



The Majesty of Vue.js 2

Alex
Kyriakidis

Kostas
Maniatis

The Majesty of Vue.js 2

Alex Kyriakidis, Kostas Maniatis and Evan You

This book is for sale at <http://leanpub.com/vuejs2>

This version was published on 2017-07-20



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2016 - 2017 Alex Kyriakidis, Kostas Maniatis and Evan You

Tweet This Book!

Please help Alex Kyriakidis, Kostas Maniatis and Evan You by spreading the word about this book on [Twitter](#)!

The suggested tweet for this book is:

I'm learning [@vuejs](#) with [@tmvuejs](#). Get it at <https://leanpub.com/vuejs2> [#vuejs](#)

The suggested hashtag for this book is [#vuejs2](#).

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

<https://twitter.com/search?q=#vuejs2>

Contents

Introduction	i
About Vue.js	ii
Vue.js Overview	ii
What people say about Vue.js	ii
Comparison with Other Frameworks	iv
Angular 1	iv
Angular 2	v
React	vii
Ember	ix
Polymer	x
Riot	xi
Welcome	xii
About the Book	xii
Who is this Book for	xii
Get In Touch	xii
Homework	xiii
Sample Code	xiii
Errata	xiii
Conventions	xiii
I 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	3
1.3 Download using Bower	3
2. Getting Started	4
2.1 Hello World	4

CONTENTS

2.2	Two-way Binding	6
2.3	Comparison with jQuery.	7
2.4	Homework	9
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	22
4.2.1	Range v-for	22
4.3	Array Rendering	24
4.3.1	Loop Through an Array	24
4.3.2	Loop Through an Array of Objects	26
4.4	Object v-for	29
4.5	Homework	31
5.	Interactivity	32
5.1	Event Handling	32
5.1.1	Handling Events Inline	32
5.1.2	Handling Events using Methods	34
5.1.3	Shorthand for v-on	35
5.2	Event Modifiers	36
5.3	Key Modifiers	40
5.4	Computed Properties	41
5.5	Homework	47
6.	Filters	49
6.1	Filtered Results	49
6.1.1	Using Computed Properties	52
6.2	Ordered Results	59
6.3	Custom Filters	63
6.4	Utility Libraries	64
6.5	Homework	68
7.	Components	69
7.1	What are Components?	69
7.2	Using Components	69

CONTENTS

7.3	Templates	71
7.4	Properties	72
7.5	Reusability	75
7.6	Altogether	79
7.7	Homework	86
8.	Custom Events	87
8.1	Emit and Listen	87
8.1.1	Lifecycle Hooks	89
8.2	Parent-Child Communication	90
8.3	Passing Arguments	92
8.4	Non Parent-Child Communication	97
8.5	Removing Event Listeners	100
8.6	Back to stories	101
8.7	Homework	104
9.	Class and Style Bindings	106
9.1	Class binding	106
9.1.1	Object Syntax	106
9.1.2	Array Syntax	109
9.2	Style binding	111
9.2.1	Object Syntax	111
9.2.2	Array Syntax	112
9.3	Bindings in Action	113
9.4	Homework	116
II	Consuming an API	117
10.	Preface	118
10.1	CRUD	118
10.2	API	118
10.2.1	Download Book's Code	119
10.2.2	API Endpoints	121
11.	Working with real data	123
11.1	Get Data Asynchronous	123
11.2	Refactoring	127
11.3	Update Data	129
11.4	Delete Data	131
12.	HTTP Clients	134
12.1	Introduction	134

CONTENTS

12.2	Vue-resource	134
12.3	Axios	135
12.4	Integrating axios	136
12.5	Enhancing Functionality	138
12.5.1	Edit Stories	138
12.5.2	Create New Stories	141
12.5.3	Store & Update Unit	147
12.6	JavaScript File	148
12.7	Source Code	149
12.8	Homework	154
12.8.1	Preface	154
12.8.2	API Endpoints	155
12.8.3	Your Code	155
13.	Pagination	157
13.1	Implementation	158
13.2	Pagination Links	161
13.3	Homework	164
III	Building Large-Scale Applications	165
14.	ECMAScript 6	166
14.1	Introduction	166
14.1.1	Compatibility	167
14.2	Variable Declarations	167
14.2.1	Let Declarations	167
14.2.2	Constant Declarations	168
14.3	Arrow Functions	168
14.4	Modules	169
14.5	Classes	170
14.6	Default Parameter Values	171
14.7	Template literals	172
15.	Advanced Workflow	174
15.1	Compiling ES6 with Babel	174
15.1.1	Installation	176
15.1.2	Configuration	178
15.1.3	Build alias	179
15.1.4	Usage	179
15.1.5	Homework	181
15.2	Workflow Automation with Gulp	183
15.2.1	Task Runners	183

CONTENTS

15.2.2	Installation	184
15.2.3	Usage	184
15.2.4	Watch	185
15.2.5	Homework	186
15.3	Module Bundling with Webpack	187
15.3.1	Module Bundlers	187
15.3.2	Webpack	190
15.3.3	Installation	191
15.3.4	Usage	191
15.3.5	Automation	192
15.4	Summary	194
16.	Working with Single File Components	196
16.1	The <code>vue-cli</code>	196
16.1.1	Vue's Templates	196
16.1.2	Installation	197
16.1.3	Usage	197
16.2	Webpack Template	200
16.2.1	Project Structure	202
16.2.2	<code>index.html</code>	203
16.2.3	<code>Hello.vue</code>	204
16.2.4	<code>App.vue</code>	205
16.2.5	<code>main.js</code>	207
16.3	Forming <code>.vue</code> Files	208
16.3.1	Nested Components	217
17.	Eliminating Duplicate State	222
17.1	Sharing with Properties	222
17.2	Global Store	227
18.	Swapping Components	231
18.1	Dynamic Components	231
18.1.1	The <code>is</code> special attribute	231
18.1.2	Navigation	234
19.	Vue Router	239
19.1	Installation	239
19.2	Usage	240
19.3	Named Routes	242
19.4	History mode	243
19.5	Nested routes	245
19.6	Auto-CSS active class	247
19.6.1	Custom Active Class	249

CONTENTS

19.7	Route Object	250
19.8	Dynamic Segments	251
19.9	Route Alias	258
19.10	Programmatic Navigation	260
19.11	Transitions	261
19.11.1	Introduction	261
19.11.2	Usage	263
19.11.3	3rd-party CSS animations	264
19.12	Navigation Guards	265
19.13	Homework	267
20.	Closing Thoughts	270
21.	Further Learning	271
21.1	Tutorials	271
21.2	Videos	271
21.3	Books	272
21.4	Open source projects	272
21.5	Awesome Vue	273

Introduction

About Vue.js

Vue.js Overview

Vue (pronounced /vju:/, like view) is a progressive framework for building user interfaces. Unlike other monolithic frameworks, Vue is designed from the ground up to be incrementally adoptable. The core library is focused on the **view layer only**, and is very easy to pick up and integrate with other libraries or existing projects. On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with modern tooling and [supporting libraries](#).¹

If you are an experienced frontend developer and you 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](#)².

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 it would be 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<https://github.com/vuejs/awesome-vue#libraries--plugins>

²<https://vuejs.org/v2/guide/>

“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**

“Looking for an alternative to Angular and jQuery I remembered that I once stumbled over Vue.js. Back then I recongnized it just marginally but looking at it closer now, I found it ideal to fit my primary needs: Enhancing the reactivity of my ASP.NET MVC Views. And so I used Vue.js instead of Angular and jQuery. I learned to love and esteem Vue.js because it was easy to learn and also easy to use. Meanwhile I came to the conclusion that I don’t need jQuery or Angular anymore because there is Vue.js and this awesome framework can be used as both: As an enhancement for views or as a tool to develop fully fledged SPAs and even PWAs.”

—Michael Seeger

Comparison with Other Frameworks

Angular 1

Some of Vue’s syntax will look very similar to Angular (e.g. `v-if` vs `ng-if`). This is because there were a lot of things that Angular got right and these were an inspiration for Vue very early in its development. There are also many pains that come with Angular however, where Vue has attempted to offer a significant improvement.

Complexity

Vue is much simpler than Angular 1, both in terms of API and design. Learning enough to build non-trivial applications typically takes less than a day, which is not true for Angular 1.

Flexibility and Modularity

Angular 1 has strong opinions about how your applications should be structured, while Vue is a more flexible, modular solution. That’s why a [Webpack template³](https://github.com/vuejs-templates/webpack) is provided, that can set you up within minutes, while also granting you access to advanced features such as hot module reloading, linting, CSS extraction, and much more.

Data binding

Angular 1 uses two-way binding between scopes, while Vue enforces a one-way data flow between components. This makes the flow of data easier to reason about in non-trivial applications.

Directives vs Components

Vue has a clearer separation between directives and components. Directives are meant to encapsulate DOM manipulations only, while components are self-contained units that have their own view and data logic. In Angular, there’s a lot of confusion between the two.

³<https://github.com/vuejs-templates/webpack>

Performance

Vue has better performance and is much, much easier to optimize because it doesn't use dirty checking. Angular 1 becomes 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 many watchers.

Vue doesn't suffer from this at all because it uses a transparent dependency-tracking observation system with async queueing - all changes trigger independently unless they have explicit dependency relationships.

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

Angular 2

There is a separate section for Angular 2 because it really is a completely new framework. For example, it features a first-class component system, many implementation details have been completely rewritten, and the API has also changed quite drastically.

Size and Performance

In terms of performance, both frameworks are exceptionally fast and there isn't enough data from real world use cases to make a verdict. However if you are determined to see some numbers, Vue 2.0 seems to be ahead of Angular 2 according to this [3rd party benchmark](http://stefankrause.net/js-frameworks-benchmark4/webdriver-ts/table.html)⁴.

Size wise, although Angular 2 with offline compilation and tree-shaking is able to get its size down considerably, a full-featured Vue 2.0 with compiler included (23kb) is still lighter than a tree-shaken bare-bone example of Angular 2 (50kb).

Flexibility

Vue is much less opinionated than Angular 2, offering official support for a variety of build systems, with no restrictions on how you structure your application. Many developers enjoy this freedom, while some prefer having only one Right Way to build any application.

Learning Curve

To get started with Vue, all you need is familiarity with HTML and ES5 JavaScript (i.e. plain JavaScript). With these basic skills, you can start building non-trivial applications within less than a day of reading the guide.

⁴<http://stefankrause.net/js-frameworks-benchmark4/webdriver-ts/table.html>

Angular's learning curve is much steeper. The API surface of the framework is simply huge and as a user you will need to familiarize yourself with a lot more concepts before getting productive. Obviously, the complexity of Angular is largely due to its design goal of targeting only large, complex applications - but that does make the framework a lot more difficult for less-experienced developers to pick up.

React

React and Vue share many similarities. They both:

- utilize a virtual DOM
- provide reactive and composable view components
- maintain focus in the core library, with concerns such as routing and global state management handled by companion libraries

Performance Profiles

In every real-world scenario that has been tested so far, Vue outperforms React by a fair margin.

Render Performance

When rendering UI, manipulating the DOM is typically the most expensive operation and unfortunately, no library can make those raw operations faster. The best it can be done is:

1. Minimize the number of necessary DOM mutations. Both React and Vue use virtual DOM abstractions to accomplish this and both implementations work about equally well.
2. Add as little overhead as possible on top of those DOM manipulations. This is an area where Vue and React differ. In React, let's say the additional overhead of rendering an element is 1 and the overhead of an average component is 2. In Vue, the overhead of an element would be more like 0.1, but the overhead of an average component would be 4, due to the setup required for the reactivity system.

This means that in typical applications, where there are many more elements than components being rendered, Vue will outperform React by a significant margin. In extreme cases however, such as using 1 normal component to render each element, Vue will usually be slower.

Both Vue and React also offer functional components, which are stateless and instanceless - and therefore, require less overhead. When these are used in performance-critical situations, Vue is once again faster.

Update Performance

In React, you need to implement `shouldComponentUpdate` everywhere and use immutable data structures to achieve fully optimized re-renders. In Vue, a component's dependencies are automatically tracked so that it only updates when one of those dependencies change. The only further optimization that sometimes can be helpful in Vue is adding a key attribute to items in long lists.

This means updates in unoptimized Vue will be much faster than unoptimized React and actually, due to the improved render performance in Vue, even fully-optimized React will usually be slower than Vue is out-of-the-box.

In Development

Obviously, performance in production is the most important and that's what we've been discussing so far. Performance in development still matters though. The good news is that both Vue and React remain fast enough in development for most normal applications.

However, if you're prototyping any high-performance data visualizations or animations, you may find it useful to know that in scenarios where Vue can't handle more than 10 frames per second in development, we've seen React slow down to about 1 frame per second.

This is due to React's many heavy invariant checks, which help it to provide many excellent warnings and error messages.

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 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 core and Ember's templating and object model layers:

- 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 intentionally 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 as well. Vue's 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 works without any dependencies or polyfills down to IE9.

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

Polymer custom elements are authored in HTML files, which limits you to plain JavaScript/CSS (and language features supported by today's browsers). In comparison, Vue's single file components allows you to easily use ES2015+ and any CSS preprocessors you want.

When deploying to production, Polymer recommends loading everything on-the-fly with HTML Imports, which assumes browsers implementing the spec, and HTTP/2 support on both server and client. This may or may not be feasible depending on your target audience and deployment environment. In cases where this is not desirable, you will have to use a special tool called Vulcanizer to bundle your Polymer elements. On this front, Vue can combine its async component feature with Webpack's code-splitting feature to easily split out parts of the application bundle to be lazy-loaded. This ensures compatibility with older browsers while retaining great app loading performance.

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. Riot and Vue probably share a lot in design philosophies. However, despite being a bit heavier than Riot, Vue does offer some significant advantages:

- True conditional rendering. Riot renders all if branches and simply shows/hides them.
- A far more powerful router. Riot’s routing API is extremely minimal.
- More mature tooling support. Vue provides official support for Webpack, Browserify, and SystemJS, while Riot relies on community support for build system integration.
- Transition effect system. Riot has none.
- Better performance. Despite advertising use of a virtual DOM, Riot in fact uses dirty checking and thus suffers from the same performance issues as Angular 1.

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 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. Now that Vue.js 2 is out we decided it was time to update our book by publishing a second version where all examples and their relative contents are rewritten.

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 everyone.

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 2 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 useful for developers who are new to Vue.js, as well as those who already use Vue.js and want to expand their knowledge. It is also helpful for developers who are looking to migrate to Vue.js 2.

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 to our attention, 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 through 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!

Sample Code

You can find most of the code examples used in the book on GitHub. You can browse around the code [here](#)⁵.

If you prefer to download it, you will find a .zip file [here](#)⁶.

This will save you from copying and pasting things out of the book, which would probably be terrible.

Errata

Although every care have been taken 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](#)⁷.

Conventions

The following notational conventions are used throughout the book.

A block of code is set as follows:

JavaScript

⁵<https://github.com/hootlex/the-majesty-of-vuejs-2>

⁶<https://github.com/hootlex/the-majesty-of-vuejs-2/archive/master.zip>

⁷<https://github.com/hootlex/the-majesty-of-vuejs-2>

```
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.



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.



This is a Terminal Command

Commands to run in terminal.



This is a Comparison text

A small text comparing things relative to the subject.



This is a link to [Github](#).

Links lead to the repository of this book, where you can find the code samples and potential homework solutions of each chapter.

I 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

[Vue.js.org](http://vuejs.org)³ recommends [unpkg](https://unpkg.com/vue/dist/vue.js)⁴, which will reflect the latest version as soon as it is published to npm.

You can find Vue.js also on [jsdelivr](https://cdn.jsdelivr.net)⁵ or [cdnjs](https://cdnjs.cloudflare.com)⁶



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

¹<http://vuejs.org/js/vue.js>

²<http://vuejs.org/js/vue.min.js>

³<https://vuejs.org/v2/guide/installation.html#CDN>

⁴<https://unpkg.com/vue/dist/vue.js>

⁵<https://cdn.jsdelivr.net/vue/2.3.2/vue.min.js>

⁶<https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.min.js>

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](http://webpack.github.io/)⁷ or [Browserify](http://browserify.org/)⁸.

```
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)⁹

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://webpack.github.io/>

⁸<http://browserify.org/>

⁹<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/2.3.4/vue.min.js"></scri\
11 pt>
12 <script>
13     new Vue({
14         el: '#app',
15     })
16 </script>
17 </html>
```

For starters, we have included Vue.js from [cdnjs](https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.min.js)¹ 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/2.3.4/vue.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 is called *binding expression*. 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.

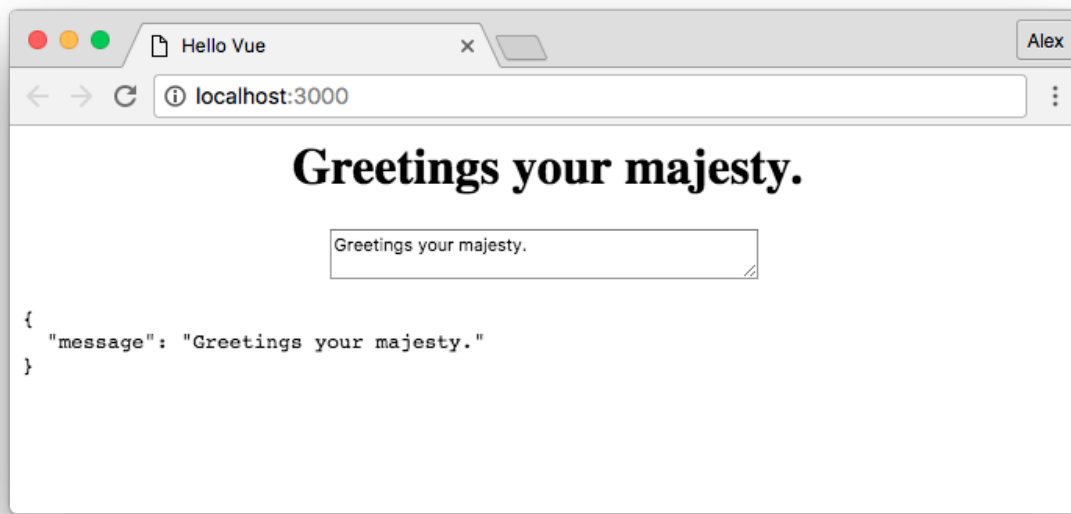
2.2 Two-way Binding

What is cool about Vue is that it makes our lives easier. Assume we want to change the message on user input, how can we easily accomplish it? 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 bound! 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 some 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!



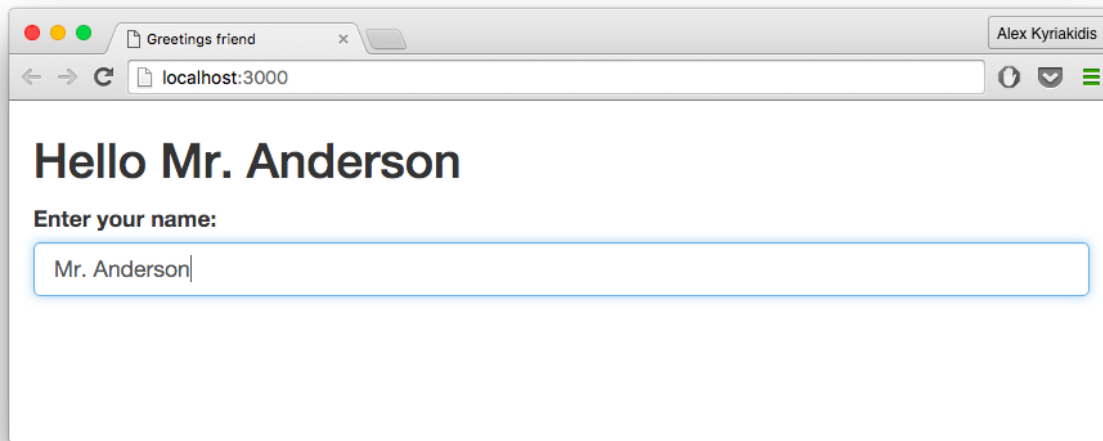
Code Examples

You can find the code examples of this chapter on [GitHub²](https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/codes/chapter2.html).

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

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



Note

The example's output makes use of Bootstrap. If you are not familiar with bootstrap you can ignore it for now, it is covered in a [later chapter](#).



Potential Solution

You can find a potential solution to this exercise [here](#)³.

³<https://github.com/hootlex/the-majesty-of-vuejs-2/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 Angular.js before, you probably don't know what a directive is. Essentially, a directive is a 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** 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 can 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/2.3.4/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 bound and by this point you may 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   <pre>
9     {{ $data }}
10  </pre>
11 </div>
12 </body>
13 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/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. Vue will nicely format the output for us automatically

whether it's a string, number, array, or a plain object. 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 }}` is bound 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 }}
14   </pre>
15 </div>
16 </body>
17 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/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

¹<http://www.json.org/>

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.



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

At 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-if="message">
10     Send word to allies for help!
11   </button>
12   <pre>
13     {{ $data }}
14   </pre>
15 </div>
16 </body>
17 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js"></script>

```

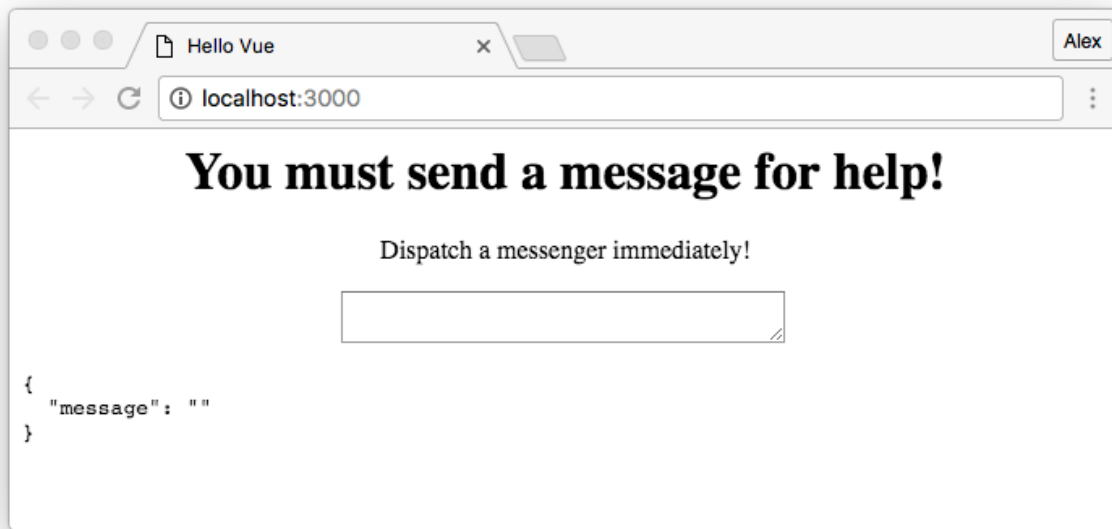
```
18 <script>
19   new Vue({
20     el: '#app',
21     data: {
22       message: 'Our king is dead! Send help!'
23     }
24   })
25 </script>
26 </html>
```

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, we can use `v-if` on a `<template>` element. In occasions where the use of `div` or `span` doesn't seem appropriate, the `<template>` element can also serve as an invisible wrapper. 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 }}
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.



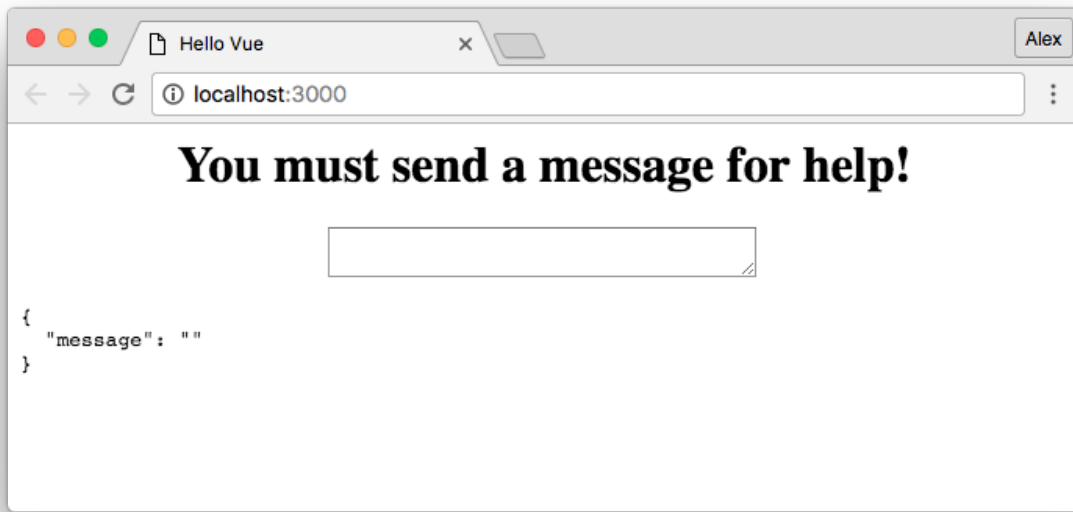
Warning

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

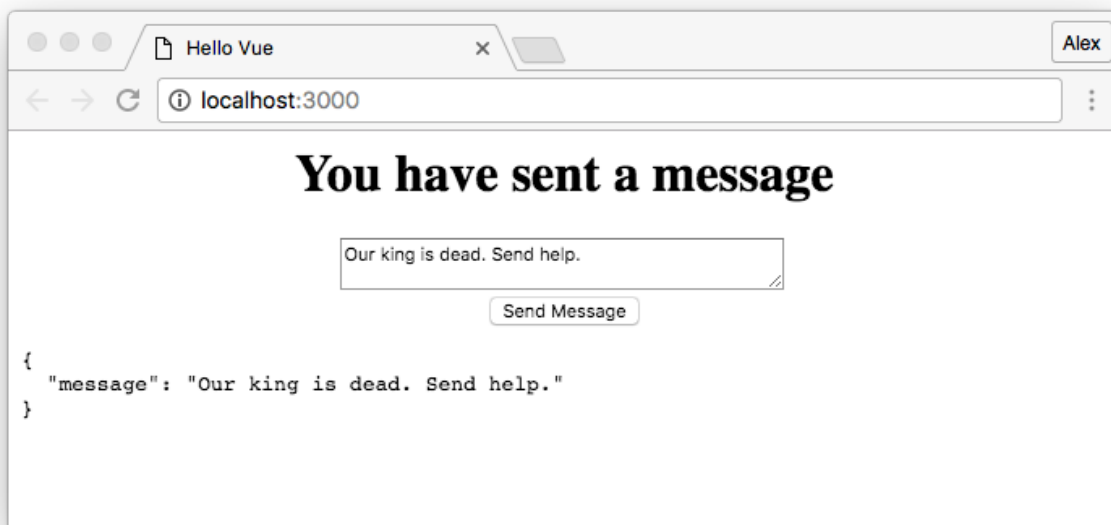
3.3 v-else

When using **v-if** 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** directive - otherwise it will not be recognized.

```
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 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 }}
15     </pre>
16   </div>
17 </body>
18 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js"></script>
19 <script>
20   new Vue({
21     el: '#app',
22     data: {
23       message: 'Our king is dead! Send help!'
24     }
25   })
26 </script>
27 </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 no message is presented, 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`. Simple as a pimple!



Warning

The **v-show** directive doesn't work anymore with **v-else**, in Vue 2.0.

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 -- the conditional block won't be rendered 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.



Code Examples

You can find the code examples of this chapter on [GitHub](https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter3)².

²<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter3>

3.5 Homework

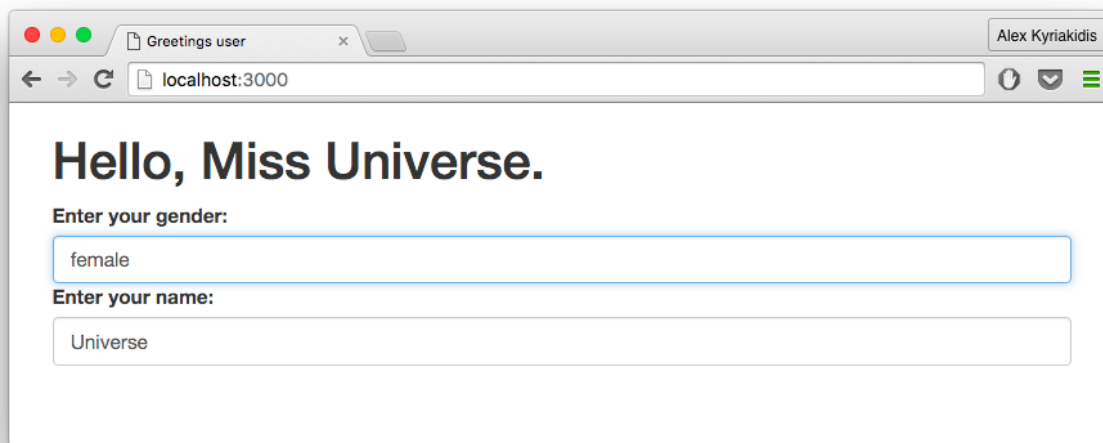
Following the previous homework exercise, you should try to expand it a bit. The user now types in his gender along with his name. If user is a male, then the heading will greet the user with “**Hello Mister {{name}}**”. If user is a female, then “**Hello Miss {{name}}**” should appear instead.

When gender is neither male or female then the user should see the warning heading “**So you can’t decide. Fine!**”.



Hint

A logical operator would come handy to determine user title.



Example Output



Potential Solution

You can find a potential solution to this exercise [here](https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter3.html)³.

³<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter3.html>

4. List Rendering

In the fourth 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.

4.1 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 in 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.

```
1 <div class="container">
2   ...
3 </div>
```

- Use `.container-fluid` for a full width container, spanning the entire width of your viewport.

¹<http://getbootstrap.com/>

²<https://www.bootstrapcdn.com/>

```
1   <div class="container-fluid">
2       ...
3   </div>
```

At this point, we would like to make an example of Vue.js with Bootstrap classes. Of course, not much study or experimentation is required in order to 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 </div>
11 </body>
12 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js"></script>
13 <script type="text/javascript">
14   new Vue({
15     el: '.container'
16   })
17 </script>
18 </html>
```

Notice this time, instead of targeting `app` id, we have targeted the `container` class within the `el` option inside the Vue instance.



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**.

The `el` property can be a CSS selector or an actual HTML Element. *It is not recommended to mount the root instance to `<html>` or `<body>`.*

4.2 v-for

In order to loop through each item in an array, we will use the **v-for** directive.

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.

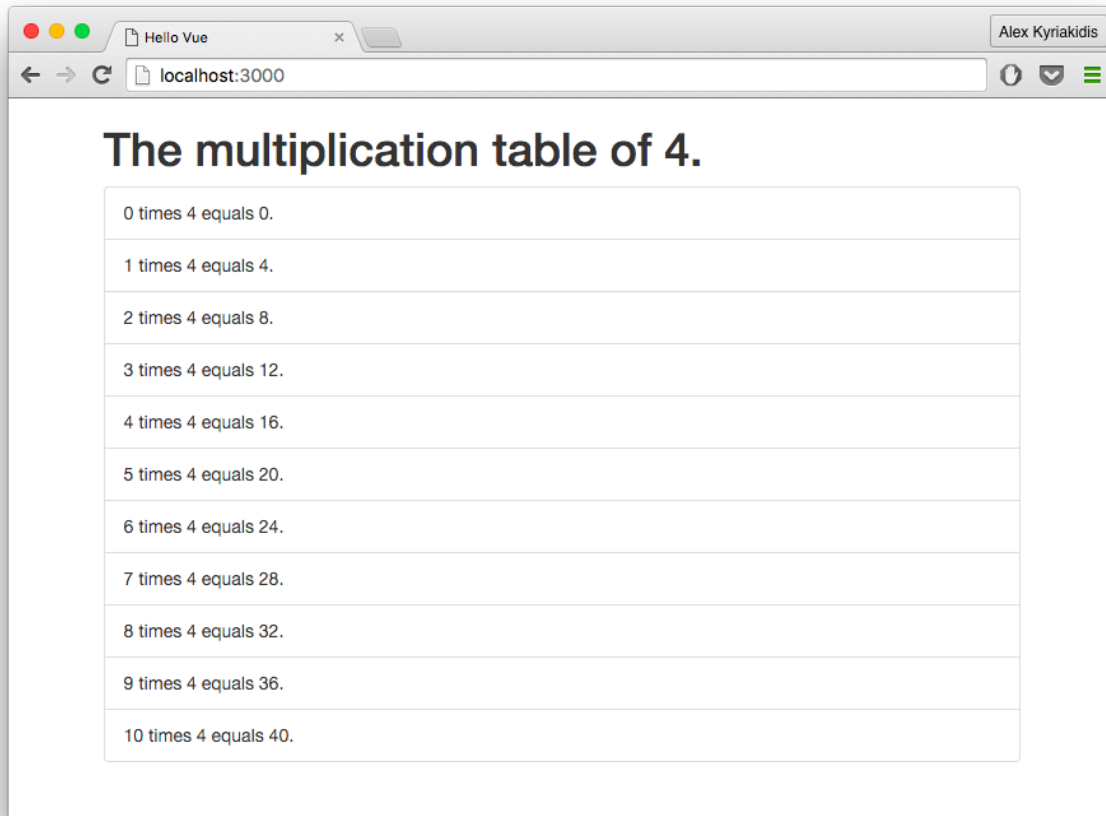
4.2.1 Range v-for

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-1 }} times 4 equals {{ (i-1) * 4 }}.
13             </li>
14         </ul>
15     </div>
16 </body>
17 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/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 1.

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.

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

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 }}  
19     </pre>  
20   </div>  
21 </body>  
22 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js"></script>  
23 <script type="text/javascript">  
24   new Vue({  
25     el: '.container',  
26     data: {
```

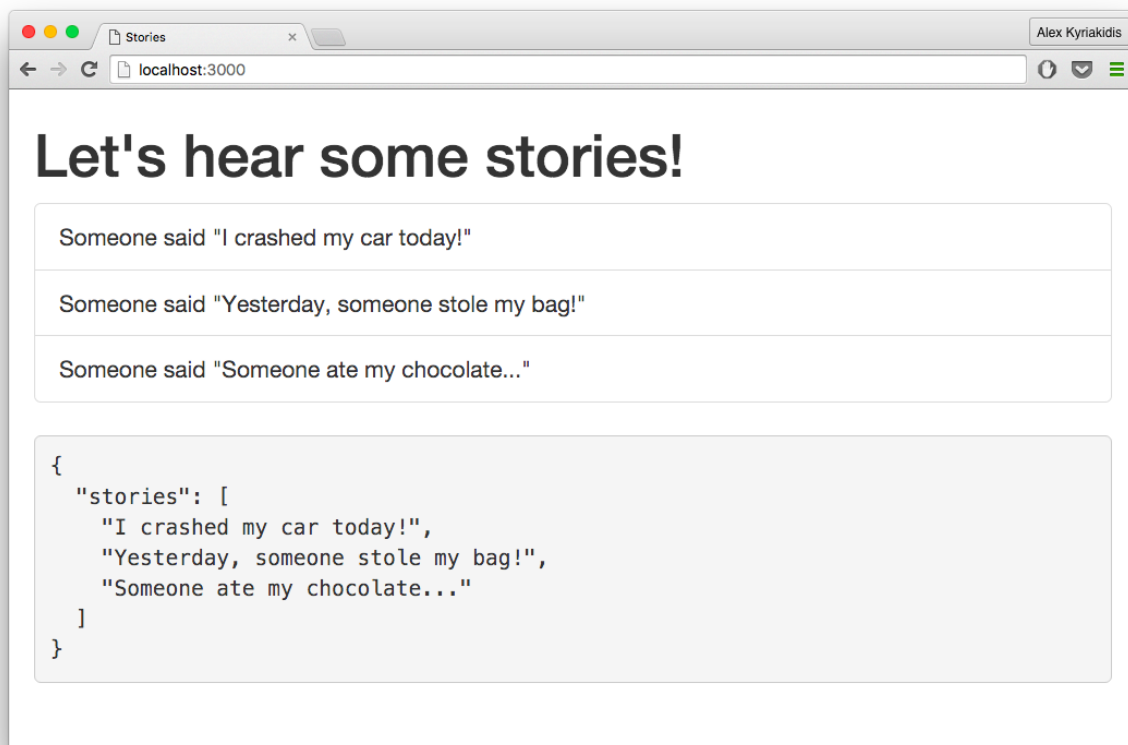


```
27     stories: [  
28         "I crashed my car today!",  
29         "Yesterday, someone stole my bag!",  
30         "Someone ate my chocolate...",  
31     ]  
32 }  
33 })  
34 </script>  
35 </html>
```



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>

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 alter 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.

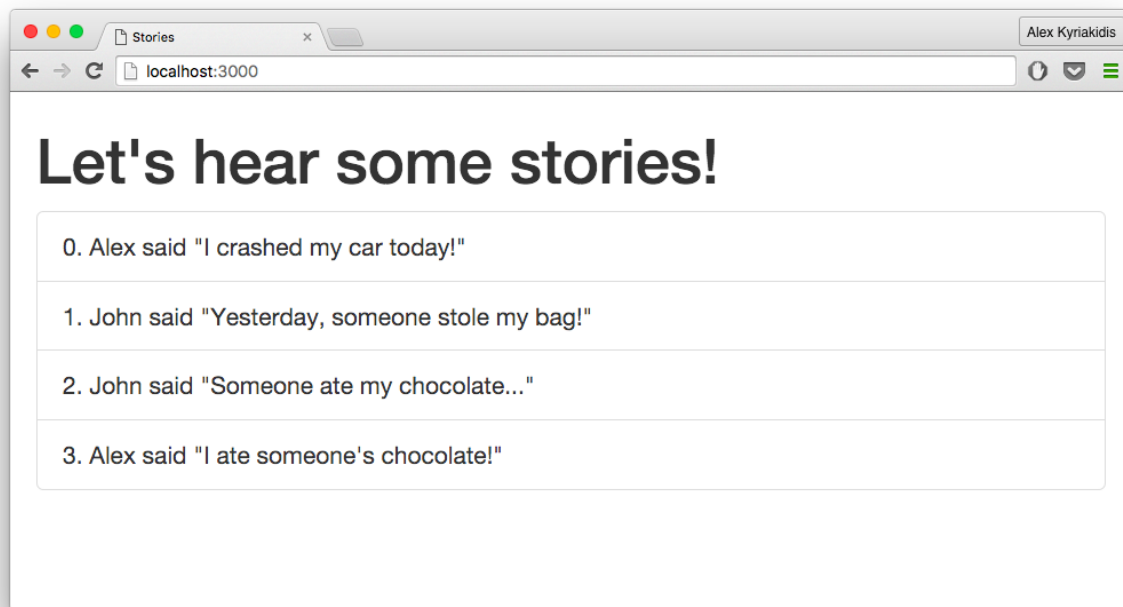
```
1  <html>
2  <head>
3  <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4  s" rel="stylesheet">
5    <title>Stories</title>
6  </head>
7  <body>
8    <div class="container">
9      <h1>Let's hear some stories!</h1>
10     <div>
11       <ul class="list-group">
12         <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 }}
21     </pre>
22   </div>
23 </body>
24 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/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 the `index` special variable. It works like this:

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

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



Rendered array with index

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**.

```
<div class="container">
  <h1>Let's hear some stories!</h1>
  <ul class="list-group">
    <li v-for="value in story" class="list-group-item">
      {{ value }}
    </li>
  </ul>
</div>
```

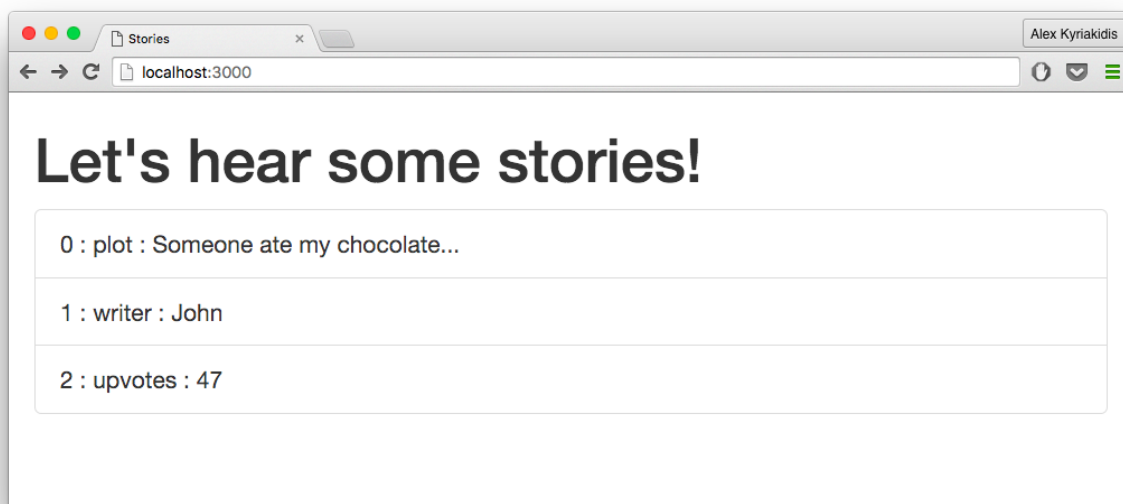
```
new Vue({
  el: '.container',
  data: {
    story: {
      plot: "Someone ate my chocolate...",
      writer: 'John',
      upvotes: 47
    }
  }
})
```

We can provide a second and third argument, for the **key** and **index** respectively.

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

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.

The result will be:



Iterate through object's properties.



Code Examples

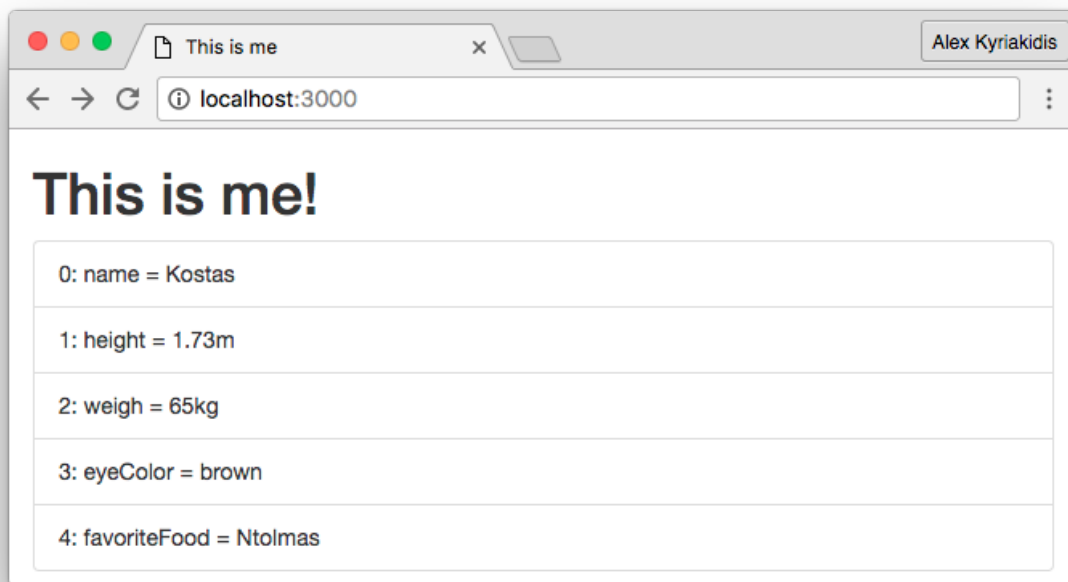
You can find the code examples of this chapter on [GitHub](https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter4)⁴.

⁴<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter4>

4.5 Homework

Keeping in mind what we reviewed in this chapter, for this homework, create an object with your personal attributes. By personal attributes, I mean your name, weight, height, eyeColor, and your favoriteFood.

Using `v-for`, iterate through each property and bring it into display in the format of: **index: key = value**.



Example Output



Potential Solution

You can find a potential solution to this exercise [here](https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter4.html)⁵.

⁵<https://github.com/hootlex/the-majesty-of-vuejs-2/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 DOM elements. When Vue.js is used in HTML pages, it can **react** to these events.

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

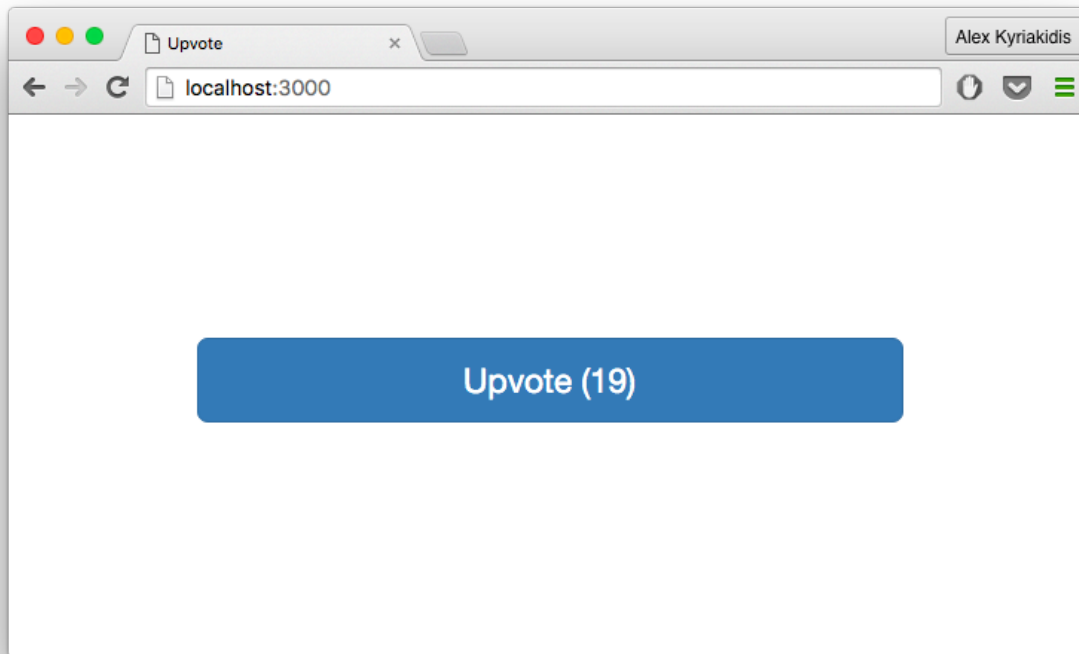
The **keyup** event occurs when the user releases a key. You can find a full list of HTML events [here](http://www.w3schools.com/tags/ref_eventattributes.asp)¹.

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 type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/vue/2\
15 .3.4/vue.js"></script>
16 <script type="text/javascript">
17 new Vue({
18   el: '.container',
19   data: {
20     upvotes: 0
21   }
22 })
23 </script>
24 </html>
```



Upvotes counter

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, each time the button is pressed we're simply increasing the count of upvotes by one, using the increment operator (`upvotes++`).

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 type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/vue/2\
15 .3.4/vue.js"></script>
16 <script type="text/javascript">
17 new Vue({
18   el: '.container',
19   data: {
20     upvotes: 0
21   },
22   // define methods under the **`methods`** object
23   methods: {
24     upvote: function(){
25       // **`this`** inside methods points to the Vue instance
26       this.upvotes++;
27     }
28   }
29 })
30 </script>
31 </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*. Using the shorthand is totally optional.

With the use of `@` the button of our previous example will be:

Listening to 'click' using v-on:

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

Listening to 'click' using @ shorthand

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

5.2 Event Modifiers

Now we will move on and create a Calculator app. To do so, we'll 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 modifier 'number'
12      is passed in order to parse inputs as numbers.-->
13      <input v-model.number="a" class="form-control">
14      <select v-model="operator" class="form-control">
15        <option>+</option>
16        <option>-</option>
17        <option>*</option>
18        <option>/</option>
19      </select>
20      <!-- Notice here the special modifier 'number'
21      is passed in order to parse inputs as numbers.-->
22      <input v-model.number="b" 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 }}
31 </pre>
32 </div>
33 </body>
34 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/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>
```

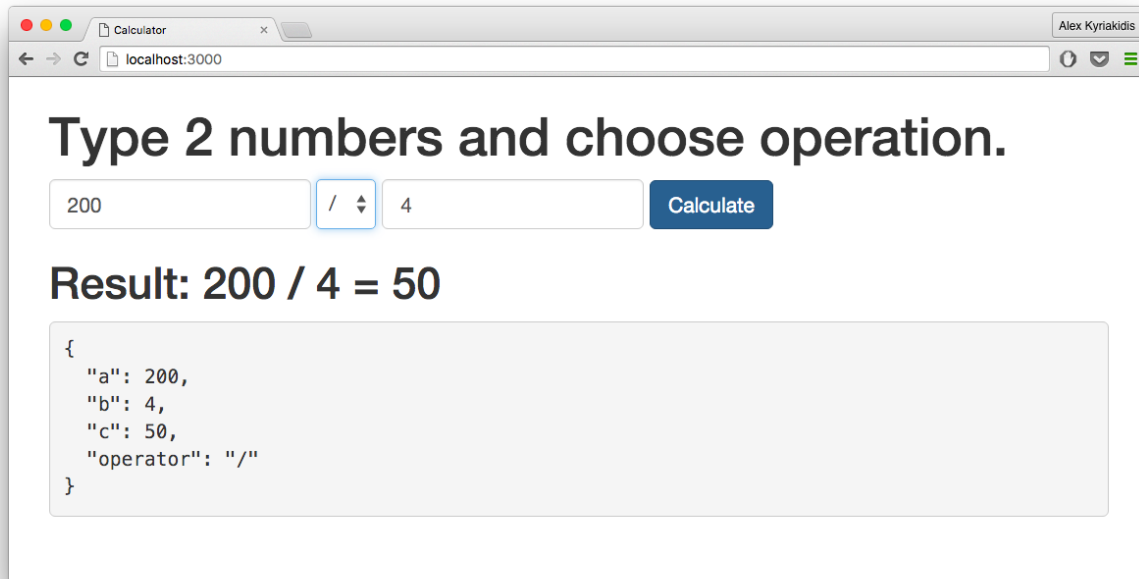
If you try to run this code yourself, you will find out that when the “calculate” button is clicked, instead of calculating, it reloads the page.

