



La Elegancia de Vue.js 2

Alex
Kyriakidis



Kostas
Maniatis

La Elegancia de Vue 2

Kostas Maniatis, Alex Kyriakidis y Styde.net

Este libro está a la venta en <http://leanpub.com/vuejs2-spanish>

Esta versión se publicó en 2018-07-03



Este es un libro de [Leanpub](#). Leanpub anima a los autores y publicadoras con el proceso de publicación. [Lean Publishing](#) es el acto de publicar un libro en progreso usando herramientas sencillas y muchas iteraciones para obtener feedback del lector hasta conseguir tener el libro adecuado.

© 2016 - 2018 Kostas Maniatis, Alex Kyriakidis y Styde.net

Índice general

Introducción	i
Sobre Vue.js	ii
Un resumen de Vue.js	ii
Qué dice la gente sobre Vue.js	ii
Comparación con Otros Frameworks	iv
Bienvenido	xii
Sobre este libro	xii
Sobre esta traducción	xii
Para quién es este libro	xii
Ponerse en Contacto	xiii
Ejercicios	xiii
Código de ejemplo	xiii
Errata	xiii
Convenciones	xiv
Fundamentos de Vue.js	1
Instalar Vue.js	2
Versión Independiente	2
Descargar usando NPM	3
Descargar usando Bower	3
Primeros Pasos	4
Hola Mundo	4
Enlace de datos en dos direcciones	6
Comparación con jQuery	7
Ejercicios	9
Un Vistazo a las Directivas	10
v-show	10
v-if	13
v-else	15

ÍNDICE GENERAL

v-if vs. v-show	18
Ejercicios	20
Renderización de Listas	21
Instalar y Usar Bootstrap	21
v-for	23
Renderización de Arreglos	24
Objeto v-for	29
Ejercicio	32
Interactividad	33
Manejo de Eventos (#von)	33
Modificadores de Eventos	37
Modificadores de Teclas	41
Propiedades Computadas	42
Ejercicio	49
Filtros	51
Resultados Filtrados	51
Resultados Ordenados	60
Filtros personalizados	64
Bibliotecas de Utilidades	65
Ejercicio	69
Componentes	71
¿Qué son los Componentes?	71
Usando Componentes	71
Plantillas	73
Propiedades	74
Reusabilidad	77
En conjunto	80
Ejercicio	88
Eventos Personalizados	89
Emitir y Escuchar	89
Comunicación Padre-Hijo	92
Pasando Argumentos	94
Comunicación Entre Todos los Componentes	99
Eliminando listeners de Eventos	102
De Vuelta a Historias	103
Ejercicio	106
Enlaces de Clase y Estilo	108
Enlace de Clases	108

ÍNDICE GENERAL

Enlaces de Estilos	113
Enlaces en Acción	115
Ejercicio	118
Consumiendo una API	119
Introducción	120
CRUD	120
API	120
Trabajando con Datos Reales	125
Obtener Datos de Forma Asíncrona	125
Refactorizando	129
Actualizar Datos	131
Eliminar datos	133
Integrando Vue-resource	136
Resumen	136
Migración	137
Mejorando la Funcionalidad	138
Archivo JavaScript	150
Código Fuente	151
Ejercicios	156
Paginación	159
Implementación	160
Enlaces de Paginación	163
Ejercicios	166
Resumen de axios	167
Retiro de vue-resource	167
Integrando axios	168
Migración	168
Mejorando la Funcionalidad	170
Archivo JavaScript	182
Código Fuente	183
Ejercicios	188
Construyendo Aplicaciones Complejas	191
ECMAScript 6	192
Introducción	192
Declaraciones de Variables	193

ÍNDICE GENERAL

Funciones Flecha	194
Módulos	195
Clases	196
Valores de Parámetros Predeterminados	197
Plantillas de literales	198
Flujo de Trabajo Avanzado	200
Compilando ES6 con Babel	200
Configuración	204
Ejercicios	207
Automatización del Flujo de Trabajo con Gulp	209
Empaquetado de Módulos con Webpack	213
Resumen	220
Trabajando con Componentes de un Solo Archivo	222
vue-cli	222
Instalación	223
Plantilla de webpack	226
Formando Archivos .vue	234
Eliminando Estado Duplicado	248
Compartiendo con Propiedades	248
Almacenamiento Global	252
Intercambiando Componentes	256
Componentes Dinámicos	256
Vue Router	263
Instalación	263
Uso	264
Rutas con nombres	266
Modo de historial	267
Rutas anidadas	269
Clases activas Auto-CSS	271
Objeto de Ruta	273
Segmentos Dinámicos	274
Alias de Ruta	281
Navegación Programática	282
Transiciones	284
Guardias de Navegación	288
Ejercicios	290

Introducción

Sobre Vue.js

Un resumen de Vue.js

Vue (pronunciado /vju:/, como view en inglés) es un framework progresivo para construir interfaces de usuario. A diferencia de otros frameworks monolíticos, Vue está diseñado completamente para ser incrementalmente adoptable. La librería principal está enfocada solo en la capa de vista y es muy fácil de integrar con otras librerías o proyectos existentes. Por otra parte, al combinarlo con herramientas modernas y [librerías de apoyo](#)¹ Vue es también perfectamente capaz de soportar Aplicaciones de Página Única sofisticadas.

Si eres un desarrollador frontend experimentado y quieres saber como Vue.js se compara con otras librerías y frameworks, revisa el capítulo [Comparación con otros frameworks](#).

Si estás interesado en aprender más sobre el núcleo de Vue.js echa un vistazo a [la guía oficial de Vue.js](#)².

Qué dice la gente sobre Vue.js

“Vue.js es lo que me hizo amar JavaScript. Es extremadamente fácil y divertido de usar. Tiene un gran ecosistema de plugins y herramientas que amplían sus servicios básicos. Rápidamente puedes incluirlo en cualquier proyecto, pequeño o grande, escribir unas pocas líneas de código y listo. ¡Vue.js es rápido, liviano y es el futuro del desarrollo frontend!”

—**Alex Kyriakidis**

“Cuando comencé a aprender JavaScript me entusiasmé con la cantidad de posibilidades, pero cuando un amigo me sugirió que aprendería Vue.js y seguí su consejo, este entusiasmo fue mayor. Mientras leía y veía tutoriales me mantenía pensando en todas las cosas que había hecho hasta ahora y cuan fácil habría sido si hubiera invertido tiempo en aprender Vue antes. Mi opinión es que si quieres hacer tu trabajo rápido, agradable y fácil, Vue es el framework JavaScript que necesitas.”

—**Kostas Maniatis**

¹<https://github.com/vuejs/awesome-vue#libraries--plugins>

²<http://vuejs.org/guide/overview.html>

“Recuerden mis palabras: Vue.js se disparará en popularidad en 2016. Es así de bueno.”

—Jeffrey Way

“Vue es lo que siempre desee en un framework de JavaScript. Es un framework que escala contigo como desarrollador. Puedes colocarlo dentro de una sola página o construir una aplicación de página única avanzada con Vuex y Vue Router. Es verdaderamente el framework de JavaScript más pulido que he visto.”

—Taylor Otwell

“Vue.js es el primer framework que he encontrado que se siente igual de natural de usar en una aplicación renderizada en el servidor así como en una completa SPA. Ya sea que necesite un pequeño widget en una sola página o que esté construyendo un complejo cliente de JavaScript, nunca se siente insuficiente o excesiva.”

—Adam Wathan

“Vue.js ha sido capaz de crear un framework que es a la vez simple de usar y fácil de entender. Es una bocanada de aire fresco en un mundo donde otros están luchando para ver quien puede hacer el framework más complejo.”

