuery function that loads data from the server using a HTTP GET request. Full documentation for `$.get()` can be found [here](#)².



Info

`vue-resource` is a plugin for `Vue.js` that provides services for making web requests and handle responses.

The `$.get()` method's syntax is

```
1 $.get(  
2   url,  
3   success  
4 );
```

which is actually a shorthand for

```
1 $.ajax({  
2   url: url,  
3   success: success  
4 });
```

So what we do now? We want to get the stories from the server using `$.get('/api/stories')` passing the appropriate URL and put the response data we get, inside the `stories` array.

There is a common catch here, we have to make the call **after the documented has finished rendering**. Fortunately, there is a helper function called `ready` in `Vue.js` (*similar to* `$(document).ready()`) which triggers once the page Document Object Model (DOM) is ready.

Lets see this in action.

¹<https://github.com/vuejs/vue-resource>

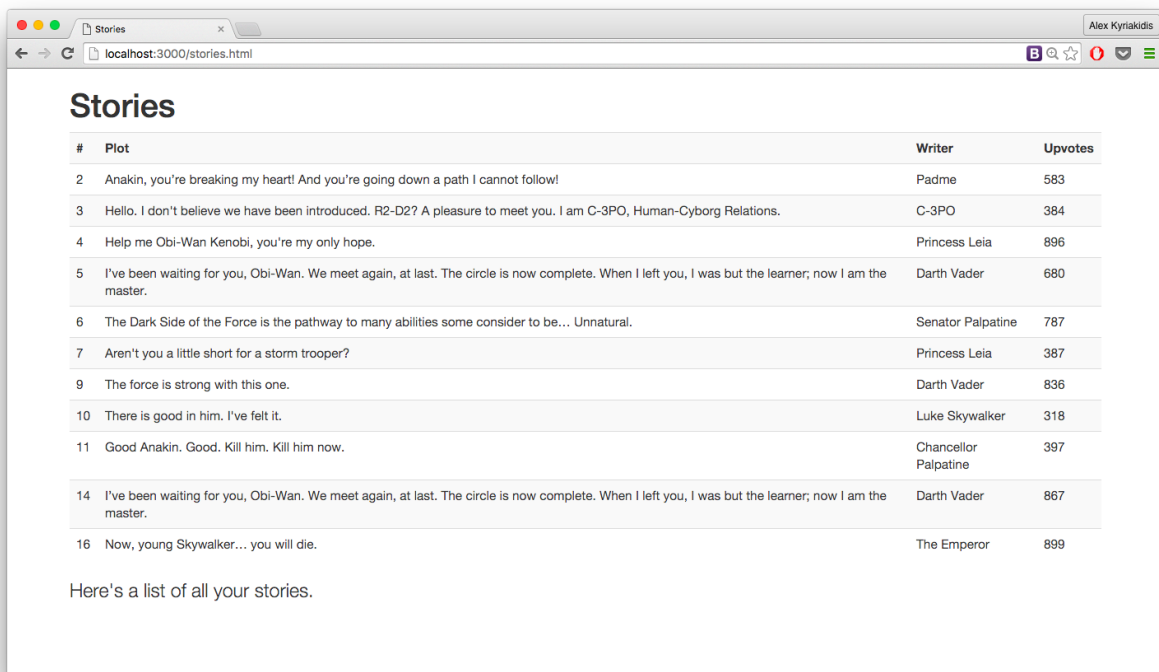
²<https://api.jquery.com/jquery.get/>

```
1 <div id="app">
2   <div class="container">
3     <h1>Let's hear some stories!</h1>
4     <ul class="list-group">
5       <story v-for="story in stories" :story="story">
6         </story>
7     </ul>
8     <pre>{{ $data | json }}</pre>
9   </div>
10 </div>
11 <template id="template-story-raw">
12   <li class="list-group-item">
13     {{ story.writer }} said "{{ story.plot }}"
14     <span>{{story.upvotes}}</span>
15   </li>
16 </template>

1 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.16/vue.js"></script>
2 <script src="https://code.jquery.com/jquery-2.1.4.min.js"></script>
3 <script type="text/javascript">
4 Vue.component('story', {
5   template: "#template-story-raw",
6   props: ['story'],
7 });
8
9 var vm = new Vue({
10   el: '#app',
11   data: {
12     stories: []
13   },
14   ready : function(){
15     $.get('/api/stories', function(data){
16       vm.stories = data;
17     })
18   }
19 })
20 </script>
```

We start by pulling in the jQuery from the [cdnjs](https://cdnjs.com/libraries/jquery/)³. Then use the ready function and inside it, perform the GET request. After the request is successfully finished we set the response data (inside the callback) to stories array.

³<https://cdnjs.com/libraries/jquery/>



Get stories



Notice here, that inside the callback we are referring to `stories` variable using `vm.stories` instead of `this.stories`. We do so because variable `this` does not represent the `Vue` instance inside the callback. So, we set the whole `Vue` instance to a variable called `vm`, in order to have access to it from anywhere within our code. To learn more about `this`, have a look at the [MDN documentation](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this)⁴.