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:

```
calculate: function(event){
    event.preventDefault();
    switch (this.operator) {
        case "+":
            this.c = this.a + this.b
            break;
        case "-":
            this.c = this.a - this.b
            break;
        case "*":
            this.c = this.a * this.b
            break;
        case "/":
            this.c = this.a / this.b
            break;
    }
}
```



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 four event modifiers for `v-on` to prevent the event default behavior:

1. `.prevent`
2. `.stop`
3. `.capture`
4. `.self`

So, using `.prevent`, our submit button will change from:

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

to:

```
1 <!-- the submit event will no longer reload the page -->
2 <button type="submit" @click.prevent="calculate">Calculate</button>
```

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



Note

`.capture` and `.self` are rarely used so we won't bother elaborating any further. If you are interested in learning more about *Event Order* have a look at this [tutorial](http://www.quirksmode.org/js/events_order.html)².

²http://www.quirksmode.org/js/events_order.html

5.3 Key Modifiers

When you focus on one of the inputs and you hit enter, you will notice that the `calculate` method is getting invoked. If the button wasn't inside the form, or if there was no button at all, you could listen for a keyboard event instead.

When listening for keyboard events, we often need to check for key codes. The key code for **Enter** button is 13. So we could use it like this:

```
1 <input v-model="a" @keyup.13="calculate">
```

Remembering all the `keyCodes` is a hassle, so Vue provides aliases for the most commonly used keys:

- enter
- tab
- delete
- esc
- space
- up
- down
- left
- right

So, to execute `calculate` method when Enter is pressed in our example, the inputs will be like this:

```
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.

For 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 }}
12     </pre>
13 </div>
14 </body>
15 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/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>
```

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 }}
13     </pre>
14 </div>
15 </body>
16 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/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>
```

The above example is the same as the previous one, but with one difference. An input is bound to the **a** variable. The desired outcome would be to change the value of the binded attribute and immediately update the result of **b**. But notice here, that it does not work as we would expect.

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

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

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

```
new Vue({
  el: '.container',
  data: {
    a: 1,
  },
  computed: {
    b: function () {
      return parseFloat(this.a) + 1
    }
  }
});
```

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

But there is a more neat way. With Vue.js, whenever you want user's input to be automatically persisted as number, you can append the special modifier `.number`.

```
<body>
<div class="container">
  a={{ a }}, b={{ b }}
  <input v-model.number="a">
  <pre>
    {{ $data }}
  </pre>
</div>
</body>
```

The `number` modifier 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 already shown, 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.number="a" class="form-control">
12      +
13      <input v-model.number="b" class="form-control">
14    </form>
15    <h2>Result: {{a}} + {{b}} = {{c}}</h2>
16    <pre>{{ $data }}</pre>
17  </div>
18 </body>
19 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/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 their sum. 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 going to show only the Javascript codeblock.

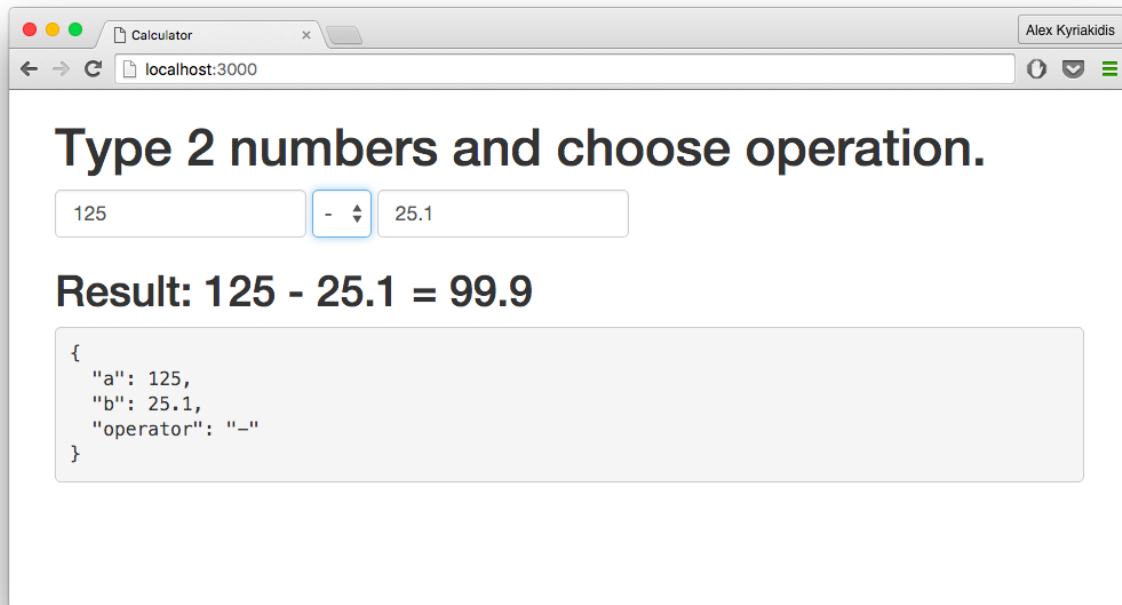
```
1  new Vue({
2    el: '.container',
3    data: {
4      a: 1,
5      b: 2,
6      operator: "+",
7    },
8    computed: {
9      c: function () {
10         switch (this.operator) {
11           case "+":
12             return this.a + this.b
13             break;
14           case "-":
15             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   });
```

The calculator is ready for use. The only thing we had to do, was to move whatever was inside **calculate** method to the computed property **c**! Whenever you change the value of **a** or **b** the result updates in real time! We don't need any buttons, events, or anything. **How awesome is that??**



Note

Note here that a normal approach would be to have an **if** statement to avoid error of division. But, 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.



Code Examples

You can find the code examples of this chapter on [GitHub](https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter5)³.

³<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter5>

5.5 Homework

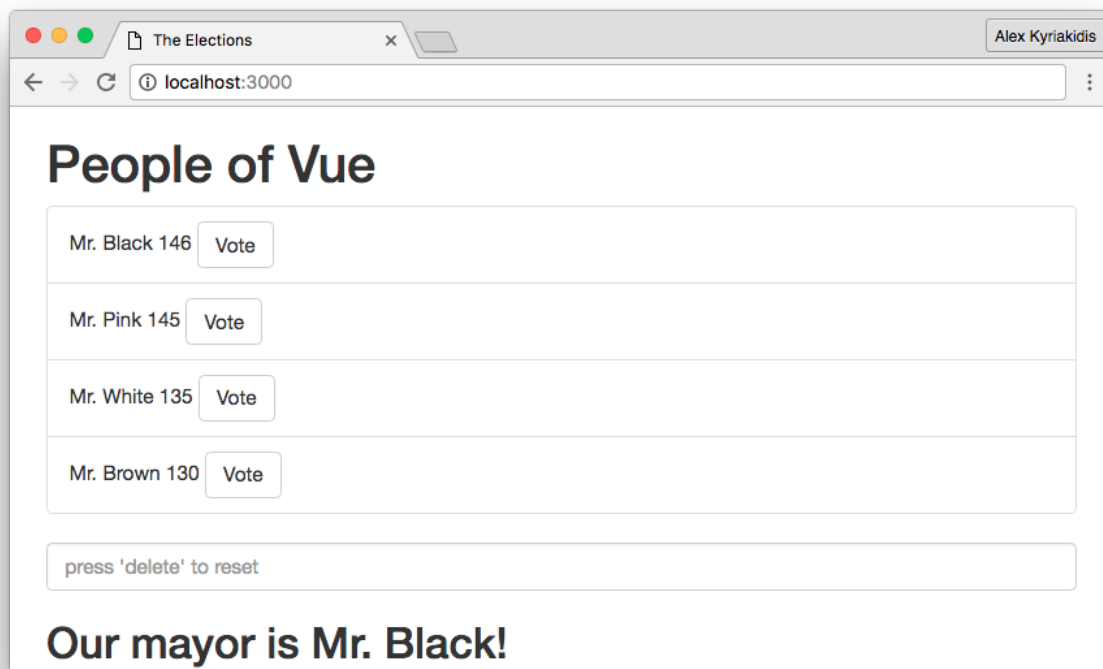
Now that you have a basic understanding of Vue's event handling, methods, computed properties etc, you should try something a bit more challenging. Start by creating an array of “Mayor” candidates. Each candidate has a “name” and a number of “votes”. Use a button to increase the count of votes for each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

Finally, add an input. When this input is focused, and key ‘**delete**’ is pressed, the elections start from the beginning. This means that all votes become 0.



Hint

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



Example Output



Potential Solution

You can find a potential solution to this exercise [here](https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter5.html)⁴.

⁴<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter5.html>

6. Filters

In the two previous chapters we reviewed list rendering, methods, and computed properties. Now it is a good time to make some examples using all the information above. In this chapter, we will cover how to:

1. Filter an array of items.
2. Order an array of items.
3. Apply a custom filter.
4. Use utility libraries.

The plan is to go through similar examples as before, combining some or all of the techniques we saw.

6.1 Filtered Results

Sometimes we need to display a filtered version of an array without actually mutating or resetting the original data. Continuing the previous example, [Loop Through an Array of Objects](#), we would like to display one list with the stories written by *Alex* and one list with the stories written by *John*. We can achieve this, by creating a method which filters our array and returns the results to be rendered.



Info

As of Vue 2.0, Vue filters cannot be used within `v-for`. Filters can now only be used inside text interpolations (`{{ }}`). Vue's team suggests to move filters' logic into JavaScript, so that it can be reused throughout your component.

```
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 storiesBy('Alex')"
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 storiesBy('John')"
22           class="list-group-item"
23         >
24           {{ story.writer }} said "{{ story.plot }}"
25         </li>
26       </ul>
27     </div>
28     <pre>
29       {{ $data }}
30     </pre>
31   </div>
32 </body>
33 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/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 methods:
58 {
59     // a method which filters the stories depending on the writer
60     storiesBy: function (writer) {
61         return this.stories.filter(function (story) {
62             return story.writer === writer
63         })
64     },
65 }
66 })
67 </script>
68 </html>
```

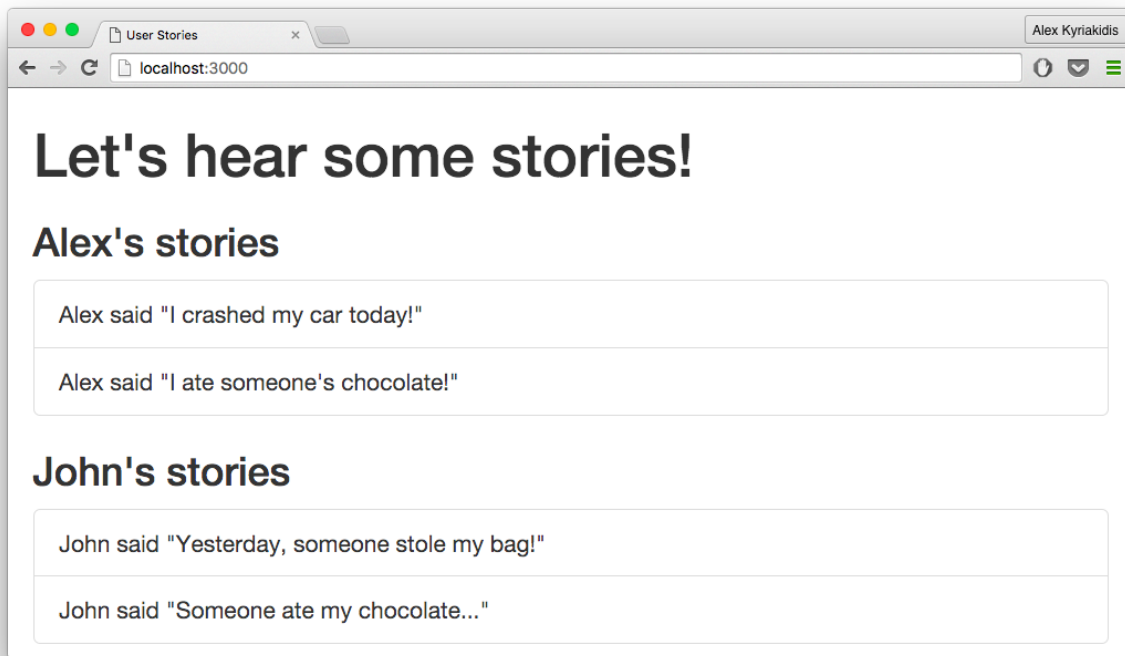


Info

Within `storiesBy` method, we use [javascript's built-in `filter` function](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Array/filter).¹ The `filter()` function creates a new array with all elements that pass the test, implemented by the provided function.

As it appears we have created a method named `storiesBy` which takes in a *writer*, as argument, and returns a filtered array with the writer's stories. We can then use this, to display each writer's stories using the `v-for` directive in the format of `story in storiesBy('Alex')`, as you can see in the example above.

¹https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Array/filter



Stories filtered by writer.



Note

As you may noticed, our `li` tag is getting really big, so we have splitted it in more lines. The actual result remains the same as with the use of filters, introduced in Vue 1.x.

Simple enough, right?

6.1.1 Using Computed Properties

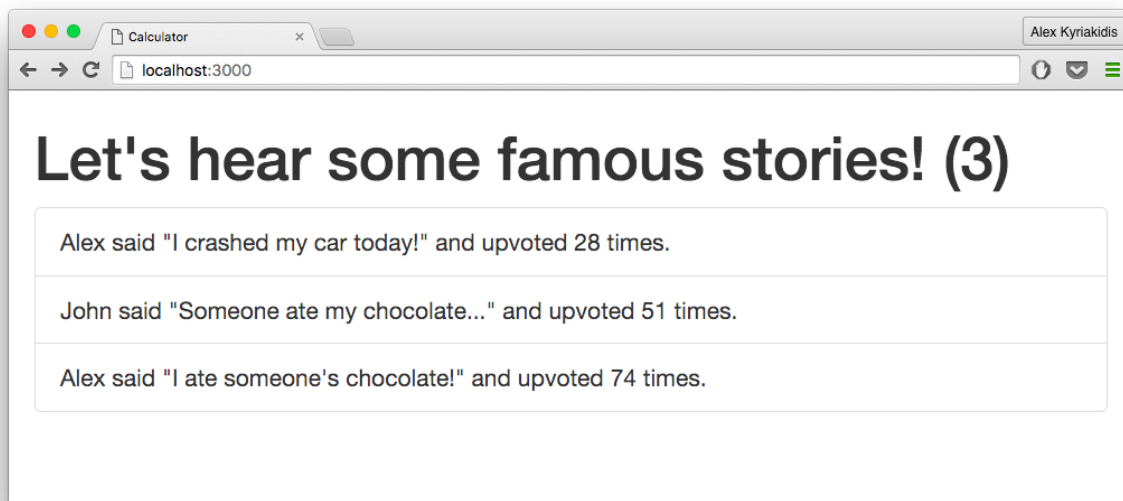
A *computed property* can also be used to filter an array. Using a computed property to perform array filtering gives you in-depth control and more flexibility, since it's full JavaScript, and allows you to access the filtered result elsewhere. For example you can get the length of a filtered array anywhere in your code.

First we will enhance our Stories with a new property, called *upvotes*. Then, we will filter the **famous** stories. As **famous**, we define the *stories* that have more than 25 upvotes. This time, we will create a computed property that returns the filtered Array.

```
new Vue({
  el: '.container',
  data: {
    stories: [
      {
        plot: "I crashed my car today!",
        writer: "Alex",
        upvotes: 28
      },
      {
        plot: "Yesterday, someone stole my bag!",
        writer: "John",
        upvotes: 8
      },
      {
        plot: "Someone ate my chocolate...",
        writer: "John",
        upvotes: 51
      },
      {
        plot: "I ate someone's chocolate!",
        writer: "Alex",
        upvotes: 74
      },
    ]
  },
  computed: {
    famous: function() {
      return this.stories.filter(function(item){
        return item.upvotes > 25;
      });
    }
  }
})
```

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

```
<body>
  <div class="container">
    <h1>Let's hear some famous stories! ({{famous.length}})</h1>
    <ul class="list-group">
      <li v-for="story in famous"
        class="list-group-item"
      >
        {{ story.writer }} said "{{ story.plot }}"
        and upvoted {{ story.upvotes }} times.
      </li>
    </ul>
  </div>
</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}}`?

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     ...
5     <div class="form-group">
6       <label for="query">
7         What are you looking for?
8       </label>
9       <input v-model="query" class="form-control">
10    </div>
11    <h3>Search results:</h3>
12    <ul class="list-group">
13      <li v-for="story in search"
14        class="list-group-item"
15      >
16        {{ story.writer }} said "{{ story.plot }}"
17      </li>
18    </ul>
19  </div>
20 </div>

```

Then we will create a computed property named **search**. Along with the built-in **filter** function, we are going to use the [includes](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/includes)² Javascript's function, which determines whether one string may be found within another string.

```

1 new Vue({
2   el: '.container',
3   data: {
4     stories: [...],
5     query: ''
6   },
7   methods: {
8     storiesBy: function (writer) {
9       return this.stories.filter(function (story) {
10         return story.writer === writer
11       })
12     }
13   },
14   computed: {
15     search: function () {
16       var query = this.query

```

²https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/includes

```
17         return this.stories.filter(function (story) {
18             return story.plot.includes(query)
19         })
20     }
21 }
22 })
```


User Stories x Alex Kyriakidis

localhost:3000

Lets hear some stories!

Alex's stories

Alex said "I crashed my car today!"

Alex said "I ate someone's chocolate!"

John's stories

John said "Yesterday, someone stole my bag!"

John said "Someone ate my chocolate..."

What are you looking for?

Search results:

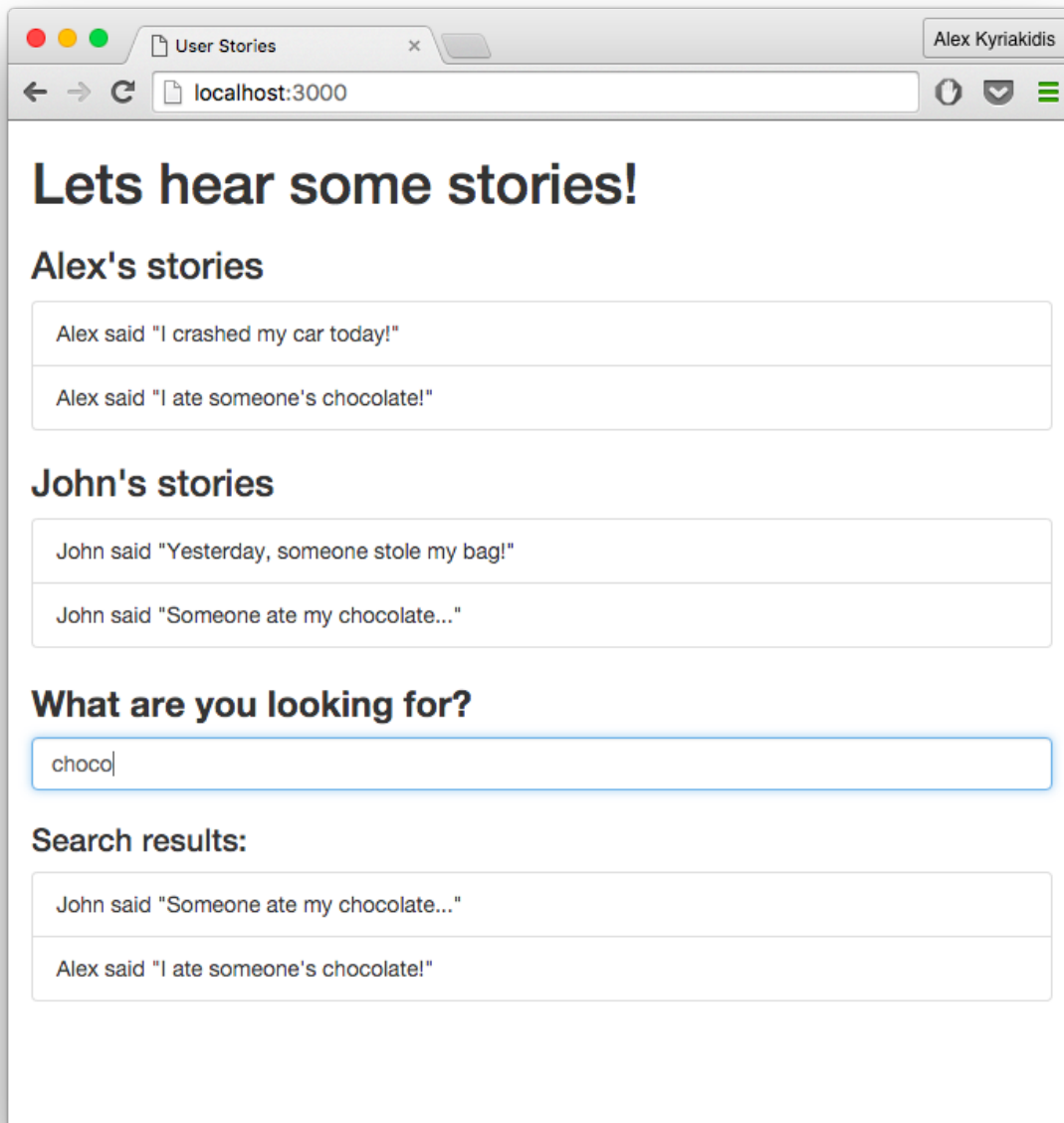
Alex said "I crashed my car today!"

John said "Yesterday, someone stole my bag!"

John said "Someone ate my chocolate..."

Alex said "I ate someone's chocolate!"

Search Stories.



Searching for 'choco'.

Isn't that awesome??

6.2 Ordered Results

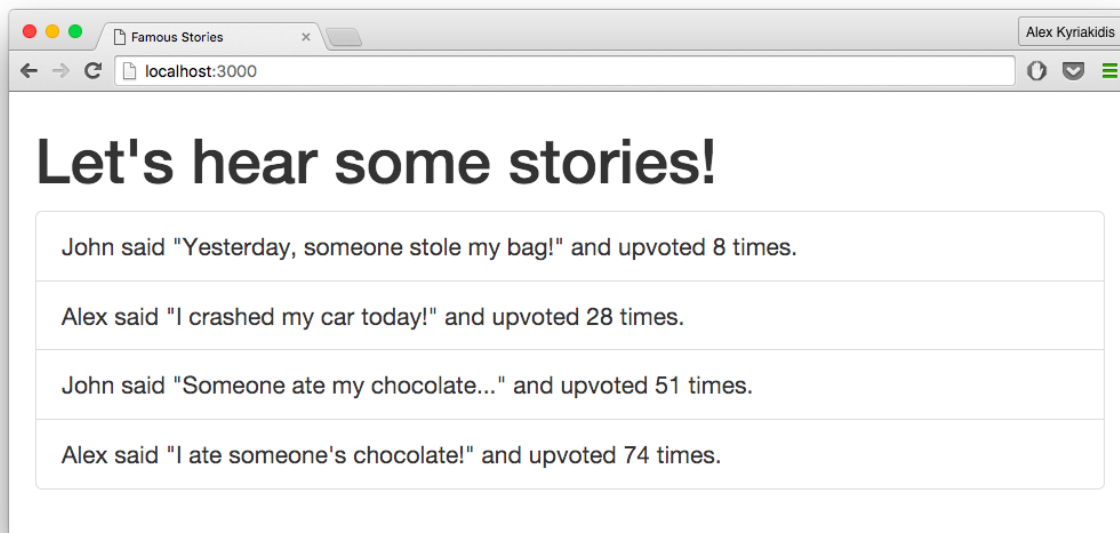
Sometimes, we may want to display the items of an Array ordered by some criteria. We can use a *computed property* to display our array, ordered by the count of each story's *upvotes*. To sort the array we are going to use JavaScript's `sort`³ function, which sorts the elements of an array in place and returns the array.

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 orderedStories"
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 }}
20 </pre>
21 </div>
22 </body>
23 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js"></script>
24 <script type="text/javascript">
25 new Vue({
26   el: '.container',
27   data: {
28     stories: [...]
29   },
30   computed: {
31     orderedStories: function () {
32       return this.stories.sort(function(a, b){
```

³https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort

```
33         return a.upvotes - b.upvotes;
34     })
35 }
36 }
37 })
38 </script>
39 </html>
```



Stories array ordered by upvotes.

Hmmm, the array is ordered but this is not what we expected. We wanted the **famous stories first**.

To change the order of the sorted array we have to take a look at the **sort** function. In JavaScript's **sort(compareFunction)**, if **compareFunction** is supplied, the array elements are sorted according to the return value of **compareFunction**. If **a** and **b** are two elements being compared, then:

- If **compareFunction(a, b)** is less than 0, sort **a** to a lower index than **b**.
- If **compareFunction(a, b)** is 0, leave **a** and **b** unchanged.
- If **compareFunction(a, b)** is greater than 0, sort **b** to a lower index than **a**.

In our use case, the **compareFunction** will be:

compareFunction

```
function(a, b){
  return a.upvotes - b.upvotes;
}
```

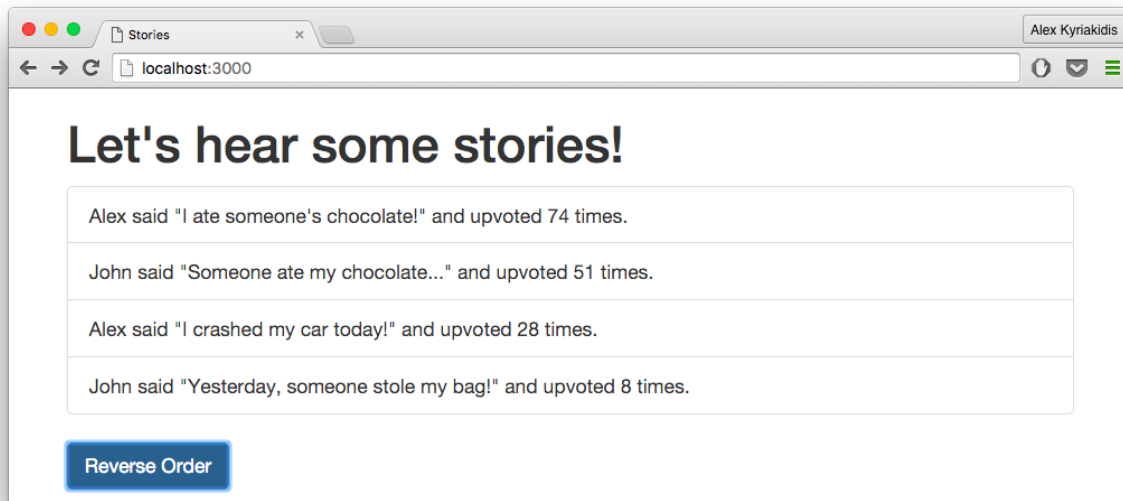
So, to change the order from ascending to descending, we can multiple the returned value by **-1**.
(return (a.upvotes - b.upvotes) * -1)

We can change the order dynamically, by using a variable, **order**. A **button** will be used, which will toggle the value of the new variable, between **-1** and **1**.

```
1 <div class="container">
2   <h1>Let's hear some stories!</h1>
3   <ul class="list-group">
4     <li v-for="story in orderedStories"
5       class="list-group-item"
6     >
7       {{ story.writer }} said "{{ story.plot }}"
8       and upvoted {{ story.upvotes }} times.
9     </li>
10  </ul>
11  <button @click="order = order * -1">Reverse Order</button>
12  <pre>
13    {{ $data }}
14  </pre>
15 </div>
```

```
1 new Vue({
2   el: '.container',
3   data: {
4     stories: [...],
5     order : -1
6   },
7   computed: {
8     orderedStories: function () {
9       var order = this.order;
10      return this.stories.sort(function(a, b) {
11        return (a.upvotes - b.upvotes) * order;
12      })
13    }
14  }
15 })
```

We initialize **order** variable with the value of **-1** and then we pass it to our computed property, so every time the button is clicked, the variable changes value and the array changes order.



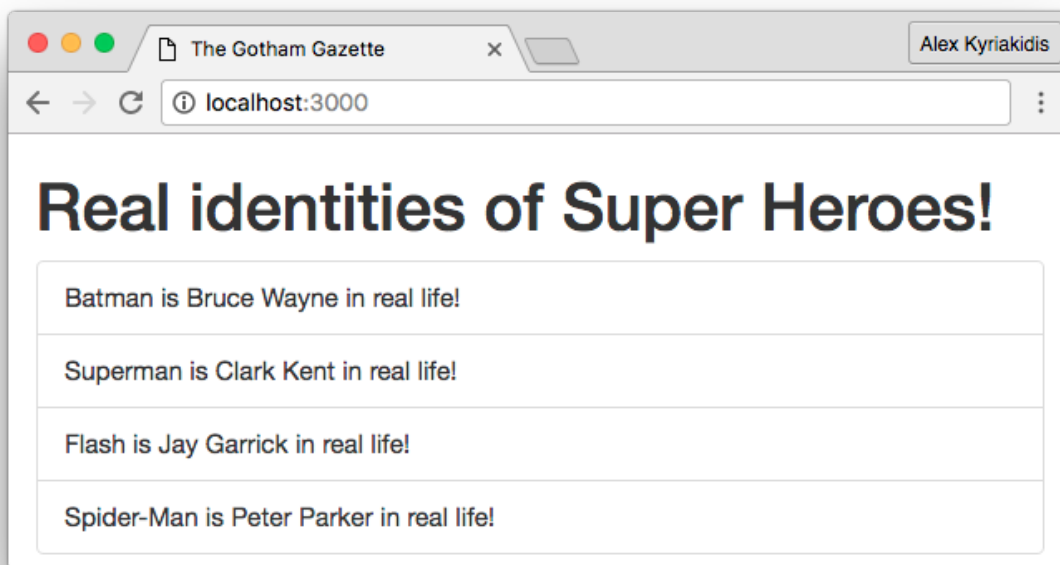
Array in descending order

6.3 Custom Filters

To demonstrate custom filters, we will make a new simple example. Assume we are now in charge of the Gotham's city news paper the "The Gotham Gazette". Our first job is to spread the news of the secret identities of heroes. We know the first and last names of them, and we want to make a nice list where each secret identity will be exposed. This is where the global `Vue.filter()` method comes in, to create a filter which can take a *hero* and return all his information for display, without polluting our HTML code. To register a filter we can pass in a `filterID` and a `filterFunction`, which returns a processed value. Then we will use the filter inside text interpolations like so:

```
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>The Gotham Gazette</title>
6 </head>
7 <body>
8 <div class="container">
9 <h1>Real identities of Super Heroes!</h1>
10 <ul class="list-group">
11 <li v-for="hero in heroes"
12 class="list-group-item"
13 >
14 {{ hero | snitch }}
15 </li>
16 </ul>
17 </div>
18 </body>
19 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js">
20 </script>
21 <script>
22 Vue.filter('snitch', function (hero) {
23   return hero.secretId + ' is '
24     + hero.firstname + ' '
25     + hero.lastname + ' in real life!'
26 })
27
28 new Vue({
29   el: '.container',
30   data: {
31     heroes: [
32       { firstname: 'Bruce', lastname: 'Wayne', secretId: 'Batman'},
```

```
33     { firstname: 'Clark', lastname: 'Kent', secretId: 'Superman'},
34     { firstname: 'Jay', lastname: 'Garrick', secretId: 'Flash'},
35     { firstname: 'Peter', lastname: 'Parker', secretId: 'Spider-Man'}
36   ]
37 }
38 })
39 </script>
40 </html>
```



Custom filter 'famous' in action.

6.4 Utility Libraries

At this point, we would like to point out that when you need to sort/filter/index data in more advanced ways, you should consider using a JavaScript utility library. There are some great utility libraries out there, such as [Lodash](https://lodash.com)⁴, [Underscore](http://underscorejs.org/)⁵, [Sugar](https://sugarjs.com/)⁶, etc.

To get a better understanding, we are going to include *Lodash* and update the previous example.

If you follow along, make sure you include *Lodash* from a cdn in your **HTML** file.

⁴<https://lodash.com>

⁵<http://underscorejs.org/>

⁶<https://sugarjs.com/>

Lodash's `orderBy` method returns a new sorted array. So, we'll use it within our computed property to sort the `Stories` array.

Syntax

Lodash's `orderBy` syntax is:

```
_.orderBy(collection, [iteratees=[_identity]], [orders])
```

Don't let the second argument confuse you. It is really simple. The first argument represents the array you want to sort. The second argument expects an array of keys, that the sorting will be based on. The third argument, expects an array of *orders* for each key.

For example if we had an array:

```
var kids = [
  { name: 'Stan', strength: 70, intelligence: 70},
  { name: 'Kyle', strength: 40, intelligence: 80},
  { name: 'Eric', strength: 45, intelligence: 80},
  { name: 'Kenny', strength: 100, intelligence: 70}
]
```

And we run:

```
_.orderBy(kids, ['intelligence', 'strength'], ['desc', 'asc'])
```

Our array will have this order:

```
var kids = [
  { name: 'Kyle', strength: 40, intelligence: 80},
  { name: 'Eric', strength: 45, intelligence: 80},
  { name: 'Stan', strength: 70, intelligence: 70},
  { name: 'Kenny', strength: 100, intelligence: 70}
]
```

Because the array is **primarily** sorted by kid's **intelligence in descending order** and **secondary**, by kid's **strength in ascending order**.

We will use `_.orderBy` within our computed property like this:

```
computed: {  
  orderedStories: function () {  
    var order = this.order  
    return _.orderBy(this.stories, 'upvotes')  
  }  
}
```

This works, but if no *orders* argument is passed, the array will be sorted in ascending order. Additionally, it should be possible to change the order of the array with a button, just like before. To do it dynamically, we can set a data property **order** : **'desc'** and create a method to change its value:

```
methods: {  
  reverseOrder: function () {  
    this.order = (this.order === 'desc') ? 'asc' : 'desc'  
  }  
},  
computed: {  
  orderedStories: function () {  
    var order = this.order  
    return _.orderBy(this.stories, 'upvotes', [order])  
  }  
}
```

And the button will be almost the same, but not quite.

```
<button v-on:click="reverseOrder">
```

That's enough. We have achieved the self-same functionality using *Lodash*.



Tip

When using a utility library to filter/order data, you can iterate the resulted array without using any computed properties.

We can update this example to get the idea. Our HTML which renders the sorted array would be:

```
<div class="container">
  <h1>Let's hear some stories!</h1>
  <ul class="list-group">
    <li v-for="story in _.orderBy(stories, ['upvotes'], ['desc'])">
      {{ story.writer }} said "{{ story.plot }}"
      and upvoted {{ story.upvotes }} times.
    </li>
  </ul>
</div>
```

That's it. No need to write any *JavaScript*.



Code Examples

You can find the code examples of this chapter on [GitHub](https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter6)⁷.

⁷<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter6>

6.5 Homework

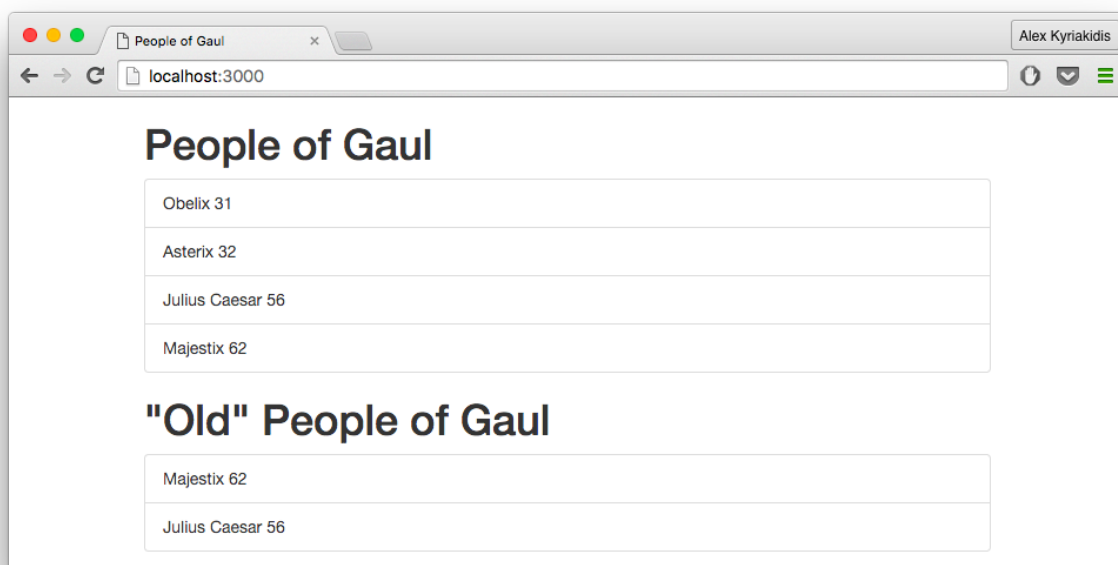
For this chapter's exercise you should do the following. Start by creating an array of people. Each person has a name and an age. Using what you've learned so far, try to render the array in a list and sort it by "age". After that, create a second list below and create a computed property named "old", which will return all people older than 65 years old.

Feel free to fill the array with your own data. **Be careful** to add people with age older and younger than 65 to ensure your filter is working properly. Go ahead!



Hint

Built in `.filter` is necessary here.



Example Output



Potential Solution

You can find a potential solution to this exercise [here](https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter6.html)⁸.

⁸<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter6.html>

7. Components

7.1 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 in order to do new things. In this chapter we are going to start with an extremely simple example. Next, we are going to see how Components can help us improve the code that we have created, in previous chapters.

7.2 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/2.3.4/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>
```

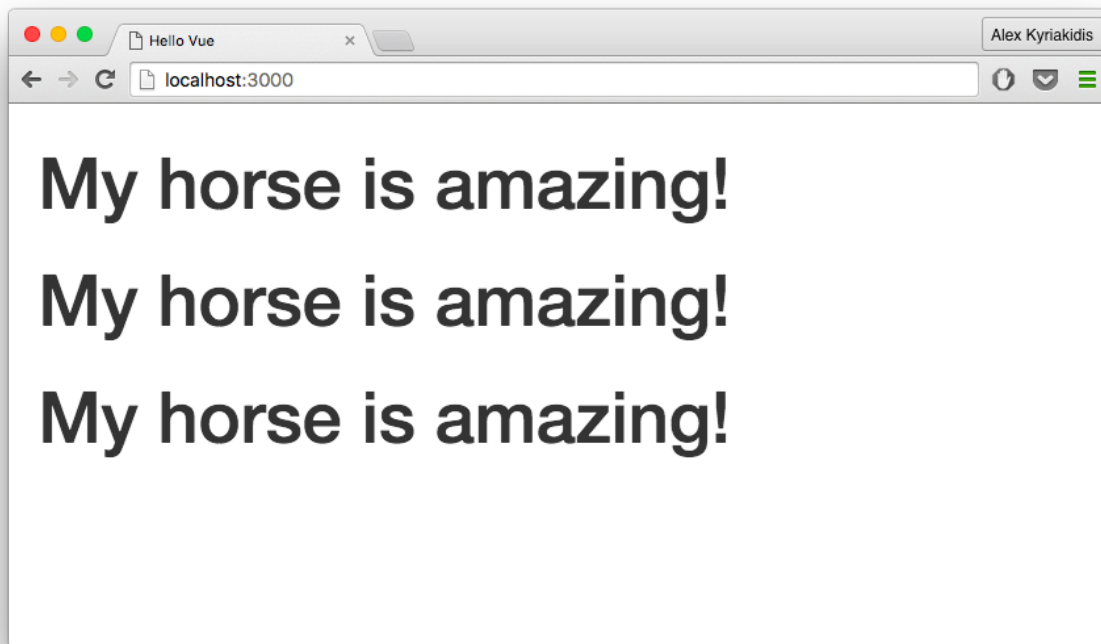


Note

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.

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

```
<body>
  <div class="container">
    <story></story>
    <story></story>
    <story></story>
  </div>
</body>
```



Displaying story component

7.3 Templates

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

Another way, is to create a **script** tag with type set to **text/template** with 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.

```
<script type="text/template" id="story-template">
  <h1>My horse is amazing!</h1>
</script>

<script type="text/javascript">
  Vue.component('story', {
    template: "#story-template"
  });
</script>
```



Info

The "**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 going to use in the examples of this book) 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:

```
<template id="story-template">
  <h1>My horse is amazing!</h1>
</template>

<script type="text/javascript">
  Vue.component('story', {
    template: "#story-template"
  });
</script>
```

7.4 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.

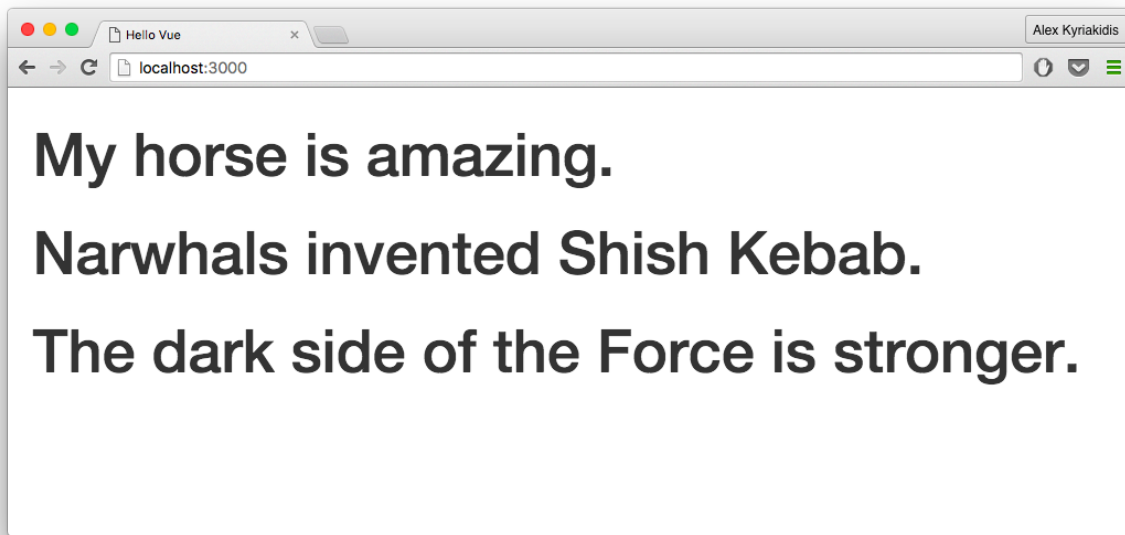
```
<template id="story-template">
  <h1>{{ plot }}</h1>
</template>
```

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.

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

Now we can pass a **plot** every time we use the **<story>** element.


```
<div class="container">
  <story plot="My horse is amazing."></story>
  <story plot="Narwhals invented Shish Kebab."></story>
  <story plot="The dark side of the Force is stronger."></story>
</div>
```



Display different 'stories'.



Warning

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

So, camelCase in JavaScript, kebab-case in HTML. For example, for `props: ['isUser']`, the HTML attribute would be `<story is-user="true"></story>`.

As you have probably 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.

```
<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   </div>
21   <template id="story-template">
22     <h1>{{ story.writer }} said "{{ story.plot }}"</h1>
23   </template>
24 </body>
25 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/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 going to use it like this: **:story="..."**.

7.5 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, decided to rename story's **plot** 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 storiesBy('Alex')"
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 storiesBy('John')"
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 search"
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* provides 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.

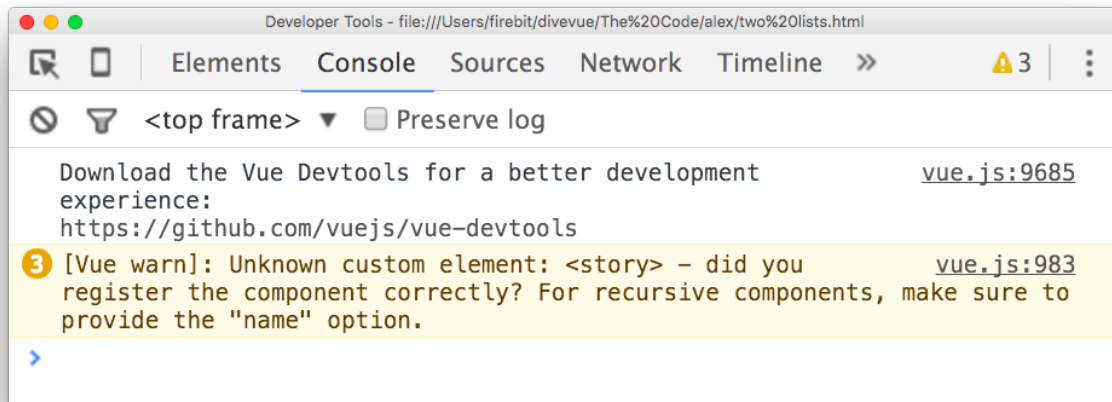
Luckily, we have created a **story** Component in the previous example, which displays the writer and the body for a specified story. We can use the custom element **<story>** inside our **HTML** and pass each story, as we did before, with the **:story** tag. This time we will use it inside a **v-for**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

So our code will be:

```
<div class="container">
  <h1>Lets hear some stories!</h1>
  <div>
    <h3>Alex's stories</h3>
    <ul class="list-group">
      <story v-for="story in storiesBy('Alex')"
        :story="story"></story>
    </ul>
    <h3>John's stories</h3>
    <ul class="list-group">
      <story v-for="story in storiesBy('John')"
        :story="story"></story>
    </ul>
    <div class="form-group">
      <label for="query">What are you looking for?</label>
      <input v-model="query" class="form-control">
    </div>
    <h3>Search results:</h3>
    <ul class="list-group">
      <story v-for="story in search"
        :story="story"></story>
    </ul>
  </div>
</div>
```

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

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



Vue warning

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:

```
<template id="story-template">
  <li class="list-group-item">
    {{ story.writer }} said "{{ story.body }}"
  </li>
</template>
```

The component will remain 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 the same as before, but this time with the use of a custom component.

Pretty neat huh?



Warning

Please be responsible. Don't drink and drive.