—Eric Barnes

“La razón por la que me gusta Vue.js es que soy un diseñador/desarrollador híbrido. He visto a React, Angular y algunos otros pero la curva de aprendizaje y la terminología siempre me han desanimado. Vue.js es el primer framework de JavaScript que entiendo. Además, no solamente es fácil de aprender para los menos desenvueltos en JavaScript, como yo, pero he notado desarrolladores experimentados en Angular y React tomar en cuenta, y gustarles, Vue.js. Esto es completamente sin precedentes en el mundo de JavaScript y es la razón por la que comencé el London Vue.js Meetup.”

—Jack Barham

Comparación con Otros Frameworks

Angular 1

Parte de la sintaxis de Vue se verá muy similar a la de Angular (por ejemplo `v-if` vs `ng-if`). Esto es porque hubo muchas cosas que Angular hizo de forma correcta y estas fueron una inspiración para Vue muy temprano en su desarrollo. Sin embargo hay muchos dolores que vienen con Angular, donde Vue ha intentado ofrecer una mejora significativa.

Complejidad

Vue es mucho más simple que Angular 1, tanto en términos de API como de diseño. Aprender lo suficiente para construir aplicaciones complejas normalmente toma menos de un día, lo que no es cierto para Angular 1.

Flexibilidad y Modularidad

Angular 1 tiene fuertes opiniones sobre como tus aplicaciones deberían estar estructuradas, mientras que Vue es una solución más flexible y modular. Es por esto que Vue proporciona una [plantilla de Webpack³](#), que puedes configurar en minutos, pero que a su vez también te otorga acceso a características avanzadas como recarga de módulos (hot module reloading), linting, extracción de CSS y mucho más.

Enlace de datos

Angular 1 utiliza enlace en dos direcciones entre ámbitos, mientras que Vue fuerza un flujo único de datos entre componentes. Esto hace que el flujo de datos sea más fácil de comprender en aplicaciones complejas.

Directivas Vs Componentes

Vue posee una clara separación entre directivas y componentes. Las directivas están pensadas para encapsular únicamente manipulaciones del DOM, mientras que los componentes son unidades autónomas que poseen sus propias vistas y lógica de datos. En Angular hay mucha confusión entre los dos.

Desempeño

Vue posee un mejor desempeño y es mucho más fácil de optimizar porque no usa “comprobación sucia” (dirty checking). Angular 1 se vuelve lento cuando hay muchos observadores, debido a que cada vez que algo cambia en el ámbito, todos estos observadores deben ser reevaluados de nuevo.

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

También, si algún observador activa otra actualización, la comprobación sucia puede tener que ejecutarse multiples veces para “estabilizar”. Los usuarios de Angular a menudo tienen que recurrir a técnicas esotéricas para lidiar con la comprobación sucia, y en algunas situaciones, simplemente no hay forma de optimizar un ámbito con muchos observadores.

Vue no sufre de esto en absoluto ya que usa un sistema de observación transparente para el rastreo de dependencias con cola asincrónica - todos los cambios se activan de forma independiente a no ser que tengan relaciones de dependencia explícitas.

Curiosamente, hay bastantes similitudes en como Angular 2 y Vue estan abordando estos problemas de Angular 1.

Angular 2

Hay una sección separada para Angular 2 porque realmente es un framework completamente nuevo. Por ejemplo, cuenta con un sistema de componentes de primera clase, muchos detalles de implementación han sido completamente reescritos y la API también ha cambiado drásticamente.

Tamaño y Desempeño

En terminos de desempeño ambos frameworks son excepcionalmente rápidos y no hay suficientes datos sobre casos de uso en el mundo real para hacer un veredicto. Sin embargo, si estás determinado a ver algunos números, Vue 2.0 parece estar a la delantera de Angular 2 de acuerdo a [esta comparación](#)⁴.

En cuanto a tamaño, a pesar de que Angular 2 con compilación offline y eliminación de código innecesario (tree-shaking) es capaz de reducir su tamaño considerablemente, Vue 2.0 completo con compilador incluído (23kb) es todavía mucho más liviano que un ejemplo básico con eliminación de código innecesario de Angular 2 (50kb).

Flexibilidad

Vue es mucho menos dogmático que Angular 2, ofreciendo soporte oficial para una variedad de sistemas de compilación, sin restricciones en como estructuras tu aplicación. Muchos desarrolladores disfrutan esta libertad, mientras que algunos prefieren tener solo una forma correcta de construir cualquier aplicación.

Curva de aprendizaje

Para comenzar con Vue todo lo que necesitas es familiaridad con HTML y JavaScript ES5 (JavaScript puro). Con esas habilidades básicas puedes comenzar a construir aplicaciones complejas a menos de un día de haber leído la guía.

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

La curva de aprendizaje de Angular 2 es mucho más empinada. Incluso sin TypeScript, su [Guía de inicio⁵](#) comienza con una aplicación que usa JavaScript ES2015, NPM con 18 dependencias, 4 archivos y más de 3000 palabras para explicar todo, sólo para decir Hola Mundo.

⁵<https://angular.io/docs/js/latest/quickstart.html>

React

React y Vue comparten muchas similitudes. Ambas:

- Utilizan un DOM virtual.
- Proporcionan componentes de vista reactivos y acoplables.
- Mantienen el foco en el núcleo de la librería, con asuntos como rutas y administración global de estado siendo manejado por librerías complementarias.

Perfiles de Desempeño

En cada escenario del mundo real que ha sido probado hasta el momento, Vue supera por un margen razonable a React.

Desempeño de Renderizado

Al renderizar la interfaz de usuario, manipular el DOM es típicamente la operación más costosa y desafortunadamente ninguna librería puede hacer que esas operaciones sean más rápidas. Lo mejor que se puede hacer es:

1. Minimizar el número necesario de mutaciones del DOM. Tanto React como Vue usan abstracciones virtuales del DOM para realizar esto y ambas implementaciones funcionan igualmente bien.
2. Agrega la menor carga adicional posible sobre esas manipulaciones del DOM. Esta es un área en la que Vue y React difieren. En React digamos que la carga adicional de renderizar un elemento es 1 y la carga adicional del componente promedio es 2. En Vue, la carga adicional de un elemento sería más como 0.1, pero la carga adicional de un componente promedio sería 4, debido a la configuración requerida por el sistema de reactividad.

Esto significa que en aplicaciones típicas, donde hay muchos más elementos que componentes siendo renderizados, Vue superará a React por un margen significativo. Sin embargo, en casos extremos como al usar un componente normal para renderizar cada elemento, Vue normalmente será más lento.

Tanto Vue como React ofrecen componentes funcionales sin estado ni instancia y por tanto requieren menor carga adicional. Cuando estos son usados en situaciones de desempeño critico, Vue es nuevamente más rápido.

Desempeño de Actualización

En React, necesitas implementar `shouldComponentUpdate` en todas partes y usar estructuras de datos inmutables para conseguir re-renderizaciones completamente optimizadas. En Vue, las dependencias de un componente son automáticamente rastreadas de forma que este solo se actualice cuando alguna de esas dependencias cambie. La única optimización adicional que algunas veces puede ser útil en Vue es agregar atributos a los elementos en listas largas.

Esto significa que actualizaciones en Vue sin optimizar serán mucho más rápidas que en React sin optimizar y de hecho, debido al desempeño mejorado de renderizado en Vue, incluso React totalmente optimizado usualmente será más lento que Vue sin optimizar.