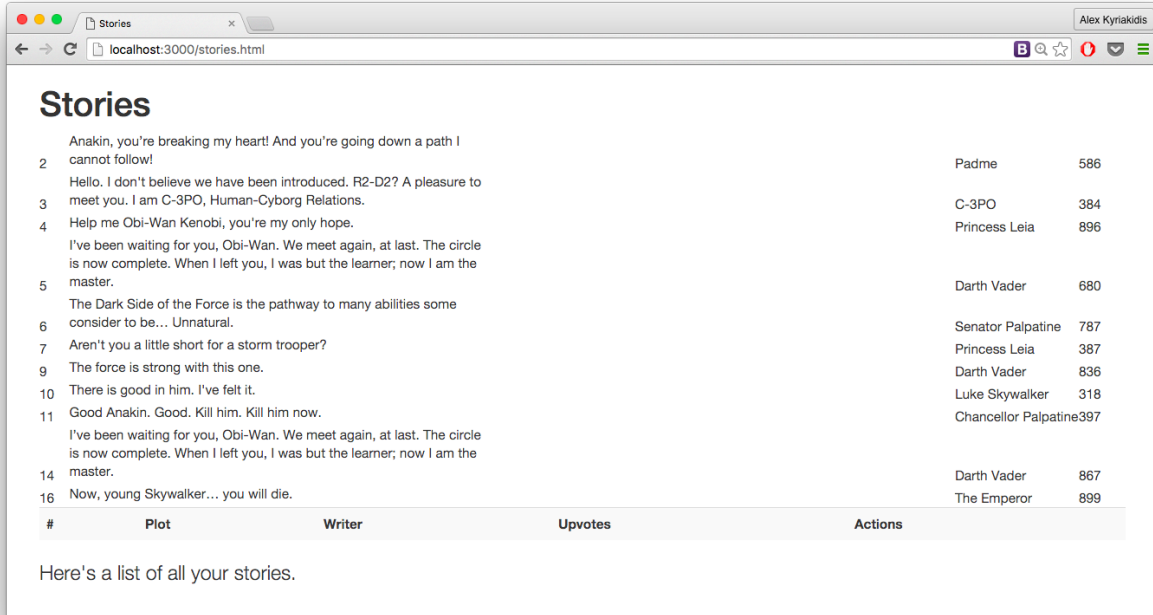
9.2 Refactoring

Having large amounts of code can be confusing, in our text editor, as well as in the browser, if not displayed properly. For that reason, we are going to refactor our example code, to render the list of stories using a `<table>` element instead of the ``.

⁴<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this>

```
1 <div id="app">
2   <table class="table table-striped">
3     <tr>
4       <th>#</th>
5       <th>Plot</th>
6       <th>Writer</th>
7       <th>Upvotes</th>
8       <th>Actions</th>
9     </tr>
10    <story v-for="story in stories" :story="story">
11    </story>
12  </table>
13  <template id="template-story-raw">
14    <tr>
15      <td>
16        {{story.id}}
17      </td>
18      <td>
19        <span>
20          {{story.plot}}
21        </span>
22      </td>
23      <td>
24        <span>
25          {{story.writer}}
26        </span>
27      </td>
28      <td>
29        {{story.upvotes}}
30      </td>
31    </tr>
32  </template>
33  <p class="lead">Here's a list of all your stories.
34  </p>
35  <pre>{{ $data | json }}</pre>
36 </div>
```

But there is an issue.



Rendering issues

Our table does not render properly, [but why?](#)⁵. The reason behind this is that:

Some HTML elements, for example `<table>`, have restrictions on what elements can appear inside them. Custom elements that are not in the whitelist will be hoisted out and thus not render properly. In such cases you should use the `is` special attribute to indicate a custom element.

Therefore, to solve this issue we have to use Vue's special attribute `is`.

```
1 <table>
2   <tr is="my-component"></tr>
3 </table>
```

So our example will become

```
1 <tr v-for="story in stories" is="story" :story="story"></tr>
```

⁵<http://goo.gl/Xr9RoQ>

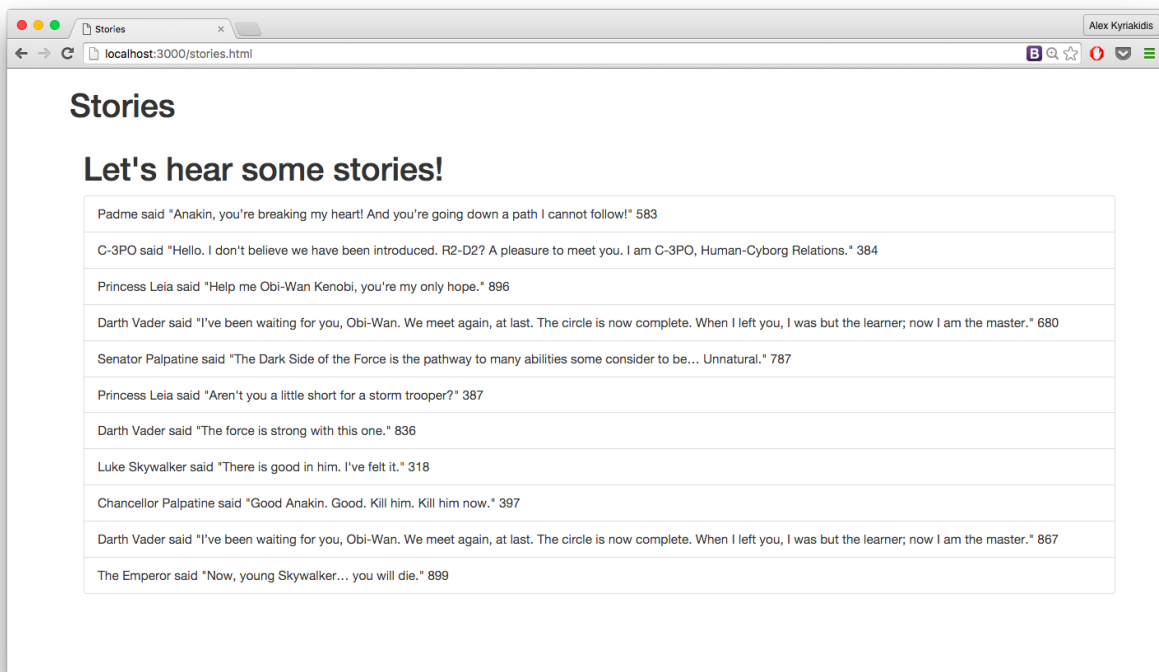


Table renders properly

Well this looks better!

9.3 Update Data

We used to have a function that allowed the user to vote any story he wanted to. This time we want each time a story is voted to inform the server, ensuring that story votes are updated in the database as well.