7.6 Altogether

Using our newly acquired knowledge, we should be able to build something a bit more complex. Based on the structure of the example above, we are going to create a voting system for our **stories**, and add a *mark as favorite* feature. The way to accomplish this, 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 }}</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/2.3.4/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 allows the user to give a vote to the story he prefers. To apply this limit (1 vote per story) we will display the ‘Upvote’ button only if the user has not already voted. So, every story must have a **voted** property, that becomes true when **upvote** function executes.

```

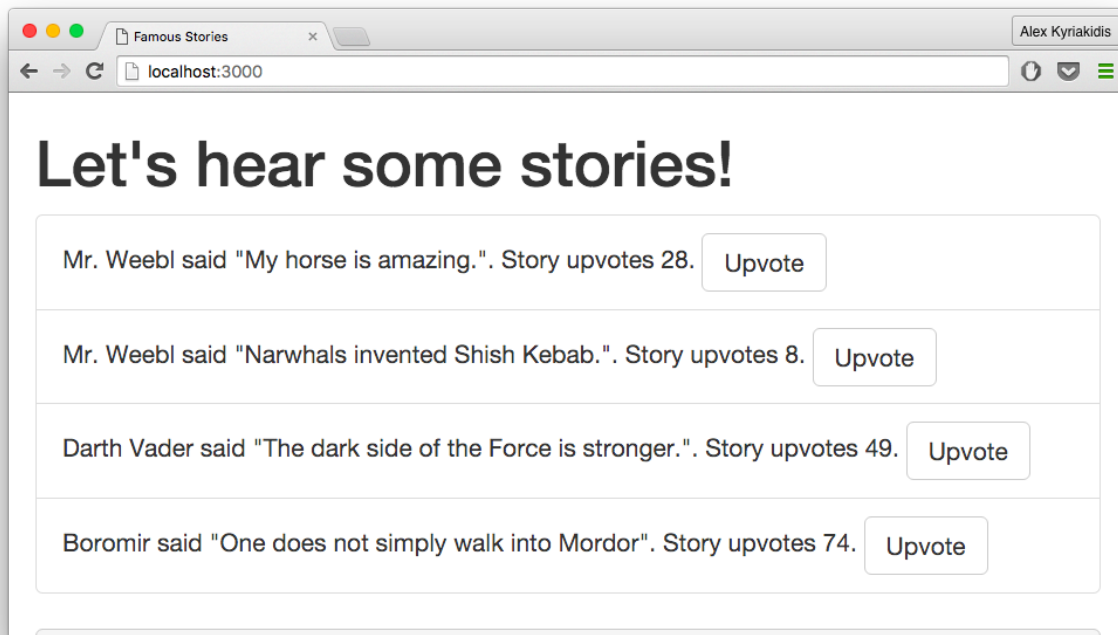
<template id="story-template">
  <li class="list-group-item">
    {{ story.writer }} said "{{ story.plot }}".
    Story upvotes {{ story.upvotes }}.
    <button v-show="!story.voted" @click="upvote"
      class="btn btn-default"
    >
      Upvote
    </button>
  </li>
</template>

```



```
Vue.component('story', {
  template: "#story-template",
  props: ['story'],
  methods: {
    upvote: function(){
      this.story.upvotes += 1;
      this.story.voted = true;
    },
  },
});

new Vue({
  el: '#app',
  data: {
    stories: [
      {
        plot: 'My horse is amazing.',
        writer: 'Mr. Weebl',
        upvotes: 28,
        voted: false,
      },
      {
        plot: 'Narwhals invented Shish Kebab.',
        writer: 'Mr. Weebl',
        upvotes: 8,
        voted: false,
      },
      {
        plot: 'The dark side of the Force is stronger.',
        writer: 'Darth Vader',
        upvotes: 49,
        voted: false,
      },
      {
        plot: 'One does not simply walk into Mordor',
        writer: 'Boromir',
        upvotes: 74,
        voted: false,
      },
    ]
  }
})
```



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.

```
<template id="story-template">
  <li class="list-group-item">
    {{ story.writer }} said "{{ story.plot }}".
    Story upvotes {{ story.upvotes }}.
    <button v-show="!story.voted" @click="upvote"
      class="btn btn-default">
      Upvote
    </button>
    <button v-show="!isFavorite" @click="setFavorite"
      class="btn btn-primary">
      Favorite
    </button>
    <span v-show="isFavorite"
      class="glyphicon glyphicon-star pull-right" aria-hidden="true">
    </span>
```

```

    </li>
  </template>

```

```

Vue.component('story', {
  template: "#story-template",
  props: ['story'],
  methods: {
    upvote: function() {
      this.story.upvotes += 1;
      this.story.voted = true;
    },
    setFavorite: function() {
      this.favorite = this.story;
    },
  },
  computed: {
    isFavorite: function() {
      return this.story == this.favorite;
    },
  },
});

```

```

new Vue({
  el: '#app',
  data: {
    stories: [
      ...
    ],
    favorite: {}
  }
})

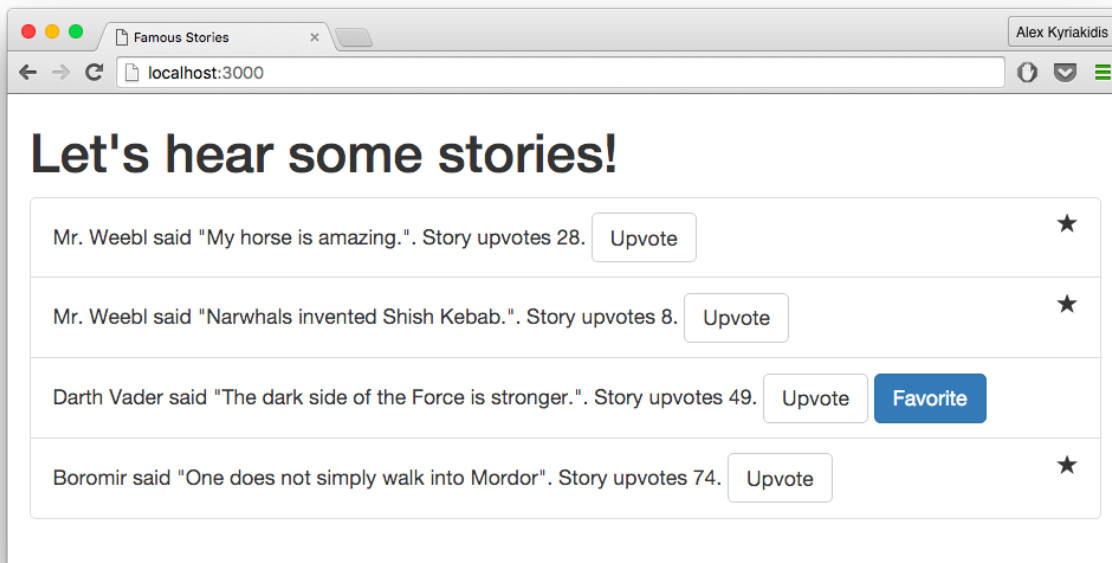
```

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.

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.

```
<ul class="list-group">
  <story v-for="story in stories"
        :story="story"
        :favorite="favorite">
  </story>
</ul>
```

```
Vue.component('story', {
  ...
  props: ['story', 'favorite'],
  ...
});
```



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. When the parent property updates, it will flow down to the child, but not the other way around.

We can't synchronize child's data with parent's, with what we know so far. So, we'll take a break and study Vue's **Custom Events**, before we go any further.



Code Examples

You can find the code examples of this chapter on [GitHub](https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter7)¹.

¹<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter7>

7.7 Homework

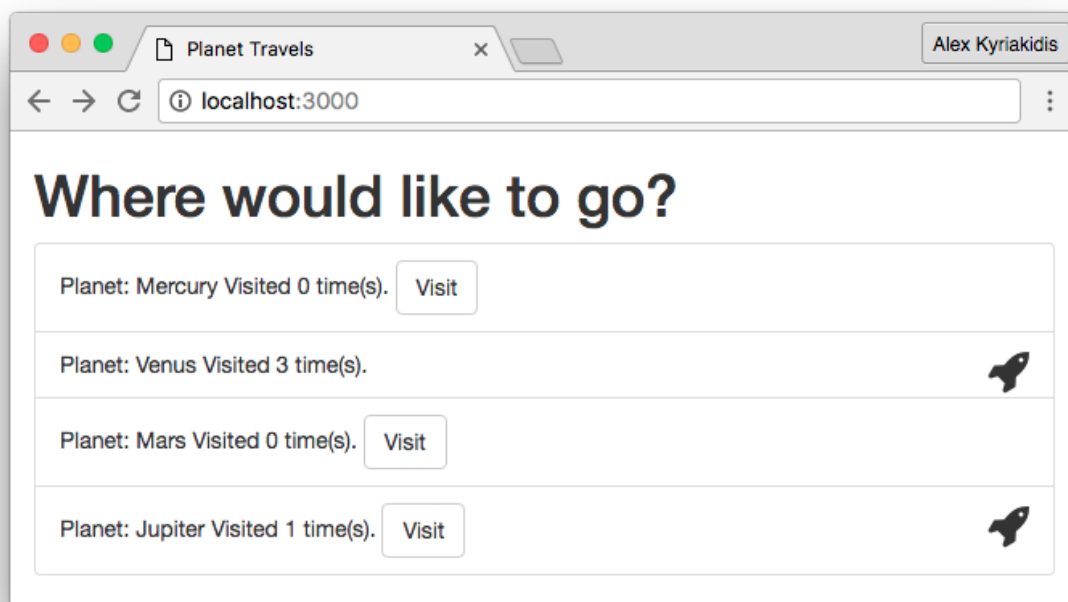
Create an array of planets. Each planet must have a name and a number of visits.

You can choose to travel to any planet, but you are limited to 3 visits per planet due to shortage of fuel.

You should have a *Planet* component with the appropriate methods/computed properties.

When rendered, each planet should display:

- its name
- the number of visits
- a *Visit* button (if max number of visits has not been reached)
- an icon to indicate if planet has been visited at least once



Example Output



Potential Solution

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

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

8. Custom Events

Some times it is needed to fire a custom event. To do so, we can use Vue Instance methods. Every Vue instance implements the [Events interface](#)¹.

This means that it can:

- Listen to an event using `$on(event)`.
- Trigger an event using `$emit(event)`.

It can also:

- Listen to an event, but only once, using `$once(event)`.
- Remove event listeners using `$off()`.

8.1 Emit and Listen

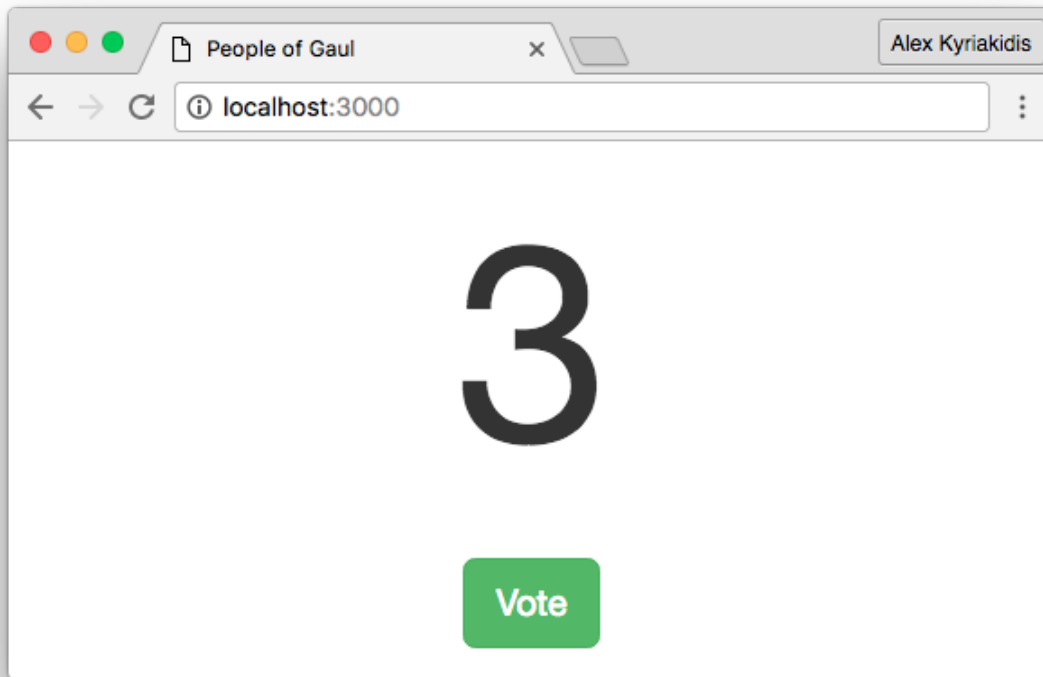
Let's start with a dead simple example.

The following codeblock represents a page with a counter and a *Vote* button. When the button is clicked, it emits an event, named 'voted'. There is also an *Event Listener* for the event, which increases the number of votes when the event is triggered.

```
1 <html>
2 <head>
3   <title>Emit and Listen</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 text-center">
9     <p style="font-size: 140px;">
10       {{ votes }}
11     </p>
12     <button class="btn btn-primary" @click="vote">Vote</button>
13   </div>
```

¹<http://vuejs.org/api/#Instance-Methods-Events>

```
14 </body>
15 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js"></script>
16 <script type="text/javascript">
17 new Vue({
18   el: '.container',
19   data: {
20     votes: 0
21   },
22   methods:
23   {
24     vote: function (writer) {
25       this.$emit('voted')
26     },
27   },
28   created () {
29     this.$on('voted', function(button) {
30       this.votes++
31     })
32   }
33 })
34 </script>
35 </html>
```

Example Output

We register the event listener within the **created** *Lifecycle Hook*. **this** is bound to the Vue Instance within **vote** method and **created** hook. So, we can access **\$on** and **\$emit** functions using **this.\$on** and **this.\$emit**.

8.1.1 Lifecycle Hooks

Lifecycle Hooks are functions who execute when Vue related events happen.

In Vue 2, these hooks are:

Hook	Called
beforeCreate	After the instance has just been initialized, before data observation and event/watcher setup.
created	After the instance is created.
beforeMount	Right before the mounting begins.
mounted	After the instance has just been mounted to the DOM.
beforeUpdate	When the data changes, before the virtual DOM is re-rendered and patched.
updated	After a data change causes the virtual DOM to be re-rendered and patched.
activated	When a kept-alive component is activated.
deactivated	When a kept-alive component is deactivated.
beforeDestroy	Right before a Vue instance is destroyed.
destroyed	After a Vue instance has been destroyed.

You don't have to know all these, but it is good to be aware of their existence. If you want to learn more about Lifecycle Hooks check [Vue's API²](http://vuejs.org/api/#Options-Lifecycle-Hooks).

8.2 Parent-Child Communication

Things are getting a bit different when a parent component needs to listen to an event of a *child* component. We can't use `this.$on`/`this.$emit` since `this` will be bound to different instances.

Remember the `v-on` (`@`) [event listener](#)? A parent component can listen to the events emitted from a child component using `v-on` directly in the template, where the child component is used.

Following the previous example, I'll create a food component which will have a *name* property. In its template, it will show a button displaying its *name*. When the button is clicked, we want to emit the *vote* event.

Food Component

```
Vue.component('food', {  
  template: '#food',  
  props: ['name'],  
  methods: {  
    vote: function () {  
      this.$emit('voted')  
    }  
  },  
})
```

²<http://vuejs.org/api/#Options-Lifecycle-Hooks>

Food Component's Template

```
<template id="food">
  <button class="btn btn-default" @click="vote">{{ name }}</button>
</template>
```

In the parent instance, the `<button>` will be replaced by `<food @voted="countVote"></food>`.

`@voted="countVote"` means that when child's `voted` event is emitted, the `countVote` method will be executed. We can also get rid of the `this.$on` listener, since we don't need it any more.

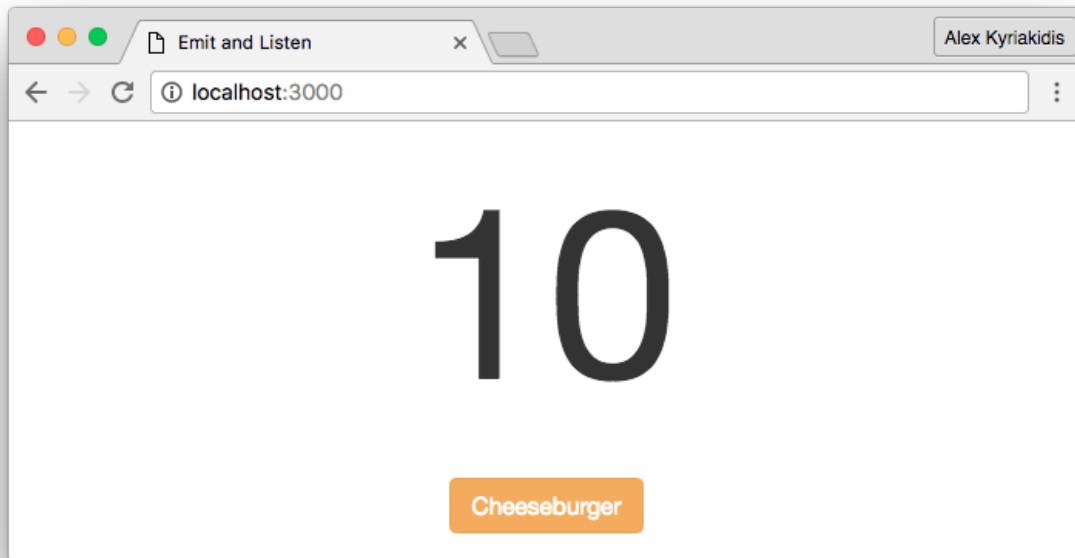
Parent Component

```
new Vue({
  el: '.container',
  data: {
    votes: 0
  },
  methods:
  {
    countVote: function () {
      this.votes++
    },
  }
})
```

Parent Component's template

```
<div class="container text-center">
  <p style="font-size: 140px;">
    {{ votes }}
  </p>
  <food @voted="countVote" name="Cheeseburger"></food>
</div>
```

If you run this in your browser, you will see that the output is the same.



Parent-Child Communication

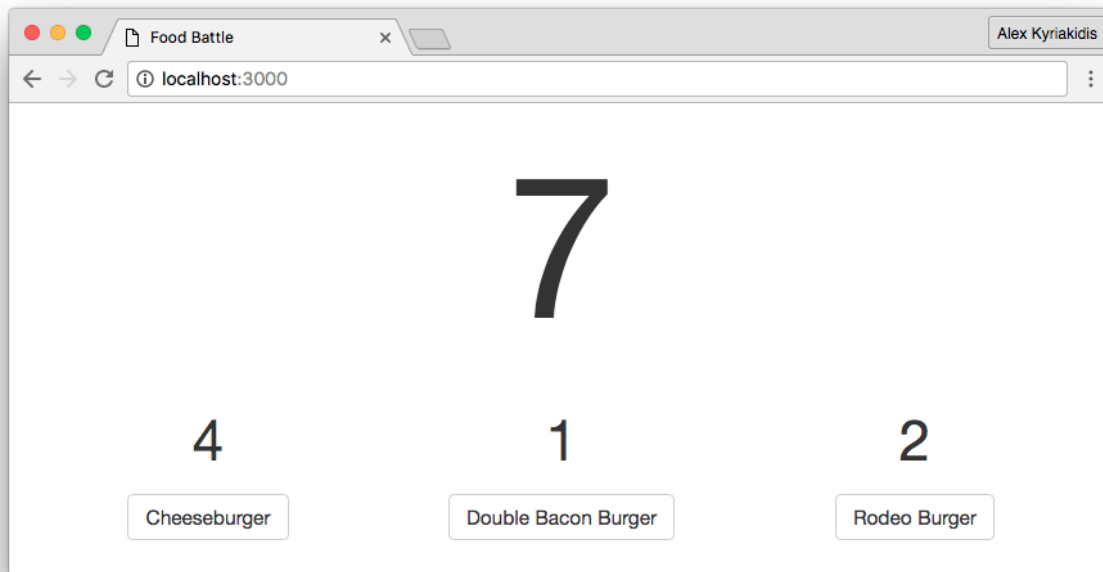
8.3 Passing Arguments

Let's create 3 instances of the *food component*. Each instance will have its own number of votes. When any of the foods gets voted, it will increase its own votes **and** it will emit an event to update the total votes, located in the parent component, as well.

```
1 <html>
2 <head>
3   <title>Food Battle</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 text-center">
9     <p style="font-size: 140px;">
10       {{ votes }}
11     </p>
12
13     <div class="row">
14       <food @voted="countVote" name="Cheeseburger"></food>
```

```
15     <food @voted="countVote" name="Double Bacon Burger"></food>
16     <food @voted="countVote" name="Rodeo Burger"></food>
17 </div>
18 </div>
19
20 </body>
21 <template id="food">
22   <div class="text-center col-lg-4">
23     <p style="font-size: 40px;">
24       {{ votes }}
25     </p>
26     <button class="btn btn-default" @click="vote">{{ name }}</button>
27   </div>
28 </template>
29 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js"></script>
30 <script type="text/javascript">
31   var bus = new Vue()
32
33   Vue.component('food', {
34     template: '#food',
35     props: ['name'],
36     data: function () {
37       return {
38         votes: 0
39       }
40     },
41     methods: {
42       vote: function () {
43         this.votes++
44         this.$emit('voted')
45       }
46     }
47   })
48   new Vue({
49     el: '.container',
50     data: {
51       votes: 0
52     },
53     methods:
54     {
55       countVote: function () {
56         this.votes++
```

```
57     }  
58   }  
59 })  
60 </script>  
61 </html>
```



Multiple Component Instances

Nothing new so far. To make the App more fancy, we can add a *Vote Log*. The log will get updated every time a food is voted. We have to update the child component, to pass the food's name when emitting the voted event.



Info

The `$emit` function, along with the event name argument, it passes any additional arguments to listener's callback function. For example: `vm.$emit('voted', 'Alex', 'Sunday', 'Bob Ross')`

We have two options to access food's name. One is obviously from component's name property. The second one, is to access the element which triggered the event and find its text content. We'll go with the second one.

We can log the event variable to the console, within **Food**'s **vote** method, to find out how we can access the clicked element.

Food Component

```
Vue.component('food', {  
  ...  
  methods: {  
    vote: function (event) {  
      console.log(event)  
      this.votes++  
      this.$emit('voted')  
    }  
  }  
})
```



`event.srcElement`

If you are following along, you will see that we have access to the clicked element within `event.srcElement` attribute. The name can be found in both `event.srcElement.outerText` and `event.srcElement.textContent`.

So, lets pass one of these to the `$emit` function.

Food Component

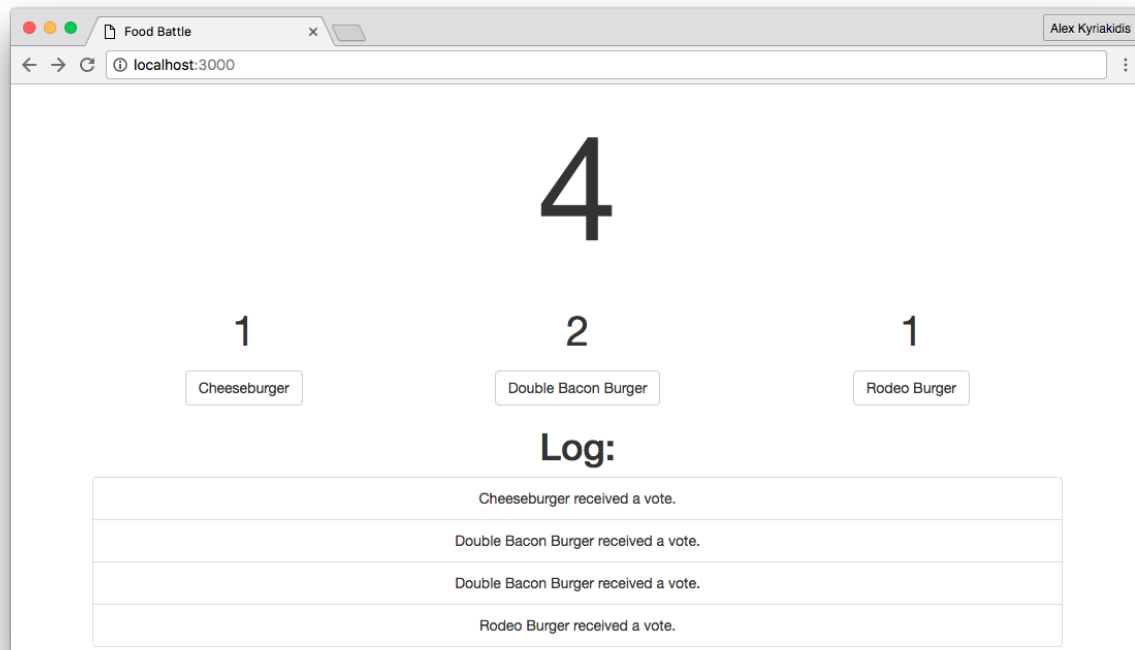
```
Vue.component('food', {
  ...
  methods: {
    vote: function () {
      this.votes++
      this.$emit('voted', event.srcElement.textContent)
    }
  }
})
```

Within the parent, we will push the incoming vote to a log array.

```
new Vue({
  el: '.container',
  data: {
    votes: 0,
    log: []
  },
  methods: {
    {
      countVote: function (food) {
        this.votes++
        this.log.push(food + ' received a vote.')
      }
    }
  }
})
```

To display the log, we can add a list to the HTML template.

```
<h1>Log:</h1>
<ul class="list-group">
  <li class="list-group-item" v-for="vote in log"> {{ vote }} </li>
</ul>
```

Votes Log

Pretty easy, huh?

8.4 Non Parent-Child Communication

We'll take the previous example a step further, by adding a Reset button. When it's clicked, it will reset all vote counters. As you can imagine, the reset button will emit an event that should be handled by all components. But how can we catch this event within the child components?

If we go with the `<food @voted="countVote">` way, we can listen for the `voted` event, but we don't have a way to emit events to the child components.

To make all components able to communicate with each other, we will use an **empty Vue instance** as a central event bus. Then, within the components **created** hook, we will register the event listeners using `bus.$on` instead of `this.$on`. Accordingly, we will use `bus.$emit` to fire all events.

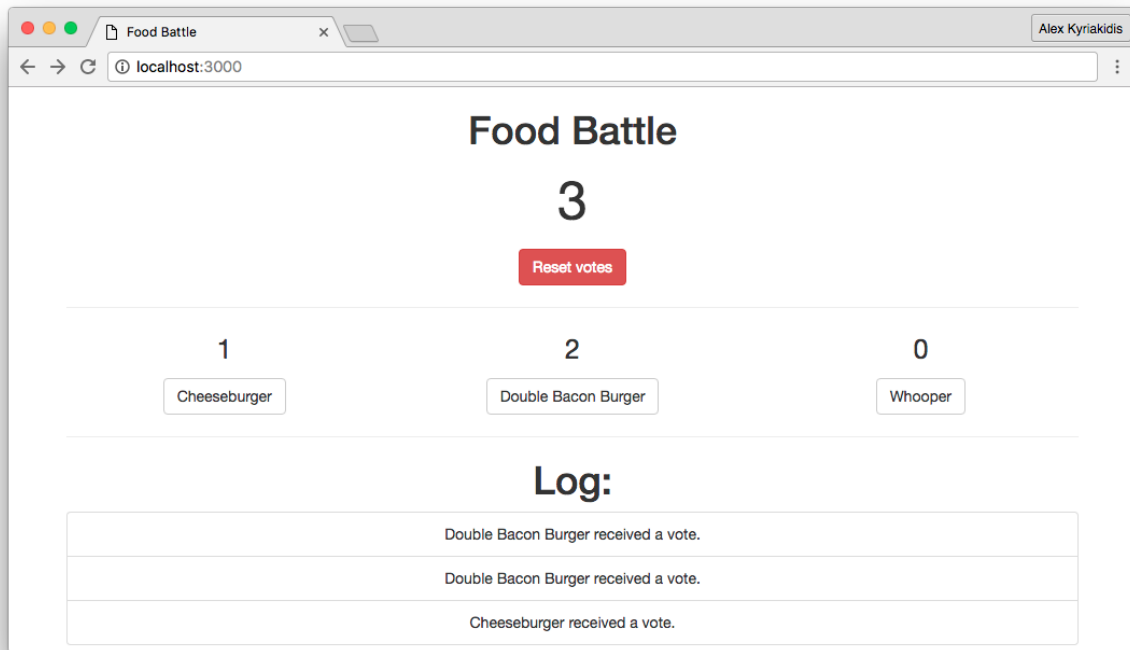
HTML

```
1 <body>
2   <div class="container text-center">
3     <h1>Food Battle</h1>
4     <p style="font-size: 140px;">
5       {{ votes.count }}
6     </p>
7     <button class="btn btn-danger" @click="reset">Reset votes</button>
8     <hr>
9
10    <div class="row">
11      <food name="Cheeseburger"></food>
12      <food name="Double Bacon Burger"></food>
13      <food name="Whooper"></food>
14    </div>
15    <hr>
16
17    <h1>Log:</h1>
18    <ul class="list-group">
19      <li class="list-group-item" v-for="vote in votes.log"> {{ vote }} </li>
20    </ul>
21  </div>
22 </body>
```

JavaScript

```
1 var bus = new Vue()
2
3 Vue.component('food', {
4   template: '#food',
5   props: ['name'],
6   data: function () {
7     return {
8       votes: 0
9     }
10  },
11  methods: {
12    vote: function (event) {
13      // instead of using this.name
14      // we can access event's element's text
15      var food = event.srcElement.textContent;
```

```
16         this.votes++
17         bus.$emit('voted', food)
18     },
19     reset: function () {
20         this.votes = 0
21     }
22 },
23 created () {
24     bus.$on('reset', this.reset)
25 }
26 })
27 new Vue({
28   el: '.container',
29   data: {
30     votes: {
31       count: 0,
32       log: []
33     }
34   },
35   methods:
36   {
37     countVote: function (food) {
38       this.votes.count++
39       this.votes.log.push(food + ' received a vote.')
40     },
41     reset: function () {
42       this.votes = {
43         count: 0,
44         log: []
45       }
46       bus.$emit('reset')
47     }
48   },
49   created () {
50     bus.$on('voted', this.countVote)
51   }
52 })
```



Non Parent-Child Communication



Warning

Notice here that we are using:

```
bus.$on('voted', this.countVote)
```

If instead we were trying to use it like:

```
bus.$on('voted', function(){
  this.vote(food)
})
```

it would have thrown an error, since **this** would be bounded to the *bus instance* instead of the current component's instance.

8.5 Removing Event Listeners

To remove one or more event listeners, we can use **\$off**. The **\$off([event, callback])** method can be used in several ways.

1. `$off()`, with no arguments, removes all event listeners.
2. `$off([event])`, removes all event listeners for the specified event.
3. `$off([event, callback])` removes event's listener for the specific callback.

To see it in action, we'll add a *Stop* button to stop the votes from being counted/logged. We'll place a `stop` method in the Vue instance.

```
new Vue({
  ...
  methods:
  {
    ...
    stop: function () {
      bus.$off(['voted'])
    }
  }
})
```

If we go with `bus.$off(['voted'])`, you see that after *Stop button* is clicked, the upcoming votes are not added to the total count. Also, they don't show up in the log. Though, if you hit the *Reset Votes* button, the Food components' votes are being reseted to 0.

To disable the *reset listener* too, we can remove all event listeners using `bus.$off()`.

8.6 Back to stories

Remember the [Stories example](#) of the previous chapters? We were about to synchronize *component's data* with *parent's data*. The solution seems obvious now.

We'll use *Story* component like this:

```
<story v-for="story in stories"
:story="story"
:favorite="favorite"
@update="updateFavorite"></story>
```

Within *Story* component, we'll emit the `update` event when a story is marked as favorite. The story being favorite, should be passed as an argument on emit.

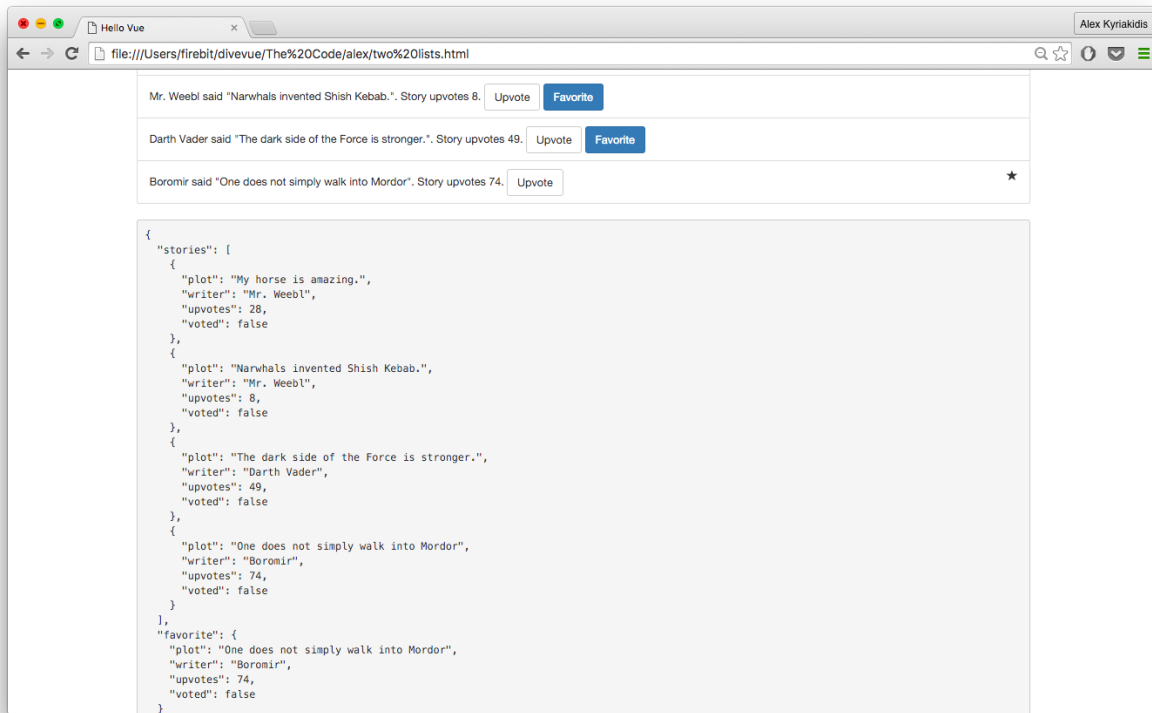
Story Component

```
Vue.component('story', {  
  ...  
  methods: {  
    ...  
    updateFavorite: function(){  
      // 'update' is just the name of the custom event  
      // it could be anything. ex: fav-update  
      this.$emit('update', this.story)  
    }  
  }  
  ...  
});
```

In the parent instance, we'll add a `favorite` variable to the data. Also, we'll create a new method, which will update `favorite` variable when called.

Parent Instance

```
new Vue({  
  ...  
  data: {  
    ...  
    favorite: {}  
  },  
  methods: {  
    updateFavorite: function(story) {  
      this.favorite = story;  
    }  
  },  
});
```



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.



Info

In Vue 2, bindings are **always** one-way. To keep data in sync between Parent-Child you have to use *Events*.



Code Examples

You can find the code examples of this chapter on [GitHub](https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter8)³.

³<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter8>

8.7 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 must also 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 its button says ‘Dismiss Horses’. I think you got the idea..



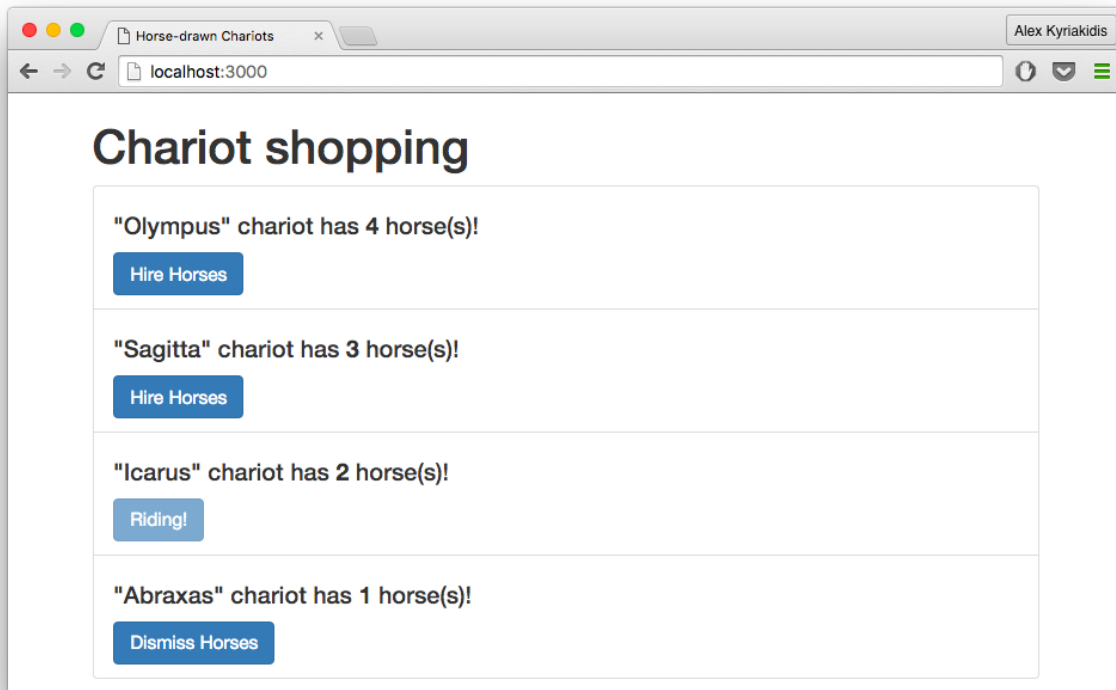
Hint

You need to keep in sync child’s and parent’s `currentChariot` property.



Hint

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



Example Output



Potential Solution

You can find a potential solution to this exercise [here](https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter8.html)⁴.

⁴<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter8.html>

9. Class and Style Bindings

9.1 Class binding

9.1.1 Object Syntax

A common need for data binding is to manipulate 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

```
{  
  'classA': true,  
  'classB': false,  
  'classC': true  
}
```

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

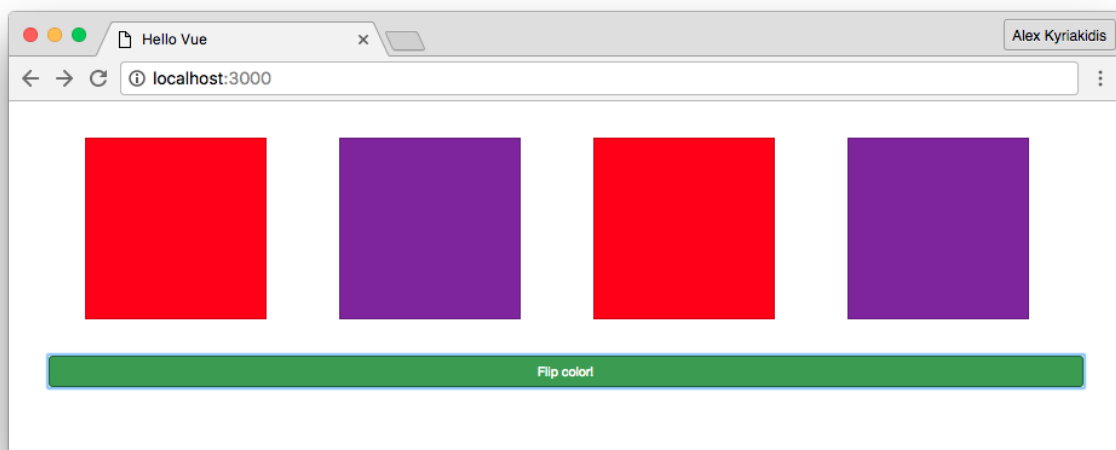
```
<div v-bind:class="elClasses"></div>
```

```
data: {  
  elClasses:  
    {  
      'classA': true,  
      'classB': false,  
      'classC': true  
    }  
}
```

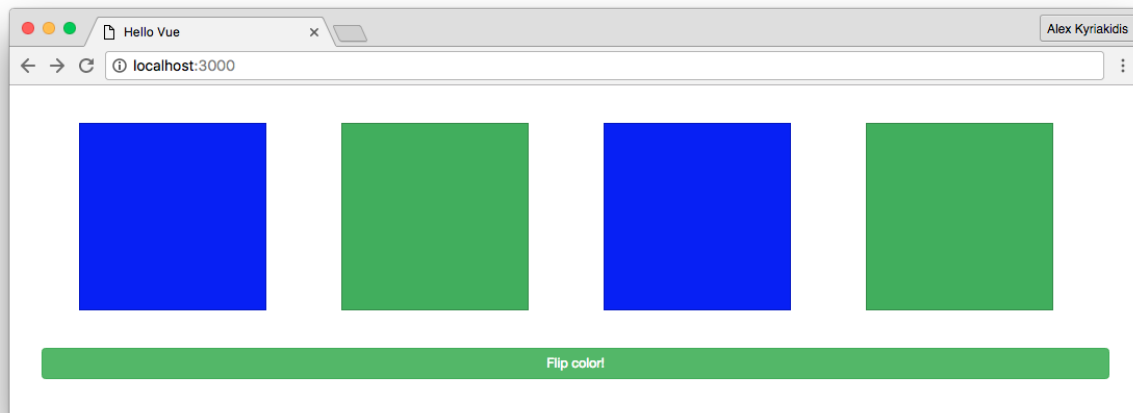
To demonstrate how `v-bind` is used with class attributes, we are going to make an example of class toggling. Using `v-bind:class` directive, we are going to dynamically toggle the class of `div` elements.

```
1 <html>
2 <head>
3   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.\
4 css" rel="stylesheet">
5   <title>Hello Vue</title>
6 </head>
7 <body>
8   <div class="container text-center">
9     <div class="box" v-bind:class="{ 'red' : color, 'blue' : !color }"></div>
10    <div class="box" v-bind:class="{ 'purple' : color, 'green' : !color }"></div>
11    <div class="box" v-bind:class="{ 'red' : color, 'blue' : !color }"></div>
12    <div class="box" v-bind:class="{ 'purple' : color, 'green' : !color }"></div>
13    <button v-on:click="flipColor" class="btn btn-block btn-success">
14      Flip color!
15    </button>
16  </div>
17 </body>
18 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.3.4/vue.js"></script>
19 <script type="text/javascript">
20 new Vue({
21   el: '.container',
22   data: {
23     color: true
24   },
25   methods: {
26     flipColor: function() {
27       this.color = !this.color;
28     }
29   }
30 });
31 </script>
32 <style type="text/css">
33 .red {
34   background: #ff0000;
35 }
36 .blue {
37   background: #0000ff;
38 }
39 .purple {
40   background: #7B1FA2;
41 }
42 .green {
```

```
43   background: #4CAF50;  
44 }  
45 .box {  
46   float: left;  
47   width: 200px;  
48   height: 200px;  
49   margin: 40px;  
50   border: 1px solid rgba(0, 0, 0, .2);  
51 }  
52 </style>  
53 </html>
```



Toggle boxes' color



Toggle boxes' color

We have applied a class of **box** to each **div**, for our convenience. What this code actually does, is “flipping” the color of the boxes with a hit of the button. When pressing it, 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 from ‘red’ to ‘blue’ or from ‘purple’ to ‘green’ conditionally, depending on the truthfulness of *color* value. That given, the style is going to apply on each class and give us the desired output.



Info

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

So, in our example, **divs** always have the **box** class and conditionally one of **red**, **blue**, **purple** or **green**.

9.1.2 Array Syntax

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

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

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

```
<div v-bind:class="['classA', condition ? 'classsB' : '']"></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 <div class="container text-center">
2   <div class="box" v-bind:class="[ color ? 'red' : 'blue' ]"></div>
3   <div class="box" v-bind:class="[ color ? 'purple' : 'green' ]"></div>
4   <div class="box" v-bind:class="[ color ? 'red' : 'blue' ]"></div>
5   <div class="box" v-bind:class="[ color ? 'purple' : 'green' ]"></div>
6   <button v-on:click="flipColor" class="btn btn-block btn-success">
7     Flip color!
8   </button>
9 </div>

```

```

1 new Vue({
2   el: '.container',
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']"`

9.2 Style binding

9.2.1 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 that 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.

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

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

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



Style object binding

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

9.2.2 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 `font-size` of `niceStyle` and the font style 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     fontStyle: 'italic'
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 the appropriate prefixes to the applied styles.

You can find more information about vendor prefixes [here](https://developer.mozilla.org/en-US/docs/Glossary/Vendor_Prefix)¹.

9.3 Bindings in Action

```

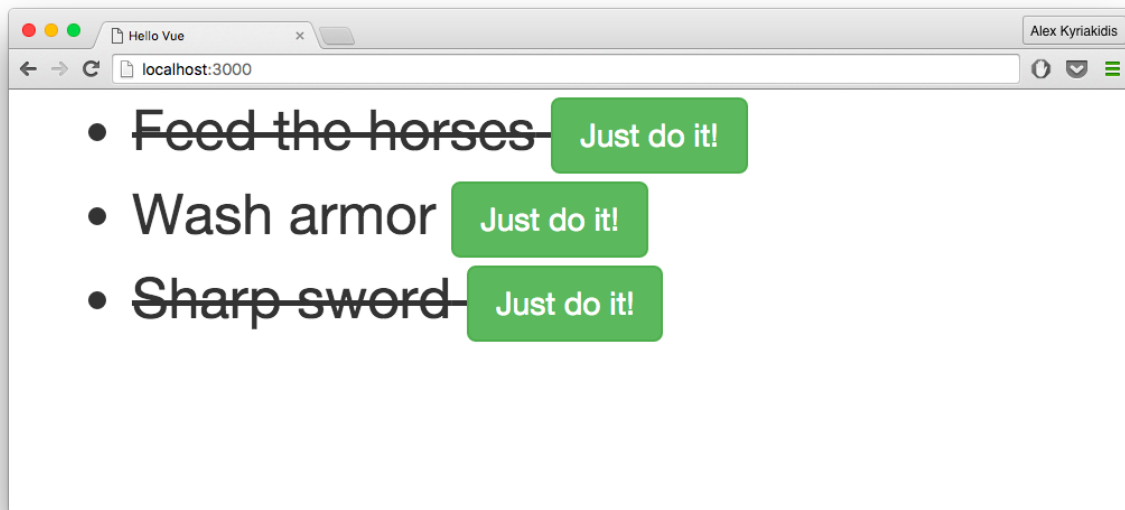
1  <html>
2  <head>
3  <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.cs\
4  s" rel="stylesheet">
5  <title>Hello Vue</title>
6  </head>
7  <body class="container-fluid">
8    <div id="app">
9      <ul>
10         <li :class="{ 'completed' : task.done}"
11             :style="styleObject"
12             v-for="task in tasks">
13             {{task.body}}
14             <button @click="completeTask(task)" class="btn">
15               Just do it!
16             </button>
17         </li>
18     </ul>
19 </div>
20 </body>
21 <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/vue/2\
22 .3.4/vue.js"></script>
23 <script type="text/javascript">
24 new Vue({
25   el: '#app',
26   data: {
27     tasks: [
28       {body: "Feed the horses", done: true},
29       {body: "Wash armor", done: true},
30       {body: "Sharp sword", done: false},

```

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

```
31      ],
32      styleObject: {
33        fontSize: '25px'
34      }
35    },
36    methods: {
37      completeTask: function(task) {
38        task.done = !task.done;
39      }
40    },
41  });
42 </script>
43 <style type="text/css">
44   .completed {
45     text-decoration: line-through;
46   }
47 </style>
48 </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 the task gains 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 **font-size** of all tasks. As you can see, the **completeTasks** method takes in the parameter **task**.



Styling completed tasks



Code Examples

You can find the code examples of this chapter on [GitHub](https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter9)².