En Desarrollo

Por supuesto, el rendimiento en producción es el más importante y eso es lo que hemos estado discutiendo hasta el momento. Sin embargo, el rendimiento en desarrollo también importa. La buena noticia es que tanto Vue como React permanecen lo suficientemente rápido en desarrollo para la mayoría de las aplicaciones normales.

Sin embargo, si estás creando una visualización de datos o animaciones, encontrarás útil saber que en escenarios en los que Vue no puede manejar más de 10 cuadros por segundo en desarrollo, hemos visto a React ralentizarse a alrededor de 1 cuadro por segundo.

Esto es debido a los múltiples y pesados controles invariantes de React, que le ayudan a proveer excelentes mensajes de advertencia y errores.

Ember

Ember es un completo framework que está diseñado para ser altamente dogmático. Proporciona una gran cantidad de convenciones y una vez que eres bastante familiar con ellas, pueden hacerte muy productivo. Sin embargo, eso también significa que la curva de aprendizaje es alta y la flexibilidad sufre. Es un compromiso cuando intentas elegir entre un framework dogmático y una librería con un conjunto de herramientas acopiables que trabajan juntas. La última te da más libertad pero también requiere que tomes más decisiones arquitectónicas.

Dicho esto, una mejor comparación sería entre el núcleo de Vue y las capas de plantillas y modelo de objetos de Ember:

- Vue proporciona reactividad no obstructiva en objetos de JavaScript y propiedades computadas que se calculan de una forma totalmente automática. En Ember, necesitas envolver todo en Objetos de Ember y declarar manualmente las dependencias de las propiedades computadas.
- La sintaxis de plantillas de Vue asegura todo el poder de las expresiones de JavaScript, mientras que la sintaxis de expresión y helpers de Handlebars es intencionalmente bastante limitada en comparación.
- A nivel de desempeño, Vue supera a Ember por un margen razonable, incluso después de la última actualización del motor Glimmer en Ember 2.0. Vue automáticamente agrupa las actualizaciones, mientras que en Ember tienes que gestionar manualmente bucles de ejecución en situaciones complejas.

Polymer

Polymer es otro proyecto patrocinado por Google y de hecho también fue una fuente de inspiración para Vue. Los componentes de Vue pueden ser libremente comparados con los elementos personalizados de Polymer y ambos proveen un estilo similar de desarrollo. La mayor diferencia es que Polymer está construido sobre las últimas características de los Componentes Web y requiere Polyfills complejos para funcionar (con rendimiento reducido) en navegadores que no soportan esas características de forma nativa. En contraste, Vue funciona sin ninguna dependencia o Polyfills hasta IE9.

En Polymer 1.0, el equipo también ha hecho su sistema de enlace de datos bastante limitado con el fin de compensar por el rendimiento. Por ejemplo, las únicas expresiones soportadas en las plantillas de Polymer son la negación booleana y llamadas a métodos únicos. Su implementación de propiedades computadas tampoco es muy flexible.

Los elementos personalizados de Polymer están realizados en archivos HTML, lo que te limita a JavaScript/CSS plano (y características del lenguaje soportadas por los navegadores actuales). En comparación, los componentes de un solo archivo de Vue te permiten usar fácilmente ES2015+ y cualquier preprocesador de CSS que quieras.

Al desplegar a producción, Polymer recomienda cargar todo en el momento con importaciones HTML, lo que asume navegadores implementando la especificación y soporte HTTP/2 tanto en el servidor como el cliente. Esto puede o no ser factible dependiendo de tu público objetivo y entorno de despliegue. En casos donde esto no es deseable, tendrás que usar una herramienta especial llamada Vulcanizer para agrupar tus elementos de Polymer. En este frente, Vue puede combinar su componente asíncrono con la característica de división de código de Webpack para fácilmente separar partes del paquete de la aplicación para ser cargados de forma diferida. Esto garantiza compatibilidad con navegadores antiguos al mismo tiempo que conserva un buen rendimiento en la carga de la aplicación.

Riot

Riot 2.0 proporciona un modelo de desarrollo similar basado en componentes (llamado “tag” en Riot), con una API minimalista y excelentemente diseñada. Riot y Vue probablemente comparten mucho en filosofías de diseño. Sin embargo, a pesar de ser un poco más pesada que Riot, Vue ofrece algunas ventajas significativas:

- Renderización condicional real. Riot renderiza todas sus ramas y simplemente las muestra u oculta.
- Un sistema de rutas más poderoso. La API de rutas de Riot es extremadamente mínima.
- Soporte de herramientas más maduro. Vue proporciona soporte oficial para Webpack, Brow-serify y SystemJS mientras que Riot depende del soporte de la comunidad para la integración con sistemas de compilación.
- Sistema de efectos de transición. Riot no tiene ninguno.
- Mejor rendimiento. A pesar de anunciar el uso de un DOM virtual, Riot en realidad usa comprobación sucia (dirty checking) y por tanto sufre de los mismos problemas de rendimiento que Angular 1.

Para comparaciones actualizadas sientete libre de revisar [la guía de Vue.js](#).

Bienvenido

Sobre este libro

Este libro te guiará a través del framework de JavaScript de rápido crecimiento llamado Vue.js.

Hace un tiempo atrás, comenzamos un nuevo proyecto basado en Laravel y Vue.js. Luego de leer a fondo la guía de Vue.js y algunos tutoriales, descubrimos la falta de recursos sobre Vue.js en la web. Durante el desarrollo de nuestro proyecto ganamos mucha experiencia, así que tuvimos la idea de escribir este libro para compartir con el mundo los conocimientos que adquirimos. Ahora que Vue.js 2 ha salido, decidimos que era hora de actualizar nuestro libro publicando una segunda versión donde todos los ejemplos y el contenido relacionado han sido reescritos.

Este libro está escrito en un formato informal, intuitivo y fácil de seguir, donde todos los ejemplos están detallados de forma apropiada para proporcionar orientación adecuada a todos.

Comenzaremos desde lo más básico y mediante muchos ejemplos cubriremos las características más significativas de Vue.js. Al finalizar este libro serás capaz de crear aplicaciones frontend rápidas e incrementar el desempeño de tus proyectos existentes con la integración de Vue.js 2.

Sobre esta traducción

Este libro es una adaptación al español del libro **The Majesty of Vue.js 2**. Parte del equipo de Styde ha estado trabajando en esta adaptación durante varios meses. Si encuentras algún error, por favor envíanos un email a admin+tmvue@styde.net y lo corregiremos pronto. Esta traducción está en progreso, recibirás las actualizaciones del libro en tu correo electrónico.

Para quién es este libro

Todo el que ha invertido tiempo en aprender desarrollo web moderno ha visto Bootstrap, JavaScript y muchos frameworks de JavaScript. Este libro es para cualquier persona interesada en aprender un framework de JavaScript sencillo y liviano. No se requiere de conocimiento excesivo, aunque sería bueno estar familiarizado con HTML y JavaScript. Si no sabes cuál es la diferencia entre una cadena de texto y un objeto, quizás primero necesites revisar eso en profundidad.

Este libro es útil para desarrolladores que son nuevos en Vue.js, así como para aquellos que ya usan Vue.js y quieren expandir sus conocimientos. También es útil para desarrolladores que buscan migrar a Vue.js 2.

Ponerse en Contacto

En caso de que quieras ponerte en contacto con nosotros por algo relacionado con el libro, enviarnos feedback u otros asuntos que te gustaría traer a nuestra atención, no dudes en contactarnos.