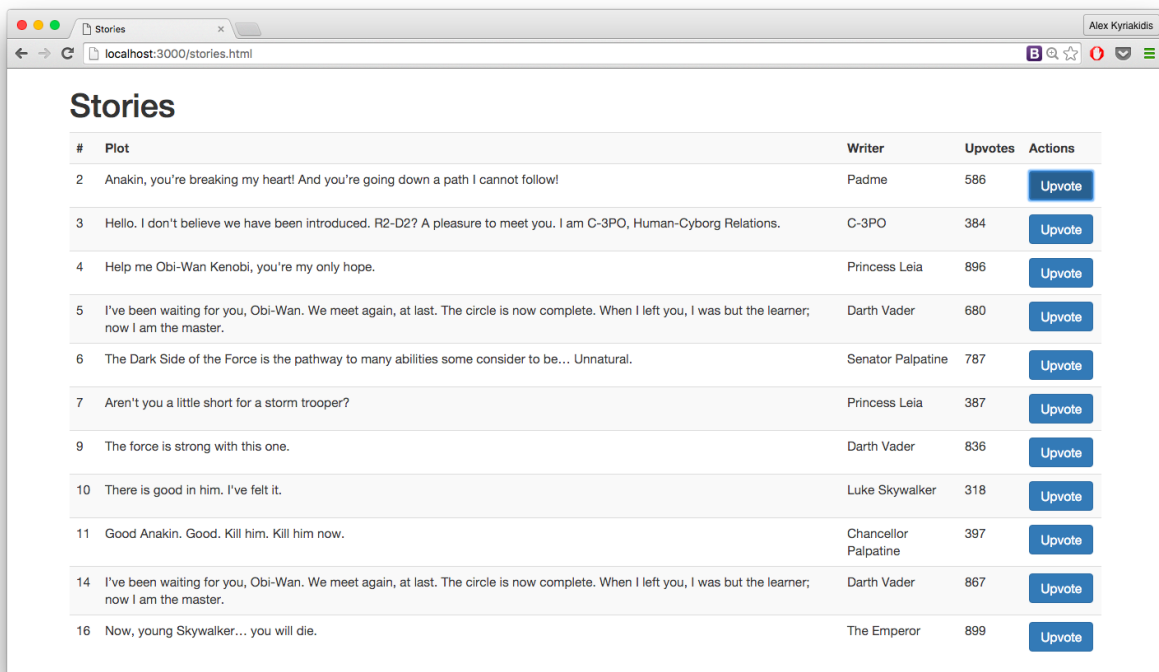
To update an existing story we have to make an HTTP PATCH or PUT request to `api/stories/{storyID}`.

Inside the `upvoteStory` function which is to be created, we are going to make a HTTP call after we have increased `story upvotes`. We will pass the newly updated `story` variable in the Request Payload in order to update the data in our server.

```
1 <td>
2   <div class="btn-group">
3     <button @click="upvoteStory(story)" class="btn btn-primary">
4       Upvote
5     </button>
6   </div>
7 </td>
```

```
1 Vue.component('story',{
2   template: '#template-story-raw',
3   props: ['story'],
4   methods: {
5     upvoteStory: function(story){
6       story.upvotes++;
7       $.ajax({
8         url: '/api/stories/'+story.id,
9         type: 'PATCH',
10        data: story,
11      });
12    }
13  },
14 })
```

We brought back the upvote method and placed it inside our story component. Making a PATCH request now, providing the new data, the server updates the upvotes count.



#	Plot	Writer	Upvotes	Actions
2	Anakin, you're breaking my heart! And you're going down a path I cannot follow!	Padme	586	Upvote
3	Hello. I don't believe we have been introduced. R2-D2? A pleasure to meet you. I am C-3PO, Human-Cyborg Relations.	C-3PO	384	Upvote
4	Help me Obi-Wan Kenobi, you're my only hope.	Princess Leia	896	Upvote
5	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	680	Upvote
6	The Dark Side of the Force is the pathway to many abilities some consider to be... Unnatural.	Senator Palpatine	787	Upvote
7	Aren't you a little short for a storm trooper?	Princess Leia	387	Upvote
9	The force is strong with this one.	Darth Vader	836	Upvote
10	There is good in him. I've felt it.	Luke Skywalker	318	Upvote
11	Good Anakin. Good. Kill him. Kill him now.	Chancellor Palpatine	397	Upvote
14	I've been waiting for you, Obi-Wan. We meet again, at last. The circle is now complete. When I left you, I was but the learner; now I am the master.	Darth Vader	867	Upvote
16	Now, young Skywalker... you will die.	The Emperor	899	Upvote

Upvote stories

9.4 Delete Data

Now let us proceed to another piece of functionality our `stories` list should have, deleting a story we don't like. To remove a story from the array and the DOM, we are going to use Vue's `$remove()` method, which searches for an item and removes it from target Array.



Info

`$remove()` method works as follows. When you want to remove an `item` from an array called `items` you can do:

```
vm.items.$remove(item)
```

```
1 <td>
2   <div class="btn-group">
3     <button @click="upvoteStory(story)" class="btn btn-primary">
4       Upvote
5     </button>
6     <button @click="deleteStory(story)" class="btn btn-danger">
7       Delete
8     </button>
9   </div>
10 </td>
```

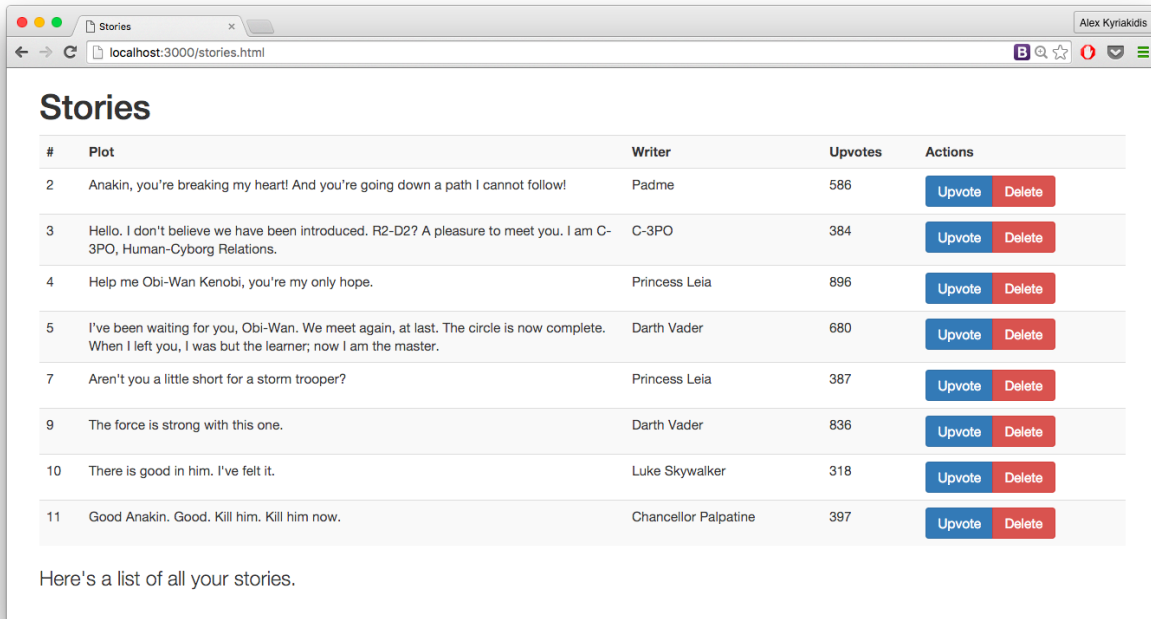
We append a ‘Delete’ button to the ‘actions’ column, binded to a method to delete the story. The `deleteStory` method will be:

```
1 Vue.component('story', {
2   ...
3   methods: {
4     ...
5     deleteStory: function(story){
6       vm.stories.$remove(story)
7     }
8   }
9   ...
10 })
```

But of course, this way, we will only remove the story temporary. In order to delete the story from the database, we have to perform an AJAX DELETE request.

```
1 Vue.component('story', {
2   ...
3   methods: {
4     ...
5     deleteStory: function(story){
6       vm.stories.$remove(story)
7       $.ajax({
8         url: '/api/stories/'+story.id,
9         type: 'DELETE'
10      });
11     },
12   }
13   ...
14 })
```

We are passing in the URL, as we did before. The type here should be equal to `DELETE`. Our method is now ready and we can delete the story from our database as well as the DOM.