²<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter9>

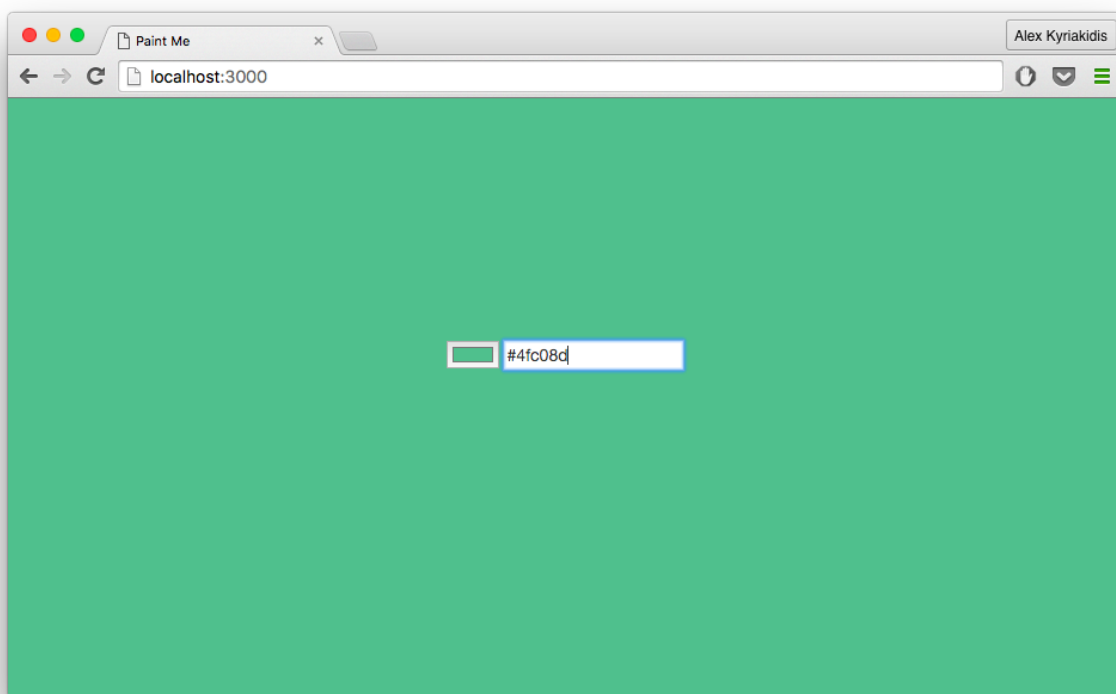
9.4 Homework

A fun and maybe tricky exercise for this chapter. Create an input where the user can choose a color. When a color is chosen, apply it to an element of your choice. That's it, **let's paint!!** :)



Hint

You could use `input type="color"` for your ease (supported in most browsers).



Example Output



Potential Solution

You can find a potential solution to this exercise [here](https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter9.html)³.

³<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter9.html>

II Consuming an API

10. Preface

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

Following the *story* examples of previous chapters, we are now going to 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.

10.1 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).

10.2 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 you to use the *demo API* that we have built exclusively for the examples of this book.

¹<https://laravel.com/>

10.2.1 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 '~/themajestyofvuejs2')

```
>_ mkdir ~/themajestyofvuejs2
```

1. Download the source code from github

```
>_ cd ~/themajestyofvuejs2  
git clone https://github.com/hootlex/the-majesty-of-vuejs-2 .
```

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

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

```
>_ cd ~/themajestyofvuejs2/apis/stories
```

1. Run the installation script

```
>_ sh setup.sh
```

1. 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.

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

Installation Script: setup.sh

```
# navigate to chapter directory
$ cd ~/themajestyofvuejs2/apis/stories

# install dependencies
$ composer install

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

# Migrate & Seed
$ php artisan migrate;
$ 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
id	Integer, Auto Increment
plot	String
writer	String
upvotes	Integer, Unsigned

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

10.2.2 API Endpoints

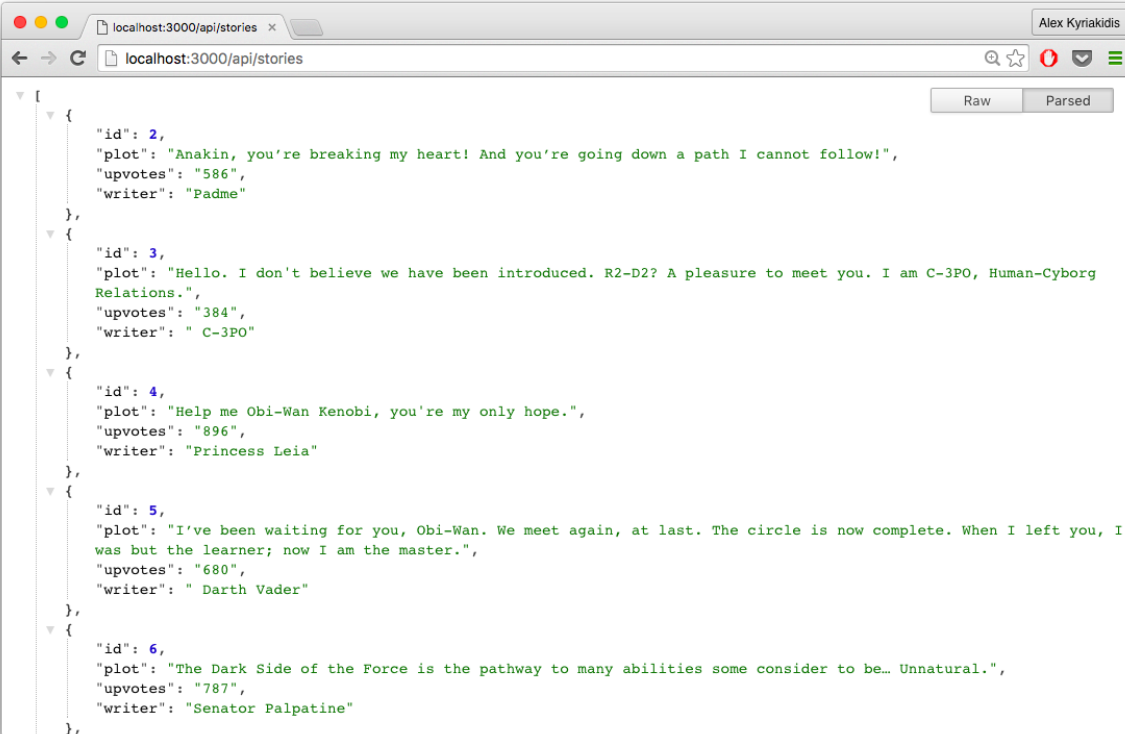
An endpoint is simply a URL. When you go to *http://example.com/foo/bar* it is an endpoint and you simply need to call it */foo/bar* because the domain will be the same for all the endpoints.

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

HTTP Method	URI	Action
GET/HEAD	api/stories	<i>Fetches</i> all stories
GET/HEAD	api/stories/{id}	<i>Fetches</i> specified story
POST	api/stories	<i>Creates</i> a new story
PUT/PATCH	api/stories/{id}	<i>Updates</i> an existing story
DELETE	api/stories/{id}	<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 update, 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 painful. 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](https://chrome.google.com/webstore/detail/json-formatter/bcjindcccaagfpapjjmafapmmgkkhgoa) 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/bcjindcccaagfpapjjmafapmmgkkhgoa>

11. Working with real data

It is time to actually put in 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 come 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.

11.1 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

```
new Vue({
  data: {
    stories: [
      {
        plot: 'My horse is amazing.',
        writer: 'Mr. Weebl',
      },
      {
        plot: 'Narwhals invented Shish Kebab.',
        writer: 'Mr. Weebl',
      },
      ...
    ]
  }
})
```

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 handling responses.

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

```
$.get(  
  url,  
  success  
);
```

which is actually a shorthand for

```
$.ajax({  
  url: url,  
  success: success  
});
```

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

There is a common catch here, we have to make the call **after the Instance is ready**. Do you remember [Vue's Lifecycle Hooks](#)?

There is a *hook*, called **mounted**, which is called just after the instance has been mounted.



Warning

The **mounted** hook is not equivalent to jQuery's `$(document).ready()`. When using **mounted**, **there's no guarantee to be in-document**. If you need to execute something once the page Document Object Model (DOM) is ready, you can use:

```
mounted: function () {  
  this.$nextTick(function () {  
    // code that assumes this.$el is in-document  
  })  
}
```

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

²<https://api.jquery.com/jquery.get/>

Lets see this in action.

```

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 }}</pre>
9    </div>
10 </div>
11 <template id="template-story-row">
12   <li class="list-group-item">
13     {{ story.writer }} said "{{ story.plot }}"
14     <span>{{ story.upvotes }}</span>
15   </li>
16 </template>

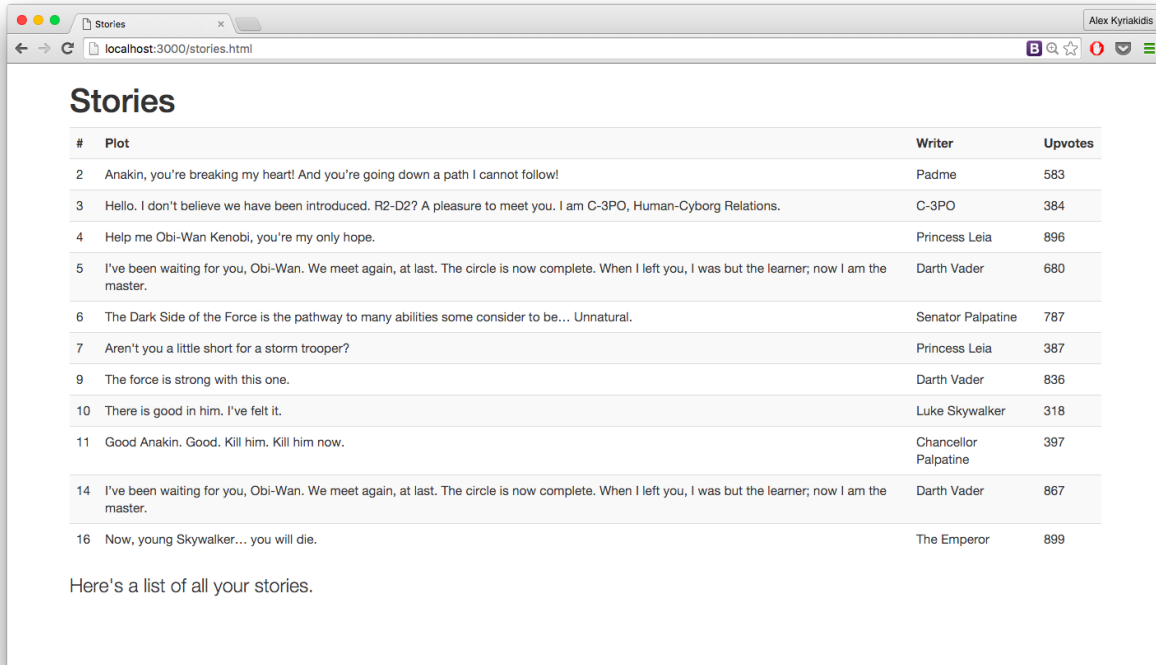
1  <script src="https://unpkg.com/vue@2.3.2/dist/vue.js"></script>
2  <script src="https://code.jquery.com/jquery-2.1.4.min.js"></script>
3  <script type="text/javascript">
4    Vue.component('story', {
5      template: "#template-story-row",
6      props: ['story'],
7    });
8
9    var vm = new Vue({
10      el: '#app',
11      data: {
12        stories: []
13      },
14      mounted: 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, within mounted hook, we perform the GET

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

request. After the request is successfully finished, we store the response data (inside the callback) into stories array.



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` is not bound to the `Vue` instance inside the callback. So, we save 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: [documentation](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this)⁴.



Warning

In order to be able to fetch the data from an API, you have to open the file on the browser behind the server that provides the API. If you are using the provided API you should open the file on `http://localhost:3000/stories.html`. In case you've made your own, make sure you host the `stories.html` under a route of your app. If you open the `stories.html` file directly on the browser it won't work.

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

11.2 Refactoring

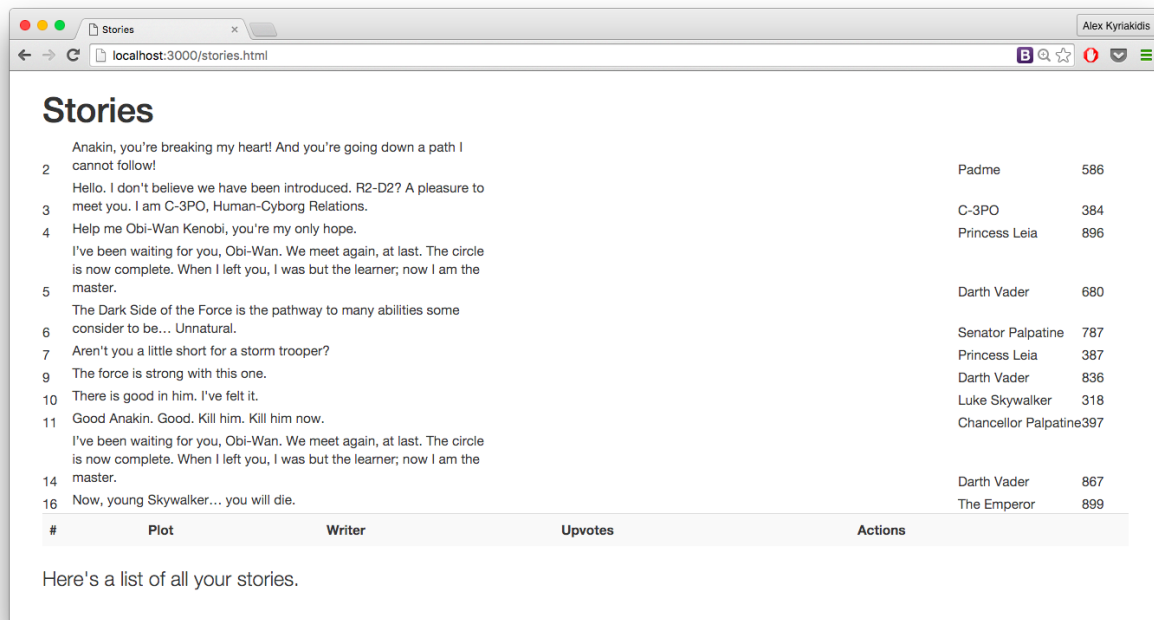
Having large amounts of code in our text editor can be confusing if not displayed properly, as well as in the browser. For that reason, we are going to refactor our example code, to render the list of stories using a `<table>` element instead of the ``.

```

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         <tr v-for="story in stories" is="story" :story="story"></tr>
11     </table>
12 </div>
13 <template id="template-story-row">
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 }}</pre>

```

But there is an issue.



Rendering issues

Our table does not render properly, [but why?](#)⁵

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`.

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

So our example will become

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

⁵<http://goo.gl/Xr9RoQ>

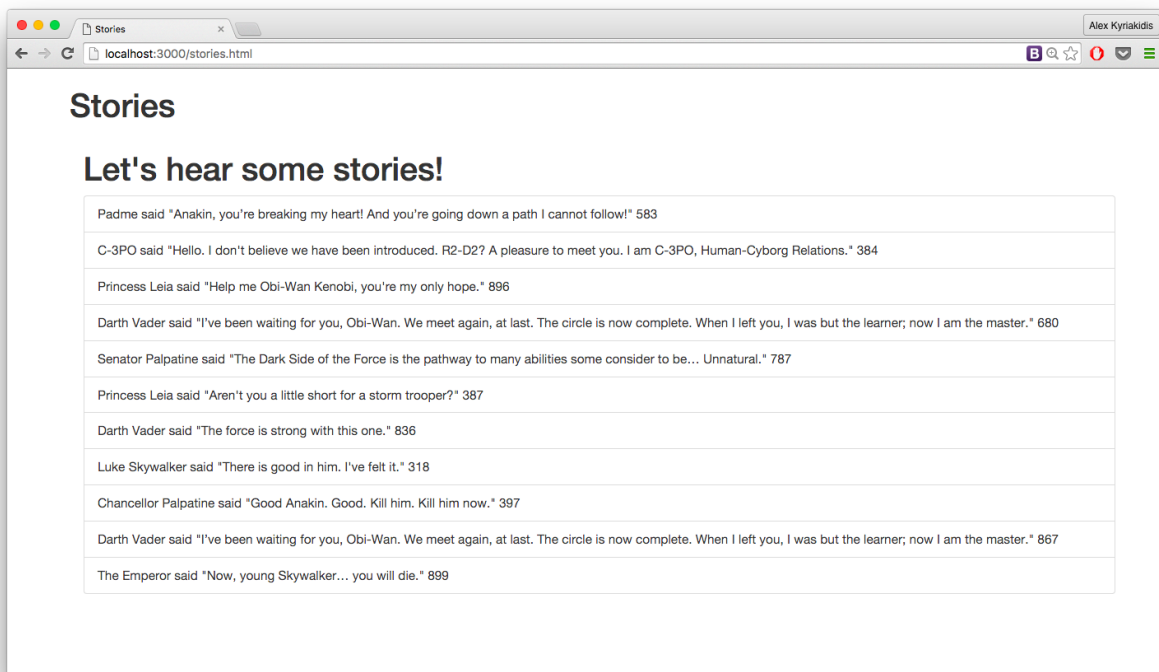


Table renders properly

Well, this looks better!

11.3 Update Data

We used to have a function that allowed the user to vote any story he wanted to. But now we want something more. We want the server to be informed every time a story is voted, ensuring that story's votes are updated in the database as well.

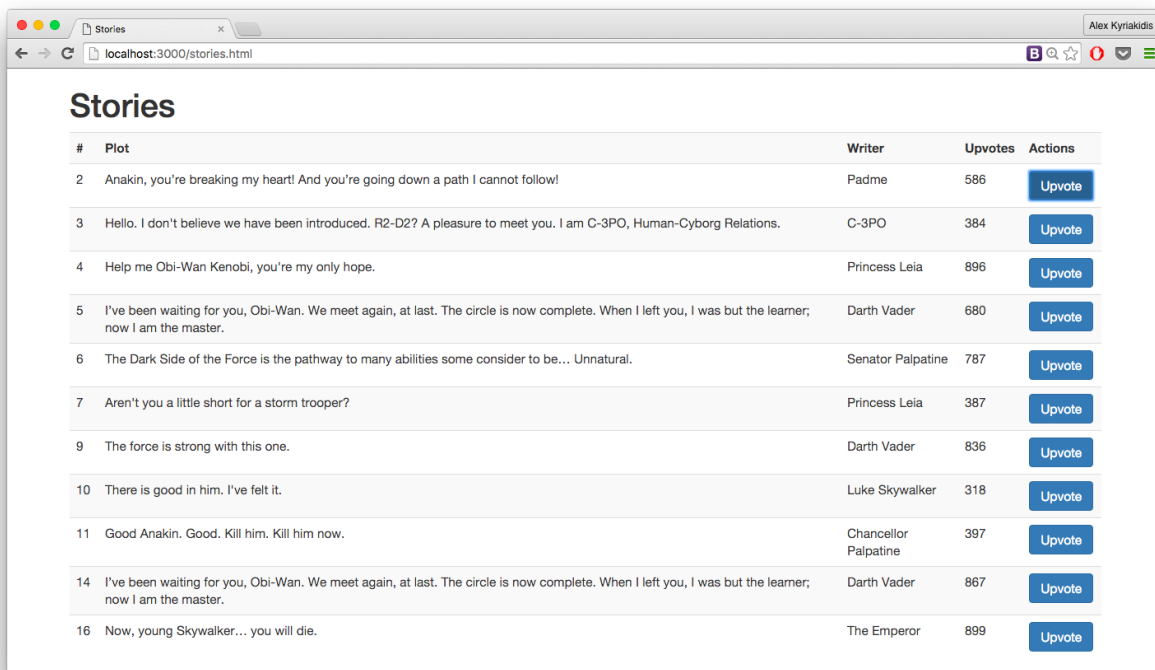
To update an existing story, we have to make an HTTP PATCH or PUT request to `api/stories/{storyID}`.

Inside the `upvoteStory` function, which is to be created, we are going to make an HTTP call after we have increased `story's 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	<button>Upvote</button>
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	<button>Upvote</button>
4	Help me Obi-Wan Kenobi, you're my only hope.	Princess Leia	896	<button>Upvote</button>
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	<button>Upvote</button>
6	The Dark Side of the Force is the pathway to many abilities some consider to be... Unnatural.	Senator Palpatine	787	<button>Upvote</button>
7	Aren't you a little short for a storm trooper?	Princess Leia	387	<button>Upvote</button>
9	The force is strong with this one.	Darth Vader	836	<button>Upvote</button>
10	There is good in him. I've felt it.	Luke Skywalker	318	<button>Upvote</button>
11	Good Anakin. Good. Kill him. Kill him now.	Chancellor Palpatine	397	<button>Upvote</button>
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	<button>Upvote</button>
16	Now, young Skywalker... you will die.	The Emperor	899	<button>Upvote</button>

Upvote stories

11.4 Delete Data

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 have to search for it and remove it from **stories** array.

```

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>

```

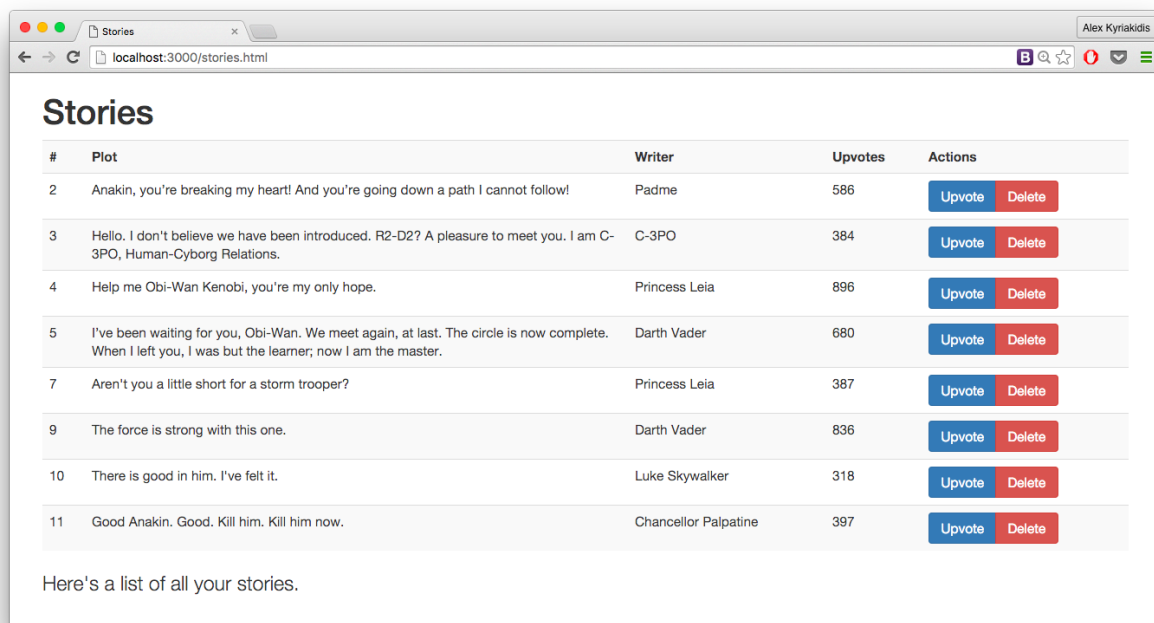
We append a *Delete* button to the *actions* column, bound to a method to delete the story. The `deleteStory` method will be:

```
1 Vue.component('story',{
2   ...
3   methods: {
4     ...
5     deleteStory: function(story){
6       // find story
7       var index = vm.stories.indexOf(story);
8
9       // delete it
10      vm.stories.splice(index, 1)
11    }
12  }
13  ...
14 })
```

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       // find story
7       var index = vm.stories.indexOf(story);
8
9       // delete it
10      vm.stories.splice(index, 1)
11
12      // make DELETE request
13      $.ajax({
14        url: '/api/stories/'+story.id,
15        type: 'DELETE'
16      });
17    },
18  }
19  ...
20 })
```

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**.

12. HTTP Clients

12.1 Introduction

As you may know, Vue used to have its own HTTP client. Though, the Vue.js team decided to retire it from official recommendation status. You can read Evan You's announcement, [Retiring vue-resource](https://medium.com/the-vue-point/retiring-vue-resource-871a82880af4#lew43e17f)¹, which explains the reasons behind this decision.

We are going to implement, again, all the web requests we made in the previous chapter, using [axios](https://github.com/mzabriskie/axios)² this time.

Before diving into axios, we'd like to show you one example of the vue-resource plugin. This way you can see the differences and decide which one works better for you. jQuery is also fine, but if you are using it only to perform AJAX calls you should consider removing it.

12.2 Vue-resource

Vue-resource provides services for making web requests and handles responses using an XMLHttpRequest or JSONP.

You can find the installation instructions and the documentation on [GitHub](https://github.com/vuejs/vue-resource)³. As usual, we are going to "pull it in" from a [cdn](https://cdnjs.com/libraries/vue-resource)⁴.

To fetch data from a server, we can use vue-resource's `$http` method with the following syntax:

```
// GET request
this.$http({url: '/someUrl', method: 'GET'})
  .then(function (response) {
    // success callback
  }, function (response) {
    // error callback
  });
```



Info

When using vue-resource, a Vue instance will provide 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.

¹<https://medium.com/the-vue-point/retiring-vue-resource-871a82880af4#lew43e17f>

²<https://github.com/mzabriskie/axios>

³<https://github.com/vuejs/vue-resource>

⁴<https://cdnjs.com/libraries/vue-resource>

Instead of passing the method option, there are shorthand methods available for all the requested types.

Request shorthands

```
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);
```

According to Evan's post, it's totally fine to keep using it if you are happy with it. We will use axios from now on, since it is the recommended one.

12.3 Axios

Axios is a promise based HTTP client for the browser and Node.js. In addition, it covers almost everything vue-resource provides with a very similar API, it is universal, supports cancellation, and has [TypeScript](http://www.typescriptlang.org/)⁵ definitions.

Here is how one can perform a GET request using axios:

```
// Make a request for a user with a given ID
axios({
  method: 'get',
  url: '/user/kostas'
})
.then(function (response) {
  console.log(response);
})
.catch(function (error) {
  console.log(error);
});
```

For convenience aliases have been provided for all supported request methods.

⁵<http://www.typescriptlang.org/>

Request method aliases

```
1 axios.request(config)
2 axios.get(url[, config])
3 axios.delete(url[, config])
4 axios.head(url[, config])
5 axios.post(url[, data[, config]])
6 axios.put(url[, data[, config]])
7 axios.patch(url[, data[, config]])
```



When using the alias methods, you don't need to specify the `url`, `method`, and `data` properties in the `config`.



Tip

If you'd like to use axios under `this.$http`, like in `vue-resource`, you can set `Vue.prototype.$http = axios`. This will be really handy if you want to replace `vue-resource` with axios.

12.4 Integrating axios

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

stories.html

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

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

app.js

```
mounted: function() {  
  // GET request  
  axios.get('/api/stories')  
    .then(function (response) {  
      Vue.set(vm, 'stories', response.data)  
      // Or we as we did before  
      // vm.stories = response.data  
    })  
}
```

Our list of stories comes without any issue.

Let's move on now, with the PATCH and DELETE requests, using the alias methods.

app.js

```
....  
upvoteStory: function (story) {  
  story.upvotes++;  
  axios.patch('/api/stories/' + story.id, story)  
}
```

app.js

```
deleteStory: function (story) {  
  var index = this.$parent.stories.indexOf(story);  
  this.$parent.stories.splice(index, 1)  
  axios.delete('/api/stories/' + story.id)  
}
```

We have replaced the AJAX methods with these ones, in no time!



Info

Since, the *story* component doesn't have access to the *stories* array, we access the array using **this.\$parent.stories**. We could also use **vm.stories** or emit an event to update the array within the parent Vue instance.

12.5 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 of a story, its writer, and also create new stories.**

12.5.1 Edit Stories

Let's start with the first task and give the user some inputs to manipulate the story's attributes. Two bound 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 **editing 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's 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's 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. Though, this approach won't work. It seems that the DOM isn't updating after setting `story.editing` to `true`. But why does this happen?

It turns out, according to [this post from Vue.js blog](http://vuejs.org/2016/02/06/common-gotchas/)⁶, 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.

⁶<http://vuejs.org/2016/02/06/common-gotchas/>

```

mounted: function() {
  var vm = this;

  // GET request
  axios.get('/api/stories')
    .then(function (response) {
      // set data on vm
      var storiesReady = response.data.map(function (story) {
        story.editing = false;
        return story
      })
      vm.stories = storiesReady
      //Vue.set(vm, 'stories', storiesReady)
    });
}

```



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](https://msdn.microsoft.com/en-us/library/ff679976(v=vs.94).aspx)⁷.

This function adds the **editing** attribute to 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 editing, we will give the user two options: to update the story with new values or to cancel the edit.

#	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			
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

Form inputs for story editing

So, let's move on and add two buttons, that should be displayed only when the user is editing a story. Additionally, a new method called `updateStory` will be created. It is going to update the current editing story, after the *Update Story* button is pressed.

⁷[https://msdn.microsoft.com/en-us/library/ff679976\(v=vs.94\).aspx](https://msdn.microsoft.com/en-us/library/ff679976(v=vs.94).aspx)

```

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

```

```

Vue.component('story', {
  ...
  methods: {
    ...
    updateStory: function (story) {
      axios.patch('/api/stories/' + story.id, story)
      //Set editing to false to show actions again and hide the inputs
      story.editing = false;
    },
  },
  ...
})

```

#	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` back to `false`, in order to hide the inputs and bring back the action buttons.

12.5.2 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 append it to the `stories` array using the `push()` javascript method. We will initialize all the story's attributes to `null`, except from `editing`. Since, we want to immediately manipulate the new story, the `editing` property 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 [here](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/push)⁸.

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`.

⁸https://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        axios.post('/api/stories/', story).then(function () {
7          story.editing = false;
8        });
9      },
10   }
11   ...
12 })

```

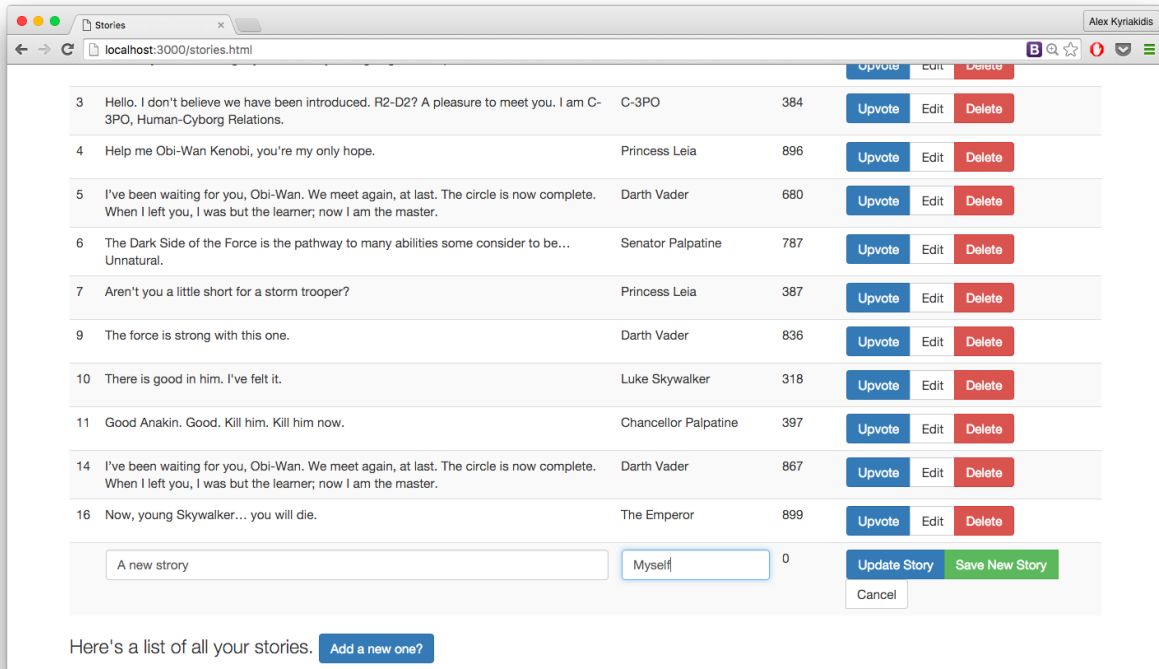
In the success callback function, we have set editing to **false** in order to show the *action* buttons again and hide the form's inputs and *editing* buttons. Below, we are going to update the button *groups*, in accordance to the new method.

```

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) we see that 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. On the other hand, 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 Story 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

```



```

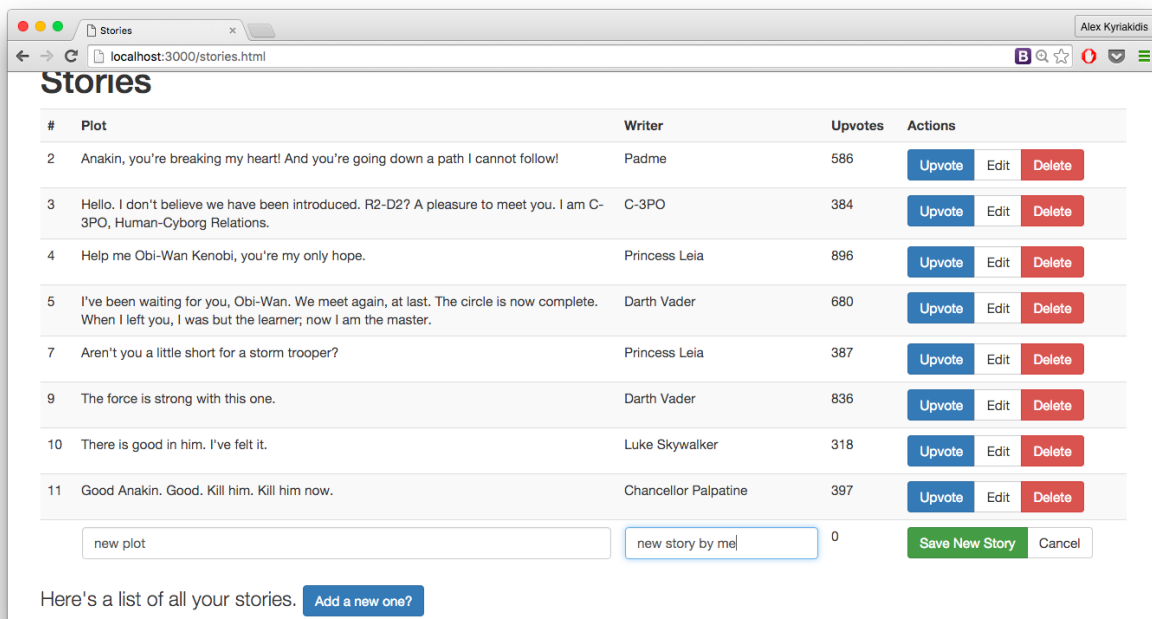
10     </button>
11     <!--always show cancel-->
12     <button @click="story.editing=false" class="btn btn-default">
13         Cancel
14     </button>
15 </div>

```



Tip

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



Adding new story

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

After finishing this part, testing our app brings up 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 again fetch the stories from the server, just after we store a new one into the database.

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    mounted: function(){
7      this.fetchStories()
8    },
9    methods: {
10     createStory: function(){
11       var newStory={
12         "plot": "",
13         "upvotes": 0,
14         "editing": true
15       };
16       this.stories.push(newStory);
17     },
18     fetchStories: function () {
19       var vm = this;
20       axios.get('/api/stories')
21         .then(function (response) {
22           var storiesReady = response.data.map(function (story) {
23             story.editing = false;
24             return story
25           })
26           // vm.stories = storiesReady
27           Vue.set(vm, 'stories', storiesReady)
28           // or: vm.stories = storiesReady
29         });
30     },
31   },
32 }
33 });
```

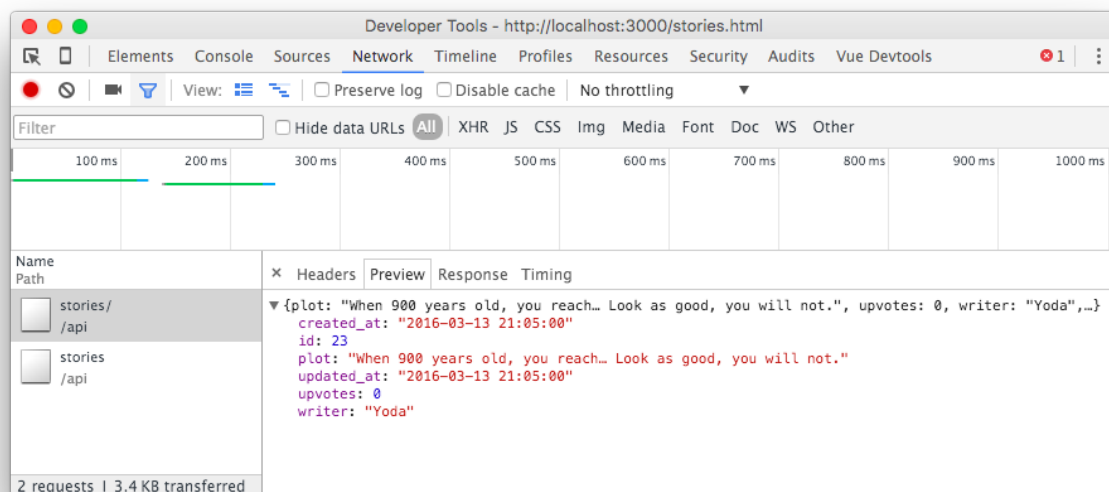
In our situation, we'll call `fetchStories()` inside the success callback of the POST request.

```
1 Vue.component('story',{
2   ...
3   methods: {
4     ...
5     storeStory: function(story){
6       axios.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.

12.5.3 Store & Update Unit

A better way to fix the previous issue, is to fetch only the newly created *story* from the database, instead of fetching and overwriting all the *stories*. If you check 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        axios.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 it is 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.

12.6 JavaScript File

As you may have noticed, our code is starting to become big. As our project grows, it is getting harder 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

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

and you are **ready to go!**

12.7 Source Code

Below is the whole source code of the previous *Managing Stories* example. If you have downloaded our repo, I suggest you to open your local files with your favorite text editor, because the code is quite big. The files are located at `~/themajestyofvuejs2/apis/stories/public`.

If you haven't downloaded the repository you can still view the [stories.html](https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/apis/stories/public/stories.html)⁹ and [app.js](https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/apis/stories/public/js/app.js)¹⁰ files on *github*.

stories.html

```

1 <html lang="en">
2 <head>
3   <title>Stories</title>
4   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.2\
5 /css/bootstrap.min.css">
6 </head>
7
8 <body>
9 <main>
10   <div class="container">
11     <h1>Stories</h1>
12     <div id="v-app">
13       <table class="table table-striped">
14         <tr>
15           <th>#</th>
16           <th>Plot</th>
17           <th>Writer</th>
18           <th>Upvotes</th>
19           <th>Actions</th>
20         </tr>
21         <tr v-for="story in stories" is="story" :story="story"></tr>
22       </table>
23       <p class="lead">Here's a list of all your stories.
24         <button @click="createStory()" class="btn btn-primary">
25           Add a new one?
26         </button>
27       </p>
28       <pre>{{ $data }}</pre>
29     </div>
30   </div>

```

⁹<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/apis/stories/public/stories.html>

¹⁰<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/apis/stories/public/js/app.js>

```
31 </main>
32 <template id="template-story-row">
33   <tr>
34     <td>
35       {{story.id}}
36     </td>
37     <td class="col-md-6">
38       <input v-if="story.editing"
39         v-model="story.plot"
40         class="form-control">
41       </input>
42       <!--in other occasions show the story plot-->
43       <span v-else>
44         {{story.plot}}
45       </span>
46     </td>
47     <td>
48       <input v-if="story.editing"
49         v-model="story.writer" class="form-control">
50       </input>
51       <!--in other occasions show the story writer-->
52       <span v-else>
53         {{story.writer}}
54       </span>
55     </td>
56     <td>
57       {{story.upvotes}}
58     </td>
59     <td>
60       <div class="btn-group" v-if="!story.editing">
61         <button @click="upvoteStory(story)"
62           class="btn btn-primary">
63           Upvote
64         </button>
65         <button @click="editStory(story)" class="btn btn-default">
66           Edit
67         </button>
68         <button @click="deleteStory(story)"
69           class="btn btn-danger">
70           Delete
71         </button>
72       </div>
```

```

73     <div class="btn-group" v-else>
74         <!--If the story is taken from the db then it will have an id-->
75         <button v-if="story.id"
76             class="btn btn-primary"
77             @click="updateStory(story)">
78             Update Story
79         </button>
80
81         <!--If the story is new we want to store it-->
82         <button v-else class="btn btn-success"
83             @click="storeStory(story)">
84             Save New Story
85         </button>
86
87         <!--Always show cancel-->
88         <button @click="story.editing=false"
89             class="btn btn-default">
90             Cancel
91         </button>
92     </div>
93 </td>
94 </tr>
95 </template>
96 <script src="https://unpkg.com/vue@2.3.2/dist/vue.js"></script>
97 <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
98 <script src='/js/app.js' type="text/javascript"></script>
99 </body>
100 </html>

```

app.js

```

1  Vue.component('story', {
2      template: '#template-story-raw',
3      props: ['story'],
4      methods: {
5          deleteStory: function (story) {
6              var index = this.$parent.stories.indexOf(story);
7              this.$parent.stories.splice(index, 1)
8              axios.delete('/api/stories/' + story.id)
9          },
10         upvoteStory: function (story) {
11             story.upvotes++;

```

```
12         axios.patch('/api/stories/' + story.id, story)
13     },
14     editStory: function (story) {
15         story.editing = true;
16     },
17     updateStory: function (story) {
18         axios.patch('/api/stories/' + story.id, story)
19         //Set editing to false to show actions again and hide the inputs
20         story.editing = false;
21     },
22     storeStory: function (story) {
23         axios.post('/api/stories/', story).then(function (response) {
24             //After the the new story is stored in the database fetch again \
25 all stories with
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
36 // Vue.prototype.$http = axios
37
38 new Vue({
39     el: '#v-app',
40     data: {
41         stories: [],
42         story: {}
43     },
44     mounted: function () {
45         this.fetchStories()
46     },
47     methods: {
48         createStory: function () {
49             var newStory = {
50                 plot: "",
51                 upvotes: 0,
52                 editing: true
53             };
```



```
54         this.stories.push(newStory);
55     },
56     fetchStories: function () {
57         var vm = this;
58         axios.get('/api/stories')
59             .then(function (response) {
60                 // set data on vm
61                 var storiesReady = response.data.map(function (story) {
62                     story.editing = false;
63                     return story
64                 })
65                 // vm.stories = storiesReady
66                 Vue.set(vm, 'stories', storiesReady)
67             });
68     },
69 }
70 });
```



Code Examples

You can find the code examples of this chapter on [GitHub](#)¹¹

12.8 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.

12.8.1 Preface

If you have followed the instructions from [Chapter 10](#), 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.

¹¹<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter12>

12.8.2 API Endpoints

The API Endpoints you are going to need are:

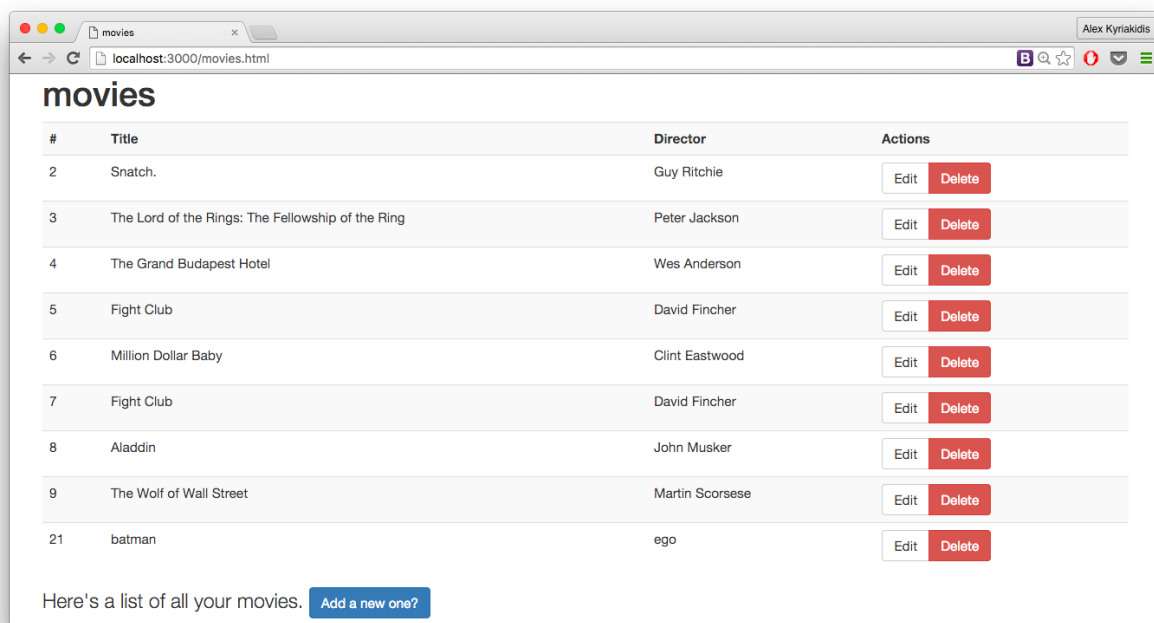
HTTP Method	URI	Action
GET/HEAD	api/movies	<i>Fetches</i> all movies
GET/HEAD	api/movies/{id}	<i>Fetches</i> specified movie
POST	api/movies	<i>Creates</i> a new movie
PUT/PATCH	api/movies/{id}	<i>Updates</i> an existing movie
DELETE	api/movies/{id}	<i>Deletes</i> specified movie

12.8.3 Your Code

Put your HTML code inside `~/themajestyofvuejs2/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` on your browser.

I hope you will enjoy this one. Good luck!



Example Output



Potential Solution

You can find a potential solution to this exercise [here](https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/homework/Chapter12)¹².

¹²<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/homework/Chapter12>

13. Pagination

In the previous chapter we managed to fetch all the records from the database and display them inside a table. That implementation is fine for a bunch of records, but in the real world, when you have to work with thousands or millions of records, you cannot simply fetch them and place them inside an array. If you do so, your browser will not be happy to load such an amount of data, but even if it manages to do that, then I assure you that no user likes dealing with a table containing 100,000 rows.



Info

Pagination is used in some form in almost every web application to divide returned data and display it on multiple pages. Pagination also includes the logic of preparing and displaying the links to the various pages, and it can be handled client-side or server-side. Server-side pagination is more common.

In situations like this, the developers who designed the API will (hopefully) divide the returned data in pages.

The HTTP response will contain some simple **meta-data** next to the main data, to inform you about *total* items, *per page* items, etc. To help you traverse through the pages, it will provide information such as the **current page**, the **next page**, and the **previous page**.

Example Response with Paginated data

```
{
  "total": 10000,
  "per_page": 50,
  "current_page": 15,
  "last_page": 200,
  "next_page_url": "/api/stories?page=16",
  "prev_page_url": "/api/stories?page=14",
  "from": 751,
  "to": 800,
  "data": [...]
}
```

The pagination's meta-data could also be inside an object next to **data**, or anywhere the API developers have decided.

Example Response with Paginated data

```
{
  "data": [...],
  "pagination": {
    "total": 10000,
    "per_page": 50,
    "current_page": 15,
    "last_page": 200,
    "next_page_url": "/api/stories?page=16",
    "prev_page_url": "/api/stories?page=14",
    "from": 751,
    "to": 800,
  }
}
```

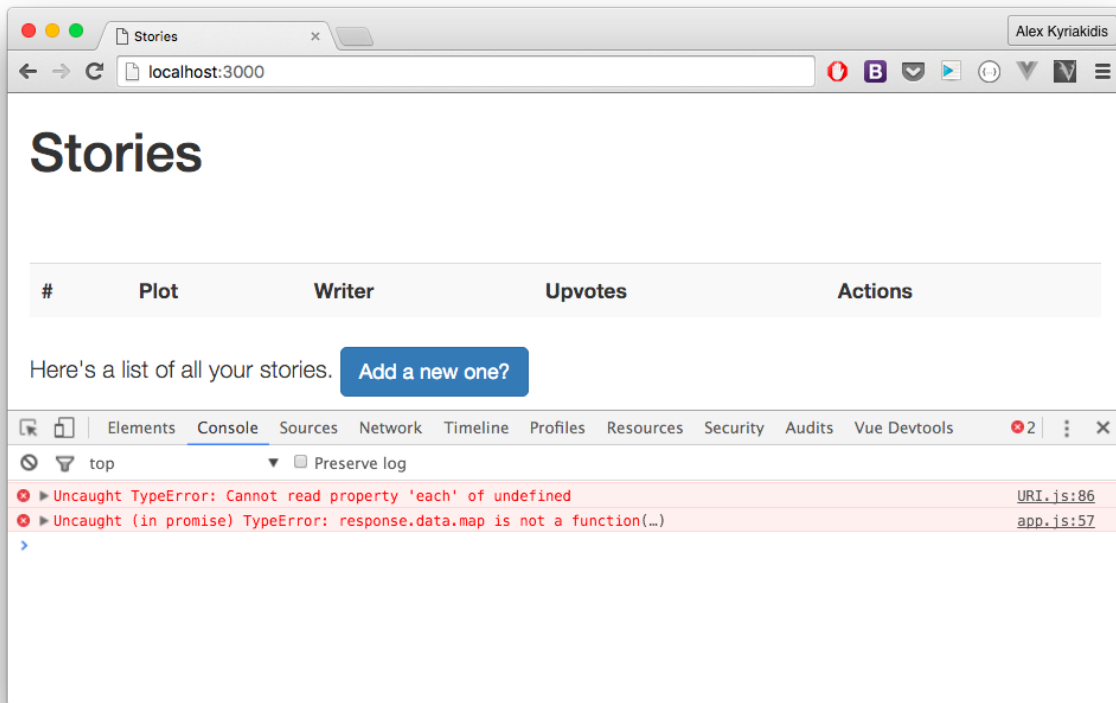
13.1 Implementation

We are going to continue working with the *story* examples from the previous chapter, using the slightly improved paginated API. So, we are going to modify the code, to be able to access and use these data.