Nombre	Correo electrónico	Twitter
Styde	admin+tmvue@styde.net.	@Stydenet
The Majesty of Vue.js	hello@tmvuejs.com	@tmvuejs
Alex Kyriakidis	alex@tmvuejs.com	@hootlex
Kostas Maniatis	kostas@tmvuejs.com	@kostaskafcas

Ejercicios

La mejor forma de aprender código es escribiendo código, así que hemos preparado un ejercicio al final de la mayoría de los capítulos para que lo resuelvas y te pruebes a ti mismo en lo que has aprendido. Te recomendamos fuertemente que intentes, en la medida de lo posible, resolverlos y a través de ellos te beneficijes con una mejor comprensión de Vue.js. No tengas miedo de probar tus ideas, cada pequeño esfuerzo contribuirá grandemente en tu aprendizaje. Tal vez diferentes ejemplos o formas de hacer algo te darán la idea correcta. ¡Por supuesto no somos despiadados!, te daremos pistas y posibles soluciones.

¡Ahora puedes comenzar tu viaje!

Código de ejemplo

Puedes encontrar la mayoría de los ejemplos usados en este libro en Github. Puedes navegar a través del código [aquí](#)⁶.

Si prefieres descargarlo, encontrarás un archivo .zip [aquí](#)⁷.

Esto te ahorrará tener que copiar y pegar cosas desde el libro, lo que probablemente sería terrible.

Errata

A pesar de que se han tomado todos los cuidados necesarios para asegurar la precisión del contenido, los errores suceden. Si encuentras un error en el libro estaríamos agradecidos si lo reportaras a nosotros. Al hacer eso, puedes proteger a otros lectores de encontrarse frustrados y ayudarnos a mejorar subsecuentes versiones de este libro. Si consigues cualquier error, por favor haznoslo saber en nuestro [repositorio de GitHub](#)⁸.

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

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

⁸<https://github.com/hootlex/the-majesty-of-vuejs-2>

Convenciones

Las siguientes convenciones de notación son usadas a lo largo del libro.

Un bloque de código está establecido de la siguiente manera:

JavaScript

```
1 function(x, y){  
2     // esto es un comentario  
3 }
```

El código dentro del texto es mostrado como: “Usar `.container` para un contenedor responsivo de anchura fija.”

Nuevos términos y palabras **importantes** son mostradas en negrita.

Tips, notas y advertencias son mostradas como:



Esto es una advertencia

Este elemento indica una advertencia o precaución.



Esto es un Tip

Este elemento representa un tip o una sugerencia.



Esto es una caja de información

Alguna información especial aquí.



Esto es una nota

Una nota sobre el tema.



Esto es una pista

Una pista sobre el tema



Esto es un video

Un video con material relacionado a la lección.



Esto es un comando de la terminal

Comandos para ejecutar en la terminal



Esto es un texto de comparación

Un pequeño texto comparando cosas relativas al tema



Esto es un enlace a Github.

Los enlaces dirigen al repositorio de este libro, donde puedes encontrar código de ejemplo y posibles soluciones a los ejercicios de cada capítulo.

Fundamentos de Vue.js

Instalar Vue.js

Cuando se trata de descargar Vue.js tienes diferentes opciones para elegir.

Versión Independiente

Descargar desde vuejs.org

Para instalar Vue puedes simplemente descargarlo e incluirlo con una etiqueta script. Vue será registrada como una variable global.

Puedes descargar dos versiones de Vue.js:

1. Versión de desarrollo desde <http://vuejs.org/js/vue.js>⁹.
2. Versión de producción desde <http://vuejs.org/js/vue.min.js>¹⁰.



Tip: No uses la versión comprimida durante el desarrollo, te perderás todas las advertencias para errores comunes.

Incluir desde un CDN

[Vuejs.org](#)¹¹ recomienda [unpkg](#)¹², que reflejará la última versión tan pronto como sea publicada a NPM.

También puedes encontrar Vue.js en [jsdelivr](#)¹³ o [cdnjs](#)¹⁴.



Toma algo de tiempo sincronizar con la última versión, así que tienes que verificar frecuentemente por actualizaciones.

⁹<http://vuejs.org/js/vue.js>

¹⁰<http://vuejs.org/js/vue.min.js>

¹¹<https://vuejs.org/v2/guide/installation.html#CDN>

¹²<https://unpkg.com/vue/dist/vue.js>

¹³<https://cdn.jsdelivr.net/vue/2.0.1/vue.min.js>

¹⁴<https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.min.js>

Descargar usando NPM

NPM es el método de instalación recomendado al construir aplicaciones complejas con Vue.js. Funciona muy bien junto a un empaquetador de módulos de CommonJS como [Webpack¹⁵](#) o [Browserify¹⁶](#).

```
1 # Última versión estable
2 $ npm install vue
3 # Última versión estable + CSP-compliant
4 $ npm install vue@csp
5 # versión en desarrollo (directamente desde GitHub):
6 $ npm install vuejs/vue#dev
```

Descargar usando Bower

```
1 # Última versión estable
2 $ bower install vue
```



Para más instrucciones de instalación y actualizaciones echa un vistazo a la [Guía de instalación de Vue.js¹⁷](#).

En la mayoría de los ejemplos del libro incluimos Vue.js desde el CDN, pero eres libre de instalarlo usando cualquier método que quieras.



Video

Puedes ver cómo instalar Vue.js en la [lección 4 del curso de Vue 2 en Styde¹⁸](#).

¹⁵<http://webpack.github.io/>

¹⁶<http://browserify.org/>

¹⁷<http://vuejs.org/guide/installation.html>

¹⁸<https://styde.net/primeros-pasos-con-vue-2/>

Primeros Pasos

Comencemos dando un vistazo rápido a las funciones de enlace de datos de Vue. Vamos a hacer una sencilla aplicación que nos permitirá ingresar un mensaje y mostrarlo en la página en tiempo real. Esto demostrará el poder de los enlaces de datos en dos direcciones de Vue. Para crear nuestra aplicación con Vue necesitamos hacer algo de configuración, lo que sólo implica crear una página HTML.

En el proceso obtendrás una idea de la cantidad de tiempo y esfuerzo que ahorraremos usando un Framework de JavaScript como Vue.js en lugar de una librería de JavaScript como jQuery.

Hola Mundo

Crearemos un nuevo archivo con algo de código. Lo llamaremos `hello.html`, aunque puedes asignarle el nombre que quieras.

```
1 <html>
2 <head>
3   <title>Hola Vue</title>
4 </head>
5 <body>
6   <h1> ¡Saludos su Majestad! </h1>
7 </body>
8 </html>
```

Este es simplemente un archivo HTML con un mensaje de saludo.

Ahora continuaremos y haremos lo mismo usando Vue.js. En primer lugar incluiremos Vue.js y crearemos una nueva Instancia.

```

1 <html>
2 <head>
3   <title>Hola Vue</title>
4 </head>
5 <body>
6   <div id="app">
7     <h1>iSaludos su Majestad!</h1>
8   </div>
9 </body>
10 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.min.js"></script>
11 <script>
12   new Vue({
13     el: '#app',
14   })
15 </script>
16 </html>

```

Para comenzar, hemos incluido Vue.js desde [cdnjs¹⁹](https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.min.js) y dentro de una etiqueta `script` tenemos nuestra instancia de Vue. Usamos un `div` con un `id` de `#app` y este es el elemento al que haremos referencia, así Vue sabrá donde ‘buscar’. Trata de pensar en esto como un contenedor en el que Vue trabaja. Vue no reconocerá nada fuera del elemento seleccionado. Usa la propiedad `el` para seleccionar el elemento que quieras.