Upvote and Delete stories

That's it for now. We will continue our example in the next chapter, by enhancing the functionality with **Creating new stories**, **Editing current stories** and more. But first of all, we will replace **jQuery** with **vue-resource**.

10. Integrating vue-resource

10.1 Overview

Vue-resource is a resource plugin for Vue.js. This plugin provides services for making web requests and handles responses using an XMLHttpRequest or JSONP.

We are going to make again all the web requests we made above, using this plugin instead. This way you can see the differences and decide for yourself which suits you best. jQuery is nice, but if you are using it only to perform AJAX calls, you may consider removing it.

Here you can find installation instructions and documentation about [vue-resource](#)¹ As usual, we are going to “pull it in” from the [cdnjs](#)² page.

To fetch data from a server and bring them up to display in the browser, we can use vue-resource `$http` method with the following syntax:

```
1 ready: function() {
2     // GET request
3     this.$http({url: '/someUrl', method: 'GET'})
4     .then(function (response) {
5         // success callback
6     }, function (response) {
7         // error callback
8     });
9 }
```



Info

A Vue instance provides the `this.$http(options)` function which takes an options object for generating an HTTP request and returns a promise. Also the Vue instance will be automatically bound to `this` in all function callbacks.

Instead of passing the method option, there are shortcut methods available for all request types.

¹<https://github.com/vuejs/vue-resource>

²<https://cdnjs.com/libraries/vue-resource>

Request shortcuts

```

1 this.$http.get(url, [data], [options]).then(successCallback, errorCallback);
2 this.$http.post(url, [data], [options]).then(successCallback, errorCallback);
3 this.$http.put(url, [data], [options]).then(successCallback, errorCallback);
4 this.$http.patch(url, [data], [options]).then(successCallback, errorCallback);
5 this.$http.delete(url, [data], [options]).then(successCallback, errorCallback);

```

10.2 Migrating

It is time to use vue-resource in our example. First of all, we have to include it. We will add this line to our HTML file.

```

1 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue-resource/0.7.0/vue-resou\
2 rce.js"></script>

```

To fetch the stories we will make a GET request in the corresponding form.

```

1 ready: function() {
2   // GET request
3   this.$http({url: '/api/stories', method: 'GET'}).then(function (response) {
4     this.$set('stories', response.data)
5     //Or we as we did before
6     //vm.stories = response.data
7   })
8 }

```

Our list of stories comes without any problems using the syntax above.

Lets move on now with the DELETE and PATCH requests using the shortcut methods.

PATCH request

```

1 upvoteStory: function(story){
2   story.upvotes++;
3   this.$http.patch('/api/stories/'+story.id , story)
4 },

```

We have replaced the AJAX method with this one, in no time!

DELETE request

```

1 deleteStory: function(story){
2     this.$parent.stories.$remove(story)
3     this.$http.delete('/api/stories/'+story.id )
4 },

```

We observe that the delete function works as expected and the story is deleted from the database.

10.3 Enhancing Functionality

We should add a couple more features to make our list of stories neat. We can give the user the ability to change the plot and the writer of a story and create new stories.

10.3.1 Edit Stories

Let's start with the first task and give the user some inputs to manipulate the story's attributes. Two binded inputs should do the job, but we should display them **only** when the user is **editing** a story. It seems like the kind of work we did in previous chapters.

To define if a story is in **edit state** we will use a property, **editing** which will become true when the user hits the 'Edit' button.

```

1 <td>
2     <!--if editing story display the input for plot-->
3     <input v-if="story.editing" v-model="story.plot" class="form-control">
4     </input>
5     <!--in other occasions show the story plot-->
6     <span v-else>
7         {{story.plot}}
8     </span>
9 </td>
10 <td>
11     <!-- if editing story display the input for writer -->
12     <input v-if="story.editing" v-model="story.writer" class="form-control">
13     </input>
14     <!--in other occasions show the story writer-->
15     <span v-else>
16         {{story.writer}}
17     </span>
18 </td>

```



```

19 <td>
20     {{story.upvotes}}
21 </td>
22 <td>
23     <div v-if="!story.editing" class="btn-group">
24         <button @click="upvoteStory(story)" class="btn btn-primary">
25             Upvote
26         </button>
27         <button @click="editStory(story)" class="btn btn-default">
28             Edit
29         </button>
30         <button @click="deleteStory(story)" class="btn btn-danger">
31             Delete
32         </button>
33     </div>
34 </td>

1  Vue.component('story', {
2      ...
3      methods: {
4          ...
5          editStory: function(story){
6              story.editing=true;
7          },
8      }
9      ...
10 })

```

This is our updated table with two new inputs and a button. We use the `editStory` function to set `story.editing` to `true`, so `v-if` will bring up the inputs to edit the story and hide the ‘Upvote’ and ‘Delete’ buttons. But this approach won’t work. It seems that the DOM isn’t updating after setting `story.editing` to `true`. Why this may be happening?

It turns out, according to [this post from Vue.js blog](http://vuejs.org/2016/02/06/common-gotchas/)³ that when you are adding a new property that wasn’t present when the data was observed the DOM won’t update. The best practice is to always declare properties that need to be reactive **upfront**. In cases where you absolutely need to add or delete properties at runtime, use the global `Vue.set` or `Vue.delete` methods.

For this reason, we have to initialize the `story.editing` attribute to `false` on each story, **right after** receiving the stories from the server.

To do this, we are going to use javascript’s `.map()` method within the success callback of the GET request.

³<http://vuejs.org/2016/02/06/common-gotchas/>

```

1 ready: function() {
2   // GET request
3   this.$http({url: '/api/stories', method: 'GET'}).then(function (response) {
4     var storiesReady = response.data.map(function (story) {
5       story.editing = false;
6       return story
7     })
8
9     this.$set('stories', storiesReady)
10  })
11 }

```



Info

The `.map()` method calls a defined callback function on each element of an array and returns an array that contains the results. You can find more information about the `.map()` method and its syntax [Here](#)⁴

The function passes the new attribute inside each story object and then returns the updated story. The new variable `storiesReady` is an array that contains our updated array with the new attribute on.