If you take a look at the code from the [previous example](#), you will see that our `fetchStories` method is similar to this:

```
new Vue({
  ...
  methods: {
    ...
    fetchStories: function () {
      var vm = this;
      this.$http.get('/api/stories')
        .then(function (response) {
          var storiesReady = response.data.map(function (story) {
            story.editing = false;
            return story
          })
          Vue.set(vm, 'stories', storiesReady)
        });
    },
    ...
  }
});
```

If we open our HTML file on the browser, as you may have already guessed, our table doesn't render properly.

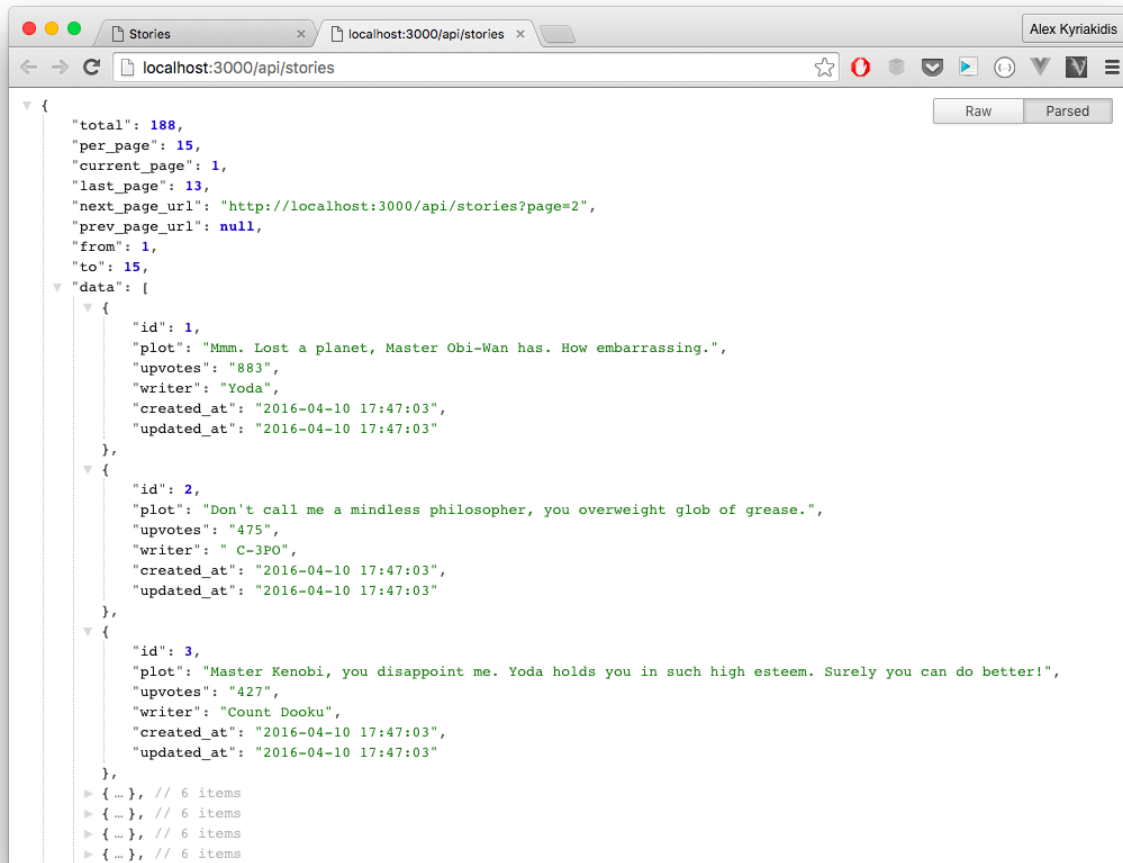


Stories' aren't displaying

This happens because the **stories** are now returned inside an array named **data**. To fix this, we have to change **response.data** to **response.data.data** (I know this is kinda weird, but...).

Except from the *stories* array, we also want to save the pagination's data inside an object in order to easily implement the pagination functionality.

To find out how we can access those data, let's have a look at the server's response.



Server's response

For starters, we don't need all those data. So, we will stick with *current_page*, *last_page*, *next_page_url*, and *prev_page_url*.

Our pagination object will be something like this:

```

pagination: {
  "current_page": 15,
  "last_page": 200,
  "next_page_url": "/api/stories?page=16",
  "prev_page_url": "/api/stories?page=14"
}

```

Let's modify our **fetchStories** method to update **pagination** object, each time it fetches stories from the database.


```

new Vue({
  ...
  methods: {
    ...
    fetchStories: function () {
      var vm = this;
      this.$http.get('/api/stories')
        .then(function (response) {
          var storiesReady = response.data.data.map(function (story) {
            story.editing = false;
            return story
          })
          //here we use response.data
          var pagination = {
            current_page: response.data.current_page,
            last_page: response.data.last_page,
            next_page_url: response.data.next_page_url,
            prev_page_url: response.data.prev_page_url
          }
          Vue.set(vm, 'stories', storiesReady)
          Vue.set(vm, 'pagination', pagination)
        });
      },
      ...
    }
  });

```

13.2 Pagination Links

By now, we have our **pagination** object but we always fetch the first page of stories since we are making a GET HTTP request to *api/stories*. We have to change the requested page, based on user interaction (next page, previous page).

First we'll update the **fetchStories** method to accept an argument with the desired page. If no argument is passed, it will fetch the first page. I'll also create a new method, **makePagination**, to make the code cleaner.

```

new Vue({
  ...
  methods: {
    ...
    fetchStories: function (page_url) {
      var vm = this;
      page_url = page_url || '/api/stories'
      this.$http.get(page_url)
        .then(function (response) {
          var storiesReady = response.data.data.map(function (story) {
            story.editing = false;
            return story
          })
          vm.makePagination(response.data)
          Vue.set(vm, 'stories', storiesReady)
        });
    },
    makePagination: function (data){
      //here we use response.data
      var pagination = {
        current_page: data.current_page,
        last_page: data.last_page,
        next_page_url: data.next_page_url,
        prev_page_url: data.prev_page_url
      }
      Vue.set(vm, 'pagination', pagination)
    }
    ...
  }
}

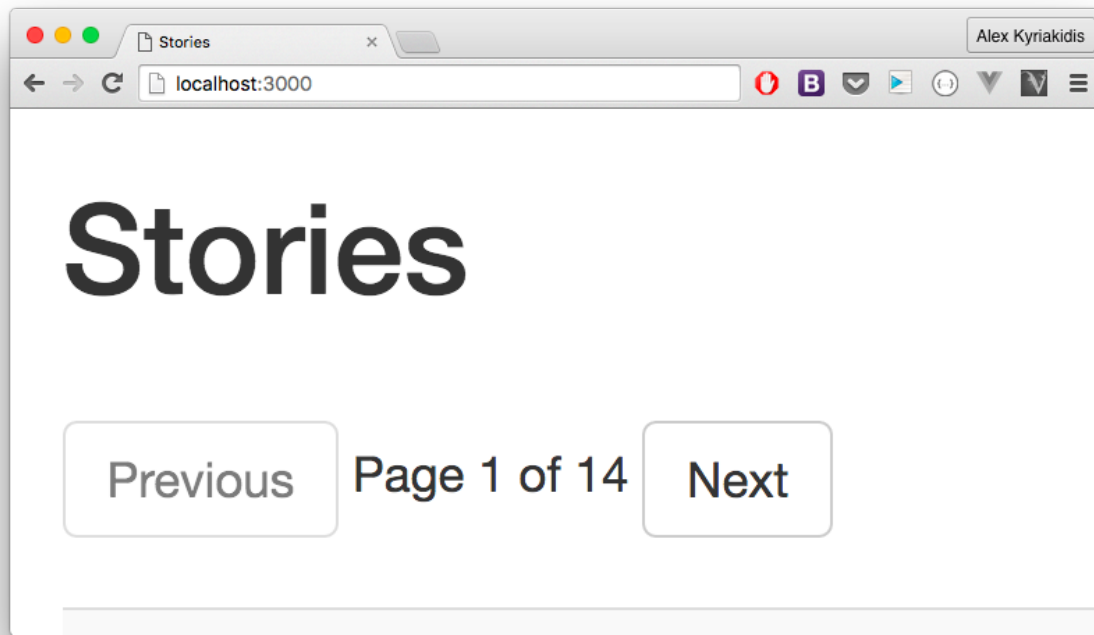
```

Now that our method is ready, we need a way to call it properly. We will add 2 buttons, one for next and one for previous page, on the top of our `#app` `div`. Each button will call `fetchStories` method when clicked, passing the corresponding `page url`.

```
1   <div class="pagination">
2     <button @click="fetchStories(pagination.prev_page_url)">
3       Previous
4     </button>
5     <button @click="fetchStories(pagination.next_page_url)">
6       Next
7     </button>
8   </div>
```

Pomp! If you try to click the buttons you'll see that they work as expected. We've got our pagination in the blink of an eye. It will be useful though to inform the user about which page he is currently looking at, and the total number of pages. Also, we can disable the **previous** button when the user is on the first page, and the **next** on the last, accordingly.

```
1   <div class="pagination">
2     <button @click="fetchStories(pagination.prev_page_url)"
3       :disabled="!pagination.prev_page_url"
4     >
5       Previous
6     </button>
7     <span>Page {{pagination.current_page}} of {{pagination.last_page}}</span>
8     <button @click="fetchStories(pagination.next_page_url)"
9       :disabled="!pagination.next_page_url"
10    >
11      Next
12    </button>
13  </div>
```



Disabled previous button



Code Examples

You can find the code examples of this chapter on [GitHub](#)¹.

13.3 Homework

There is nothing particular to do for homework in this chapter. If you actually want to work on this example, I will provide you the paginated API.

If you have solved the previous chapter's homework (downloaded the code and started a server), you are just a few clicks away. If you haven't, just follow [these instructions](#).

The paginated API lives inside `'~/themajestyofvuejs2/apis/pagination/stories'` directory.

The HTML file is in `'~/themajestyofvuejs2/apis/pagination/stories/public'` directory.

If you just want to view the final code, you can take a look at the files on [GitHub](#)².

¹<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter13>

²<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/apis/pagination/stories/public>

III Building Large-Scale Applications

14. ECMAScript 6

Before we take things a step further and see how we can build Large-Scale Applications, I would like to familiarize you with ECMAScript 6.



Info

ECMAScript is a client-side scripting language's specification, that is the basis of several programming languages including JavaScript, ActionScript, and JScript.

ECMAScript 6 (ES6), also known as ES2015, is the latest version of the ECMAScript standard. The ES6 specification was finalized in June 2015. It is a significant update to the language, and the first major update since ES5 was standardized in 2009. Implementation of ES6 features in major JavaScript engines is [underway now](#)¹.

14.1 Introduction

ES6 has a lot of new features. We are going to review those that we will use in the next chapters. If you are interested in learning more about what is new in ES6, I highly recommend you the book “Understanding ECMAScript 6” by Nicholas C. Zakas which is available on [leanpub](#)². There is also an [online version](#)³ of the book, which you can read for free.

Also, there are other useful resources and tutorials, like the one on [Babel](#)⁴, an article on [tutsplus](#)⁵, a [blog post](#)⁶ by Nicholas C. Zakas, and a ton of stuff around the web!

¹<http://kangax.github.io/compat-table/es6/>

²<https://leanpub.com/understandings6/>

³<https://leanpub.com/understandings6/read>

⁴<https://babeljs.io/docs/learn-es2015/>

⁵<http://code.tutsplus.com/articles/use-ecmascript-6-today--net-31582>

⁶<https://www.nczonline.net/blog/2013/09/10/understanding-ecmascript-6-arrow-functions/>

14.1.1 Compatibility

Unsurprisingly, support varies wildly from engine to engine, with Mozilla tending to lead the way. [ES6 compatibility table](#)⁷ is a useful start for establishing what ECMAScript 6 features your browser does and doesn't support.



Note

If you're using Chrome most of the ES6 features are hidden behind a feature toggle. Browse to `chrome://flags`, find the section titled "Enable Experimental JavaScript" and enable it to turn on support

From now on we will develop our examples using ES6 features.

14.2 Variable Declarations

14.2.1 Let Declarations

let is the new **var**. You can basically replace **var** with **let** to declare a variable, but limit the variable's scope only to the current code block. Since **let** declarations are not hoisted to the top of the enclosing block, you better always place **let** declarations first in the block, so that they are available to the entire block. For example:

Let inside if

```
1 let age = 22
2 if (age >= 18) {
3     let adult = true;
4     console.log(adult); //outputs true
5 }
6 //adult isn't accessible here
7 console.log(adult);
8 //ERROR: Uncaught ReferenceError: adult is not defined
```

⁷<https://kangax.github.io/compat-table/es6/>

Let on top

```
1 let age = 22
2 let adult
3 if (age >= 18) {
4     adult = true;
5     console.log(adult); //outputs true
6 }
7 //now adult is accessible here
8 console.log(adult); //outputs true
```

14.2.2 Constant Declarations

Constants, like `let` declarations, are block-level declarations. There is one big difference between `let` and `const`. Once you declare a variable using `const`, it is defined as a constant, which means that you can't change its value.

```
1 const name = "Alex"
2
3 name = "Kostas" //throws error
```



Info

Much like constants in other languages, their value cannot change later on. However, unlike constants in other languages, the value a constant holds **may be modified if it is an object**.

14.3 Arrow Functions

One of the most interesting new parts of ECMAScript 6 is the arrow functions. Arrow functions are functions defined with a new syntax that uses an “arrow” (\Rightarrow). They support both expression and statement bodies. Unlike functions, arrows share the same lexical `this` as their surrounding code.

For example, the following arrow function takes a single argument and returns its value increased by 1:


```
var increment = value => value + 1;  
increment(5) //returns 6
```

// equivalent to:

```
var increment = function(value) {  
    return value + 1;  
};
```

Another example with an arrow function which takes 2 arguments and returns their sum:

```
var sum = (a, b) => a + b;  
sum(5, 10) //returns 15
```

// equivalent to:

```
var sum = function(a, b) {  
    return a + b;  
};
```

An example arrow function which takes no arguments and uses a statement.

```
var sayHiAndBye = () => {  
    console.log('Hi!');  
    console.log('Bye!');  
};  
sayHiAndBye()
```

// equivalent to:

```
var sayHiAndBye = function() {  
    console.log('Hi!');  
    console.log('Bye!');  
};
```

14.4 Modules

This is, to me, the biggest improvement of the language. ES6 now supports exporting and importing modules across different files.

The simplest example is to create a `.js` file with a variable, and use it inside another file like this:

module.js

```
1 export name = 'Alex'
```

main.js

```
1 import {name} from './module'
2 console.log('Hello', name)
3 //outputs "Hello Alex"
```

You can also export variables along with functions **one by one**

module.js

```
1 export var name = 'Alex'
2 export function getAge(){
3   return 22;
4 }
```

main.js

```
1 import {name, getAge} from './module'
2 console.log(name, 'is', getAge())
```

Or inside an object:

module.js

```
1 var name = 'Alex'
2 function getAge(){
3   return 22;
4 }
5 export default {name, getAge}
```

main.js

```
1 import person from './module'
2 console.log( person.name, 'is', person.getAge() )
3 //outputs "Alex is 22"
```

14.5 Classes

JavaScript classes are introduced in ECMAScript 6 and are syntactical sugar over JavaScript's existing prototype-based inheritance. The class syntax is **not** introducing a new object-oriented inheritance model to JavaScript. JavaScript classes provide a much simpler and clearer syntax to create objects and deal with inheritance.

Class Example

```
//parent class
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }

  calcArea() {
    return this.height * this.width;
  }

  //To create a getter, use the keyword get.
  get area() {
    return this.calcArea();
  }

  //To create a setter, you do the same, using the keyword set.
}

//child class
class Square extends Rectangle{
  constructor(side) {
    //call parent's constructor
    super(side, side)
  }
}

var square = new Square(5);

console.log(square.area); //outputs 25
```

14.6 Default Parameter Values

With ES6 you can define default parameter values.

```
function divide(x, y = 2){  
    return x/y;  
}
```

// equivalent to:

```
function divide(x, y){  
    y = y == undefined ? 2 : y;  
    return x/y;  
}
```

14.7 Template literals

Template literals are string literals allowing embedded expressions. You can use multi-line strings and string interpolation features with them. They were called “template strings” in prior editions of the ES2015/ES6 specification.

Template literals are enclosed by the back-tick (`) character instead of double or single quotes. Within the back-ticks you can use `${expression}`, where expression can be a function, variable or an actual expression.

Template literals - Using Variables

```
let name = 'Alex'  
console.log(`Hello ${name}`)
```

// equivalent to:

```
console.log('Hello ' + name )
```

Template literals - Using Functions

```
add = (a, b) => a + b
```

```
let [a, b] = [10, 2]
```

```
console.log(`If you have ${a} eggs and you buy ${b}  
more you'll have ${add(a,b)} eggs!`)
```

// equivalent to:

```
console.log('If you have ' + a + ' eggs and you buy ' + b +  
'\nmore you'll have ' + add(a,b) + ' eggs!')
```

Template literals - Using Expressions

```
let [a, b] = [10, 2]
```

```
console.log(`If you have ${a} eggs and you buy ${b}  
more you'll have ${a + b} eggs!`)
```

```
// equivalent to:
```

```
console.log('If you have ' + a + ' eggs and you buy ' + b +  
'\nmore you\'ll have ' + (a + b) + ' eggs!')
```

It is also possible to split the message in multiple lines using `'\n'`.

15. Advanced Workflow

All these ES6 features (and many more) may get you excited, but there is a catch here. As we mentioned before, **not** all browsers fully support ES6/ES2015 features.

In order to be able to write this new JavaScript syntax today, we need to have a middleman which will take our code and transpile it into [Vanilla JS](#)¹, which *every browser understands*. This procedure is really **important** in production, even though you might not think so.

Let me tell you a story. A few years ago, a co-worker of mine began using some cool JS features that weren't fully supported by all browsers. A few days later our users started complaining about some pages of our website not showing properly, but we couldn't figure out why. We tested it on different PCs, Android phones, iPhones, etc, and it was 100% functional in all our browsers. Later, he found out that older versions of mobile Safari didn't support his code. *Don't be that guy!*

Some times it's **really hard** to know if the code you write is going to work well on **all browsers**, including Facebook's mobile browser, which is my worst fear.

15.1 Compiling ES6 with Babel

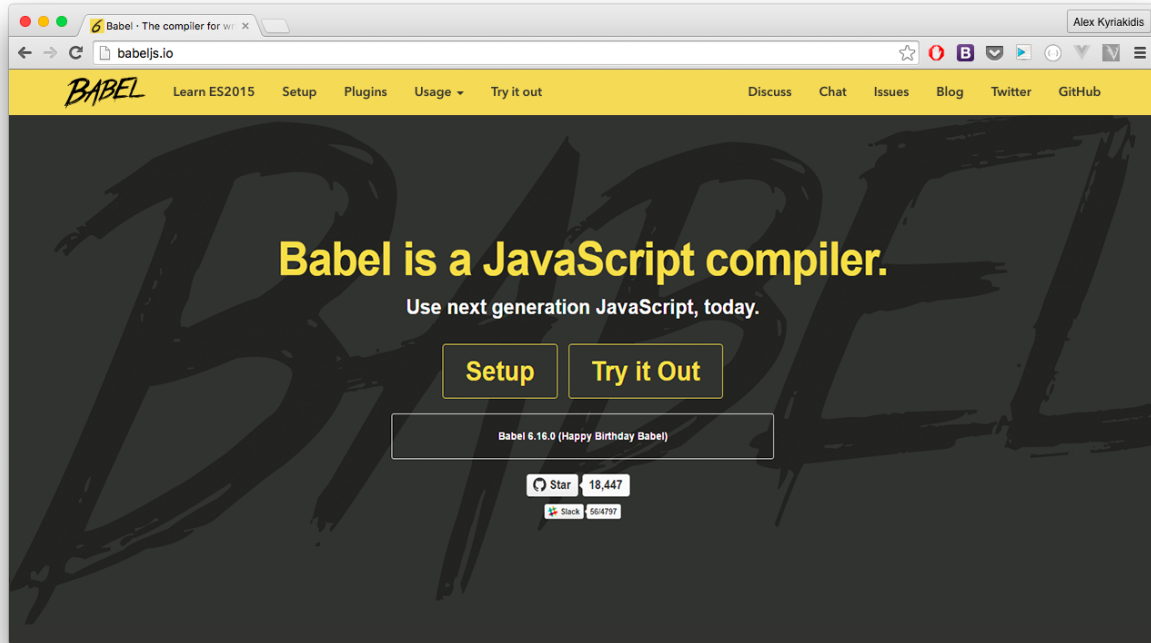
Babel will be our *middleman*. Babel is a source-to-source JavaScript compiler, which lets us use next generation JavaScript, today.



Info

A source-to-source compiler, transcompiler or transpiler, is a type of compiler that takes the source code of a program written in one programming language, as its input, and produces the equivalent source code in another programming language.

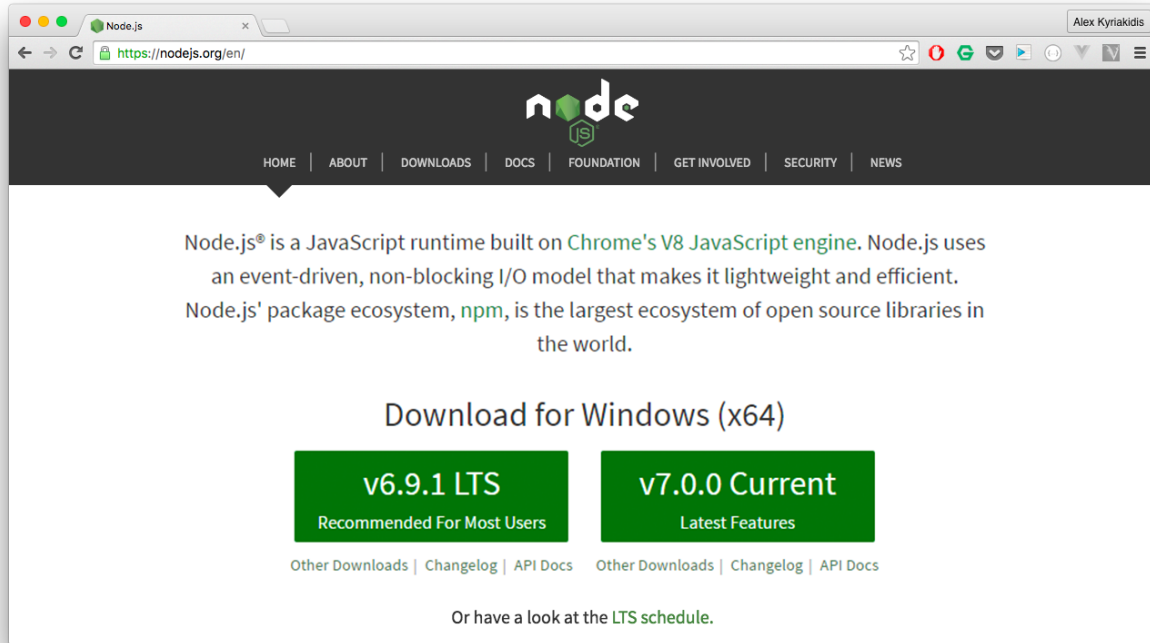
¹<http://vanilla-js.com/>



Babel

Before installing Babel, you have to install **Node.js**. To do so, head to [Node's website](https://nodejs.org/en/)² and hit the download button for the the Latest Stable Version. It is going to give you a *.pkg* file (or *.msi* if you are on Windows). When the download is finished, open the file and follow the instructions. Then, do the required restart, and you are done!

²<https://nodejs.org/en/>



Node.js

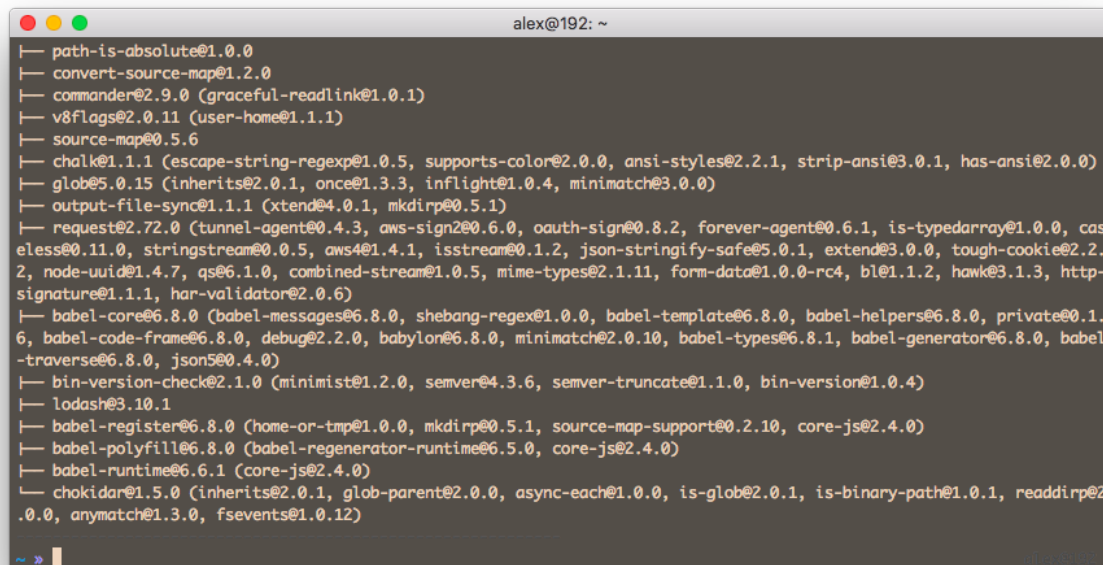
15.1.1 Installation

Create a new directory and place a file named **package.json** inside, containing an empty JSON object (`{}`). You can do it manually, or by running the following commands in your terminal.

```
>_ mkdir babel-example  
_ echo {} > package.json
```

Then run this to install Babel:

```
>_ npm install babel-cli --save-dev
```

```
alex@192: ~
├─ path-is-absolute@1.0.0
├─ convert-source-map@1.2.0
├─ commander@2.9.0 (graceful-readlink@1.0.1)
├─ v8flags@2.0.11 (user-home@1.1.1)
├─ source-map@0.5.6
├─ chalk@1.1.1 (escape-string-regexp@1.0.5, supports-color@2.0.0, ansi-styles@2.2.1, strip-ansi@3.0.1, has-ansi@2.0.0)
├─ glob@5.0.15 (inherits@2.0.1, once@1.3.3, inflight@1.0.4, minimatch@3.0.0)
├─ output-file-sync@1.1.1 (xtend@4.0.1, mkdirp@0.5.1)
├─ request@2.72.0 (tunnel-agent@0.4.3, aws-sign2@0.6.0, oauth-sign@0.8.2, forever-agent@0.6.1, is-typedarray@1.0.0, caselless@0.11.0, stringstream@0.0.5, aws4@1.4.1, isstream@0.1.2, json-stringify-safe@5.0.1, extend@3.0.0, tough-cookie@2.2.2, node-uuid@1.4.7, qs@6.1.0, combined-stream@1.0.5, mime-types@2.1.11, form-data@1.0.0-rc4, bl@1.1.2, hawk@3.1.3, http-signature@1.1.1, har-validator@2.0.6)
├─ babel-core@6.8.0 (babel-messages@6.8.0, shebang-regex@1.0.0, babel-template@6.8.0, babel-helpers@6.8.0, private@0.1.6, babel-code-frame@6.8.0, debug@2.2.0, babylon@6.8.0, minimatch@2.0.10, babel-types@6.8.1, babel-generator@6.8.0, babel-traverse@6.8.0, json5@0.4.0)
├─ bin-version-check@2.1.0 (minimist@1.2.0, semver@4.3.6, semver-truncate@1.1.0, bin-version@1.0.4)
├─ lodash@3.10.1
├─ babel-register@6.8.0 (home-or-tmp@1.0.0, mkdirp@0.5.1, source-map-support@0.2.10, core-js@2.4.0)
├─ babel-polyfill@6.8.0 (babel-regenerator-runtime@6.5.0, core-js@2.4.0)
├─ babel-runtime@6.6.1 (core-js@2.4.0)
├─ chokidar@1.5.0 (inherits@2.0.1, glob-parent@2.0.0, async-each@1.0.0, is-glob@2.0.1, is-binary-path@1.0.1, readdirp@2.0.0, anymatch@1.3.0, fsevents@1.0.12)
```

Terminal output

When it's done, your package.json file should be something like this:

package.js

```
{
  "devDependencies": {
    "babel-cli": "^6.18.0"
  }
}
```



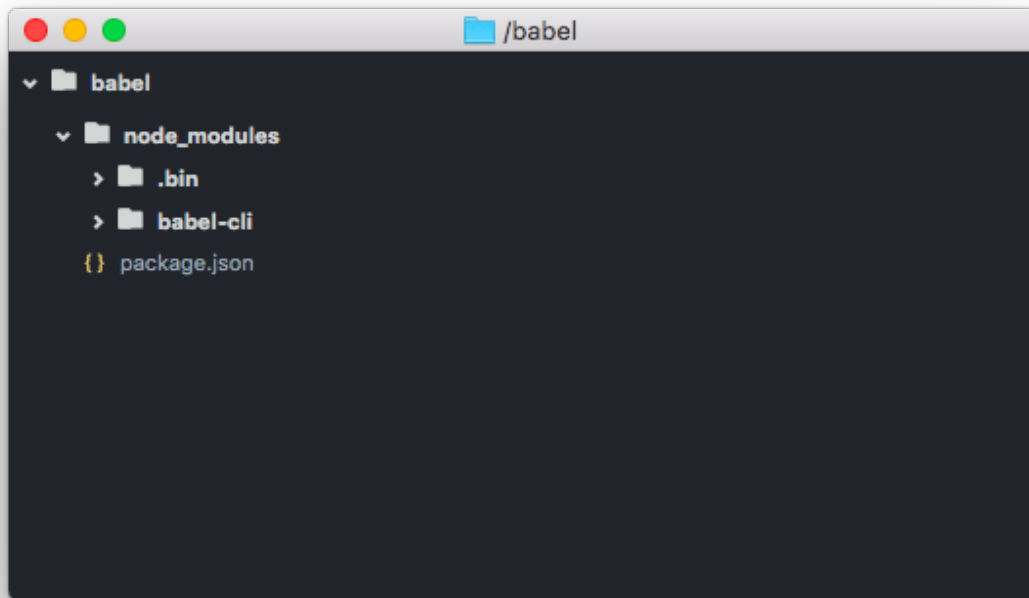
What is package.json?

A package.json file contains meta data about your app or module. Most importantly, it includes the list of dependencies to install from npm when running `npm install`. If you're familiar with Composer, it's similar to the `composer.json` file.

To learn more about package.json have a look at [npm docs](https://docs.npmjs.com/files/package.json)³.

Your project's directory should look like this:

³<https://docs.npmjs.com/files/package.json>



Project directory

15.1.2 Configuration

Now that we have babel installed, we need to explicitly tell it what transformations to run on build. Since we want to transform ES2015 code, we will install the [ES2015-Preset](https://babeljs.io/docs/plugins/preset-es2015/)⁴.

We'll also create a config file (`.babelrc`) to enable our preset.

```
>_ npm install babel-preset-es2015 --save-dev  
    echo { "presets": [ ["es2015"] ] } > .babelrc
```



Tip

If the second command fails, enclose file contents inside quotes like this:

```
echo '{ "presets": [ ["es2015"] ] }' > .babelrc
```

⁴<https://babeljs.io/docs/plugins/preset-es2015/>

15.1.3 Build alias

Instead of running Babel directly from the command line, we're going to put our commands within **npm** scripts.

We'll add a **scripts** field to our **package.json** file, and register the **babel** command there, as **build**. Our **package.json** will look like this:

package.js

```
{
  "scripts": {
    "build": "babel src -d assets/js"
  },
  "devDependencies": {
    "babel-cli": "^6.8.0",
    "babel-preset-es2015": "^6.18.0"
  }
}
```

This works like an alias. Meaning that when we run **npm run build** we are actually running **babel src -d assets/js**. This command tells Babel to transpile the code from the **src** directory to **assets/js** directory.

Before we run the **build** command we have to do a few more things. For starters, go on and create the above-mentioned dirs (**src** and **assets/js**).

15.1.4 Usage

Lets move on and put some files inside **src** folder. I will create a file with a simple **sum** function and call it **sum.js**.

src/sum.js

```
const sum = (a, b) => a + b;
console.log(sum(5,3));
```

Thats was it. We can now run:

```
> npm run build
```

When you run it, you can see in your terminal that the **src\sum.js** file has been compiled to **assets\js\sum.js** and looks like this:

assets/js/sum.js

```
"use strict";

var sum = function sum(a, b) {
  return a + b;
};
console.log(sum(5, 3));
```

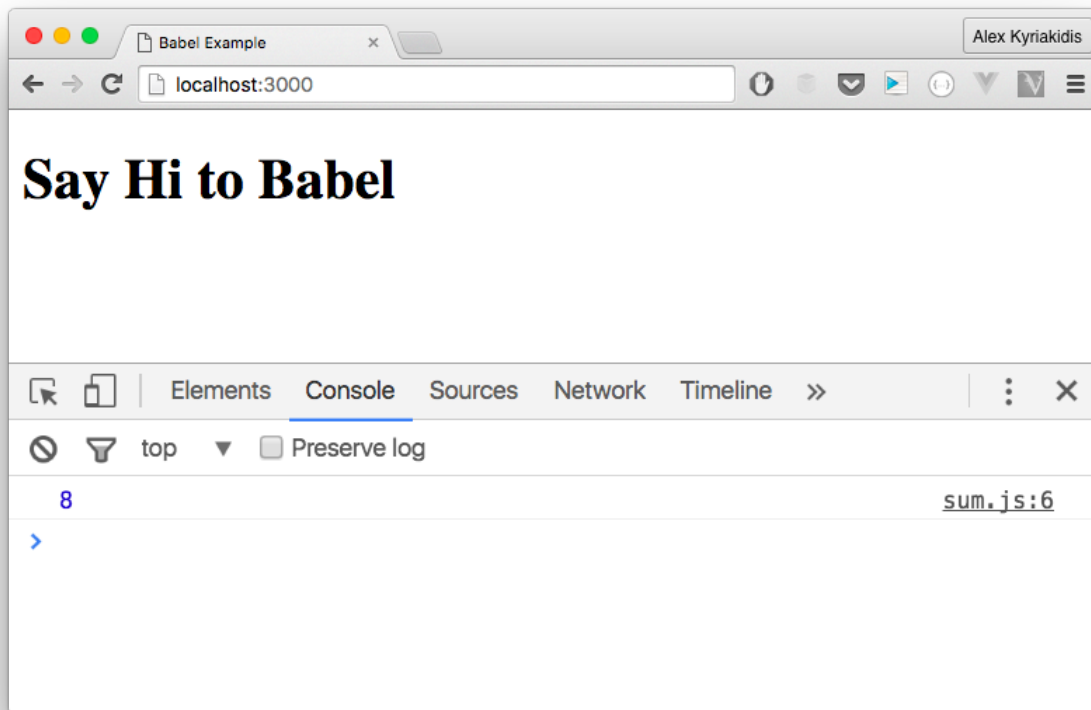
From now on, whenever you want to compile your ES6 code you can do it by running the **build** command. **Pretty neat, huh?**

It's time to see the resulted **sum.js** file in the browser. I will create **sum.html** and include our js.

sum.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Babel Example</title>
</head>
<body>
  <h1>Babel Example</h1>

  <script src="assets/js/sum.js"></script>
</body>
</html>
```



Browser output

As you can see, the result of the `sum` function is successfully printed to the console.



Info

When you want to test a `.js` file, but you don't want to get in the process of serving it to the browser, you can run it with Node.js.

In the `sum.js` example there is a `console.log(sum(5,3))` line, so if you type in your terminal `node sum.js` you'll see the result (8) pop right up!

15.1.5 Homework

This homework exercise aims to help you remember what you've learned by reproducing the example we've built. Instead of the `sum.js` go on and use *ES6 Classes* to create a `Ninja.js` file which will contain a `Ninja` class.

A `Ninja` should have a property `name` and a method `announce`, which will alert the presence of a `Ninja`.

For example

```
new Ninja('Leonardo').announce()  
//alerts "Ninja Leonardo is here!"
```

Don't forget to compile your *js* using Babel before including it in your *HTML*.



Hint

You can find an example for building classes on the previous chapter.



Hint 2

Don't forget to run `npm run build` each time you make a change in your *js* file, otherwise it won't update!



Potential Solution

You can find a potential solution to this exercise [here](https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/homework/Chapter15/chapter15.1)⁵.

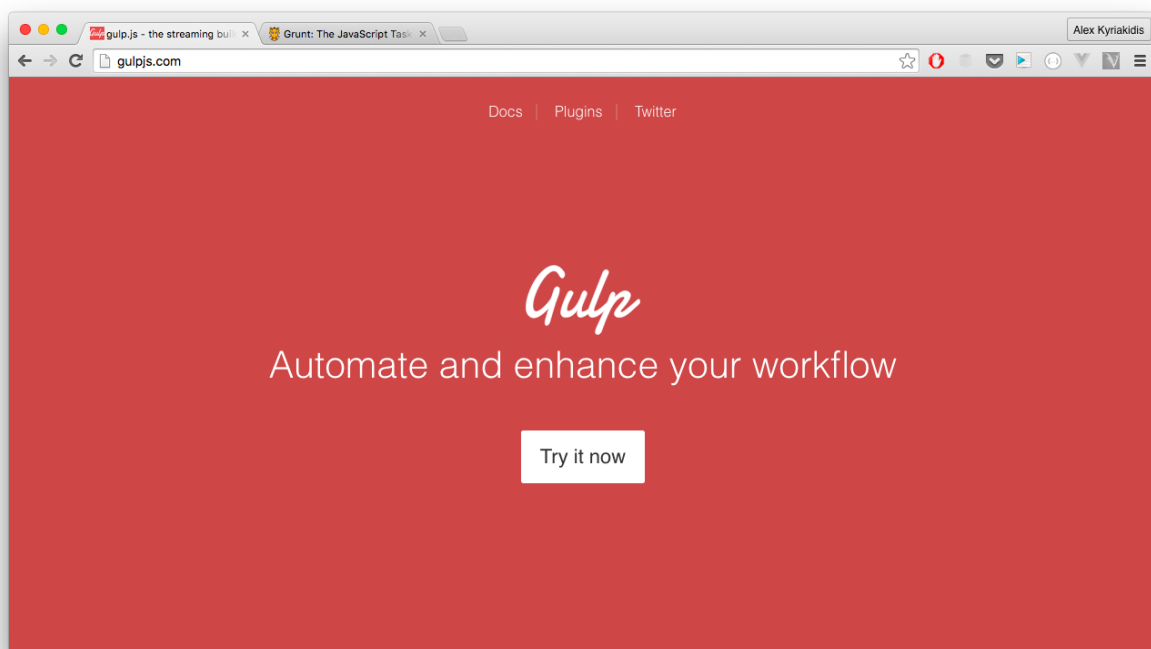
⁵<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/homework/Chapter15/chapter15.1>

15.2 Workflow Automation with Gulp

15.2.1 Task Runners

If you devoted some time to develop the app from the previous Homework section, you have probably found out that it is kinda annoying having to run `npm run build` every time you make a change in your code.

This is where **Task Runners** like [Gulp](http://gulpjs.com/)⁶ or [Grunt](http://gruntjs.com/)⁷ come in handy. Task runners let you automate and enhance your workflow.



Gulp



Why use a task runner?

In one word: **Automation**. The less work you have to do when performing repetitive tasks like minification, compilation, unit testing, linting, etc, the easier your job becomes.

⁶<http://gulpjs.com/>

⁷<http://gruntjs.com/>



Gulp vs Grunt

Grunt, like Gulp, is a tool for defining and running tasks. The major difference between Grunt and Gulp is that Grunt defines tasks using **configuration objects** while Gulp defines tasks as **JavaScript functions**. Since Gulp runs Javascript, it provides more flexibility in writing your tasks.

Both have a massive plugin library, where you may find one which implements a task you need.

15.2.2 Installation

I will show you an example of how you can use Gulp to watch for changes in your js files and automatically run the **build** command.

First we have to install Gulp globally:

```
>_ npm install gulp-cli --global
```

Then we'll install Gulp in our project's devDependencies:

```
>_ npm install gulp --save-dev
```

Now that we have gulp installed we will create a **gulpfile.js** at the root of our project:

gulpfile.js

```
const gulp = require('gulp');

gulp.task('default', function() {
  // place the code for your default task here
});
```

15.2.3 Usage

When we now run **gulp** in our console, it starts, but does nothing yet. We have to setup a default task.

In order to run **babel** directly, I'll install an npm plugin called **gulp-babel**⁸.

⁸<https://www.npmjs.com/package/gulp-babel>


```
> npm install gulp-babel --save-dev
```

I'll add a new *gulp task* named *babel* and set it as the default task. My *gulpfile* will look like this:

gulpfile.js

```
const gulp = require('gulp');
const babel = require('gulp-babel');

gulp.task('default', ['babel']);

//basic babel task
gulp.task('babel', function() {
  return gulp.src('src/*.js')
    .pipe(babel({
      presets: ['es2015']
    }))
    .pipe(gulp.dest('assets/js/'))
});
```

This task basically tells babel to transform all *js* files under *src* directory using the *es2015* preset and put them inside *assets/js* directory.

15.2.4 Watch

Currently running *gulp* on your console has the same effect with *npm run build*. What we want to achieve here is to run this task every time a *js* file has changed. To do so, we will set up a *watcher* inside our *gulpfile* like this:

gulpfile.js

```
const gulp = require('gulp');
const babel = require('gulp-babel');

gulp.task('default', ['watch']);

//basic babel task
gulp.task('babel', function() {
  return gulp.src('src/*.js')
    .pipe(babel({
      presets: ['es2015']
    }));
```

```
    )))  
    .pipe(gulp.dest('assets/js/'))  
  })  
  
  //the watch task  
  gulp.task('watch', function() {  
    gulp.watch('src/*.js', ['babel']);  
  })
```

When we run **gulp watch** on our console, gulp is watching for changes in all our `.js` files under the specified directory. Every time we make a change, gulp runs our `babel` task and the files under `assets/js` are being updated. **How awesome is that?**

15.2.5 Homework

This homework exercise is following the previous one. If you haven't done the previous one, it's never too late to begin!

Since this part of the chapter is dedicated to Task Runners, you have to to setup a watcher with Gulp and compile your code with *Babel*, when a change is detected.



Note

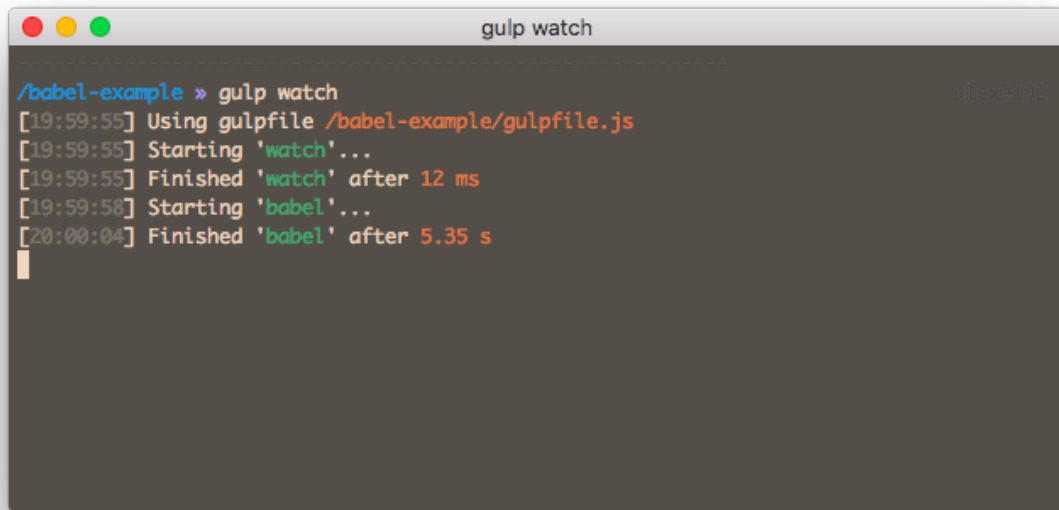
You may already have noticed that when running Gulp it prints messages in terminal (*“Starting”* - *“Finished”*), so don't be so hasty and wait for the changes to be applied.



Potential Solution

You can find a potential solution to this exercise [here](https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/homework/Chapter15/chapter15.2)⁹.

⁹<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/homework/Chapter15/chapter15.2>

A terminal window titled "gulp watch" with a dark background and light-colored text. The window shows the execution of the "gulp watch" command in a directory named "/babel-example". The output shows the command being used, the "watch" task starting and finishing quickly, and the "babel" task starting and finishing after a longer duration. The text is color-coded: blue for the prompt, green for task names, and orange for durations and file paths.

```
/babel-example » gulp watch
[19:59:55] Using gulpfile /babel-example/gulpfile.js
[19:59:55] Starting 'watch'...
[19:59:55] Finished 'watch' after 12 ms
[19:59:58] Starting 'babel'...
[20:00:04] Finished 'babel' after 5.35 s
```

Gulp is watching!

15.3 Module Bundling with Webpack

15.3.1 Module Bundlers

Our workflow is fine with the current code of *sum.js*. We will extend its features, to calculate the cost of a pizza and a beer and let the client know.

src/sum.js

```
const pizza = 10
const beer = 5

const sum = (a, b) => a + b + '$';
console.log( `Alex, you have to pay ${sum(pizza, beer)}$` )
```

This code looks good, but assuming that not everyone is called *Alex* we'll create a new file, **client.js**, which will provide the client's name.

src/client.js

```
export const name = 'Alex'
```

We'll import the name from there.

src/sum.js

```
import { name } from './client'

const pizza = 10
const beer = 5

const sum = (a, b) => a + b + '$';
console.log(`${name} you have to pay ${sum(pizza, beer)}`)
```

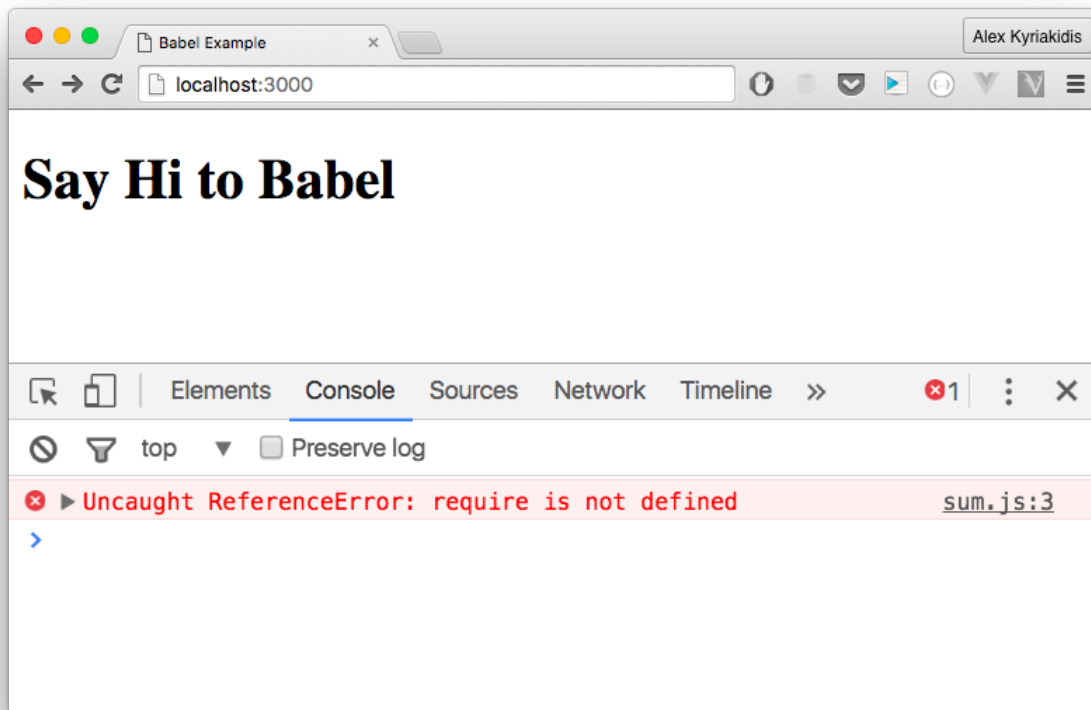
Great! If we run `node assets/js/sum.js` we get the expected output.

A terminal window titled 'alex@192: /babel-example' shows the command 'node assets/js/sum.js' being executed. The output is 'Alex you have to pay 15\$'. The prompt is '/babel-example >'.