Ahora asignaremos el mensaje que queremos mostrar, a una variable dentro de un objeto llamado `data`. Después pasaremos el objeto de datos como una propiedad en la instancia de Vue.

```

1 var data = {
2   message: ' iSaludos su Majestad!'
3 };
4 new Vue({
5   el: '#app',
6   data: data
7 })

```

Para mostrar nuestro mensaje en la página sólo necesitamos envolverlo en llaves dobles. Así, cualquier cosa que esté dentro de nuestro mensaje aparecerá automáticamente en la etiqueta `h1`.

```

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

```

Es tan simple como eso. Otra forma de definir la variable `message` es directamente dentro de la instancia de Vue, en el objeto `data`.

¹⁹<https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.min.js>

```

1 new Vue({
2   el: '#app',
3   data: {
4     message: '¡Saludos su Majestad!'
5   }
6 });

```

Ambas formas tienen el mismo resultado, por lo que nuevamente eres libre de elegir la sintaxis que prefieras.

Info

El código en llaves dobles es llamado interpolaciones, y éstas no serán renderizadas como HTML sino que serán evaluadas como expresiones de JavaScript, enlazadas al objeto de Vue. Por ejemplo, `{{ message }}` mostrará el valor de la variable de JavaScript en nuestro objeto de Vue. Mientras que el código `{{1+2}}` mostrará el número 3.

Enlace de datos en dos direcciones

Lo que es genial sobre Vue es que hace nuestras vidas más fáciles. Supongamos que queremos cambiar el mensaje en tiempo real, ¿cómo podemos lograr esto fácilmente? En el ejemplo de abajo usamos `v-model`, una directiva de Vue (cubriremos más sobre directivas en el siguiente capítulo). Luego usamos un enlace de datos en dos direcciones para cambiar dinámicamente el valor del mensaje cuando el usuario cambia el texto del mensaje dentro de un campo de texto. Los datos son sincronizados por defecto en cada evento de entrada.

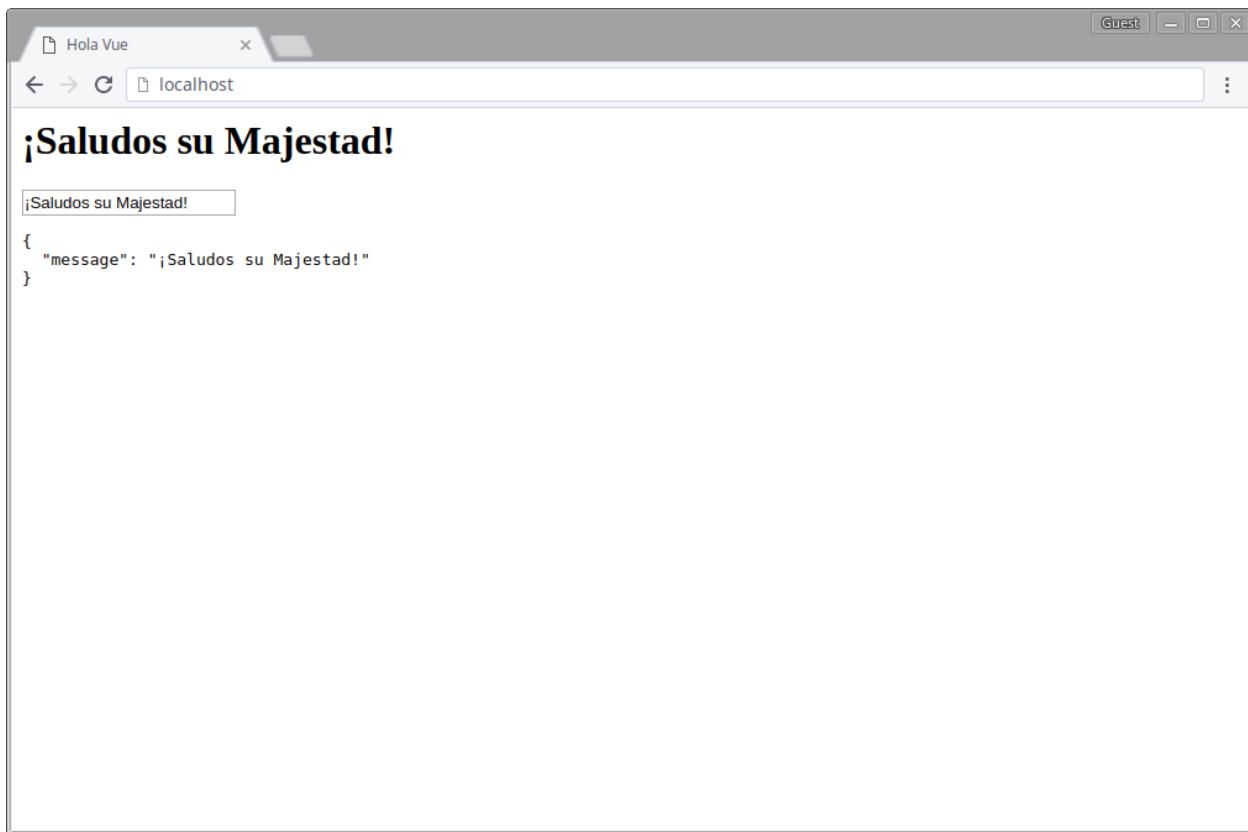
```

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: '¡Saludos su Majestad!'
5   }
6 });

```

Eso es todo. ¡Ahora nuestro mensaje y el texto ingresado por el usuario están enlazados! Al usar `v-model` dentro de la etiqueta `input` le decimos a Vue que variable debería enlazarse con ese `input`, en este caso `message`.



Enlace de datos en dos direcciones

Enlace de datos en dos direcciones significa que si cambias el valor de un modelo en tu vista, todo se mantendrá actualizado.

Comparación con jQuery.

Probablemente tienes algo de experiencia con jQuery. Si no es así, no hay problema, el uso de jQuery en este libro es mínimo. Cuando lo usamos es solo para demostrar como las cosas pueden hacerse con Vue en lugar de jQuery y nos aseguraremos de que todos lo entiendan.

De todas formas, a fin de entender mejor como los enlaces de datos nos ayudan a construir aplicaciones, toma un momento para pensar como podrías hacer el ejemplo anterior con jQuery. Probablemente crearías un elemento input y le darías un `id` o `class`, para poder seleccionarlo y manipularlo. Luego de esto, llamarías a una función que cambia el elemento deseado para que coincida con el valor ingresado cada vez que el *evento keyup* sucede. **Esto es una verdadera molestia.**

Mas o menos, tu código se verá así.

```
1 <html>
2 <head>
3     <title>Hola Vue</title>
4 </head>
5 <body>
6 <div id="app">
7     <h1>¡Saludos su Majestad!</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>
```

Esto es un ejemplo sencillo de comparación y como puedes ver, Vue parece ser mucho más agradable, consume menos tiempo y es más fácil de entender. Por supuesto, jQuery es una poderosa librería de JavaScript para la manipulación del DOM (Document Object Model), pero todo viene con sus altibajos.



Video

Puedes ver una comparación entre Vue.js y jQuery en la [lección 2 del curso de Vue 2 en Styde²⁰](#).



Video

Puedes ver una introducción al uso de directivas en la [lección 4 del curso de Vue 2 en Styde²¹](#).



Código de Ejemplo

Puedes encontrar el código de ejemplo de este capítulo en [GitHub²²](#).