When the story is under edit, we will give the user two options, to update the story with new values, and to cancel the edit.

#	Plot	Writer	Upvotes	Actions
12	Laugh it up, Fuzz ball.	Han Solo	279	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
14	<input type="text" value="Fear is the path to the dark side."/>	<input type="text" value="Yoda"/>	561	
15	I find your lack of faith disturbing.	Darth Vader	582	<button>Upvote</button> <button>Edit</button> <button>Delete</button>
16	Laugh it up, Fuzz ball.	Han Solo	772	<button>Upvote</button> <button>Edit</button> <button>Delete</button>

Form inputs for story editing

So, lets move on and add two new buttons that should be displayed only when the user is editing a story. Additionally a new method called `updateStory` will be created, which updates the current editing story, after the 'Update Story' button is pressed.

⁴[https://msdn.microsoft.com/en-us/library/ff679976\(v=vs.94\).aspx](https://msdn.microsoft.com/en-us/library/ff679976(v=vs.94).aspx)

```

1 <!-- If story is under edit, display this group of buttons -->
2 <div class="btn-group" v-else>
3   <button @click="updateStory(story)" class="btn btn-primary">
4     Update Story
5   </button>
6   <button @click="story.editing=false" class="btn btn-default">
7     Cancel
8   </button>
9 </div>

1 Vue.component('story',{
2   ...
3   methods: {
4     ...
5     updateStory: function(story){
6       this.$http.patch('/api/stories/'+story.id , story)
7       //Set editing to false to show actions again and hide the inputs
8       story.editing = false;
9     },
10  }
11  ...
12 })

```

#	Plot	Writer	Upvotes	Actions
12	Laugh it up, Fuzz ball.	Han Solo	279	Upvote Edit Delete
14	<input type="text" value="Fear is the path to the dark side."/>	<input type="text" value="Yoda"/>	561	Update Story Cancel
15	I find your lack of faith disturbing.	Darth Vader	582	Upvote Edit Delete
16	Laugh it up, Fuzz ball.	Han Solo	772	Upvote Edit Delete

Updating story actions

Here it is up and running. After the PATCH request is finished successfully, we have to set `story.editing` to `false` in order to hide the inputs and bring back the action buttons.

10.3.2 Create New Stories

Now for a bit trickier task, we are going to give the user the ability to create a new story and save it to our server. First, we must provide inputs so the new story can be typed in. To make this happen, we will create an empty story and we'll add it to the stories array using the `push()` javascript method. We will initialize all the story's attributes to null, except from `editing`. We want to immediately manipulate the story, so the `editing` will be set to `true`.

```
1 var vm = new Vue({
2   ...
3   methods: {
4     createStory: function(){
5       var newStory={
6         "plot": "",
7         "upvotes": 0,
8         "editing": true
9       };
10      this.stories.push(newStory);
11    },
12  }
13 })
```

```
1 <p class="lead">Here's a list of all your stories.
2   <button @click="createStory()" class="btn btn-primary">
3     Add a new one?
4   </button>
5 </p>
```



Info

The `push()` method adds new items to the end of an array, and returns the new length. You can find more information about the `push()` method and its syntax [Here⁵](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/push)

As soon as the new variable is set, we push it to our stories array. We named the new function `createStory` and we placed it in our Vue instance.

Right bellow our list, we have added a button. When the button is clicked, `createStory` method gets invoked. Since the `newStory.editing` is set to `true`, the binded inputs for “plot” and “writer” along with the ‘Edit action buttons’ are being rendered instantly.

Also, the new story object must be sent to the server in order to be stored in the database. We are going to perform a POST request inside a method called `storeStory`.

⁵https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/push

```

1  Vue.component('story',{
2    ...
3    methods: {
4      ...
5      storeStory: function(story){
6        this.$http.post('/api/stories/', story).then(function() {
7          story.editing = false;
8        });
9      },
10   }
11   ...
12 })

```

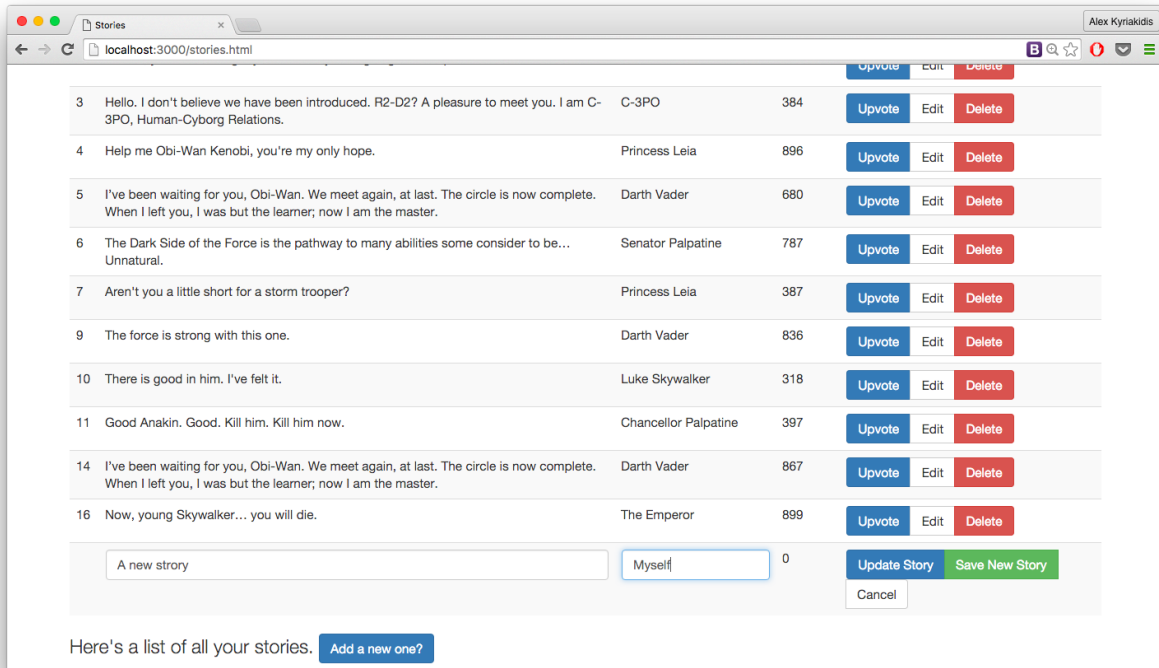
We've used the shortcut method for the request type POST and in the success callback function we set editing to false, to show the 'actions buttons' again and hide the form's inputs and 'editing' buttons. Below we update the button groups in accordance to the new method.

```

1  <td>
2  <div class="btn-group" v-if="!story.editing">
3    <button @click="upvoteStory(story)" class="btn btn-primary">
4      Upvote
5    </button>
6    <button @click="editStory(story)" class="btn btn-default">
7      Edit
8    </button>
9    <button @click="deleteStory(story)" class="btn btn-danger">
10     Delete
11   </button>
12 </div>
13 <div class="btn-group" v-else>
14   <button class="btn btn-primary" @click="updateStory(story)">
15     Update Story
16   </button>
17   <button class="btn btn-success" @click="storeStory(story)">
18     Save New Story
19   </button>
20   <button @click="story.editing=false" class="btn btn-default">
21     Cancel
22   </button>
23 </div>
24 </td>