```
alex@192: /babel-example
/babel-example » node assets/js/sum.js
Alex you have to pay 15$
/babel-example »
```

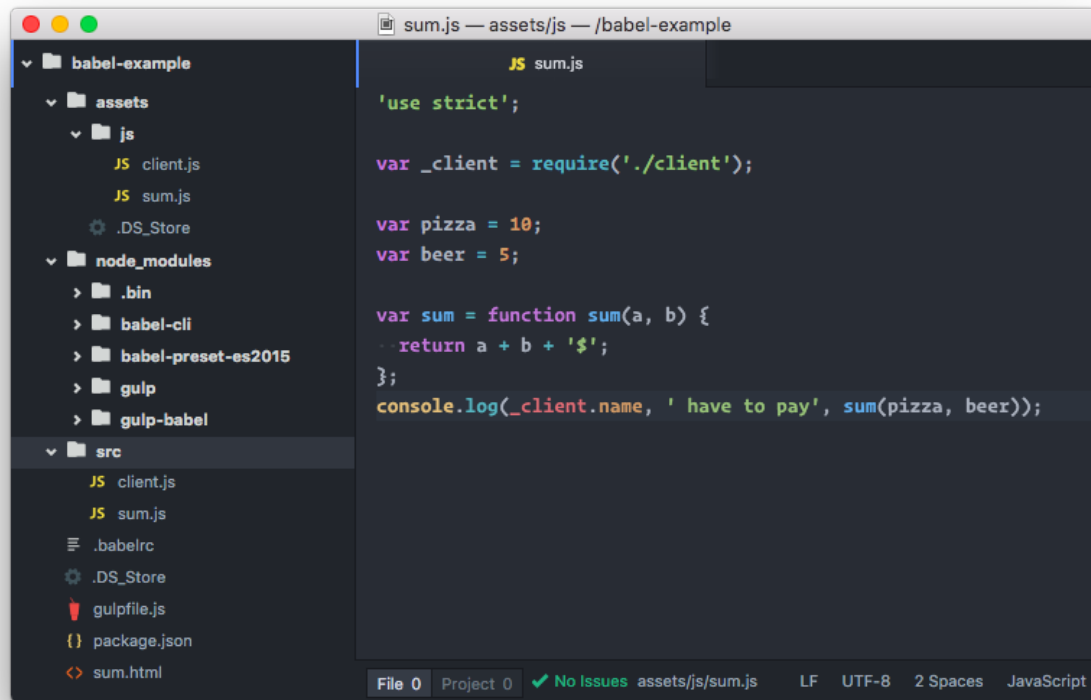
Output of sum.js

You would expect here that the same behavior will apply when we open the *html* file in the browser, but it doesn't! We get an error instead.



Require is not defined

Check the `node assets/js/sum.js` and notice the `var _client = require('./client');` on top. The reason we get the error in the browser is because `require()` does not exist in the browser/**client-side JavaScript**. What we have to do is to *bundle* the modules in one file so it can be included within a `<script>` tag.



assets/js/sum.js

This is where we need **Module Bundlers** like [Webpack](https://webpack.github.io/)¹⁰ or [Browserify](http://browserify.org/)¹¹.

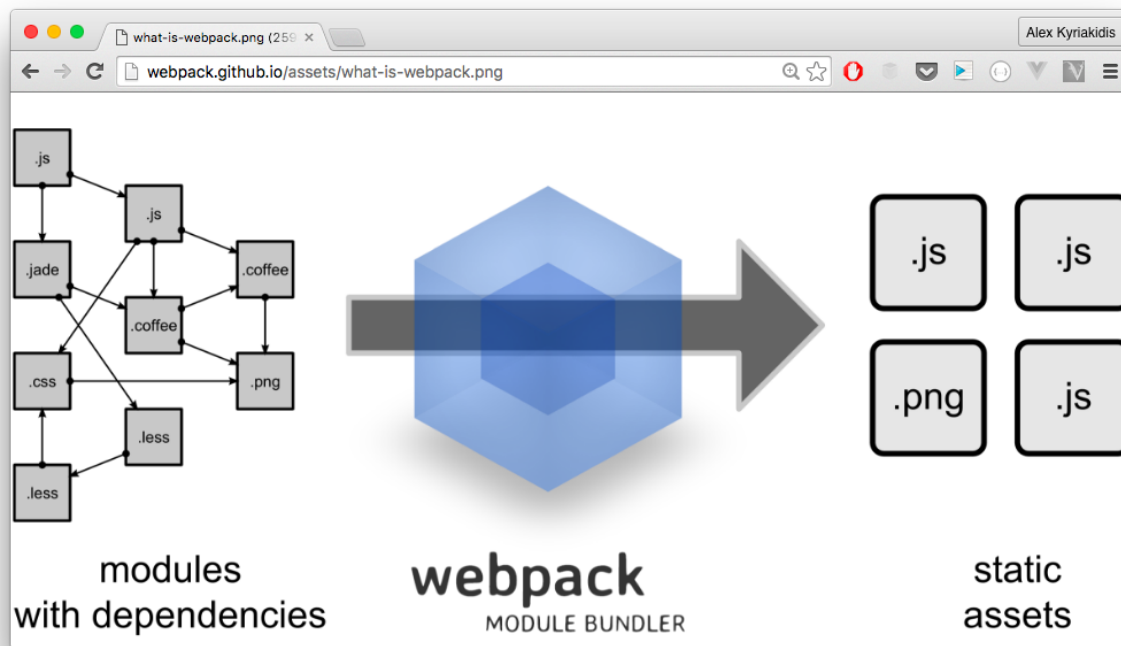
15.3.2 Webpack

Webpack is a Module Bundler. It takes JavaScript modules, understands their dependencies, and then concatenates them and produces static assets representing those modules.

I'll use Webpack in the next examples. That is because I believe it'll come handy in the future for other uses, since it can do more than bundling modules. Using *loaders*, we can teach Webpack to transform all type of files in any way we want, before outputting the final bundle.

¹⁰<https://webpack.github.io/>

¹¹<http://browserify.org/>



What Webpack does

15.3.3 Installation

I am going to install Webpack globally first, and then add it as dependency in our project.

```
>_ npm install webpack -g
    npm install webpack --save-dev
```



Tip

At the time of writing, there is a known bug when you are using [Vagrant](https://www.vagrantup.com/)¹² on Windows, running `npm install` may fail. To solve this issue, exit Vagrant, `cd` your project on your Windows terminal and run `npm install` from there.

15.3.4 Usage

In order to compile our Javascript we have to give Webpack a source point and an output. In our case the source is the babelified `assets/js/sum.js` and as output I'll set `assets/webpacked/app.js`.

¹²<https://www.vagrantup.com/>

```
>_ webpack assets/js/sum.js assets/webpacked/app.js
```

A terminal window titled 'alex@192: /babel-example' showing the output of the command 'webpack assets/js/sum.js assets/webpacked/app.js'. The output includes a hash, version, time, and a table of assets.

```
alex@192: /babel-example
/babel-example » webpack assets/js/sum.js assets/webpacked/app.js
Hash: 7d079f20fab1a5cd6563
Version: webpack 1.13.0
Time: 79ms
  Asset      Size  Chunks             Chunk Names
  app.js    1.78 kB      0  [emitted]  main
    [0]  ./assets/js/sum.js 192 bytes {0} [built]
    [1]  ./assets/js/client.js 113 bytes {0} [built]

/babel-example »
```

Webpack Output

After the build is completed, we can use the outputed *js*, *assets/webpacked/app.js*, inside *sum.html*. We can also run it in the terminal using `node assets/webpacked/app.js`.

15.3.5 Automation

If you are lazy, like me, and you don't like having to run *webpack* every time you make a change, you can automate its process. You can configure Webpack to watch your source and rebuild your bundle, when any of your files change. I won't do it here. Instead, I'm going to integrate it into Gulp, in order to demonstrate how you can combine multiple tools.



Further Learning

If you want to learn more about how Webpack works and how you can configure it, go on and read “[Beginner's guide to Webpack](#)”¹³ by Nader Dabit¹⁴.

To integrate Webpack into Gulp I'll use a plugin called [webpack-stream](#)¹⁵.

¹³<https://medium.com/@dabit3/beginner-s-guide-to-webpack-b1f1a3638460>

¹⁴<https://twitter.com/dabit3>

¹⁵<https://www.npmjs.com/package/webpack-stream>


```
> npm install webpack-stream --save-dev
```

After installing, I'll create a new task with the name of 'webpack' and tell Gulp to run it every time it detects a change, immediately after running the 'babel' task.

gulpfile.js

```
const gulp = require('gulp');
const babel = require('gulp-babel');
const webpack = require('webpack-stream');

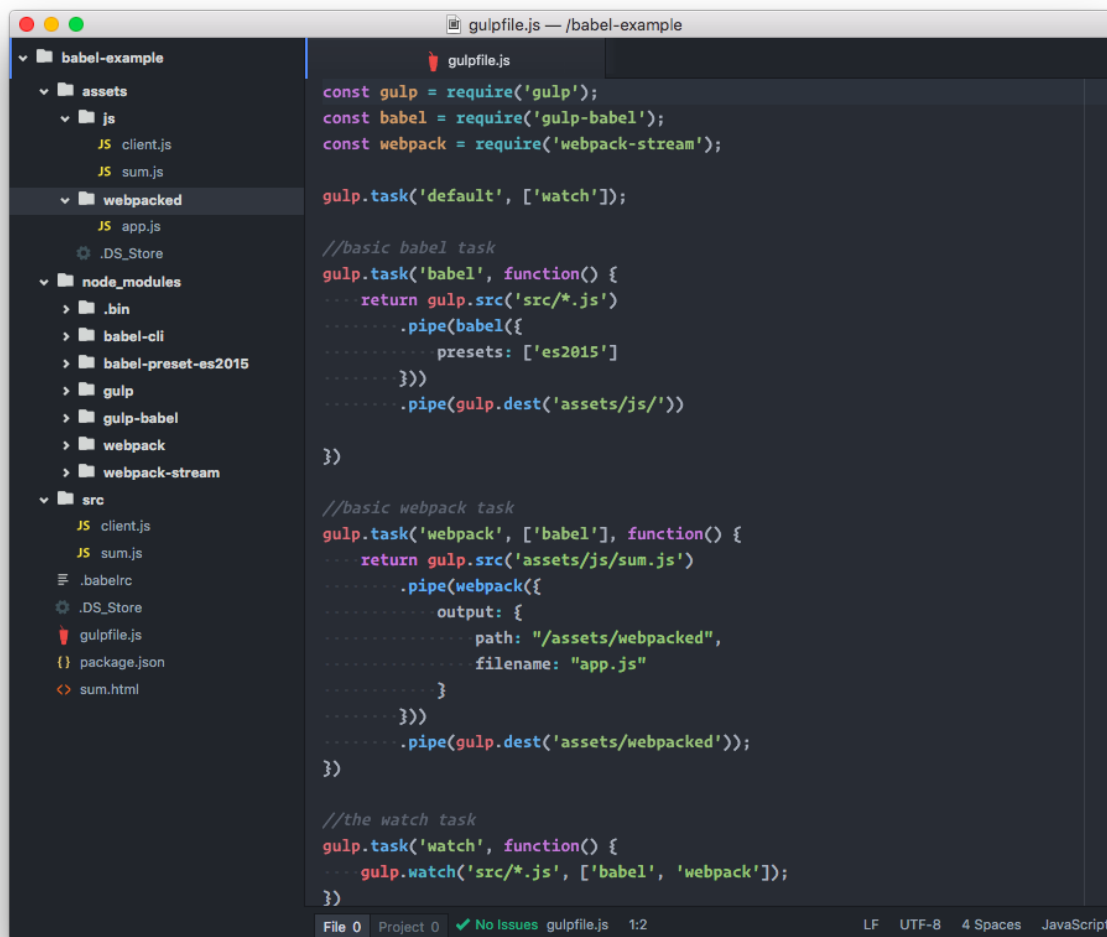
gulp.task('default', ['watch']);

//basic babel task
gulp.task('babel', function() {
  return gulp.src('src/*.js')
    .pipe(babel({
      presets: ['es2015']
    }))
    .pipe(gulp.dest('assets/js/'))
});

//basic webpack task
gulp.task('webpack', ['babel'], function() {
  return gulp.src('assets/js/sum.js')
    .pipe(webpack({
      output: {
        path: "/assets/webpacked",
        filename: "app.js"
      }
    }))
    .pipe(gulp.dest('assets/webpacked'));
});

//the watch task
gulp.task('watch', function() {
  gulp.watch('src/*.js', ['babel', 'webpack']);
});
```

This solution is not ideal. It is just a demonstration of how you can bind whatever you've learned together. When in production, there are a lot better ways to automate your webpack tasks.



Webpack in Gulp

15.4 Summary

When you want to compile ES6 you can use [Babel](http://babeljs.io/)¹⁶.

To automate operations like this and many others (such as minifying, compiling SASS/LESS, etc) you need task runners like [Gulp](http://gulpjs.com/)¹⁷ or [Grunt](http://gruntjs.com/)¹⁸.

To bundle things up you can use [Webpack](https://webpack.github.io/)¹⁹ or [Browserify](http://browserify.org/)²⁰.

¹⁶<http://babeljs.io/>

¹⁷<http://gulpjs.com/>

¹⁸<http://gruntjs.com/>

¹⁹<https://webpack.github.io/>

²⁰<http://browserify.org/>

In the next chapter we will dive into Vue's Single File Components and use several tools that Vue provides you with, along with the tools we have learned.



Note

If you found this chapter hard to understand, don't worry. You don't need to remember all these things. This was just a demonstration in order to give you a better understanding of how things work. In the next chapter we will use *project-templates*. There, things like **module bundling**, **automation**, **build on change** and much more, are already implemented and we'll take advantage of them.

16. Working with Single File Components

As we promised, in this chapter we are going to review Single File Components. To use these single-file Vue components we need tools like **Webpack** with **vue-loader**, or **Browserify** with **vueify**. For our examples, we are going to use Webpack, which we've already seen how it works. If you prefer Browserify or something else, feel free to use it.

Single File Components or Vue Components encapsulate their CSS styles, template and JavaScript code, all in one file using the `.vue` extension. And that's where webpack steps in, to bundle this new file type with the other files.

Webpack uses [vue-loader](https://github.com/vuejs/vue-loader)¹ to transform Vue components into plain JavaScript modules. *vue-loader* also provides a very nice set of features such as ES2015 enabled by default, scoped CSS for each component, and more.

16.1 The `vue-cli`

To avoid configuring Webpack and creating a new workflow from nothing, we will use `vue-cli`.



Info

`vue-cli`² is a simple *Command-line interface* for scaffolding Vue.js projects.

This awesome tool is the fastest way to get up a pre-configured build. It offers templates with hot-reload, lint-on-save, unit testing, and much more. Currently, it offers scaffold templates for webpack and browserify, but if needed, you can [create you own](#)³.

16.1.1 Vue's Templates

What CLI does, is pulling down templates from [Vue.js official templates repository](https://github.com/vuejs-templates)⁴ where there are 5 templates available now. I believe this number will grow in the near future. You can check what they include on GitHub.

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

²<https://github.com/vuejs/vue-cli>

³<https://github.com/vuejs/vue-cli#custom-templates>

⁴<https://github.com/vuejs-templates>

All templates contain a *package.json* file, which handles the project's dependencies and comes with a preset of NPM scripts.

Using Vue's project templates, you get a lot of features together. For example, the "webpack" template's description says that it is *"A full-featured Webpack + vue-loader setup with hot reload, linting, testing & css extraction."*

16.1.2 Installation

We're going to stick with the Webpack setup approach and install **vue-cli** globally using the following command.

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

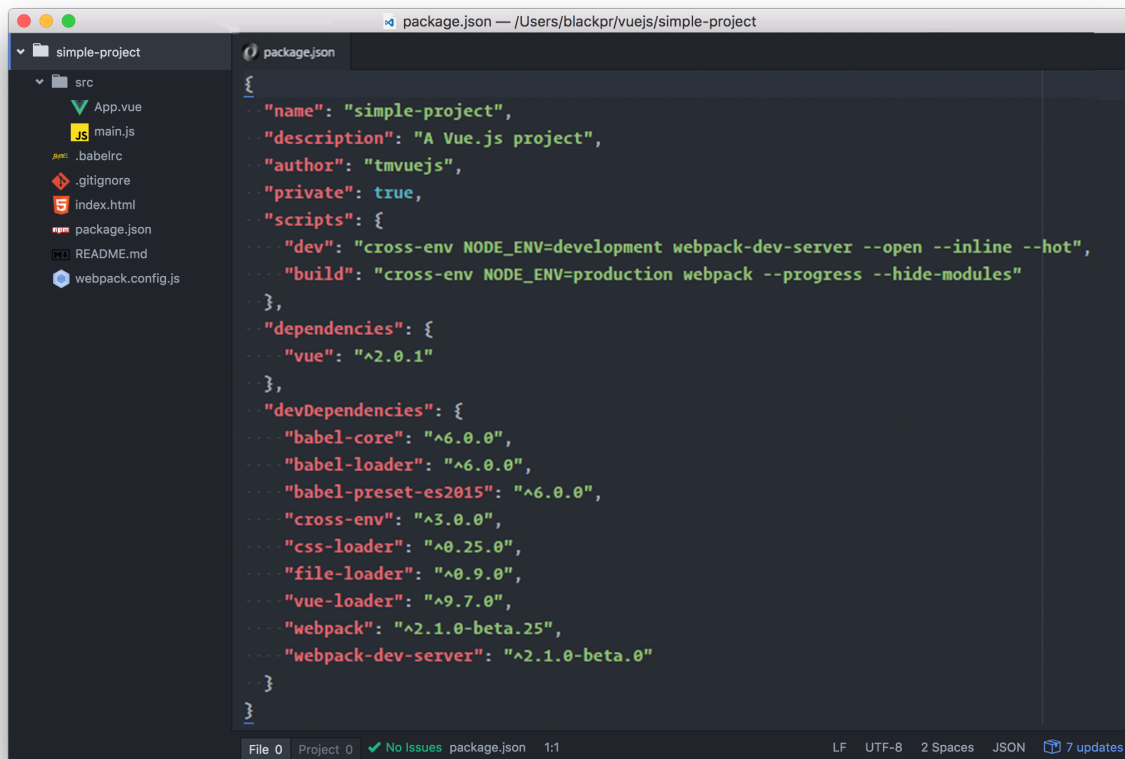
16.1.3 Usage

Using the CLI you can run **vue init <template-name> <project-name>** where the **<template-name>** is the name of the template (either official or custom) and the **<project-name>** is the name of the directory/project you are going to create.

So, if you run

```
> vue init webpack-simple simple-project
```

you are going to have a directory named `simple-project` with the following structure:



webpack-simple structure

For our example we will use the full featured webpack template, so our command will be like this:

```
>_ vue init webpack stories-classic-project
```



Tip

Use `vue list` to see all available official templates.



Info

When you are initializing a new project you will be prompt to fill in some details, like the name, the version, the author, build, etc. Every time we will use the cli to create a new project we will choose the `Runtime + Compiler` build, because we need to compile templates on the fly (e.g. passing a string to the template option, or mounting to an element using its in-DOM HTML as the template). You can find detailed explanations of different builds at the [guide](#)⁵.

At some point you will be asked to **Pick an ESLint preset**. The available options are [feross/standard](#)⁶ and [airbnb/javascript](#)⁷.

I created a table to compare the two styles, in order for you to get a better understanding of what rules each style applies.

Rules	feross/standard	airbnb/javascript
Indentation	2 spaces	2 spaces
Semicolons	No!	Yes
Unused Variables	Not allowed	Not allowed
String's quotes	Single	Single
Use === instead of ==	Yes	Yes
Number of empty lines allowed	1	2
Space after function name	Yes	No
Start a line with [No	Yes
End files w/ a newline character	Yes	Yes
Trailing commas allowed	No	No



Standard vs Airbnb

The rules of the table are some of many applied in each style. To consider and decide what fits you the best, check their Github repositories.

After you've selected a style, you'll get some prompts about installing several tools like [Karma-Mocha](#)⁸ and [Nightwatch](#)⁹. We are not going to need these tools right now, so answer the questions negative and continue.

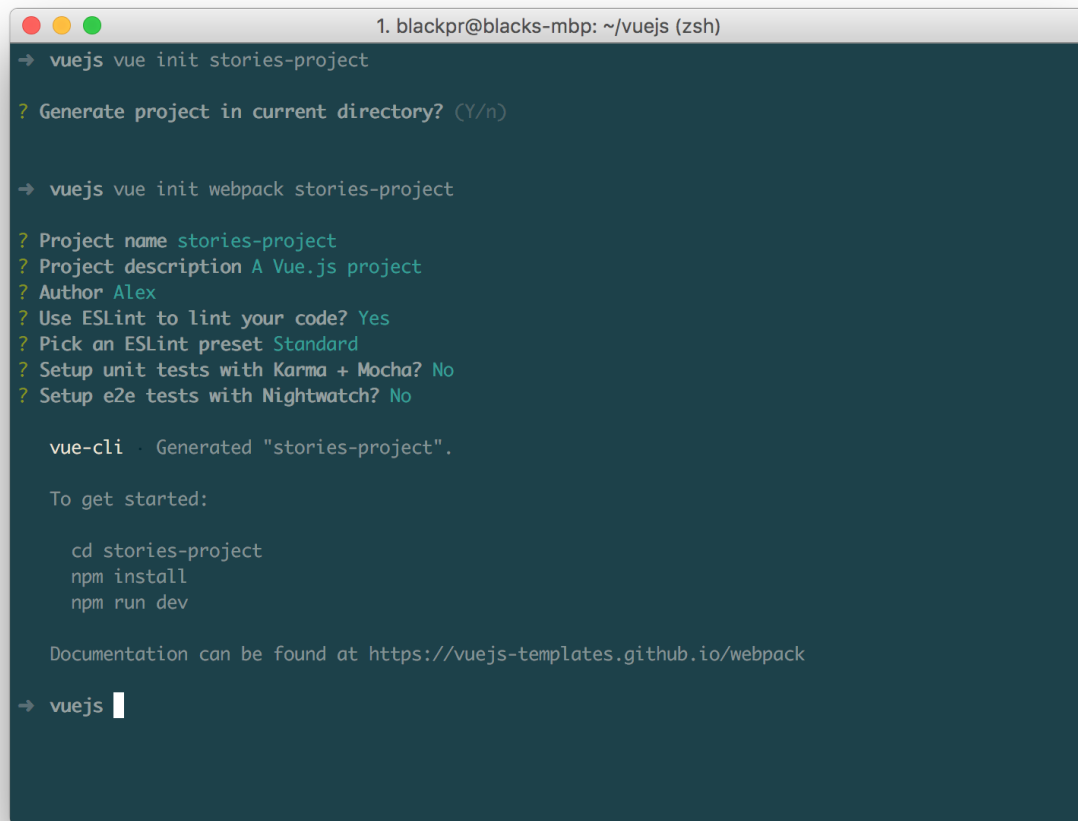
⁵<https://vuejs.org/v2/guide/installation.html#Explanation-of-Different-Builds>

⁶<https://github.com/feross/standard>

⁷<https://github.com/airbnb/javascript>

⁸<https://github.com/karma-runner/karma-mocha>

⁹<http://nightwatchjs.org/>



```
1. blackpr@blacks-mbp: ~/vuejs (zsh)
→ vuejs vue init stories-project

? Generate project in current directory? (Y/n)

→ vuejs vue init webpack stories-project

? Project name stories-project
? Project description A Vue.js project
? Author Alex
? Use ESLint to lint your code? Yes
? Pick an ESLint preset Standard
? Setup unit tests with Karma + Mocha? No
? Setup e2e tests with Nightwatch? No

vue-cli · Generated "stories-project".

To get started:

  cd stories-project
  npm install
  npm run dev

Documentation can be found at https://vuejs-templates.github.io/webpack

→ vuejs
```

Vue's Template Installation



Info

Karma is a plugin, adapter for the [Mocha](https://mochajs.org/)¹⁰ testing framework.

Nightwatch enables writing browser automated testing, that run against a [Selenium](http://www.seleniumhq.org/)¹¹ server.

16.2 Webpack Template

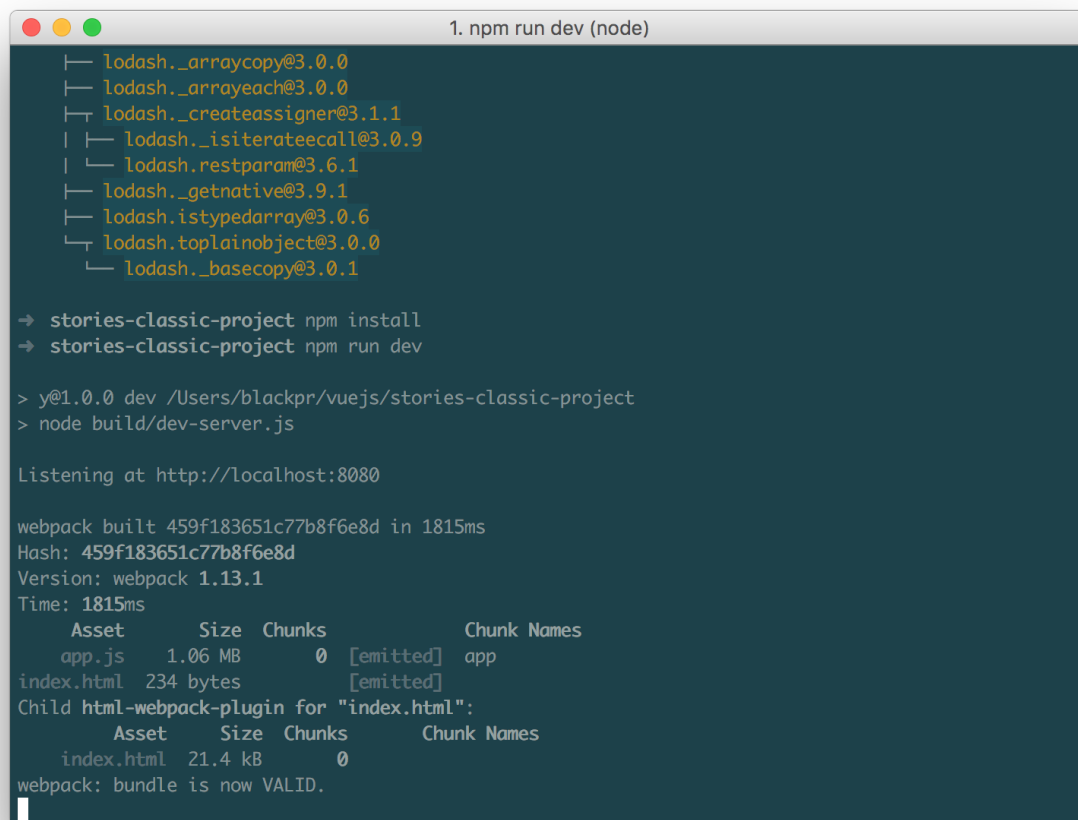
To complete the setup of our project we need to install its dependencies. Let's move on and run:

¹⁰<https://mochajs.org/>

¹¹<http://www.seleniumhq.org/>


```
>_ cd stories-classic-project
    npm install
    npm run dev
```

The terminal outputs ***Listening at http://localhost:8080***. You should wait until the **webpack: bundle is now VALID** message is posted. Then you are lit!



```
1. npm run dev (node)
└─ lodash._arraycopy@3.0.0
└─ lodash._arrayeach@3.0.0
└─ lodash._createassigner@3.1.1
├─ lodash._isiterateecall@3.0.9
├─ lodash.restparam@3.6.1
└─ lodash._getnative@3.9.1
└─ lodash.istypedarray@3.0.6
└─ lodash.toplainobject@3.0.0
└─ lodash._basecopy@3.0.1

→ stories-classic-project npm install
→ stories-classic-project npm run dev

> y@1.0.0 dev /Users/blackpr/vuejs/stories-classic-project
> node build/dev-server.js

Listening at http://localhost:8080

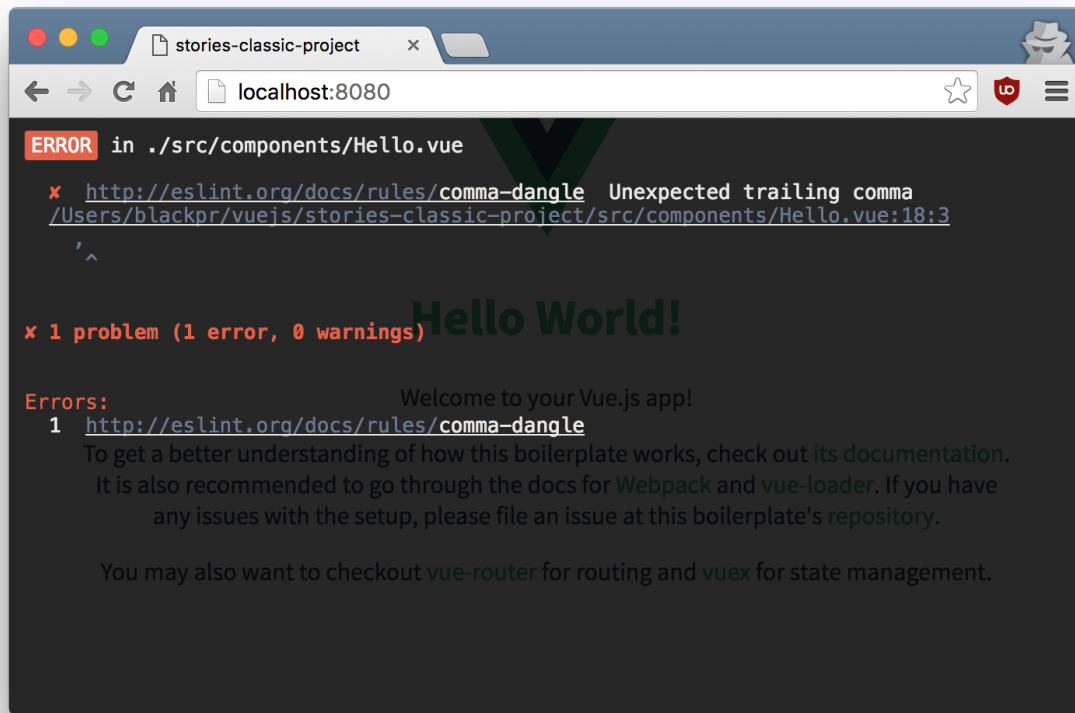
webpack built 459f183651c77b8f6e8d in 1815ms
Hash: 459f183651c77b8f6e8d
Version: webpack 1.13.1
Time: 1815ms
   Asset      Size  Chunks             Chunk Names
  app.js    1.06 MB          0  [emitted]  app
index.html  234 bytes          0  [emitted]
Child html-webpack-plugin for "index.html":
   Asset      Size  Chunks             Chunk Names
index.html  21.4 kB          0
webpack: bundle is now VALID.
```

Server Running...



Warning

Be careful, you have to be explicit in your code. Otherwise you will get errors for extra empty lines between blocks, trailing spaces, indentation other than 2 spaces and other stuff that don't follow the [selected style's rules](#).



Error overlay

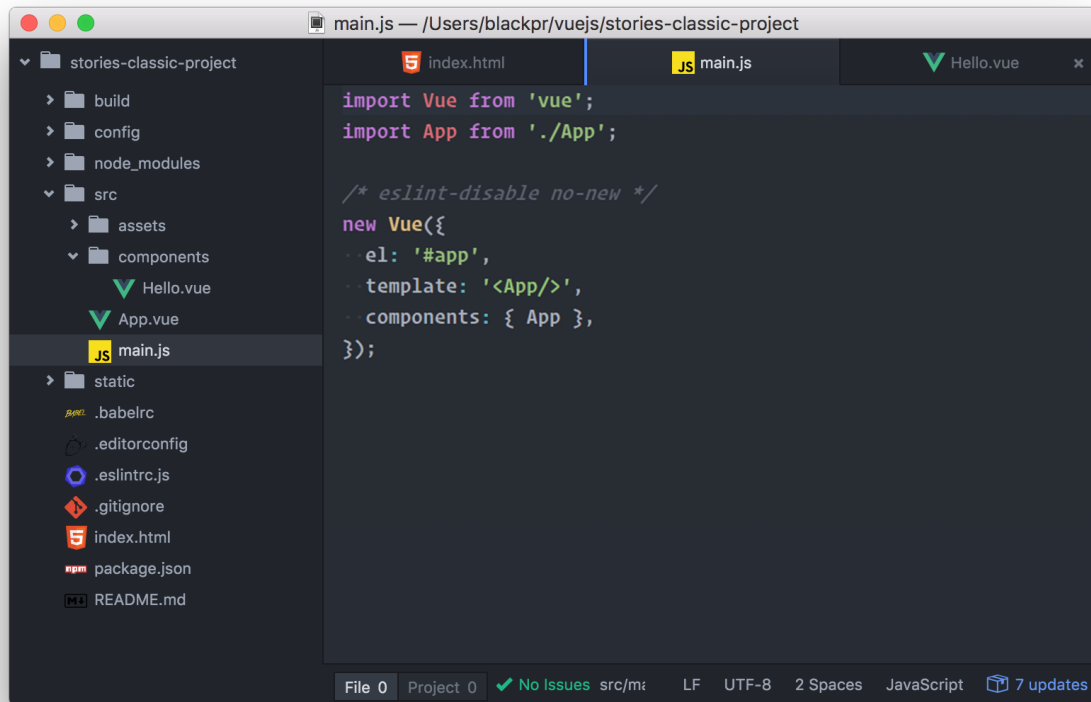


Note

If you use `webpack-simple` you will still have the basic features, but the error overlay won't display in the browser, so check the terminal for any errors.

16.2.1 Project Structure

After completing the above steps you should have a project directory filled with all the necessary files.



Webpack structure

The files that you are usually going to play with, are:

1. **index.html**
2. **main.js**
3. files under **src** and **src/components** directories

16.2.2 index.html

Lets start with the **index.html**. It should look like this

index.html

```
<html>
  <head>
    <meta charset="utf-8">
    <title>stories-classic-project</title>
  </head>
  <body>
    <app></app>
    <!-- built files will be auto injected -->
  </body>
</html>
```

As you can see, it is a pretty basic setup with a component already included. The comment refers to the script, `app.js`, which is the output of Webpack. It basically means that after Webpack has bundled the scripts, it will automatically inject the outputted script here, so that you don't have to include it manually.

16.2.3 Hello.vue

If you are following along, navigate to `src/components` and open `Hello.vue` file to see how a `.vue` file looks like.

src/components/Hello.vue

```
<template>
  <div class="hello">
    <h1>{{ msg }}</h1>
    <h2>Essential Links</h2>
    ...
  </div>
</template>

<script>
export default {
  name: 'hello',
  data () {
    return {
      msg: 'Welcome to Your Vue.js App'
    }
  }
}
```

```
</script>

<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped>
h1, h2 {
  font-weight: normal;
}
...
</style>
```

Inside the `<script>` tag, the component contains only its **data*. There is no need to define a template. It will be automatically bound if `<template>` block exists. The `<template>` block defines the template of the component, of course. Think of `<template>` like a `<template-hello>` tag in your HTML.

We could have only the `<script>` block, but this way, we wouldn't take advantage of Single File Component's benefits.

Remember that **export**, ES6 feature, is handled by those nice tools (transpilers + module bundlers) we have installed.



Warning

Each `.vue` file mustn't contain more than one `<script>` block. Every template must contain exactly **one root element**, like `<div id=hello>...</div>` which encapsulates all the other elements.

The script must export a *Vue.js component options object*. Exporting an extended constructor created by `Vue.extend()` is also supported, but a plain object is preferred.

ES6 allows us to have any numbers of exports, but in single file components that's not the case.

If you try to do additional exports you will get a warning similar to this: `[vue-loader] src/components/Hello.vue: named exports in /*.vue files are ignored.`

The `<style>` block, defines the CSS styles as expected.

16.2.4 App.vue

The `App.vue` file is located in the `src` directory and is the one which contains the main template of the application. This component is usually responsible to include the other components.

`App.vue` has a few more lines with texts and styles, but since we are focusing on the structure, we have shortened it a little.

src/App.vue

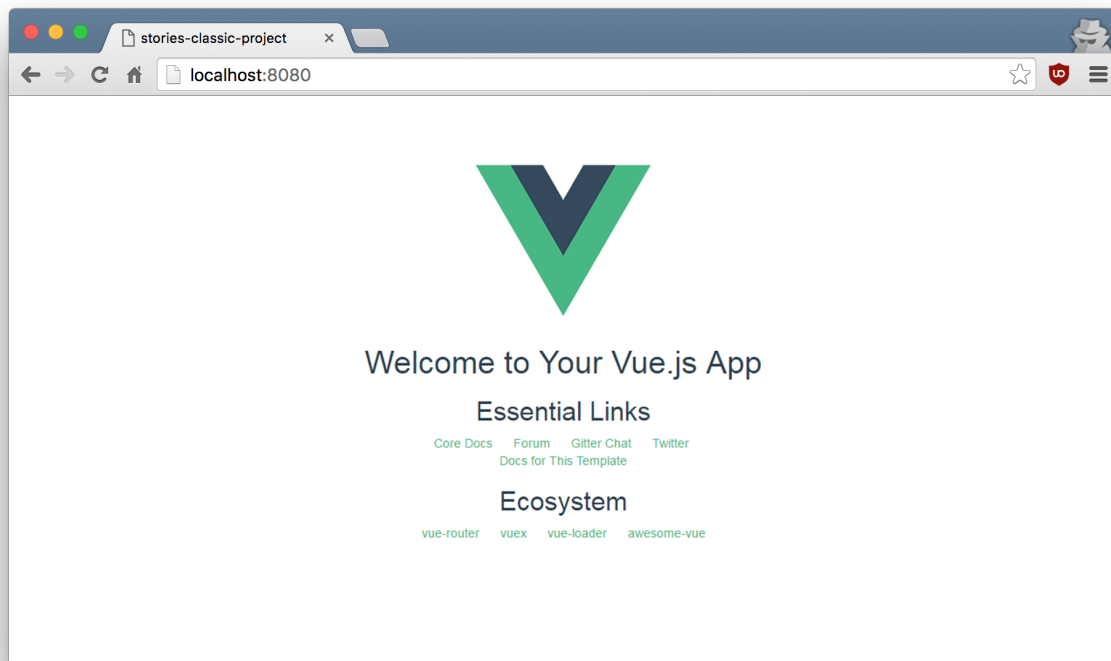
```
<template>
  <div id="app">
    
    <hello></hello>
  </div>
</template>

<script>
import Hello from './components/Hello'

export default {
  name: 'app',
  components: {
    Hello
  }
}
</script>

<style>
...
</style>
```

It has the same structure as the *Hello.vue* file we saw above. By default, there is a **components** object which contains the `Hello` component. Inside this object, we will import any new components. In the template, there is the `<hello></hello>` tag, and therefore the template of the `Hello` component will be displayed.



Project's Homepage

16.2.5 main.js

The *main.js* file within *src*, as you imagine, is our main script.

src/main.js

```
import Vue from 'vue'
import App from './App'

/* eslint-disable no-new */
new Vue({
  el: '#app',
  template: '<App/>',
  components: { App }
})
```

It imports *Vue* as a module from *node_modules* and *App* component from the *src* directory as well. Below is our *Vue* instance and in the components object, there is the *App* one.

Whenever you need to import a script or a component globally, you can put it within *main.js*.



Note

The **template**: '`<App/>`' option represents the template output of the component, which is to be injected to the `index.html` we've mentioned earlier.

`<App/>` has the same output with `<App> </App>` and `<app> </app>`.



Info

You can find more information about the Project Structure of Webpack template on its [documentation](http://vuejs-templates.github.io/webpack/structure.html)¹²

16.3 Forming `.vue` Files

We have seen how a single file component looks like and how it is used in a project. It is time to create a few, in a real-life scenario. Assume we want to create some kind of social network or forum, where users post their stories and experiences. To create our startup, we are going to need 2 forms, one for registration and login, and one page to display users' stories.

Before we begin, we'll include Bootstrap globally in order to be able to use its styles within all our components. To do so, we have to update our `index.html`.

`index.html`

```
<html>
  <head>
    <meta charset="utf-8">
    <title>stories-classic-project</title>
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6\
/css/bootstrap.min.css">
  </head>
  <body>
    <div id="app"></div>
    <!-- built files will be auto injected -->
  </body>
</html>
```

Let's create the login form, in a new, `Login.vue` file.

¹²<http://vuejs-templates.github.io/webpack/structure.html>

src/components/Login.vue

```
<template>
  <div id="login">
    <h2>Sign in</h2>
    <input type="email" placeholder="Email address">
    <input type="password" placeholder="Password">
    <button class="btn">Sign in</button>
  </div>
</template>

<script>
export default {
  created () {
    console.log('login')
  }
}
</script>
```

And there it is. In order to view the file in the browser we have to include our *Login* component somewhere. So, we'll import it into the main *App* component and append it to its components object.

src/App.vue

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

<script>
import Login from './components/Login.vue'
import Hello from './components/Hello'

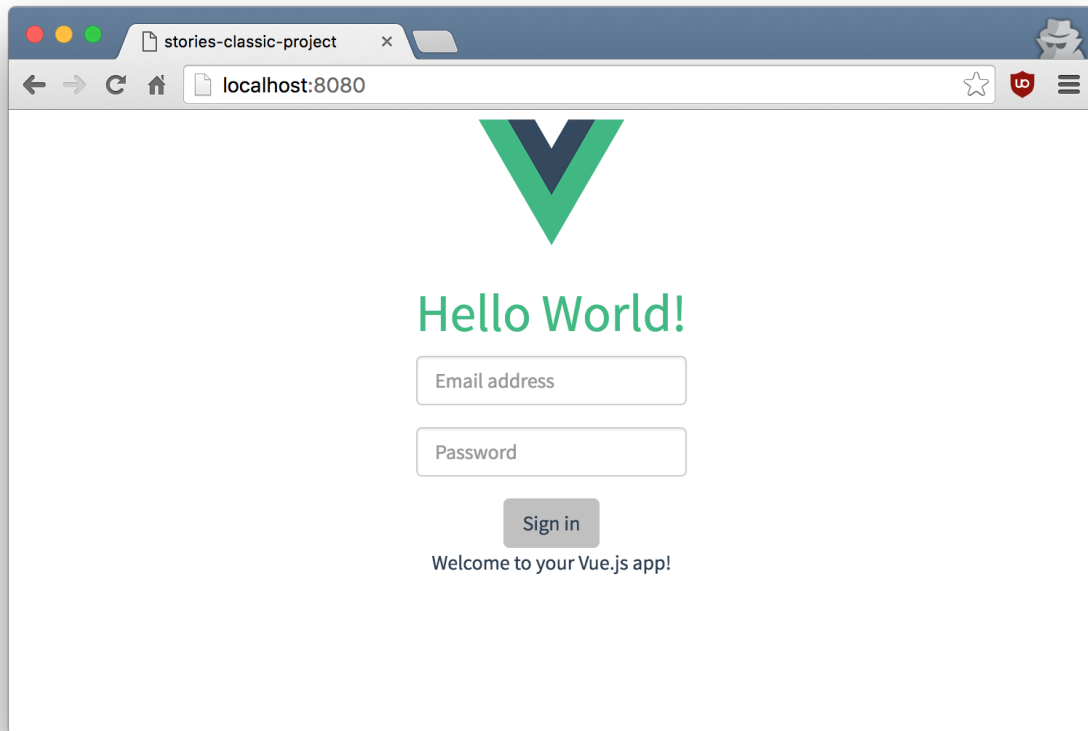
export default {
  name: 'app',
  components: {
    Hello,
    Login
  }
}
</script>
```

```
<style>
...
</style>
```

If you hit reload in your browser you won't see the *Login* component yet, because we need to reference it. Place it under the `<hello>` `</hello>` and you will have a nice login form!

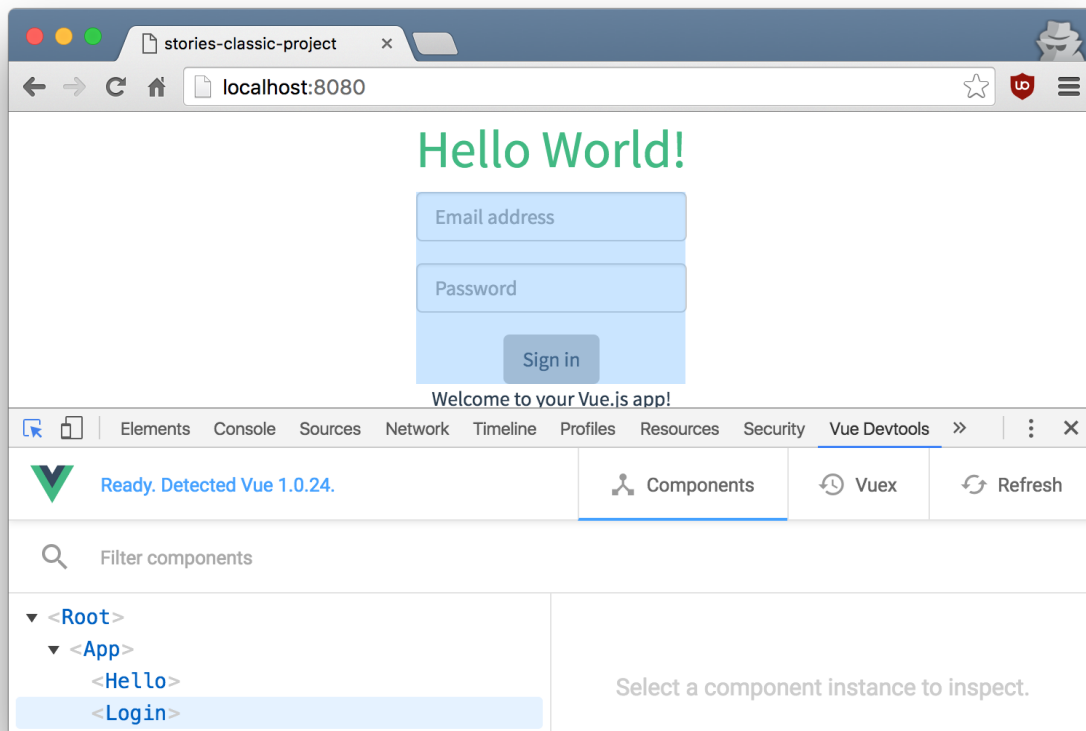
src/App.vue

```
<template>
  <div id="app">
    
    <hello></hello>
    <login></login>
  </div>
</template>
...
...
```



Login Component

If you open the browser's console, you should see the `login` message we are logging when the component is created. If you are using `vue-devtools`, which is highly recommended, you should also see it in the components tree view.



Tree View

Let's create another component, this time for registration.