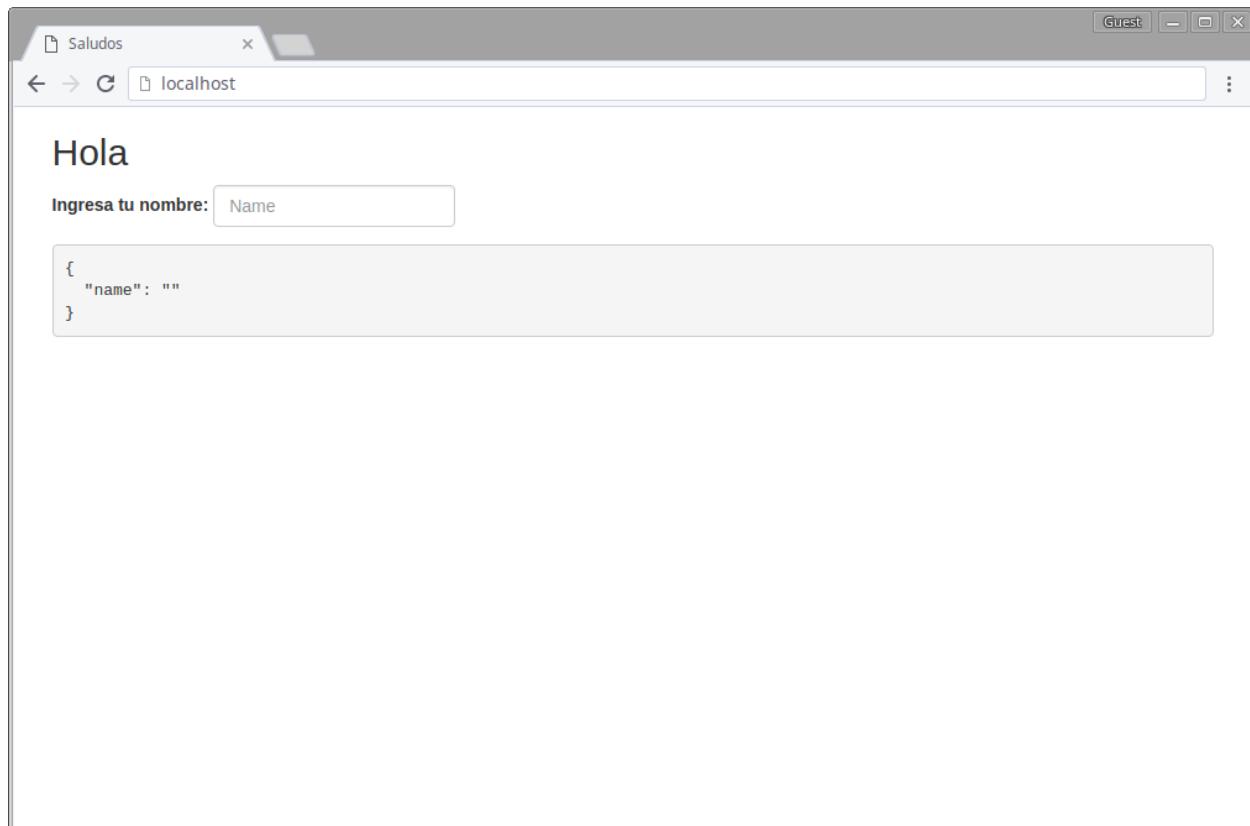
²⁰<https://styde.net/vue-vs-jquery-1/>

²¹<https://styde.net/primeros-pasos-con-vue-2/>

²²<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/codes/chapter2.html>

Ejercicios

Un genial y simple ejercicio introductorio es crear un archivo HTML con un encabezado Hola, {{name}}. Agrega un input y enlazalo a la variable `name`. Como puedes imaginarte, el encabezado deberá cambiar automáticamente cada vez que el usuario ingrese o cambie su nombre. ¡Buena suerte y diviertete!



Pantalla de Ejemplo



Nota

El ejemplo hace uso de Bootstrap. Si no estás familiarizado con Bootstrap puedes ignorarlo por ahora, será cubierto en un [capítulo posterior](#).



Possible Solución

Puedes encontrar una posible solución a este ejercicio [aquí](#)²³.

²³<https://github.com/hoottlex/the-majesty-of-vuejs-2/blob/master/homework/chapter2.html>

Un Vistazo a las Directivas

En este capítulo vamos a repasar algunos ejemplos básicos de las directivas de Vue. Bien, si no has utilizado ningún Framework como Vue.js o Angular.js anteriormente, probablemente no sabes qué es una directiva. Básicamente, una directiva es un indicador especial en la plantilla que le comunica a la librería que haga algo a un elemento del DOM. En Vue.js el concepto de directiva es drásticamente más simple que en Angular. Algunas de las directivas son:

- **v-show** usada para mostrar un elemento de forma condicional.
- **v-if** la cual puede ser usada en lugar de **v-show**.
- **v-else** la cual muestra un elemento cuando **v-if** evalúa a false.

También existe **v-for**, que requiere una sintaxis especial y su uso es para renderizar (p.ej. mostrar una lista de elementos en base a un arreglo). Profundizaremos sobre el uso de cada una más adelante en el libro.

Comenzemos y demos un vistazo a las directivas mencionadas.

v-show

Para demostrar la primera directiva, vamos a hacer algo simple. ¡Te daremos algunos tips que harán tu comprensión y trabajo mucho más fácil! Supongamos que te encuentras en la necesidad de alternar la visualización de un elemento, en base a algún criterio. Quizás un botón de submit no debería mostrarse a menos que primero se haya escrito un mensaje. ¿Cómo podemos lograr eso con Vue?.

```
1 <html>
2 <head>
3   <title>Hola 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/2.0.1/vue.js"></script>
11 <script>
12   new Vue({
```

```

13     el: '#app',
14     data: {
15       message: ' ¡Nuestro Rey ha muerto!'
16     }
17   })
18 </script>
19 </html>

```

Aquí tenemos un archivo HTML con nuestro conocido `div id="app"` y un `textarea`. Dentro del `textarea` vamos a mostrar nuestro mensaje. Por supuesto, aun no está enlazado y en este punto probablemente ya te has dado cuenta. Quizás también has notado que en este ejemplo ya no estamos usando la versión comprimida de Vue.js. Como hemos mencionado anteriormente, la versión comprimida no debería ser usada durante el desarrollo porque te perderás las advertencias para errores comunes. De ahora en adelante vamos a usar esta versión en el libro, pero por supuesto eres libre de usar la que tú quieras.

```

1 <html>
2 <head>
3   <title>Hola 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/2.0.1/vue.js"></script>
14 <script>
15   new Vue({
16     el: '#app',
17     data: {
18       message: ' ¡Nuestro Rey ha muerto!'
19     }
20   })
21 </script>
22 </html>

```

Es momento de enlazar el valor de `textarea` con nuestra variable `message` usando `v-model` para mostrar nuestro mensaje. Cualquier cosa que ingresemos cambiará en tiempo real, justo como vimos en el ejemplo del capítulo anterior, donde estabamos usando un `input`. Adicionalmente, aquí estamos usando una etiqueta `pre` para mostrar los datos. Lo que esto va a hacer, es tomar

los datos desde nuestra instancia de Vue, filtrarlo a través de `json` y finalmente mostrar los datos en nuestro navegador. Vue formateará el resultado por nosotros automáticamente ya sea este una cadena de texto, un número, un array o un objeto. Nosotros creemos que esto ofrece una mejor manera de construir y manipular nuestros datos, ya que tener todo en frente de ti es mejor que mirar constantemente en la consola para observar el valor de un elemento.