```

We observe a small mistake in this block of code. When we are in “editing” mode (`v-else` block) the buttons for update and store are being shown together, but we only need one for each story since each story will be **Stored** **or** **Updated**, it can’t do both. So, if the story is an old one and the user is about to edit it, we need the update button. Else, if the story is new, we need the store button.



A small mistake

To bypass this issue, we are going to restructure our buttons. The Update button will **only** be displayed when the **story is old**. Accordingly the Save new button will be displayed when the story is a **new one**.

You may have noticed that all stories fetched from the server have an `id` attribute. We are going to use this observation to **define if a story is new or not**.

```

1 <div class="btn-group" v-else>
2   <!--If the story is an old one then we want to update it
3   TIP: if the story is taken from the db then it will have an id-->
4   <button v-if="story.id" class="btn btn-primary" @click="updateStory(story)">
5     Update Story
6   </button>
7   <!--If the story is new we want to store it-->
8   <button v-else class="btn btn-success" @click="storeStory(story)">
9     Save New Story
10  </button>

```

```

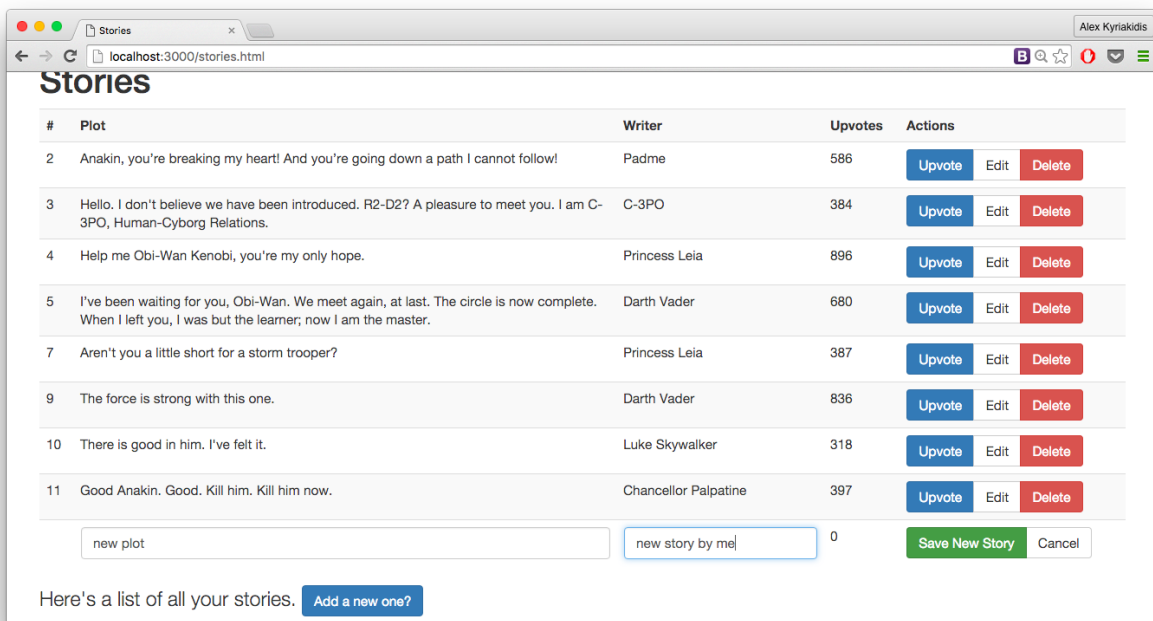
11   <!--always show cancel-->
12   <button @click="story.editing=false" class="btn btn-default">
13     Cancel
14   </button>
15 </div>

```



Tip

If the story is taken from the database then it will have an id.



Adding new story

So there we have it. It wasn't that hard, right?

After finishing this part, testing our app shows another error. After creating, saving, and trying to edit a new story, we see that the button says "Save new Story" instead of "Update Story"! That's because we are not fetching the newly created story from the server, after we send it, and it does not have an id yet. To solve this problem we can fetch the stories from server again, after finishing creating a story.

Since I don't like to repeat my code, I will extract the fetching procedure to a method called `fetchStories()`. After that, I can use this method to fetch the stories anytime.

The fetchStories method

```
1 var vm = new Vue({
2   el: '#v-app',
3   data : {
4     stories: [],
5   },
6
7   ready : function(){
8     this.fetchStories()
9   },
10  methods: {
11    createStory: function(){
12      var newStory={
13        "plot": "",
14        "upvotes": 0,
15        "editing": true
16
17      };
18      this.stories.push(newStory);
19    },
20    fetchStories: function () {
21      this.$http.get('/api/stories')
22        .then(function (response) {
23          var storiesReady = response.data.map(function(story){
24            story.editing = false;
25            return story
26          })
27          this.$set('stories', storiesReady)
28        });
29    },
30  }
31 });
```