`src/components/Register.vue`

```
<template>
<div id="register">
  <h2>Register Form</h2>
  <input placeholder="First Name" class="form-control">
  <input placeholder="Last Name" class="form-control">
  <input placeholder="Email address" class="form-control">
  <input placeholder="Pick a password" class="form-control">
  <input placeholder="Confirm password" class="form-control">
  <button class="btn">Sign up</button>
</div>
</template>

<script>
export default {
```

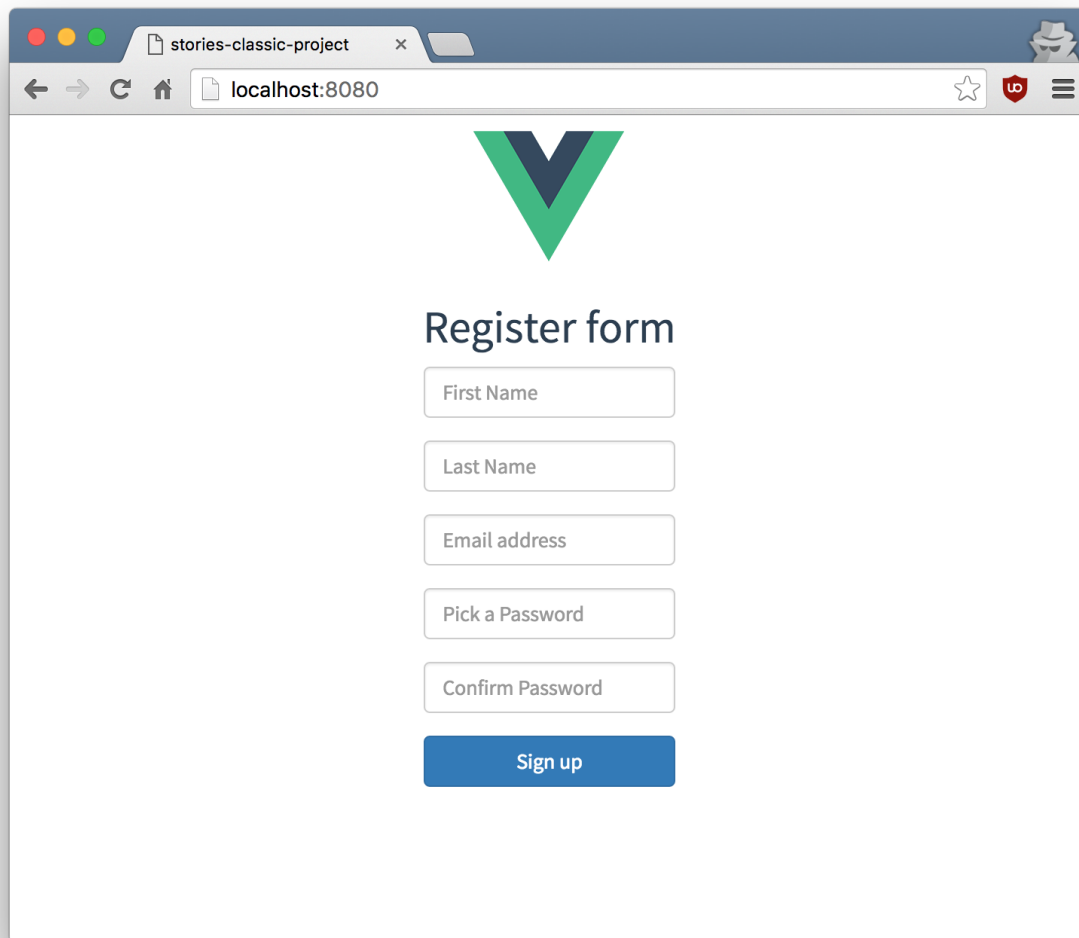
```
    created () {  
      console.log('register')  
    }  
  }  
</script>
```

Then we can import it in the *App.vue* file.

src/App.vue

```
<template>  
  <div id="app">  
    ...  
    <!-- <hello></hello> -->  
    <!-- <login></login> -->  
    <register></register>  
    ...  
  </div>  
</template>  
  
<script>  
// import Hello from './components/Hello'  
// import Login from './components/Login'  
import Register from './components/Register'  
  
export default {  
  name: 'app',  
  components: {  
    // Hello,  
    // Login,  
    Register,  
  }  
}  
</script>  
...  
...
```

The Register component's template appears, when we check the browser.



Register Component



Note

The other components are commented out because we don't want to display them one under the other. The **Hello** component is there by default, but we are not going to use it in any further examples, so we will remove it.

We said that we are working on a social network (or something relevant), so we want a place to display the stories. Thus, we are going to create a **Stories** component which when is rendered, it will bring all the stories told by the users.

src/components/Stories.vue

```
<template>
  <ul class="list-group">
    <li v-for="story in stories" class="list-group-item">
      {{ story.writer }} said "{{ story.plot }}"
      Story upvotes {{ story.upvotes }}.
    </li>
  </ul>
</template>

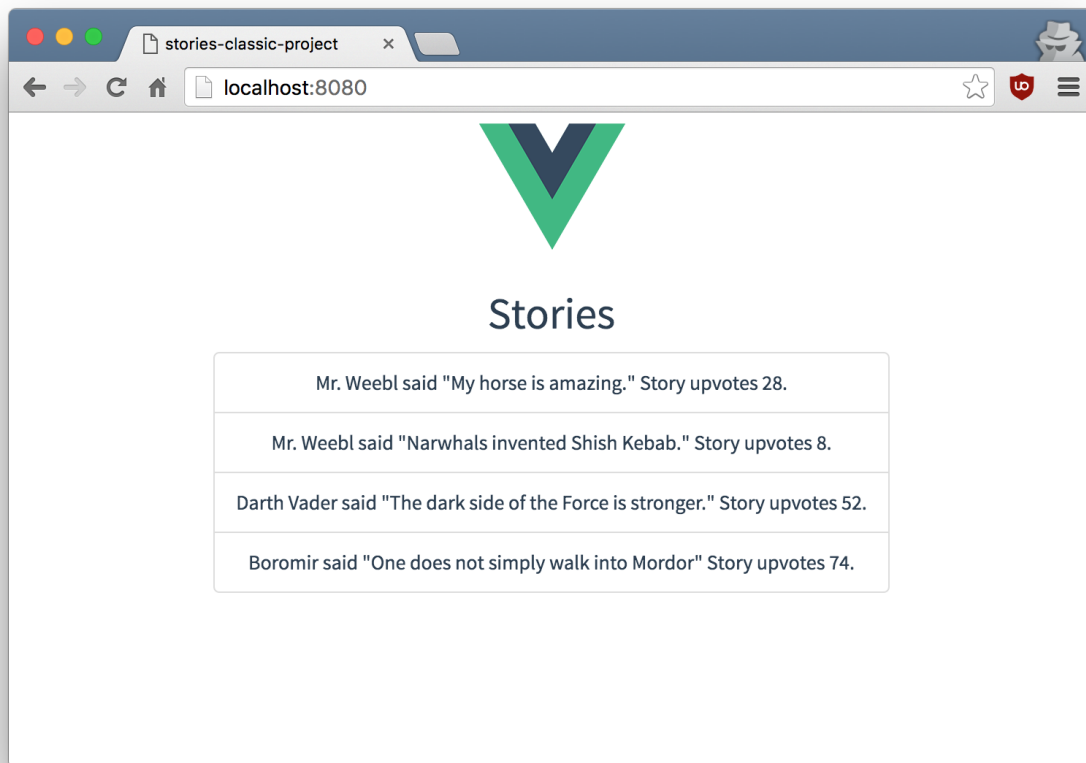
<script>
export default {
  data () {
    return {
      stories: [
        {
          plot: 'My horse is amazing.',
          writer: 'Mr. Weebl',
          upvotes: 28,
          voted: false
        },
        {
          plot: 'Narwhals invented Shish Kebab.',
          writer: 'Mr. Weebl',
          upvotes: 8,
          voted: false
        },
        {
          plot: 'The dark side of the Force is stronger.',
          writer: 'Darth Vader',
          upvotes: 52,
          voted: false
        },
        {
          plot: 'One does not simply walk into Mordor',
          writer: 'Boromir',
          upvotes: 74,
          voted: false
        }
      ]
    }
  }
}
```

```
    }  
  </script>
```

This is the *Stories.vue* file. We can use it in our main *App.vue* file. At this point the stories are hard-coded for simplicity. Time to import it just like the other components.

src/App.vue

```
<template>  
  <div id="app">  
      
    <!-- <login></login> -->  
    <!-- <register></register> -->  
    <stories></stories>  
  </div>  
</template>  
  
<script>  
// import Login from './components/Login.vue'  
// import Register from './components/Register.vue'  
import Stories from './components/Stories.vue'  
export default {  
  components: {  
    Login,  
    Register,  
    Stories  
  }  
}  
</script>  
  
<style>  
...  
</style>
```



Stories Component

Great! Now we have a page to display all the listings.

16.3.1 Nested Components

We would like to be able to display the most “famous” stories, at any place we want to. So after the creation of the *Famous* component, we should be able to use it anywhere.

src/components/Famous.vue

```
<template>
  <div id="famous">
    <h2>Trending stories<strong>({{ famous.length }})</strong></h2>
    <ul class="list-group">
      <li v-for="story in famous" class="list-group-item">
        {{ story.writer }} said "{{ story.plot }}".
        Story upvotes {{ story.upvotes }}.
      </li>
    </ul>
  </div>
```

```
</ul>
</div>
</template>

<script>
export default {
  computed: {
    famous () {
      return this.stories.filter(function (item) {
        return item.upvotes > 50
      })
    }
  },
  data () {
    return {
      stories: [
        {
          plot: 'My horse is amazing.',
          writer: 'Mr. Weebl',
          upvotes: 28,
          voted: false
        },
        {
          plot: 'Narwhals invented Shish Kebab.',
          writer: 'Mr. Weebl',
          upvotes: 8,
          voted: false
        },
        {
          plot: 'The dark side of the Force is stronger.',
          writer: 'Darth Vader',
          upvotes: 52,
          voted: false
        },
        {
          plot: 'One does not simply walk into Mordor',
          writer: 'Boromir',
          upvotes: 74,
          voted: false
        }
      ]
    }
  }
}
```

```
    }  
  }  
</script>
```

This is the whole *Famous.vue* file. We have **filtered** the *stories* array using computed properties, as we saw in previous chapters, and created a **template** to display them.



Note

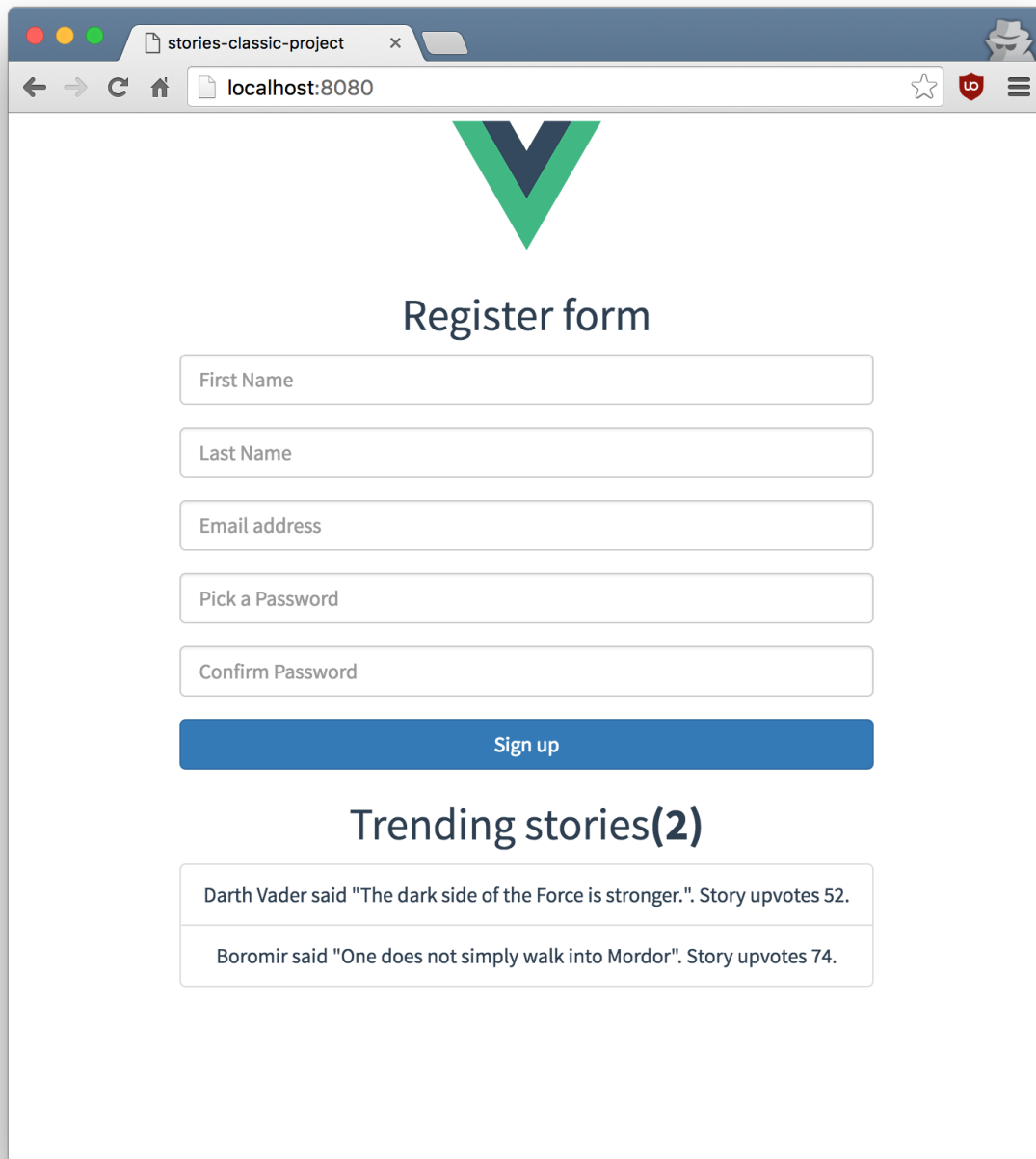
stories array is hard coded again here and data are the same as before. This is a bad practice, we will find a way later to define **stories** array once and share it between all components.

But where could we use this component? An idea is to have it within the registration page, so the user could read the most trending stories and be intrigued. This means - in the current project - that we need to have the *Famous* component within the *Register* one. Well, this can be done the same way we did it inside *App.vue*.

So, open **Register.vue**, import it there, and reference it within the template.

src/components/Register.vue

```
<template>  
  <div id="register">  
    <h2>Register Form</h2>  
    ...  
    <famous></famous>  
  </div>  
</template>  
  
<script>  
import Famous from './Famous.vue'  
  
export default {  
  components: {  
    Famous  
  },  
  created () {  
    console.log('register')  
  }  
}  
</script>
```



The screenshot shows a web browser window with the title 'stories-classic-project' and the address 'localhost:8080'. The page features a large green and blue 'V' logo at the top. Below the logo is the heading 'Register form'. The form consists of five input fields: 'First Name', 'Last Name', 'Email address', 'Pick a Password', and 'Confirm Password'. A blue 'Sign up' button is positioned below the form fields. Underneath the button is the heading 'Trending stories(2)'. Below this heading are two story entries, each in a box: 'Darth Vader said "The dark side of the Force is stronger.". Story upvotes 52.' and 'Boromir said "One does not simply walk into Mordor". Story upvotes 74.'

Registration page with top stories

Pay attention to the import file path. Now that the two files are in the same directory, you have to use `./Famous` instead of the full path. This is an easy mistake to make, especially if you're not familiar with it!



Code Examples

You can find the code examples of this chapter on [GitHub](https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter16/16.3)¹³.

¹³<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter16/16.3>

17. Eliminating Duplicate State

In the previous examples we hardcoded the data -the array of stories- within each component. **This is not a proper way to work with data.**

When more than one component uses the same data, it's a good practice to create/fetch the array once, and then find a way to share it between application's components.

Stories.vue and *Famous.vue* are using the same *stories* array. We will review two ways of sharing the data:

1. Using component properties.
2. Using a global store.

17.1 Sharing with Properties

The first thing we are going to do is to move the *stories* array to App component.

src/App.vue

```
1 <script>
2 ...
3
4 export default {
5   components: {
6     ...
7   },
8   data () {
9     // here we place the stories array
10    return {
11      stories: [
12        {
13          plot: 'My horse is amazing.',
14          writer: 'Mr. Weebl',
15          upvotes: 28,
16          voted: false
17        },
18        {
19          plot: 'Narwhals invented Shish Kebab.',
```

```
20     writer: 'Mr. Weebl',
21     upvotes: 8,
22     voted: false
23   },
24   {
25     plot: 'The dark side of the Force is stronger.',
26     writer: 'Darth Vader',
27     upvotes: 52,
28     voted: false
29   },
30   {
31     plot: 'One does not simply walk into Mordor',
32     writer: 'Boromir',
33     upvotes: 74,
34     voted: false
35   }
36 ]
37 }
38 }
39 }
40 </script>
```

The next step is to remove the `data()` from `Stories` and `Famous` components, and declare `stories` property.

Let's do it for the first component.

`src/components/Stories.vue`

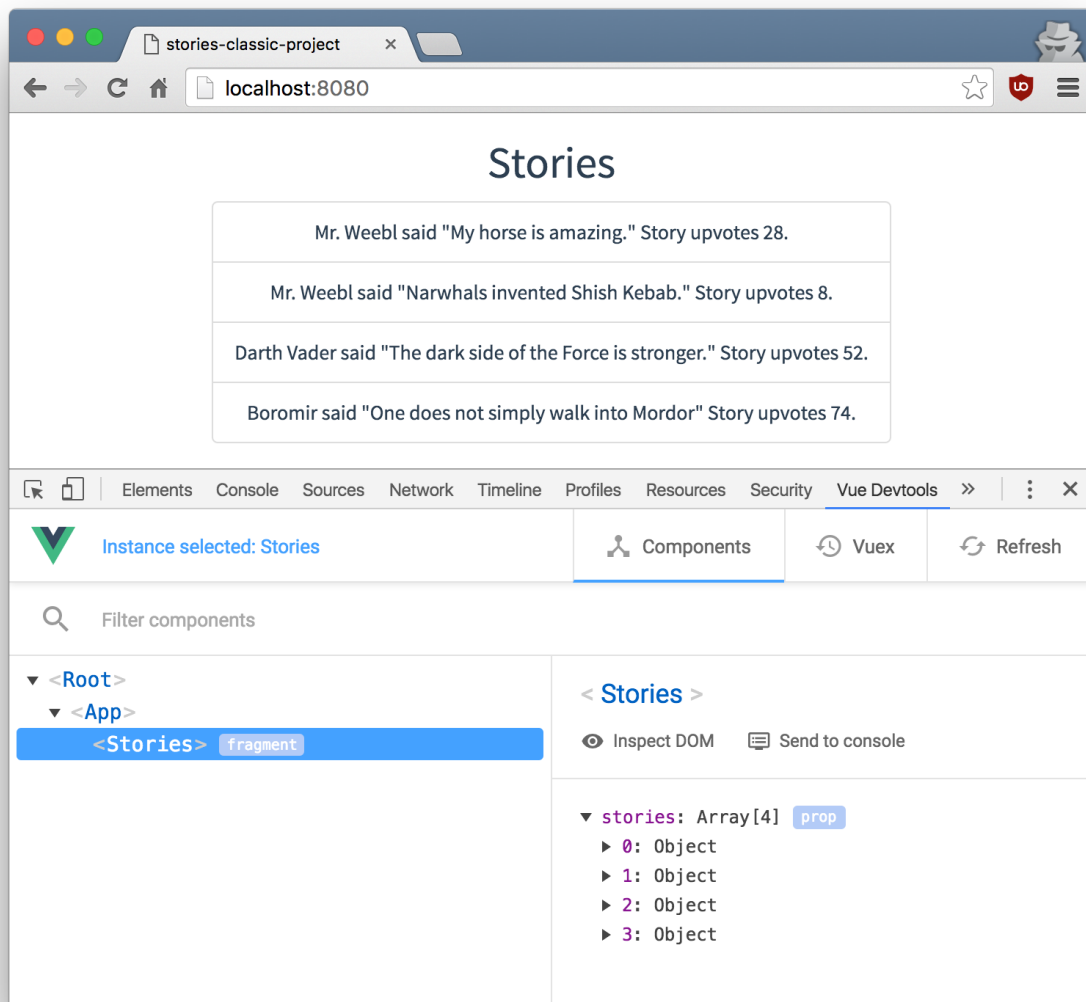
```
1 <script>
2   export default {
3     props: ['stories']
4   }
5 </script>
```

We have to update the way we reference our component within `App.vue`.

src/App.vue

```
1 <template>
2   <div id="app">
3     ...
4     <stories :stories="stories"></stories>
5     ...
6     <p>
7       Welcome to your Vue.js app!
8     </p>
9   </div>
10 </template>
```

Here we bind `stories` prop to `stories` array.



Same output, using props

Success, we got our stories again, fetched from the parent component!

We can't do the same for Famous component because it is not referenced inside `App.vue`. We will have to pass our array to Register component in order to pass it to Famous.

src/App.vue

```
1 <template>
2   <div id="app">
3     ...
4     <register :stories="stories"></register>
5     ...
6   </div>
7 </template>
```

src/components/Register.vue

```
1 <template>
2   <h2>Register Form</h2>
3   ...
4   <famous :stories="stories"></famous>
5 </template>
6
7 <script>
8   import Famous from './Famous'
9
10  export default {
11    components: {
12      Famous
13    },
14    props: ['stories']
15  }
16 </script>
```

src/components/Famous.vue

```
1 <script>
2   export default {
3     props: ['stories'],
4
5     computed: {
6       famous () {
7         return this.stories.filter(function (item) {
8           return item.upvotes > 50
9         })
10      }
11    }
12  }
```

```
12   }  
13 </script>
```

This implementation works, but is not efficient, because Famous component is **not independent**. This means that we cannot use it wherever we want, unless we pass down the data from root component, *App.vue*.

In a scenario where a not independent component is deeply nested, you will have to pass a useless property, from component to component, just to be able to use it. In our case, if we wanted to use Famous inside Register's sidebar's widgets, we would have to carry the stories array all the way long.

App -> Register -> Sidebar -> WidgetX -> Famous

17.2 Global Store

The “props way” seemed nice at first, but as seen in the Famous component, as a project gets bigger and components get nested into others, data management and sharing between them gets really hard to track.

So, let's make the data of our examples a bit easier to handle. We can extract the stories data to a *.js* file, store them to a **constant** and later import them at the desirable locations.

I'll name our *js* file *store.js* and put it under the */src* directory.

src/store.js

```
1 export const store = {  
2   stories: [  
3     {  
4       plot: 'My horse is amazing.',  
5       writer: 'Mr. Weebl',  
6       upvotes: 28,  
7       voted: false  
8     },  
9     {  
10      plot: 'Narwhals invented Shish Kebab.',  
11      writer: 'Mr. Weebl',  
12      upvotes: 8,  
13      voted: false  
14    },  
15    {  
16      plot: 'The dark side of the Force is stronger.',  
17      writer: 'Darth Vader',
```

```
18     upvotes: 52,  
19     voted: false  
20   },  
21   {  
22     plot: 'One does not simply walk into Mordor',  
23     writer: 'Boromir',  
24     upvotes: 74,  
25     voted: false  
26   }  
27 ]  
28 }
```



Warning

The stories prop must be removed from all files, because we have changed the way of data storage and there can be conflicts, which can break our build.

After we have stored all data within *store.js* we can import it within *Stories.vue* using the ES6 modules syntax.

src/components/Stories.vue

```
1  <script>  
2    import {store} from '../store.js'  
3  
4    export default {  
5      data () {  
6        return {  
7          //will give us access to store.stories  
8          store  
9        }  
10     },  
11     created () {  
12       console.log('stories')  
13     }  
14   }  
15 </script>
```

Because we are importing the store object we have to change the component's template as well.

src/components/Stories.vue

```
1 <template>
2   <ul class="list-group">
3     <li v-for="story in store.stories" class="list-group-item">
4       {{ story.writer }} said "{{ story.plot }}"
5       Story upvotes {{ story.upvotes }}.
6     </li>
7   </ul>
8 </template>
```

We are using `v-for` to render the items of the array (`store.stories`). Our list of stories is displaying as before.

We could do the same thing without having to change the template, by binding component's `stories` attribute to `store.stories` directly.

src/components/Stories.vue

```
1 <script>
2   data () {
3     return {
4       // Bind directly to stories
5       stories: store.stories,
6     }
7   }
8 </script>
```

The same thing applies for *Famous.vue*.

src/components/Famous.vue

```
1 <script>
2   import {store} from '../store.js'
3
4   export default {
5     data () {
6       return {
7         stories: store.stories
8       }
9     },
10    computed: {
11      famous () {
```

```
12         return this.stories.filter(function (item) {  
13             return item.upvotes > 50  
14         })  
15     }  
16 }  
17 }  
18 </script>
```

If we hadn't bind to `stories`, `famous()` computed property would have to be updated to filter `this.store.stories`.

Once you get used to work with global objects I believe you are going to **love it!** :)



Code Examples

You can find the code examples of this chapter on [GitHub](https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter17)¹.

¹<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter17>

18. Swapping Components

Using Single File Components is the simplest way to build a Single Page Application with Vue. We've seen so far how to setup a fresh project, create `.vue` files and manage duplicate state. Now it's time to review a way to swap view-specific components.

For reference, in the previous examples, we had 3 components inside `App.vue` and some others nested within. We need to find a way to swap components dynamically, so they won't be rendered on the page simultaneously.

18.1 Dynamic Components

18.1.1 The `is` special attribute

We can use the reserved `<component>` element and use the same mount point to dynamically switch between multiple components, using the `is` special attribute.

src/App.vue

```
<template>
  <div id="app">
    <component is="hello"></component>
    <p>
      This is very useful...
    </p>
  </div>
</template>

<script>
import Hello from './components/Hello'
// Component Hello returns a template containing a "msg" property of data
export default {
  components: {
    Hello
  }
}
```

We've created a fresh project and modified the *Hello.vue* file. We have the exact same result as before, but here we are using the `<component is="hello">` element. The `Hello` component is bound to the `is` attribute. To see how this would work dynamically, check the next example where we are changing between 2 different components by clicking on their links.

Firstly, create a similar component with a different message, named *Greet.vue*.

src/Greet.vue

```
<template>
  <div class="greet">
    <h1>{{ msg }}</h1>
  </div>
</template>

<script>
export default {
  data () {
    return {
      msg: 'No! I want to use the <component> element!'
    }
  }
}
</script>
```

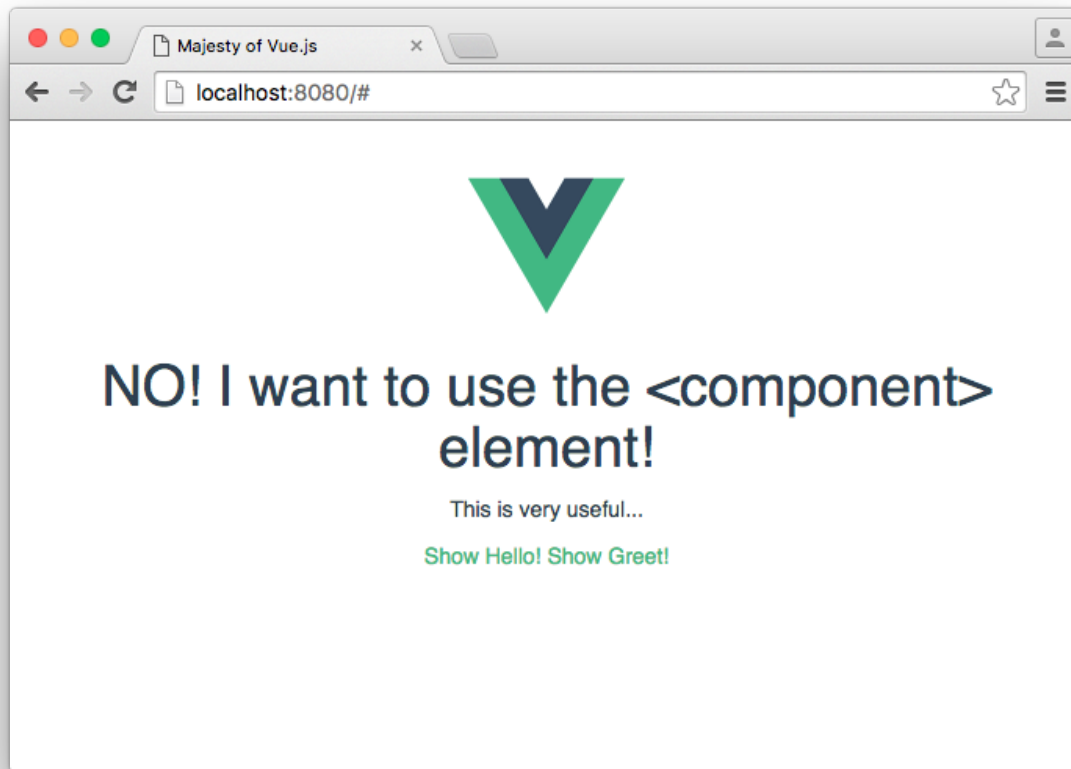
We made `Greet` to display a message to manifest its presence! Let's import it into `App` and give the user the ability to swap between the 2 components.

src/App.vue

```
<template>
  <div id="app">
    
    <component :is="currentComponent">
      <!-- component changes when this.currentComponent changes! -->
    </component>
    <p>
      This is very useful...
    </p>
    <a href="#" @click="currentComponent = 'hello'">Show Hello</a>
    <a href="#" @click="currentComponent = 'greet'">Show Greet</a>
  </div>
</template>
```

```
<script>
import Hello from './components/Hello'
import Greet from './components/Greet'

export default {
  components: {
    Hello,
    Greet
  },
  data () {
    return {
      currentComponent: 'hello'
    }
  }
}
</script>
```



Greet.vue

Well, as you can see, we are binding the special attribute `is` to `currentComponent`, so when its value changes, the displaying component will also change. To swap the view, the user just have to click on either link to change the value of `currentComponent`.

This dynamic way of switching between multiple components can prove handy.

18.1.2 Navigation

In the [previous examples](#), we used `.vue` files to simulate a social network, where we had components like `Login`, `Registration`, etc. Now we can use a tab system to navigate between these components with style.

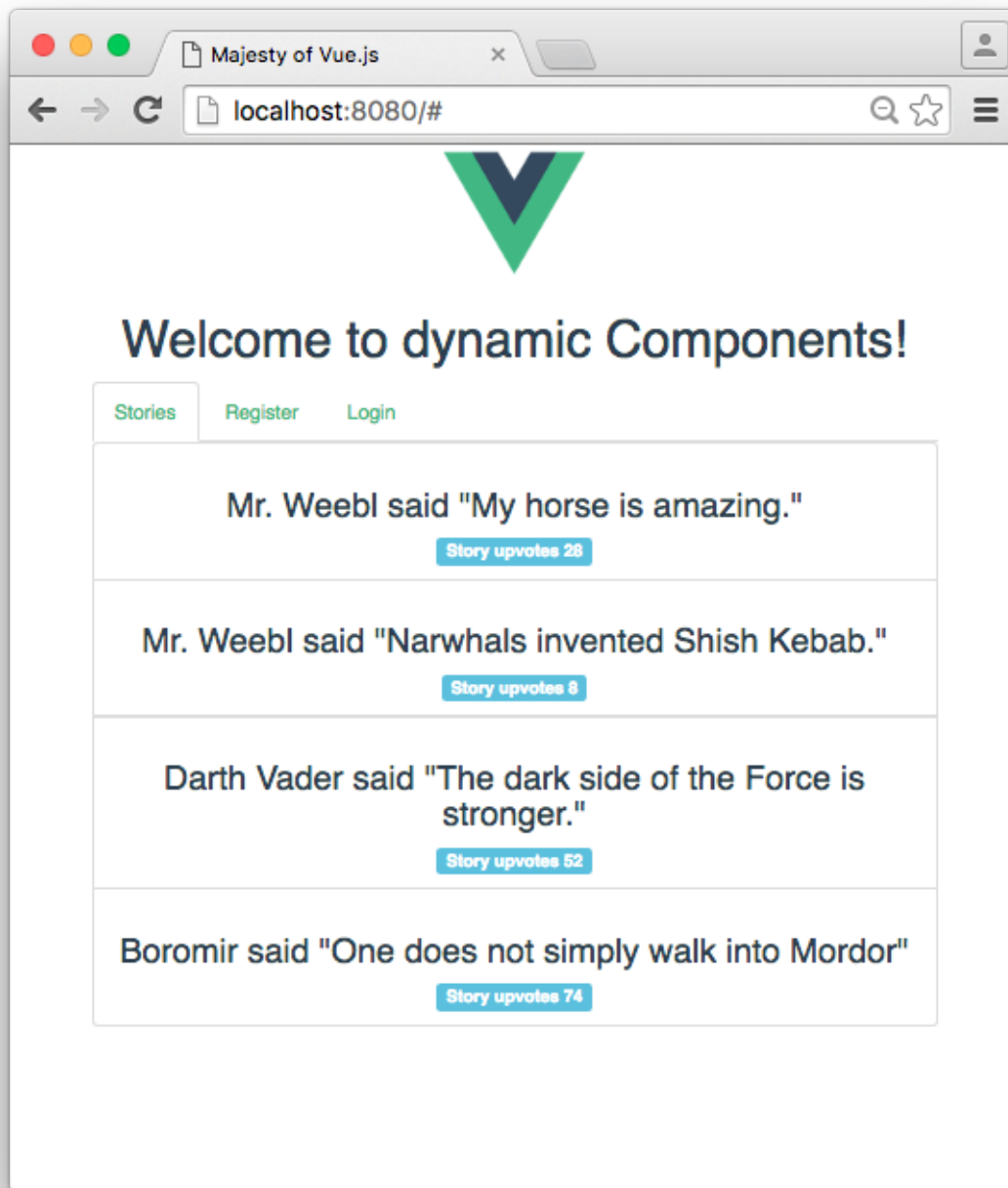
We are going to have `Stories.vue` in one tab, `Register.vue` in another, and `Login.vue` in a third. Don't forget that `Register` contains the `Famous` component, which returns the most trending stories.

Read thoroughly the next example.

src/App.vue

```
1 <template>
2   <div id="app">
3     
4     <h1>Welcome to dynamic Components!</h1>
5     <ul class="nav nav-tabs">
6       <!-- set 'active' class conditionally -->
7       <li v-for="page in pages" :class="isActivePage(page) ? 'active' : ''">
8         <!-- use links to change between tabs -->
9         <a @click="setPage(page)">{{page | capitalize}}</a>
10      </li>
11    </ul>
12    <component :is="activePage"></component>
13  </div>
14 </template>
15
16 <script>
17 import Vue from 'vue'
18 import Login from './components/Login.vue'
19 import Register from './components/Register.vue'
20 import Stories from './components/Stories.vue'
21
22 Vue.filter('capitalize', function (value) {
23   return value.charAt(0).toUpperCase() + value.substr(1)
24 })
25
26 export default {
27   components: {
28     Login,
29     Register,
30     Stories
31   },
32   data () {
33     return {
34       // the pages we want to render each time
35       pages: [
36         'stories',
37         'register',
38         'login'
39       ],
40       activePage: 'stories'
41     }
```

```
42   },
43   methods: {
44     setPage (newPage) {
45       this.activePage = newPage
46     },
47     isActivePage (page) {
48       return this.activePage === page
49     }
50   }
51 }
52
53 </script>
```



A page for each component

Let's break it down.

The array named `pages` contains the components which we would like to render. We are using the `v-for` directive to create a tab for each one.

To navigate between the tabs, we've created a method called `setPage`.

The `activePage` property is initially set to `'stories'`. When a tab is clicked, `activePage` changes in order to reflect the name of the component we wish to display.

To determine which tab must be active, an in-line `if` is applied, which sets the class `active` whether the current `activePage` property matches the current component's name.

To make the first letter of each tab capitalize, we've created a `Vue.filter()`, named `capitalize`, which is used within text interpolations.

With these few and simple lines of code, we have accomplished a simple navigation system, swapping between our components.



Code Examples

You can find the code examples of this chapter on [GitHub](https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter18)¹.

¹<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter18>

19. Vue Router

Routing, in general, refers to determining how your application responds to a client request. A web browser request wouldn't be directed to your application without some form of routing. Router helps a web server to fetch an appropriate and exact information for the user. It is like a station master at a railway station, who informs the train operator when to change tracks.

The way of swapping views we just studied, paves the way for routing. The official router for Vue.js is called **vue-router**. It is deeply integrated with Vue.js core to make building Single Page Applications a breeze. This plugin is relatively easy to understand, install, and use.

The main features are:

- Nested route/view mapping
- Modular, component-based router configuration
- Route params, query, wildcards
- View transition effects powered by Vue.js' transition system
- Fine-grained navigation control
- Links with automatic active CSS classes
- HTML5 history mode or hash mode, with auto-fallback in IE9
- Restore scroll position when going back in history mode



Info

These are just some of the provided features, you can see more on [Github](https://github.com/vuejs/vue-router)¹. Also, here is the [official documentation](http://router.vuejs.org/en/index.html)².

19.1 Installation

There are the usual ways to install the plugin; using cdn, NPM, and Bower. We are going to use the terminal to install it via NPM.

```
> npm install vue-router
```

Type this command in your terminal to have it installed in your *node_modules* folder inside your project's directory. After it is complete, go to your *main.js* file and add the following lines.

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

²<http://router.vuejs.org/en/index.html>

src/main.js

```
import Vue from 'vue'
import VueRouter from 'vue-router'
Vue.use(VueRouter)
```

You can install a Vue.js plugin using `Vue.use()` as shown. For more information about `Vue.use` check the [guide](#)³.

19.2 Usage

The first step is to create a router instance, where we will pass extra options later on, but let's keep it simple for now.

src/main.js

```
...
const router = new VueRouter({
  routes // short for routes: routes
})
```

Now, we have to define some routes. Each route should map to a component, which means that we are going to create routes for the `*.vue` files we've been using all along in our examples.

The main method to define route mappings for the router, is to create a routes array where we can pass route objects.

src/main.js

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import Hello from '../src/components/Hello.vue'
import Login from '../src/components/Login.vue'

Vue.use(VueRouter)

const routes = [
  { path: '/', component: Hello },
  { path: '/login', component: Login }
]
```

³<http://vuejs.org/api/#Vue-use>

Inside the array we define 2 routes.

1. When `http://localhost:3000/` is met (or any port you might use), the default `Hello.vue` will be rendered
2. When `http://localhost:3000/login` is met, the `Login.vue` component will be rendered.



Info

We are using `{ path: '/login', component: Login }` because the `Login` component is imported on top of the code. Alternatively you could use `require()` function like this: `{ path: '/login', component: require('Login') }`

The next step is to create an outlet for the router. The guide states that the router needs a root component to render. In our case, we will use the `App.vue` component as root.

Let's create and mount the root instance. Keep in mind that we have to inject the router with the router option to make the whole app *router-aware*.

`src/main.js`

```
...
/* eslint-disable no-new */
new Vue({
  el: '#app',
  router,
  template: '<app></app>',
  components: {
    App
  }
})
```

We set the template to `<app></app>` so Vue will replace the `div` with the id of `app` with `App.vue` component's template. Since we are already importing `App` in our `main.js` file, we are set here.

If you neglect to include the comment `/* eslint-disable no-new */` you will get an error from `eslint` stating:

<http://eslint.org/docs/rules/no-new> Do not use 'new' for side effects

What we have to do here is to disable that rule.



Tip

Every time you find yourself struggling with an `eslint` rule, you can disable it by adding a comment like the above. For example `/* eslint-disable eqeqeq */`. You can find a complete list with the rules [here](http://eslint.org/docs/rules/)⁴.

⁴<http://eslint.org/docs/rules/>

Now, we need to define some links for the navigation. Head to `App.vue`, to make the changes required.

`src/App.vue`

```
<template>
  <div id="app">
    
    <h1>Welcome to Routing!</h1>
    <router-link to="/">Home</router-link>
    <router-link to="/login">Login</router-link>
    <!-- route outlet -->
    <router-view></router-view>
  </div>
</template>
```

The `<router-view></router-view>` is the place where all the magic happens while components are being rendered for us.

`router-link` is the component that lets users navigate in a router-enabled app. The `to` property defines the target location, matching a route defined in `main.js`. By default, `router-link` will render an `<a>` tag with the `href` attribute set to the desired URI. It can take more arguments for more complex navigations, which we will review soon.



Note

If you are using the vue-cli template with router included, you have to define your routes array within `router/index.js`.

19.3 Named Routes

While the options we reviewed for routing serve our needs in a small project, like our example, presumably you will need more options as your project grows. For instance, if we decide later to change `/login` url to `/signin`, we will have to update all the links directing to the login page. To prevent this from happening, we can give each route a name.

To name a route we have to alter the route configuration.

src/main.js

```
...
const routes = [
  {
    path: '/',
    name: 'home',
    component: Hello
  },
  {
    path: '/login',
    name: 'login',
    component: Login
  }
]
```

We can give a name to a route by adding a **name** property and use it as identifier to link it afterwards.

src/App.vue

```
<template>
  <div id="app">
    ...
    <router-link :to="{ name: 'home' }">Home</router-link>
    <router-link :to="{ name: 'login' }">Login</router-link>
    <!-- route outlet -->
    <router-view></router-view>
  </div>
</template>
```

Notice here that instead of using a string to define the destination of the link (`to="/home"`), we are using an object (`:to="{ name: 'home' }"`). We will elaborate on that later.

19.4 History mode

Before moving on, I would like to point out something regarding the browser URL, when using vue-router. As you have seen, when a route changes, a `#` symbol is appended to the URL. For instance, the URL is `*/login`, when we navigate to the login page.

This is caused by the default mode for vue-router, hash mode, which uses the URL hash to simulate a full URL so that the page won't be reloaded when the URL changes. To get rid of the hash, we can change to **history mode**. Also we will set another option, **base**, which defines a root path for all

router navigations. Changing this to anything from its initial value, that equals `default`, will result in paths which will always include the new value in the actual browser URL.

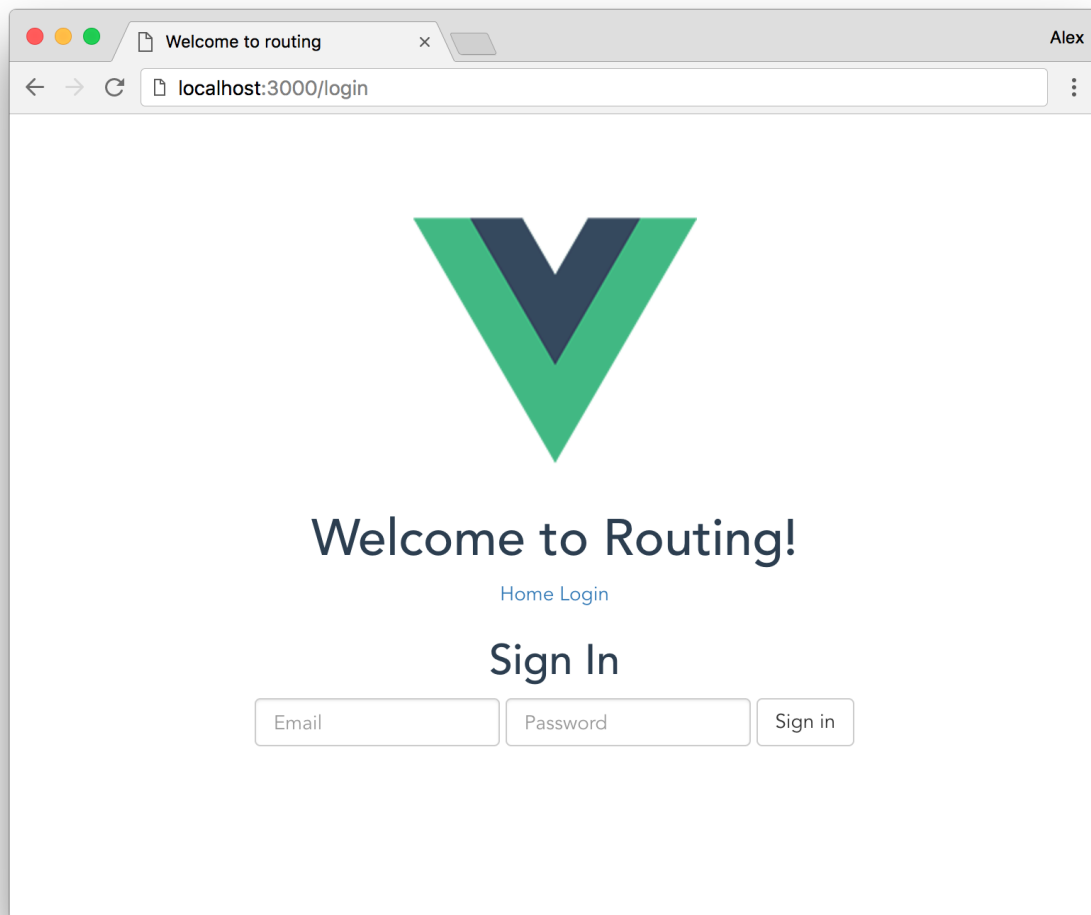
For example if we set **base** to `/vuejs` the login page will be `/vuejs/login`.

You can set anything to be the **base**, here let's just put `/`.

`src/main.js`

```
const router = new VueRouter({  
  mode: 'history',  
  base: '/',  
  routes  
})
```

This relieves us from `#` in the URLs.



Neat URLs



Info

Check the detailed list of available options on vue-router's [documentation](https://router.vuejs.org/en/api/options.html)⁵

19.5 Nested routes

Nested routes, are routes that live within other routes. Mapping nested routes to components is a common need, and it's also very simple with vue-router.

To demonstrate it, we are going to add a page to display the stories with 2 sub-pages.

⁵<https://router.vuejs.org/en/api/options.html>

- One to display all stories (StoriesAll.vue)
- One to display famous stories (StoriesFamous.vue)

We will create the above-mentioned components, plus one, which will be the wrapper, similar to App.vue.

Let's start by registering the new routes. All we have to do is to use the children option in the routes array and add a nested `<router-view>` within our wrapper view, StoriesPage.vue.

src/main.js

```
import Vue from 'vue'
import App from './App'

import Hello from './components/Hello.vue'
import Login from './components/Login.vue'
import StoriesPage from './components/StoriesPage.vue'
import StoriesAll from './components/StoriesAll.vue'
import StoriesFamous from './components/StoriesFamous.vue'
```

```
import VueRouter from 'vue-router'
```

```
Vue.use(VueRouter)
```

```
const routes = [
  {
    path: '/',
    component: Hello
  },
  {
    path: '/login',
    component: Login
  },
  {
    path: '/stories',
    component: StoriesPage,
    children: [
      {
        path: '',
        name: 'stories.all',
        component: StoriesAll
      },
      {
        path: 'famous',
```

```
      name: 'stories.famous',
      component: StoriesFamous
    }
  ]
}
```

Notice here that the path for `StoriesAll` is set to `''`. This means that it is the default child route and will be rendered when `/stories` is matched. You can also use `'/'` to define a default route.

The contents of `StoriesFamous` will be rendered when `/stories/famous` is matched.

At this point, there is no need to show what's inside these components. They both just display an array of stories.

Our wrapper component, `StoriesPage`, contains 2 links and the `<router-view>` tag, to render its child components' contents.

src/StoriesPage.vue

```
<template>
  <div>
    <h2>Stories</h2>
    <!-- navigation -->
    <router-link :to="{name: 'stories.all'}">All</router-link>
    <router-link :to="{name: 'stories.famous'}">Trending</router-link>
    <!-- route outlet -->
    <router-view></router-view>
  </div>
</template>
```

19.6 Auto-CSS active class

Wouldn't be nice if we highlighted the link that directs to the active page? Vue Router is smart enough to append a css class to the active link. This class is `vue-router-active`.

All we have to do is to add a rule to style it in our css. I will add it to `App.vue` component.

src/App.vue