Info

JSON (Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos. Puedes encontrar más información sobre JSON [aquí²⁴](#). El resultado de `{{ $data }}` está enlazado con los datos de Vue y se actualizará en cada cambio.

```

1 <html>
2 <head>
3   <title>Hola Vue</title>
4 </head>
5 <body>
6 <div id="app">
7   <h1>iDebes enviar un mensaje de ayuda!</h1>
8   <textarea v-model="message"></textarea>
9   <button v-show="message">
10    iEnviar mensaje de ayuda a los aliados!
11  </button>
12  <pre>
13    {{ $data }}
14  </pre>
15 </div>
16 </body>
17 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
18 <script>
19   new Vue({
20     el: '#app',
21     data: {
22       message: ' iNuestro Rey está muerto! iEnviar ayuda!'
23     }
24   })
25 </script>
26 </html>
```

Siguiendo adelante, ahora tenemos una simple advertencia en la etiqueta `h1` que cambiará luego en base a algún criterio. Cerca de la etiqueta está el botón que se va a mostrar de forma condicional.

²⁴<http://www.json.org/>

Sólo aparece si hay un mensaje presente. Si el `textarea` está vacío y por tanto nuestros datos, el atributo `display` del botón es automáticamente establecido a ‘none’ y el botón desaparece.

Info

Un elemento con `v-show` siempre se renderizará y permanecerá en el DOM. `v-show` simplemente cambia la propiedad CSS `display` del elemento.

```

1 <h1 v-show="!message"> iDebes enviar un mensaje de ayuda!</h1>
2 <textarea v-model="message"></textarea>
3 <button v-show="message">
4     iEnviar mensaje de ayuda a los aliados!
5 </button>
```

Lo que queremos lograr en este ejemplo, es cambiar diferentes elementos. En este paso, necesitamos ocultar la advertencia dentro de la etiqueta `h1`, si un mensaje está presente. De lo contrario ocultar el mensaje estableciendo su `style` a `display: none`.

v-if

En este punto podrías preguntar ¿Y la directiva `v-if` que mencionamos anteriormente? Así que, vamos a hacer el ejemplo anterior nuevamente, solo que esta vez usaremos `v-if`.

```

1 <html>
2 <head>
3     <title>Hola Vue</title>
4 </head>
5 <body>
6 <div id="app">
7     <h1 v-if="!message"> iDebes enviar un mensaje de ayuda!</h1>
8     <textarea v-model="message"></textarea>
9     <button v-if="message">
10        iEnviar mensaje de ayuda a los aliados!
11    </button>
12    <pre>
13        {{ $data }}
14    </pre>
15 </div>
16 </body>
17 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
18 <script>
```

```
19  new Vue({
20    el: '#app',
21    data: {
22      message: ' ¡Nuestro Rey está muerto! ¡Enviar ayuda!'
23    }
24  })
25 </script>
26 </html>
```

Como se muestra, el reemplazo de `v-show` con `v-if` funciona tan bien como esperábamos. ¡Trata de hacer tus propios experimentos para ver como funciona! La única diferencia es que un elemento con `v-if` no permanecerá en el DOM.

v-if con template

Si alguna vez nos encontramos en la posición de querer cambiar la existencia de múltiples elementos a la vez, podemos usar `v-if` en un elemento `<template>`. En ocasiones donde el uso de `div` o `span` no parece apropiado, el elemento `template` puede servir también como un contenedor invisible. `<template>` no será renderizado en el resultado final.

```
1 <div id="app">
2   <template v-if="!message">
3     <h1> ¡Debes enviar un mensaje de ayuda! </h1>
4     <p> ¡Envia un mensajero inmediatamente! </p>
5     <p> ¡Al cercano Reino de Corazones! </p>
6   </template>
7   <textarea v-model="message"></textarea>
8   <button v-show="message">
9     ¡Enviar mensaje de ayuda a los aliados!
10  </button>
11  <pre>
12    {{ $data }}
13  </pre>
14 </div>
```



v-if con template

Usando la configuración del ejemplo anterior hemos agregado la directiva **v-if** al elemento **template**, alternando la existencia de todos los elementos encapsulados.



Advertencia

La directiva **v-show** no soporta la sintaxis de **<template>**.

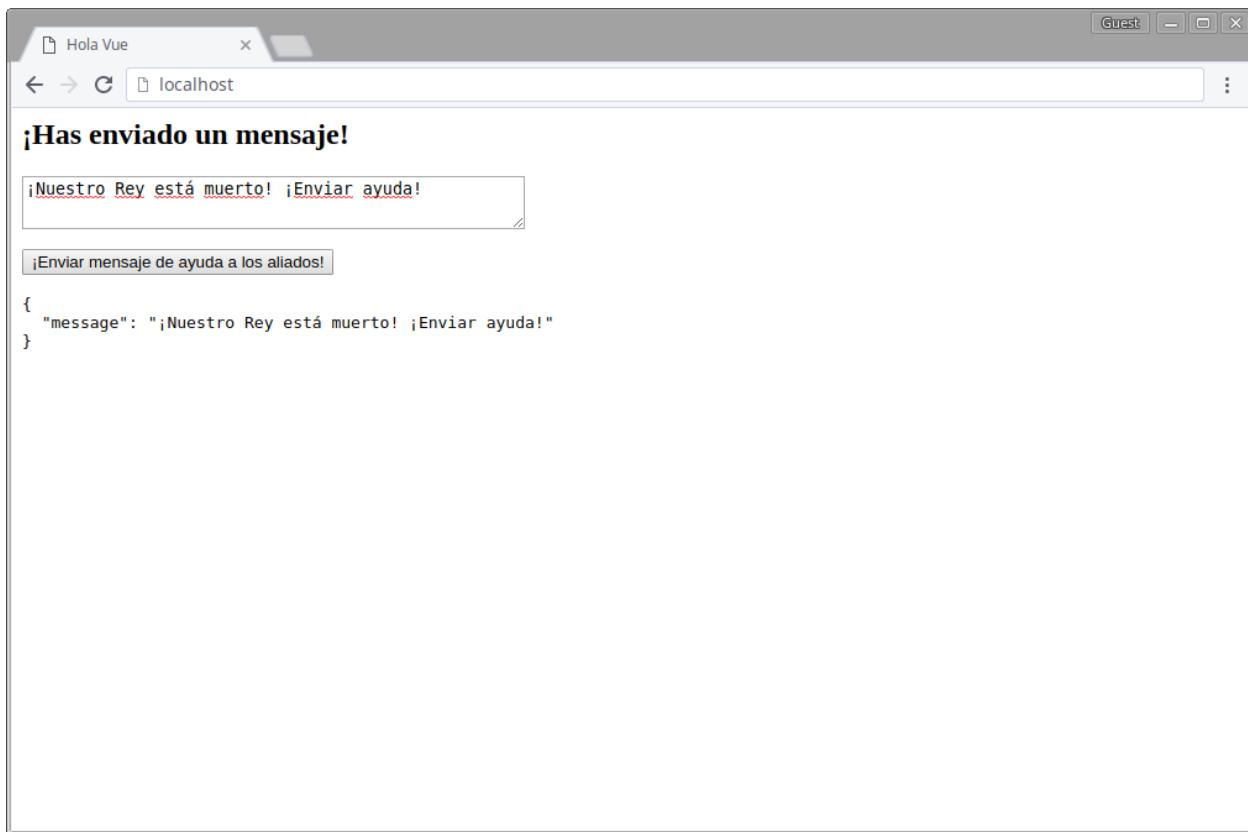
v-else

Al usar **v-if** puedes usar la directiva **v-else** para indicar un “bloque else” como ya habrás imaginado. Ten en cuenta que la directiva **v-else** debe seguir inmediatamente a la directiva **v-if** - ya que de otra manera no será reconocida-.

```
1 <html>
2 <head>
3     <title>Hola Vue</title>
4 </head>
5 <body>
6 <div id="app">
7     <h1 v-if="!message"> ¡Debes enviar un mensaje de ayuda! </h1>
8     <h2 v-else> ¡Has enviado un mensaje! </h2>
9     <textarea v-model="message"></textarea>
10    <button v-show="message">
11        ¡Enviar mensaje de ayuda a los aliados!
12    </button>
13    <pre>
14        {{ $data }}
15    </pre>
16 </div>
17 </body>
18 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
19 <script>
20     new Vue({
21         el: '#app',
22         data: {
23             message: ' ¡Nuestro Rey está muerto! ¡Enviar ayuda!'
24         }
25     })
26 </script>
27 </html>
```