In our situation, we'll call `fetchStories()` inside the success callback of the POST request.


```

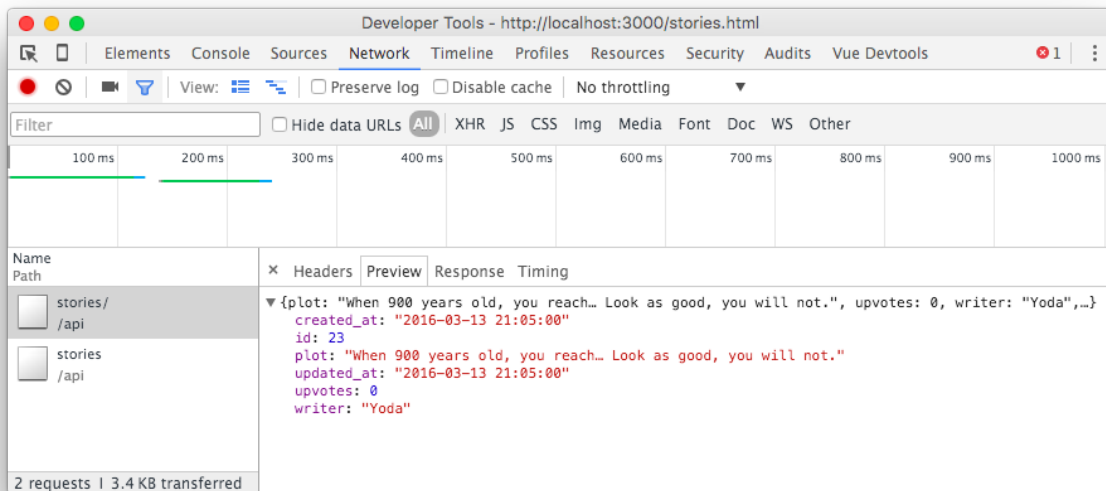
1  Vue.component('story',{
2    ...
3    methods: {
4      ...
5      storeStory: function(story){
6        this.$http.post('/api/stories/', story).then(function() {
7          story.editing = false;
8          vm.fetchStories();
9        });
10   },
11  },
12  ...
13 })

```

That's it! We can now create and edit any story we want.

10.3.3 Store & Update Unit

A better way to fix the previous issue, is to update only the newly created story from the database, instead of fetching and overwriting all the stories. If you see the server response for the POST request you will see that it returns the created story along with its id.



Server response after creating new story

The only thing we have to do, is to update our story to match the server's one. So, we will set the `id` of the response data, to `story's id` attribute. We will do this inside the POST's success callback.

```
1 Vue.component('story',{
2   ...
3   methods: {
4     ...
5     storeStory: function(story){
6       this.$http.post('/api/stories/', story).then(function(response) {
7         Vue.set(story, 'id', response.data.id);
8         story.editing = false;
9       });
10    },
11  }
12  ...
13 })
```

I use `Vue.set(story, 'id', response.data.id)` instead of `story.id = response.data.id` because inside our table we display the `id` of each story. Since the new story had no `id` when pushed to the `stories` array the DOM won't be updated when the `id` changes, so we will not be able to see the new `id`.



Tip

When you are adding a **new property that wasn't present when the data was observed**, Vue.js cannot detect the property's addition. So, if you need to add or remove properties at runtime, use the global `Vue.set` or `Vue.delete` methods.

10.4 JavaScript File

As you may have noticed, our code is becoming big, and as our project grows, it will be hard to maintain. For starters, we'll separate the JavaScript code from the HTML. I'll create a file called `app.js` and I'll save it under `js` directory.

All the JavaScript code should live inside that file from now on. To include the newly created script to any HTML page you simply have to add this tag

```
1 <script src='/js/app.js' type="text/javascript"></script>
```

and you are **ready to go!**

10.5 Source Code

Below is the whole source code of the previous *Managing Stories* example. Because the code is big enough I suggest you to open your local files with your favorite text editor, if you have downloaded our repo. The file is located at `~/themajestyofvuejs/apis/stories/`.

If you haven't downloaded the repository you can still view the [stories.html](https://github.com/hootlex/the-majesty-of-vuejs/blob/master/apis/stories/public/stories.html)⁶ and [app.js](https://github.com/hootlex/the-majesty-of-vuejs/blob/master/apis/stories/public/js/app.js)⁷ files on *github*.

stories.html

```
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1">
6   <title>Stories</title>
7   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.2\
8 /css/bootstrap.min.css">
9 </head>
10 <body>
11 <main>
12   <div class="container">
13     <h1>Stories</h1>
14     <div id="v-app">
15       <table class="table table-striped">
16         <tr>
17           <th>#</th>
18           <th>Plot</th>
19           <th>Writer</th>
20           <th>Upvotes</th>
21           <th>Actions</th>
22         </tr>
23         <tr v-for="story in stories" is="story" :story="story"></tr>
24       </table>
25       <template id="template-story-raw">
26         <tr>
27           <td>
28             {{story.id}}
29           </td>
30           <td class="col-md-6">
```

⁶<https://github.com/hootlex/the-majesty-of-vuejs/blob/master/apis/stories/public/stories.html>

⁷<https://github.com/hootlex/the-majesty-of-vuejs/blob/master/apis/stories/public/js/app.js>

```

31         <input v-if="story.editing" v-model="story.plot"
32             class="form-control">
33     </input>
34     <!--in other occasions show the story plot-->
35     <span v-else>
36         {{story.plot}}
37     </span>
38 </td>
39 <td>
40     <input v-if="story.editing" v-model="story.writer"
41         class="form-control">
42     </input>
43     <!--in other occasions show the story writer-->
44     <span v-else>
45         {{story.writer}}
46     </span>
47 </td>
48 <td>
49     {{story.upvotes}}
50 </td>
51 <td>
52     <div class="btn-group" v-if="!story.editing">
53         <button @click="upvoteStory(story)"
54             class="btn btn-primary"
55         >
56             Upvote
57         </button>
58         <button @click="editStory(story)"
59             class="btn btn-default"
60         >
61             Edit
62         </button>
63         <button @click="deleteStory(story)"
64             class="btn btn-danger"
65         >
66             Delete
67         </button>
68     </div>
69     <div class="btn-group" v-else>
70         <!--If the story is taken from the db then it will h\
71     ave an id-->
72         <button v-if="story.id"