```
...  
<style type="text/css">  
  .router-link-active {  
    color: green;  
  }  
</style>
```

So, now, every time we visit a page, the corresponding link turns green..

If you try this out in the browser, you will notice that the Home link is always green. This happens because Home's path is /, so when you visit for example /login, Home remains active. To get rid of this behavior we can add the `exact` prop to this specific link.

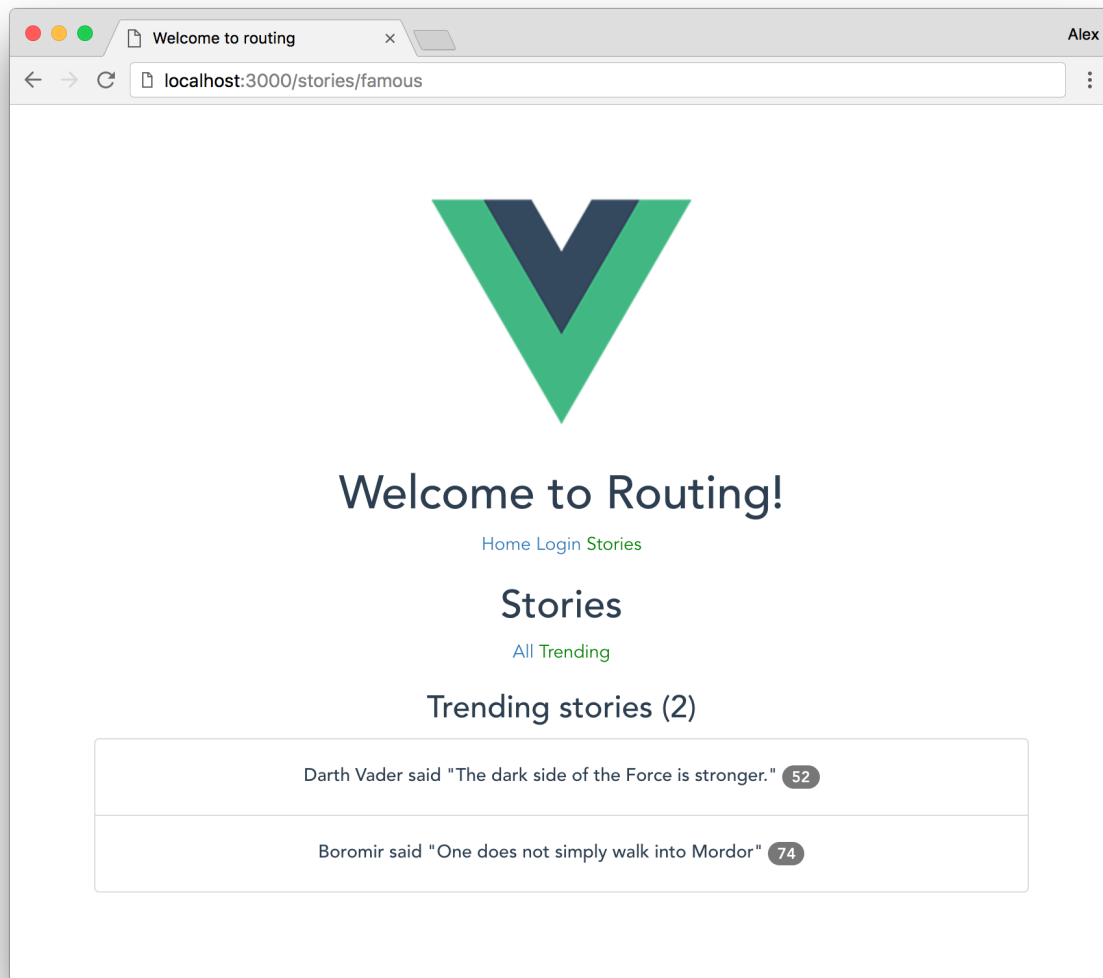
Our navigation links will look like this:

src/App.vue

```
<template>  
  <div>  
    ...  
    <router-link :to="{ name: 'hello' }" exact>Home</router-link>  
    <router-link :to="{ name: 'login' }">Login</router-link>  
    <router-link :to="{ name: 'stories.all' }">Stories</router-link>  
    <router-view></router-view>  
  </div>  
</template>
```

We have append `exact` to the first link within `StoriesPage.vue` too.

The nice part is that the active link is highlighted in the secondary navigation as well.



Active Class

19.6.1 Custom Active Class

You can change the name of the active class (`router-link-active`) for a **specific link**, using the `active-class` property or **globally** by using the `linkActiveClass` router constructor option.

Local custom active class

```
<router-link :to="{ name: 'hello'}" active-class="my-active-class" exact>
  Home
</router-link>
```

Global custom active class

```
const router = new VueRouter({
  mode: 'history',
  base: '/',
  linkActiveClass: 'my-active-class',
  routes
})
```

19.7 Route Object

All information of a parsed route will be accessible on the exposed **Route Context Object**, also called **route object**.

The route object will be injected into every component in a router-enabled app and will be accessible as **this.\$route**. It will be updated whenever a route transition is performed.

Below is a list with **\$route** object's properties.

Property	Description
<i>path</i>	A string that equals the path of the current route, always resolved as an absolute path. e.g. <code>/foo/bar</code> .
<i>params</i>	An object that contains key/value pairs of dynamic segments and star segments. More details below.
<i>query</i>	An object that contains key/value pairs of the query string. For example, for a path <code>/foo?user=1</code> , we get <code>\$route.query.user == 1</code> .
<i>hash</i>	The hash of the current route (without <code>#</code>), if it has one. If no hash is present the value will be an empty string.
<i>fullPath</i>	The full resolved URL including query and hash.
<i>matched</i>	An Array containing route records for all nested path segments of the current route. Route records are the copies of the objects in the routes configuration Array (and in children Arrays).
<i>name</i>	The name of the current route, if it has one.

19.8 Dynamic Segments

Vue Router provides the ability to form paths using dynamic segments. Dynamic segments are segments with a leading colon. They are called dynamic because their value is changeable.



Info

URL segments are the parts of a URL or path delimited by slashes. If you had the path `/user/:id/posts`, then `user`, `:id`, and `posts` would each be a segment.

In this path, the `:id` is the dynamic segment and will match any provided value, for instance `/user/11/posts`, `/user/37/posts`, etc.

When a path containing a dynamic segment is matched, the dynamic segments will be available inside `$route.params`.

In our example, we can use a dynamic segment to access a certain story by its `id`, in order to create a view where we can edit it.

`src/main.js`

```
1 const routes = [  
2   // other routes  
3   {  
4     path: ':id/edit',  
5     name: 'stories.edit',  
6     component: StoriesEdit  
7   }  
8   ...  
9 ]
```

Now, we need a way to link `StoriesAll.vue` with `StoriesEdit.vue`. Let's manipulate the file.



Note

I have created the `StoriesEdit.vue` file in the background. You will see it soon, after the routing for it is complete.

src/component/StoriesAll.vue

```
1 <template>
2   <div class="">
3     <h3>All Stories ({{stories.length}})</h3>
4     <ul class="list-group">
5       <li v-for="story in stories" class="list-group-item">
6         <div class="row">
7           <h4>{{ story.writer }} said "{{ story.plot }}"
8             <span class="badge">{{ story.upvotes }}</span>
9           </h4>
10          <router-link
11            :to="{ name: 'stories.edit' }" tag="button"
12            class="btn btn-default" exact
13          >
14            Edit
15          </router-link>
16        </div>
17      </li>
18    </ul>
19  </div>
20 </template>
21
22 <script>
23 import {store} from '../store.js'
24
25 export default {
26   data () {
27     return {
28       stories: store.stories
29     }
30   },
31   mounted () {
32     console.log('stories')
33   }
34 }
35 </script>
```



Note

Our example files are almost the same as before, with minor alterations, mostly styles, which do not affect their functionality. A noteworthy change is the addition of an `id` to each story within `store.js`.

We've added a button to link the `stories.edit` route. Well, this is not enough, because we also need to pass story's `id` to the route.

To do so, we are going to edit the `to` prop and make the `:id` turn into the corresponding `id` of each story.

src/component/StoriesAll.vue

```
1 <template>
2   <div>
3     <h3>All Stories ({{stories.length}})</h3>
4     <ul class="list-group">
5       <li v-for="story in stories" class="list-group-item">
6         <h4>{{ story.writer }} said "{{ story.plot }}"
7         <span class="badge">{{ story.upvotes }}</span>
8       </h4>
9       <router-link
10        :to="{ name: 'stories.edit', params: { id: story.id }}"
11        tag="button" class="btn btn-default" exact>
12         Edit
13       </router-link>
14     </li>
15   </ul>
16 </div>
17 </template>
```

Here we want `<router-link>` to render a `<button>` tag instead of `<a>`. We can use the `tag` property to specify which tag to render. The rendered tag will listen to click events for navigation.

Within `$route.params`, the `id` of the chosen story is available when we reach our destination. With this at hand, we can pick out the story that the user wants to edit, and bring it to him.

It's time to show the `StoriesEdit.vue` file, where the editing will take place.

src/components/StoriesEdit.vue

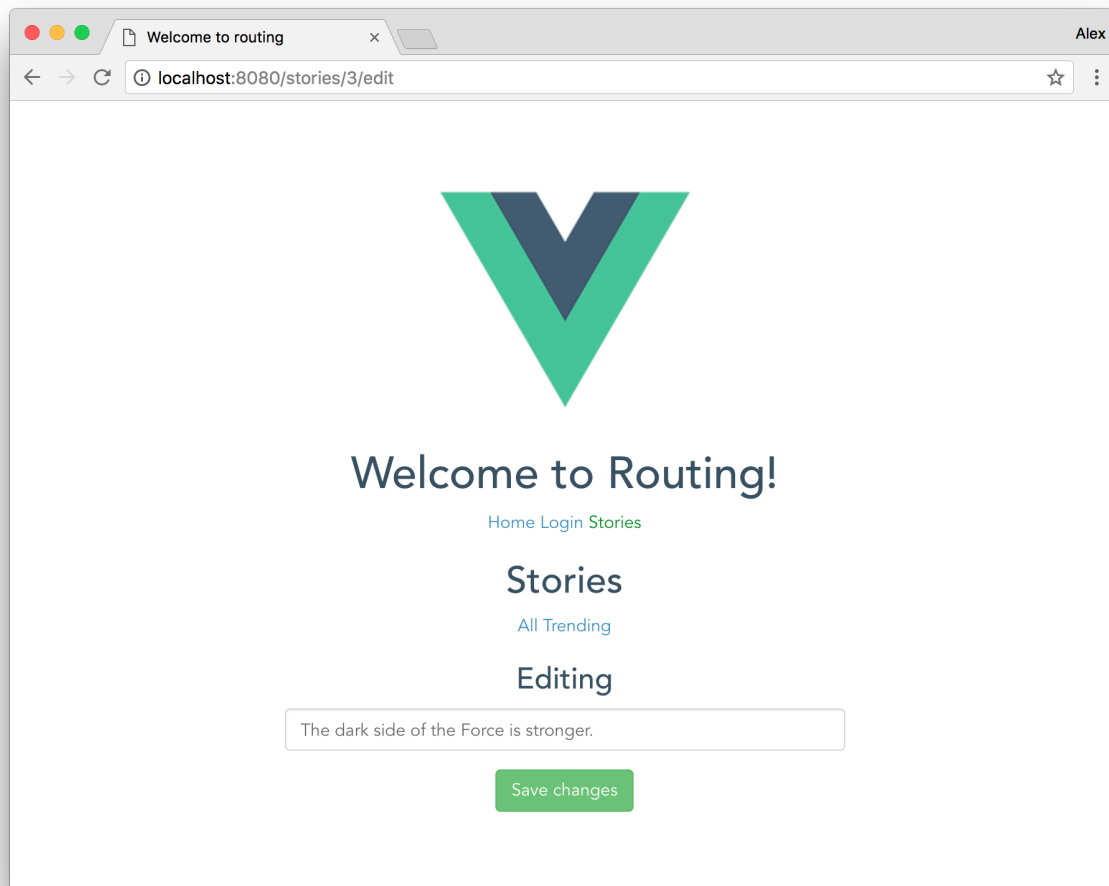
```
1 <template>
2   <div class="row">
3     <h3>Editing</h3>
4     <form>
5       <div class="form-group col-md-offset-2 col-md-8">
6         <input class="form-control" v-model="story.plot">
7       </div>
8       <div class="form-group col-md-12">
9         <button @click="saveChanges(story)" class="btn btn-success">
```

```
10         Save changes
11     </button>
12 </div>
13 </form>
14 </div>
15 </template>
16
17 <script>
18 import {store} from '../store.js'
19
20 export default {
21   data () {
22     return {
23       story: {}
24     }
25   },
26   methods: {
27     isTheOne (story) {
28       return story.id === this.id
29     },
30     saveChanges (story) {
31       // we will use that later
32     }
33   },
34   mounted () {
35     this.story = store.stories.find(this.isTheOne)
36   }
37 }
38 </script>
```

We are using story's id, to pick the desired story out of stories array with JavaScript's [find method](#)⁶.

The chosen story is ready for editing. Notice that the id of the story is shown in the URL.

⁶https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find



Editing selected story

This works fine if you visit the `StoriesEdit` page using the button but if you try to type the URL directly into the browser's address bar, e.g. `/stories/2/edit`, you'll get an error and the component won't render.

The reason behind this is that we used [strict equality](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Identity)⁷ (`===`) to find the right active story. When visiting the page directly, `id` is passed as string (not number). So, `isTheOne` always returns false.

⁷https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Identity

src/components/StoriesEdit.vue

```
isTheOne (story) {  
  // returns false when this.id is not a number  
  return story.id === this.id  
}
```

The easy-fix is to convert the `id` to number, using the **Number** JavaScript object wrapper.

src/components/StoriesEdit.vue

```
export default {  
  ...  
  mounted () {  
    this.id = Number(this.$route.params.id)  
  }  
}
```

If you check this in the browser you will find out that the story comes up as if we clicked the edit button.

Though, there is a much better solution which helps us decouple the `StoriesEdit` component from `$route`. We can cast the `id` parameter to number within route's configuration and use it as a prop in the `StoriesEdit` component.

src/main.js

```
const routes = [  
  ...  
  {  
    path: ':id/edit',  
    props: (route) => ({ id: Number(route.params.id) }),  
    name: 'stories.edit',  
    component: StoriesEdit  
  },  
  ...  
]
```

Now, when `stories/:id/edit` is met, router will pass the numeric value of `:id` as a property to the **StoriesEdit** component. What's missing is to define that property and remove `this.$route.params.id`. So, our component will look like

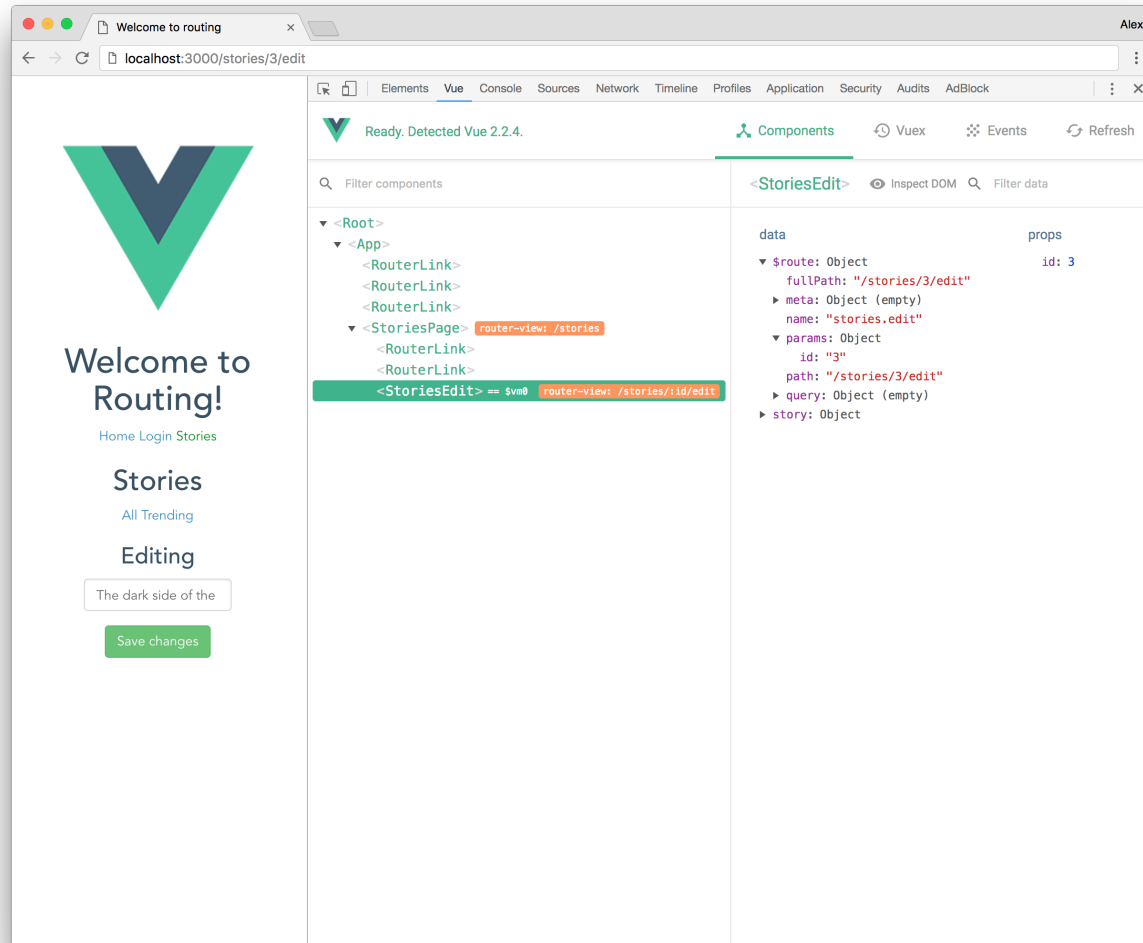
src/components/StoriesEdit.vue

```
<script>
import {store} from '../store.js'

export default {
  props: ['id'],
  data () {
    return {
      story: {}
    }
  },
  methods: {
    isTheOne (story) {
      return story.id === this.id
    }
  },
  mounted () {
    this.story = store.stories.find(this.isTheOne)
  }
}
</script>
```

That's it. Now the `StoriesEdit` component is decoupled from the route and we can use it anywhere like this: `<stories-edit :id="story.id"></stories-edit>`.

You can have a look at the attributes of the `$route` object, using Vue Devtools.



Inside \$route

19.9 Route Alias

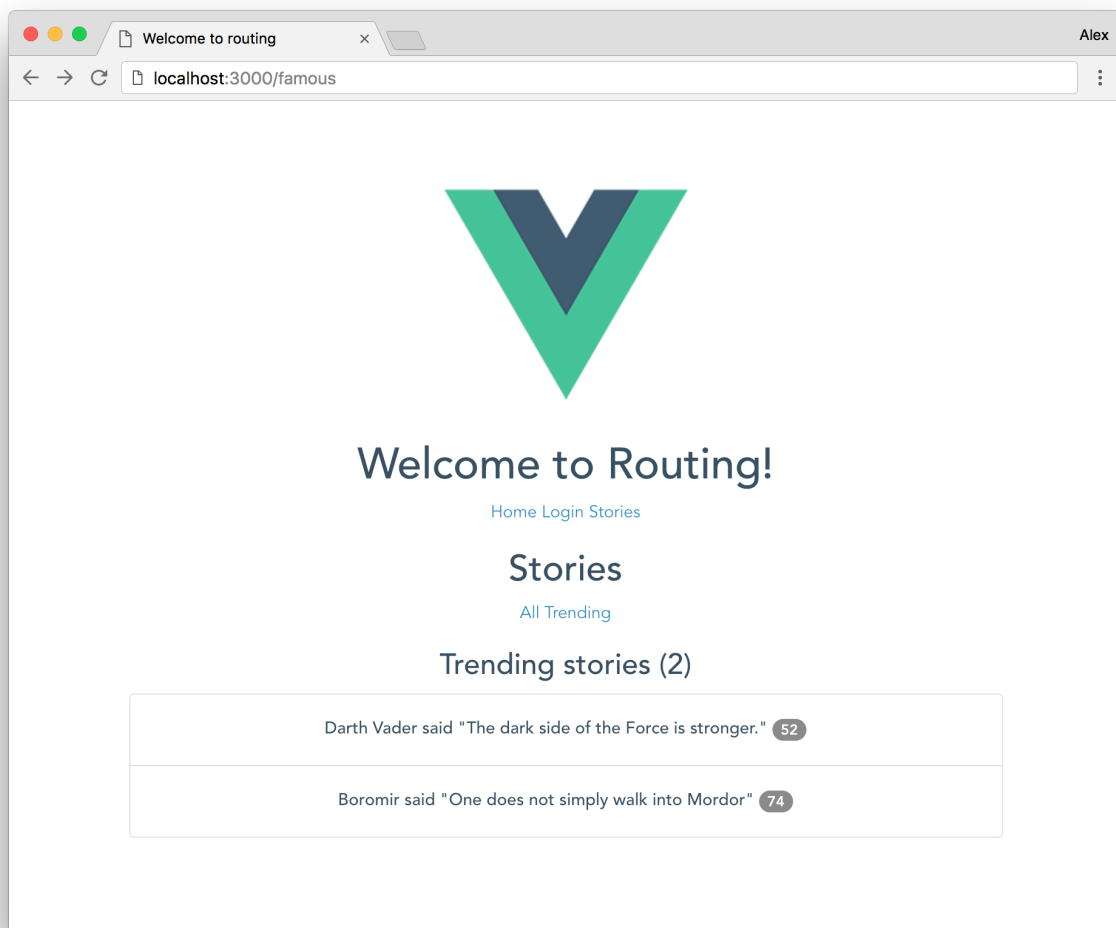
When we define new routes, we usually try to make them clear and representative. Sometimes though, we might end up with long paths or complex ones, which can be difficult to handle later.

When we want to view the famous routes through the browser, we have to visit `' /stories/famous '`. We can make this shorter by defining a global alias for this route, where a shorter URL would lead us to the same place.

src/main.js

```
{
  path: 'famous',
  name: 'stories.famous',
  // match '/famous' as if it is '/stories/famous'
  alias: '/famous',
  component: StoriesFamous
}
```

Using this configuration, we can just use the `alias` instead of the `path`.



Using route alias

19.10 Programmatic Navigation

At some point, we want to navigate to a route not through links, but programmatically.

To navigate to a route, we can use `router.push(path)`. The path can be either a string or an object.

If it is a string, the path must be in the form of plain path, meaning that it can not contain dynamic segments. For example `router.push('/stories/11/edit')`.

If it is an object, you can pass any needed arguments.

Programmatic navigation

```
router.push({ path: '/stories/11/edit' })
router.push({ name: 'stories.edit', params: {id: '11'} })
```

We can use `router.push` to navigate to the stories' listing page after editing is complete.

src/components/StoriesEdit.vue

```
1 <script>
2 import {store} from '../store.js'
3
4 export default {
5   props: ['id'],
6   data () {
7     return {
8       story: {}
9     }
10  },
11  methods: {
12    saveChanges (story) {
13      console.log('Saved!')
14      this.$router.push('/stories')
15    },
16    isTheOne (story) {
17      return story.id === this.id
18    }
19  },
20  mounted () {
21    this.story = store.stories.find(this.isTheOne)
22  }
23 }
24 </script>
```

We have updated the `saveChanges` method. When called, it logs a message to the console and using `this.$router.push()`, navigates back to `/stories`.

If you want to direct the user to the URL he visited previously, instead of a specific URL, you can use `router.back()`.

In our case we can add button and call the `router.back` function, instead of `router.push`, and this time the user will be able to navigate to the previous page (whichever this is, for example `https://google.com`).

`src/components/StoriesEdit.vue`

```
1 ...
2 <button @click="goBack">Go back</button>
3
4 methods: {
5   ...
6   goBack () {
7     this.$router.back()
8   },
9   ...
10 }
```

Another way to do this is to use the `router.go(n)` method, which takes a single integer as parameter that indicates by how many steps to go forwards or go backwards in the [history stack](#)⁸.

```
goBack () {
  this.$router.go(-1)
}
```



Warning

By using `$router.back()`, or any other router's method, we are coupling the component to the router. If you want to keep it uncoupled, you can use `window.history.back()`.

19.11 Transitions

19.11.1 Introduction

Each time we navigate to another page of our application nothing fancy happens. We can change that by using a transition to animate the component that enters the page and also the one that leaves.

⁸<http://router.vuejs.org/en/essentials/history-mode.html>

Vue provides a variety of ways to apply transition effects when items are inserted, updated, or removed from the DOM. This includes tools to:

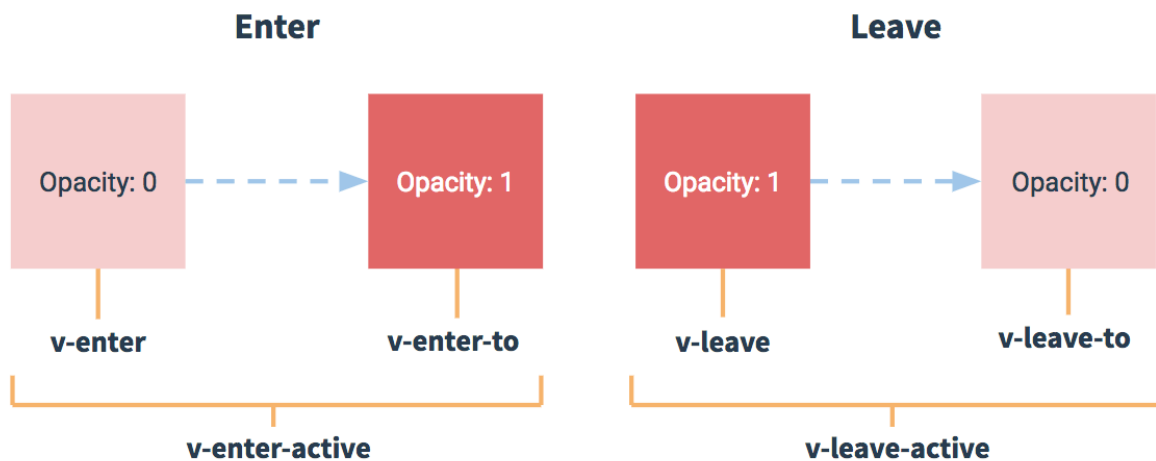
- automatically apply classes for CSS transitions and animations
- integrate 3rd-party CSS or JavaScript animation libraries, such as [Animate.css](https://daneden.github.io/animate.css/)⁹, [Velocity.js](http://velocityjs.org/)¹⁰, etc
- use JavaScript to directly manipulate the DOM during transition hooks

At this point, we'll only cover entering and leaving using CSS classes. If you are interested in learning more about transitions check the [guide](https://vuejs.org/v2/guide/transitions.html)¹¹.

To use a transition, we have to wrap the corresponding element within the transition component. In our case, `router-view` component.

Vue will append `v-enter` CSS class to the element before it's inserted and `v-enter-active` during the entering phase. `v-enter-to` is the last class to be appended before the transition is complete. This is actually the ending state for enter.

Accordingly, when the element is being removed from the DOM, `v-leave`, `v-leave-active`, and `v-leave-to` will be applied.



Transition Classes

If a name for the transition is defined, all the above mentioned classes will contain the name instead of 'v'. For example `fade-enter`, `fade-leave-to`, etc.

⁹<https://daneden.github.io/animate.css/>

¹⁰<http://velocityjs.org/>

¹¹<https://vuejs.org/v2/guide/transitions.html>

19.11.2 Usage

Lets use a transition, named `fade`, for our route outlet.

`src/App.vue`

```
<template>
  <div>
    ...
    <transition name="fade">
      <router-view></router-view>
    </transition>
  </div>
</template>

<style type="text/css">
  .fade-enter{
    opacity: 0
  }
  .fade-enter-active {
    transition: opacity 1s
  }
  .fade-enter-to {
    opacity: 0.8
  }
</style>
```

Take a look at the CSS classes. The transition will start with opacity 0 which will be gradually increasing for 1 second. `.fade-enter-to` is not necessary, defining it like this will create a blunt pop, from 0.8 to 1, just before the transition is finished.

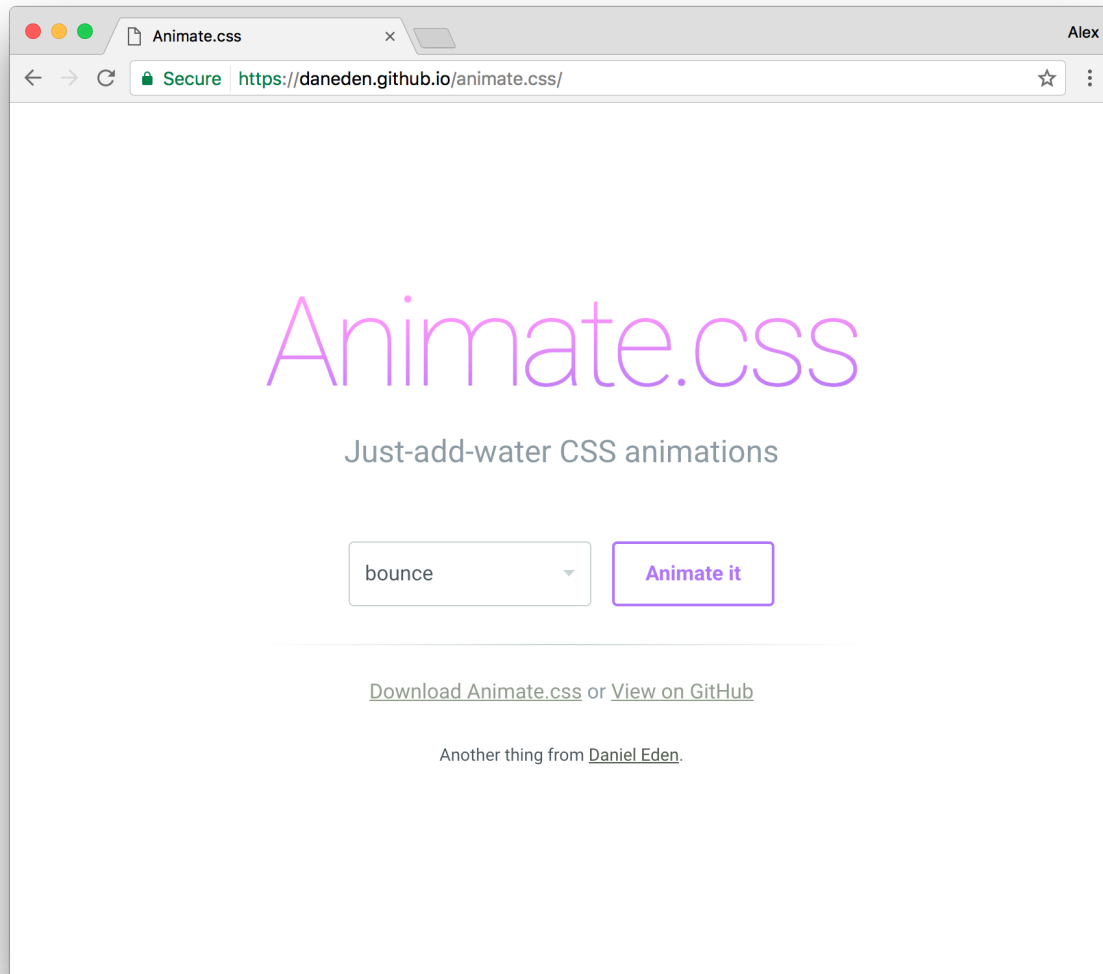
To create the reverse animation when a component leaves, we have to alter our CSS like this:

```
<style type="text/css">
  .fade-enter, .fade-leave-to{
    opacity: 0
  }
  .fade-enter-active, .fade-leave-active {
    transition: opacity 1s
  }
  .fade-enter-to, .fade-leave {
    opacity: 0.8
  }
</style>
```

19.11.3 3rd-party CSS animations

Creating animations from scratch, and designing in general, is a hard task for me. I always prefer to rely on 3rd party libraries. Fortunately for me (and you maybe), using Vue transitions is pretty easy to integrate 3rd-party CSS and JS animation libraries.

In this example I will use [Animate.css](https://daneden.github.io/animate.css/)¹².



Animate.css

According to the documentation, to use `Animate.css` you have to:

1. Include the stylesheet on your document's `<head>`.

¹²<https://daneden.github.io/animate.css/>

2. Add the class `animated` to the element you want to animate. You may also want to include the class `infinite` for an infinite loop.
3. Finally you need to add one of the available classes, such as `bounce`, `rollIn`, `fadeIn`, etc. You can find a list with all available classes [here](https://github.com/daneden/animate.css)¹³.

Lets start by importing it from CDNJS, in the header of our `index.html` file.

Instead of relying on the `name` prop of the `transition` component, this time we will use a custom class for `v-enter-active`.

`src/App.vue`

```
<template>
  <div>
    ...
    <transition enter-active-class="animated rollIn">
      <router-view></router-view>
    </transition>
  </div>
</template>
```

We can also apply an animation when the component leaves the DOM by appending a custom `leave-active-class`, like this: `leave-active-class="animated rollOut"`.

19.12 Navigation Guards

Vue Router provides a convenient mechanism for filtering transitions. To filter a transition you can use `router.beforeEach()` which is triggered before each transition, and `router.afterEach()`, which is triggered after, however `afterEach()` cannot affect the navigation.

`router.beforeEach()` can be handy in a scenario regarding authorization. For example if a user does not have permission to access a page of your app, he should be directed to the login page.

Let's see how we can accomplish that in a short example.

¹³<https://github.com/daneden/animate.css>

src/main.js

```
1 // create a dummy user object
2 let User = {
3   isAdmin: false
4 }
5
6 router.beforeEach((to, from, next) => {
7   if (to.path !== '/login' && !User.isAdmin) {
8     // if not going to login and not an admin redirect to login
9     next('/login')
10  } else {
11    // if authorized, proceed
12    next()
13  }
14 })
```

Here we apply a rule so router won't let users proceed to any page except login. Make sure to always call the `next()` function, otherwise the hook will never be resolved.



Code Examples

You can find the code examples of this chapter on [GitHub](https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter19)¹⁴.

¹⁴<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter19>

19.13 Homework

Throughout this chapter we analyzed a lot of things and it's been a while since we've assigned you some homework!

The app you have to build is a mini Pokédex.

The home page will show a list of Pokémon categories, such as *Fire*, *Water*, *etc.* From there, the user will be able to browse a category, view its Pokémon, and add new ones.

Your routes could be something like these:

Route	Description
/	List categories.
/category/:name	Show category's Pokémon.
/category/:name/pokemons/new	Add new Pokémon to category.

Each transition must be logged to the console. For example, when the user decides to browse category *Fire*, a message has to be logged, informing the user that he is going to visit `/category/Fire`.

We have created the Pokédex object to help you get started. You can find it [here](#)¹⁵.



Info

The `/category/:name/pokemons/new` route is a subroute of `/category/:name`.

When the user visits `/category/:name/pokemons/new` s/he should see a form to add a new Pokémon along with the listing of the category's Pokémon.



Hint 1

To access Pokémon of a specific category, consider using JavaScript's [find method](#)¹⁶.

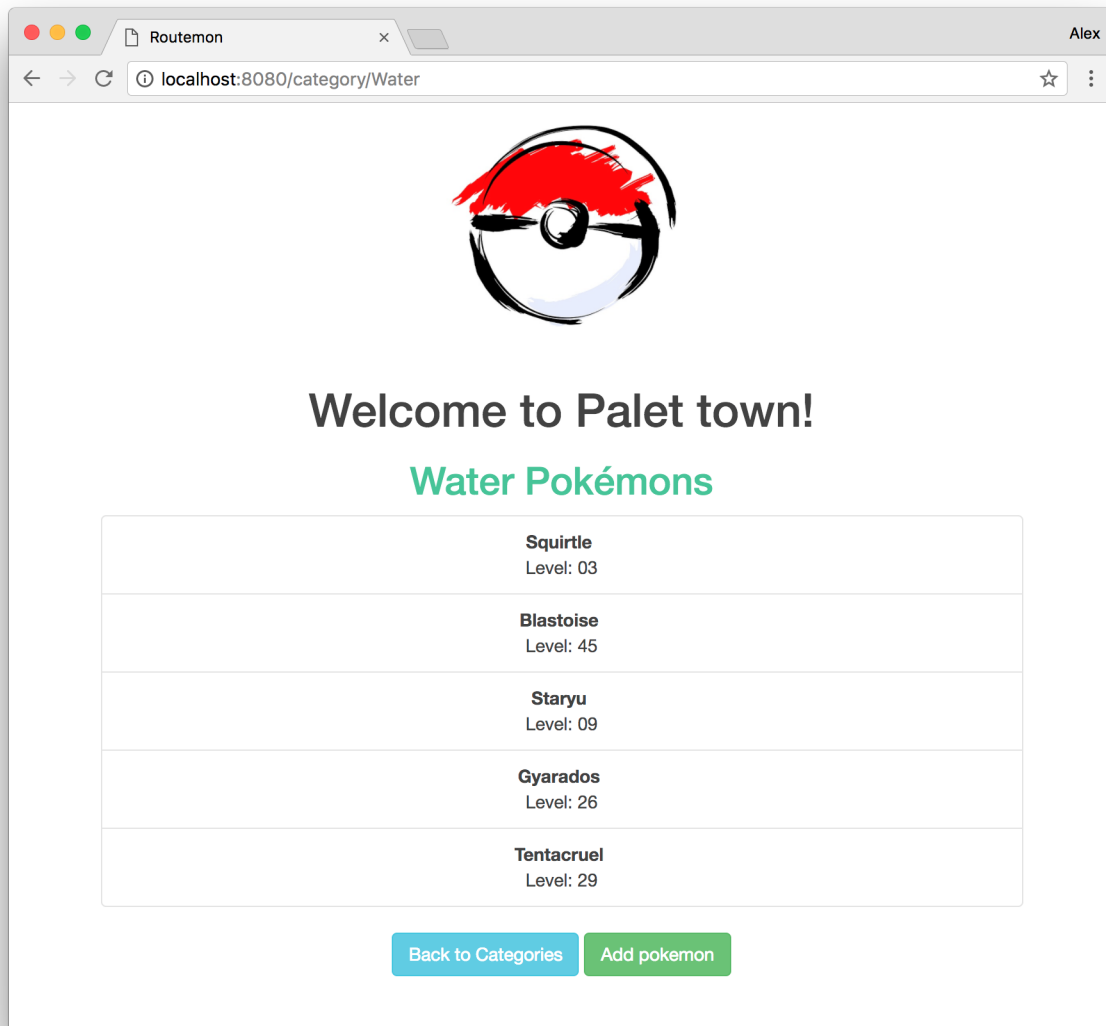


Hint 2

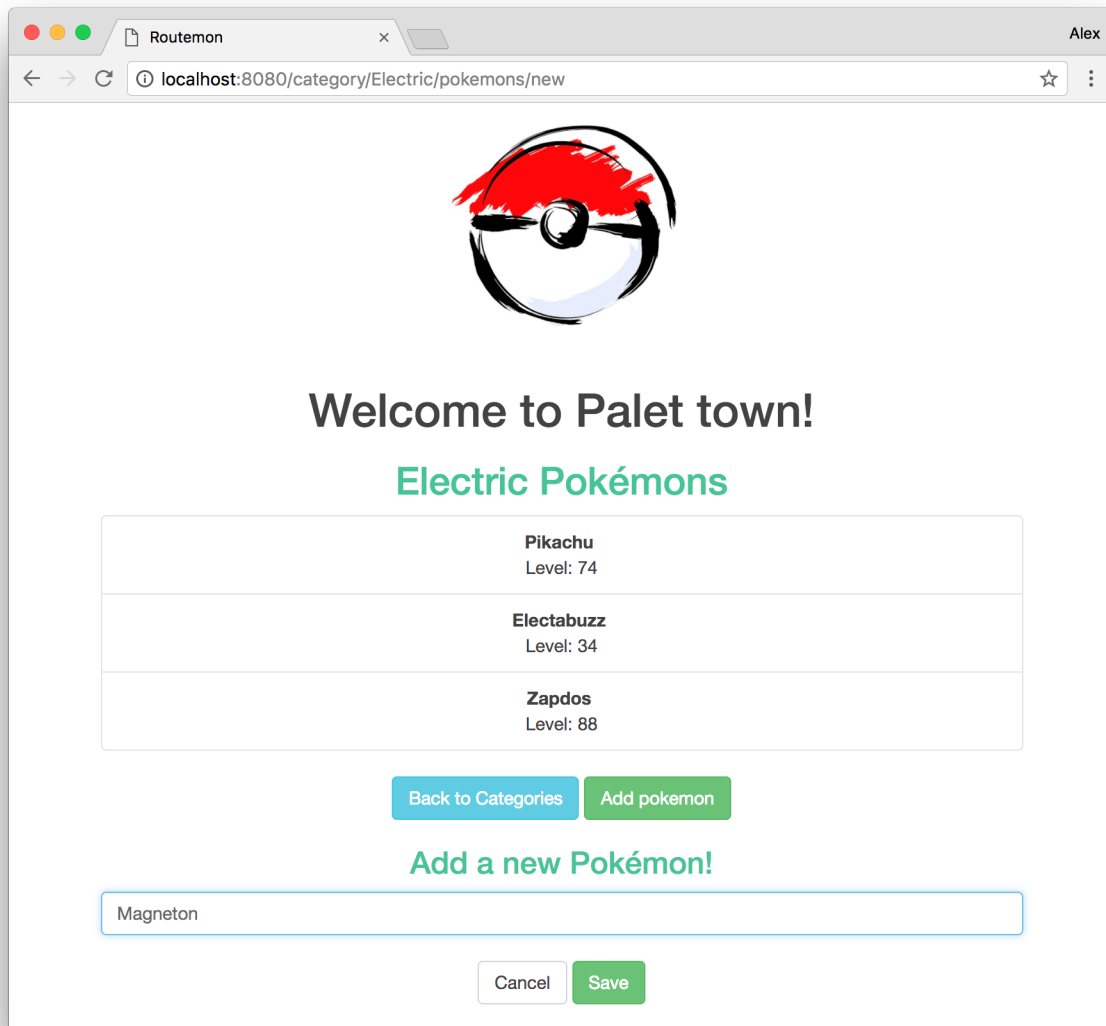
To log messages to the console before each transition use `router.beforeEach()`.

¹⁵<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/Chapter19/pokedex.js>

¹⁶https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find



Example output



Example output

You can find a potential solution to this exercise [here](https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/homework/Chapter19)¹⁷

¹⁷<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/homework/Chapter19>

20. Closing Thoughts

Well, it seems that our journey has come to an end. We hope that you have enjoyed reading the book as much we enjoyed writing it. We also hope that you liked (at least a bit) our sense of humor.

We want to thank each one of you, for reading the entire book and of course for supporting our efforts to make this book a reality. Huge thanks to all the [contributors](#)¹, who spared some of their time to report several grammatical, syntax, and coding flows.

In case you missed it, we have launched [Vue.js Feed](#)², where news, tutorials, code, and plugins, related to Vue.js, are published **daily**. We'd love it if you would like to be part of this, since we love you all. All and every single one of vue. :)



Don't be a stranger!

If you want to ask anything, or just say hi, don't hesitate to reach us on [Twitter](#)³. We'll also keep you informed about book updates.

¹<https://github.com/hootlex/the-majesty-of-vuejs-2/graphs/contributors>

²<https://vuejsfeed.com/>

³<http://twitter.com/tmvuejs>

21. Further Learning

We have curated some tutorials and books that you should consider reading. We've also gathered a few open source projects for you to see Vue in real action. Some of the resources are not directly related to Vue.js but they refer to related subjects.

If you'd like to learn more advanced Vue topics and how to build real-world applications, you should check out [Vue School](#)¹ - by Alex Kyriakidis.

21.1 Tutorials

- [Intro to Vue.js \(series\)](#)² - This series covers a lot of Vue stuff, like Rendering, Directives, Events, Components, Vuex, and Animations.
- [Authenticating a Vue JS Application With Firebase UI](#)³ - Setting up Firebase UI for a regular web application is incredibly simple. It's easy in Vue.js too, but there are a couple of steps to figure out.
- [Introducing Vue and Weex for Native Mobile Apps](#)⁴ - This tutorial shows how to use the Vue framework, in particular how to understand its concepts of data binding and templates. Then, it goes on to introduce the Weex platform, for coding native mobile apps using Vue!
- [How to build a reactive engine in JavaScript](#).⁵ - This tutorial goes the getters/setters way of observing and reacting to changes in real time.
- [Simple guide to authoring open-source Vue.js components](#)⁶ - Here are some bits of advice, for those that own or think about starting their own open source solutions for Vue.js.

21.2 Videos

- [Demystifying Frontend Framework Performance](#)⁷ - In this talk, Evan You will walk you through the techniques used in major front-end frameworks - namely dirty checking, virtual-dom diffing and dependency-tracking.

¹<https://vueschool.io>

²<https://css-tricks.com/intro-to-vue-1-rendering-directives-events/>

³<https://medium.com/dailyjs/authenticating-a-vue-js-application-with-firebase-ui-8870a3a5cff8>

⁴<https://code.tutsplus.com/tutorials/introducing-vue-and-weex-for-native-mobile-apps--cms-28782>

⁵<http://monterail.com/blog/2016/how-to-build-a-reactive-engine-in-javascript-part-1-observable-objects/>

⁶<http://monterail.com/blog/2016/simple-guide-to-authoring-open-source-vue-js-components/>

⁷<https://vuejsfeed.com/blog/demystifying-frontend-framework-performance-video>

- [Reactivity in Frontend JavaScript Frameworks](#)⁸ - How do the frameworks detect state changes, and how do they efficiently propagate the changes through the system? Evan answers these questions based on his experience building Vue.js.
- [Learning Vue 2: Step By Step \(series\)](#)⁹ - A series of lessons demonstrating the building blocks of Vue.

21.3 Books

- [Understanding ECMAScript 6](#)¹⁰ - There's a lot of new concepts to learn and understand in ES6. Get a headstart with this book!
- [Build APIs You Won't Hate](#)¹¹ - These days it's pretty standard to build your application with a separation of the frontend and backend logic. Frontend is done pretty well in JavaScript in the browser using awesome frameworks like Vue.js, and the backends will usually be some server-side language knocking out JSON.
- [SVG Animations](#)¹² - SVG is extremely powerful, with its reduced HTTP requests and crispness on any display. In this book you will learn all about SVG, including how to make SVG cross-browser compatible, backwards compatible, optimized, and responsive.

21.4 Open source projects

- [Vuedo](#)¹³ - A blog platform, built with Laravel and Vue.js.
- [Airflix](#)¹⁴ - An AirPlay friendly web interface to stream your movies and TV shows from a home server.
- [Koel](#)¹⁵ - A personal music streaming server that works.
- [Mini e-shop](#)¹⁶ - Mini online e-shop built with Vue.js. It has enough features to start building yours!
- [The Movie Database App](#)¹⁷ - A pretty application similar to imdb.
- [Vue Wordpress PWA](#)¹⁸ - An offline-first SPA using Vue.js, the WordPress REST API and Progressive Web Apps.

⁸<https://vuejsfeed.com/blog/watch-evan-you-reactivity-in-frontend-javascript-frameworks-in-dotjs-conference>

⁹<https://laracasts.com/series/learn-vue-2-step-by-step>

¹⁰<https://leanpub.com/understandings6>

¹¹<https://leanpub.com/build-apis-you-wont-hate>

¹²<http://shop.oreilly.com/product/0636920045335.do>

¹³<https://github.com/Vuedo/vuedo>

¹⁴<https://github.com/wells/airflix>

¹⁵<https://github.com/phanan/koel>

¹⁶<https://github.com/BosNaufal/vue-mini-shop>

¹⁷<https://github.com/dmtrbrl/tmdb-app>

¹⁸<https://github.com/bstavroulakis/vue-wordpress-pwa>

- [Hypersurface](https://github.com/aswdesign/hypersurface)¹⁹ - Hypersurface is a space for people to ask, answer, talk, and take notes on questions that invoke personal opinions. The data from these questions is displayed in real time, allowing anyone to impact and discover the current point of view.

21.5 Awesome Vue

[Awesome Vue.js](https://github.com/vuejs/awesome-vue)²⁰ is a curated list of awesome things related to Vue.js. You can find a ton of Vue stuff here. Currently the Vue team is working on a new project, [Curated Vue](https://curated.vuejs.org/)²¹, which will contain only approved resources and will provide the ability to search for specific resources.



The End...

¹⁹<https://github.com/aswdesign/hypersurface>

²⁰<https://github.com/vuejs/awesome-vue>

²¹<https://curated.vuejs.org/>