```

```
73         @click="updateStory(story)"
74         class="btn btn-primary"
75     >
76         Update Story
77     </button>
78     <!--If the story is new we want to store it-->
79     <button v-else
80         @click="storeStory(story)"
81         class="btn btn-success"
82     >
83         Save New Story
84     </button>
85     <!--Always show cancel-->
86     <button @click="story.editing=false"
87         class="btn btn-default"
88     >
89         Cancel
90     </button>
91 </div>
92 </td>
93 </tr>
94 </template>
95 <p class="lead">Here's a list of all your stories.
96     <button @click="createStory()"
97         class="btn btn-primary"
98     >
99         Add a new one?
100 </button>
101 </p>
102 <pre>{{ $data | json }}</pre>
103 </div>
104 </div>
105 </main>
106 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/1.0.18/vue.min.js"></scr\
107 ipt>
108 <script src="https://code.jquery.com/jquery-2.1.4.min.js"></script>
109 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue-resource/0.7.0/vue-resou\
110 rce.js"></script>
111 <script src='/js/app.js' type="text/javascript"></script>
112 </body>
113 </html>
```

```
1  Vue.component('story', {
2    template: '#template-story-raw',
3    props: ['story'],
4    methods: {
5      deleteStory: function (story) {
6        this.$parent.stories.$remove(story)
7        this.$http.delete('/api/stories/' + story.id)
8      },
9      upvoteStory: function (story) {
10       story.upvotes++;
11       this.$http.patch('/api/stories/' + story.id, story)
12     },
13     editStory: function (story) {
14       story.editing = true;
15     },
16     updateStory: function (story) {
17       this.$http.patch('/api/stories/' + story.id, story)
18       // Set editing to false to show actions again and hide the inputs
19       story.editing = false;
20     },
21     storeStory: function (story) {
22       this.$http.post('/api/stories/', story)
23       .then(function (response) {
24         // After the the new story is stored in the database
25         // we fetch again all stories
26         // vm.fetchStories();
27         // OR Better, update the id of the created story
28         Vue.set(story, 'id', response.data.id);
29         // Set editing to false to show actions again and hide the inputs
30         story.editing = false;
31       });
32     },
33   }
34 })
35 new Vue({
36   el: '#v-app',
37   data: {
38     stories: [],
39   },
40   ready: function () {
41     this.fetchStories()
42   },
```

```
43     methods: {
44         createStory: function () {
45             var newStory = {
46                 plot: "",
47                 upvotes: 0,
48                 editing: true
49             };
50             this.stories.push(newStory);
51         },
52         fetchStories: function () {
53             var vm = this;
54             this.$http.get('/api/stories')
55                 .then(function(response){
56                     // set data on vm
57                     var storiesReady = response.data.map(function (story) {
58                         story.editing = false;
59                         return story
60                     })
61                     vm.$set('stories', storiesReady)
62                 });
63         },
64     }
65 });
```

10.6 Homework

To get comfortable with making web requests and handling responses you should replicate what we did in this chapter.

What you have to do is to consume an API in order to:

- create a table and **display existing** movies
- **modify** existing movies
- **store** new movies in the database
- **delete** movies from the database

I have prepared **the database and the API** for you. You only have to write HTML and JavaScript.

10.6.1 Preface

If you have followed the [instructions from Chapter 8](#) open your terminal and run:

```
cd ~/themajestyofvuejs/apis/movies
sh setup.sh
```

If you haven't, you should run this:

```
mkdir ~/themajestyofvuejs
cd ~/themajestyofvuejs
git clone https://github.com/hootlex/the-majesty-of-vuejs .
cd ~/themajestyofvuejs/apis/movies
sh setup.sh
```

You now have a **database filled with great movies** along with a fully functional server **running on 'http://localhost:3000'**!

To ensure that everything is working fine browse to 'http://localhost:3000/api/movies' and you should see an array of movies in JSON format.

10.6.2 API Endpoints

The **API Endpoints** you are going to need are:

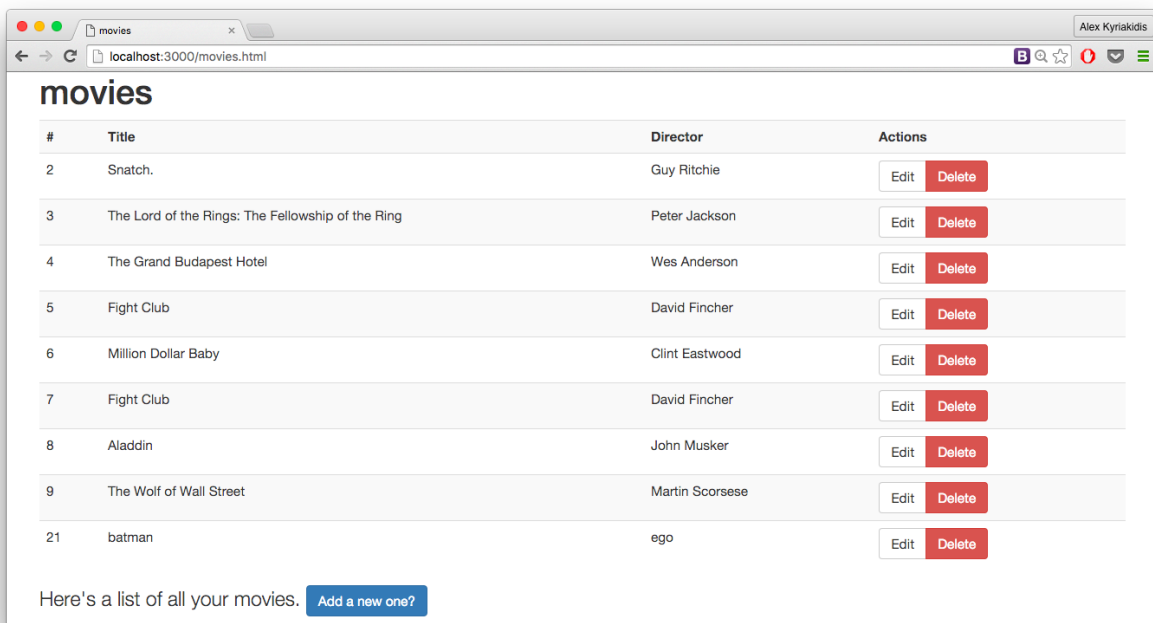
HTTP Method	URI	Action
GET/HEAD	api/movies	<i>Fetches</i> all movies
GET/HEAD	api/movies/{id}	<i>Fetches</i> specified movie
POST	api/movies	<i>Creates</i> a new movie
PUT/PATCH	api/movies/{id}	<i>Updates</i> an existing movie
DELETE	api/movies/{id}	<i>Deletes</i> specified movie

10.6.3 Your Code

Put your HTML code inside `~/themajestyofvuejs/apis/movies/public/movies.html` file we have created. You can place your JavaScript code there too, or inside `js/app.js`.

To check your work visit `'http://localhost:3000/movies.html'` with your browser.

I hope you will enjoy this one, Good luck!



Example Output

You can find a potential solution to this exercise [here](https://github.com/hootlex/the-majesty-of-vuejs/blob/master/homework/chapter10)⁸.

⁸<https://github.com/hootlex/the-majesty-of-vuejs/blob/master/homework/chapter10>