v-if en acción



v-else en acción

Sólo por motivos de ejemplo hemos usado una etiqueta **h2** con una advertencia diferente a la anterior, que se muestra de forma condicional. Si ningún mensaje está presente vemos la etiqueta **h1**. Si hay un mensaje, vemos la etiqueta **h2** usando la sintaxis **v-if** y **v-else** de Vue. ¡Simple!



Advertencia

La directiva **v-show** ya no funciona con **v-else** en Vue 2.0.

v-if vs. v-show

Aunque ya hemos mencionado la diferencia entre **v-if** y **v-show**, podemos profundizar un poco más. Algunas preguntas pueden surgir de su uso. ¿Hay alguna gran diferencia entre usar **v-show** y **v-if**? ¿Hay alguna situación en la que el desempeño se ve afectado? ¿Hay situaciones donde estarás mejor usando una o la otra? Podrías experimentar que el uso de **v-show** en muchas situaciones ocasiona un mayor tiempo de carga durante el renderizado de la página. En comparación, de acuerdo a la guía de Vue.js, **v-if** es realmente condicional.

*Al usar **v-if**, si la condición es falsa en el renderizado inicial no hará nada, esto es, el bloque condicional no será renderizado hasta que la condición se vuelva verdadera por*

primera vez. En general, **v-if** tiene un costo mayor de alteración o cambio mientras que **v-show** tiene un costo de renderización inicial mayor. Por lo tanto elige **v-show** si necesitas alterar o cambiar algo muy a menudo y elige **v-if** si la condición es poco probable que cambie durante la ejecución.

Por lo tanto, cuando usar una o la otra realmente depende de tus necesidades.



Video

Aprende más sobre directivas en el [Curso de Vue en Styde.net²⁵](#).



Video

v-if, v-else y v-show son cubiertas en la [lección 6 del curso de Vue 2 en Styde²⁶](#).



Código de Ejemplo

Puedes encontrar el código de ejemplo de este capítulo en [GitHub²⁷](#).

²⁵<https://styde.net/interpolaciones-y-directivas-en-vue-2/>

²⁶<https://styde.net/mostrar-y-ocultar-elementos-usando-v-if-y-v-show-con-vue-js-2/>

²⁷<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter3>

Ejercicios

Siguiendo con el ejercicio de tarea anterior, deberías intentar expandirlo un poco. El usuario ahora ingresa su género junto con su nombre. Si el usuario es un hombre, entonces en el encabezado se saludará al usuario con “Hola Sr. {{name}}”. Si el usuario es una mujer, entonces “Hola Sra. {{name}}” deberá aparecer en su lugar.

Cuando el género no es hombre o mujer entonces el usuario deberá ver la advertencia “**Así que no puedes decidir. ¡Bien!**”



Pista

Un operador lógico vendría bien para determinar el título del usuario.

The screenshot shows a web browser window titled "Saludos, usuario". The URL bar says "localhost". The page content is as follows:

Hola, Sr. Universo.

Ingresa tu género:

Ingresa tu nombre:

Pantalla de ejemplo



Possible Solución

Puedes encontrar una posible solución para este ejercicio [aquí²⁸](#).

²⁸<https://github.com/hoottlex/the-majesty-of-vuejs-2/blob/master/homework/chapter3.html>

Renderización de Listas

En el cuarto capítulo de este libro vamos a aprender sobre renderización de listas. Usando las directivas de Vue vamos a ver como:

1. Renderizar una lista de elementos en base a un arreglo.
2. Repetir una plantilla.
3. Iterar a través de las propiedades de un objeto.

Instalar y Usar Bootstrap

Para hacer nuestro trabajo más agradable a la vista, vamos a importar Bootstrap.



Info

Bootstrap es el framework de HTML, CSS y JS más popular para desarrollo responsive, de proyectos para la web enfocados principalmente a dispositivos móviles.

Dirígete a <http://getbootstrap.com/>²⁹ y presiona el botón de descargar. Por el momento, solo usaremos Bootstrap desde el [enlace del CDN](#)³⁰ pero puedes instalarlo en cualquier forma que se adapte a tus necesidades particulares. Para nuestro ejemplo por ahora sólo necesitamos un archivo: `css/bootstrap.min.css`. Cuando usamos este archivo `.css` en nuestra aplicación, tenemos acceso a todas las bonitas estructuras y estilos. Sólo inclúyelo dentro de la etiqueta `head` de tu página y listo.

Bootstrap requiere un elemento contenedor para envolver el contenido del sitio y alojar nuestro sistema de rejillas. Puedes elegir uno de dos contenedores para usar en tu proyecto. Ten en cuenta que, debido al `padding` y más, ninguno de los contenedores se puede anidar.

- Usa `.container` para un contenedor adaptable de ancho fijo.

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

- Usa `.container-fluid` para un contenedor de ancho completo, abarcando todo el ancho de tu vista (viewport).

²⁹<http://getbootstrap.com/>

³⁰<https://www.bootstrapcdn.com/>

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

En este punto, nos gustaría hacer un ejemplo de Vue.js con clases de Bootstrap. Por supuesto, no es requerido mucho estudio o experimentación a fin de combinar Vue y Bootstrap.

```
1  <html>
2  <head>
3  <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" r\
4  el="stylesheet">
5    <title>Hola Bootstrap</title>
6  </head>
7  <body>
8    <div class="container">
9      <h1>Hola Bootstrap, siéntate junto a Vue.</h1>
10     </div>
11   </body>
12  <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
13  <script type="text/javascript">
14    new Vue({
15      el: '.container'
16    })
17  </script>
18 </html>
```

Observa que esta vez en lugar de apuntar al id `app` hemos apuntado a la clase `container` dentro de la opción `el` en la instancia de Vue.



Tip

En el ejemplo anterior seleccionamos el elemento con la clase `.container`. Se cuidadoso cuando estás seleccionando a un elemento por clase, cuando la clase está presente más de una vez, Vue.js montará sólo el primer elemento.

La propiedad `el:` puede ser un selector CSS o un elemento HTML. *No se recomienda montar la instancia principal en <html> o <body>.*

v-for

Para iterar a través de cada uno de los elementos en un arreglo usaremos la directiva **v-for**.

Esta directiva requiere una sintaxis especial en la forma de **elemento in arreglo** donde **arreglo** es el origen de los datos del arreglo y **elemento** es un alias para los elementos que están siendo iterados en el arreglo.



Advertencia

Si vienes del mundo de PHP puedes notar que **v-for** es similar a la función **foreach** de PHP. Pero **ten cuidado** si estás acostumbrado a **foreach(\$arreglo as \$valor)**.

El **v-for** de Vue es exactamente lo contrario, **valor in arreglo**.

El singular primero, el plural después.

Range v-for

La directiva **v-for** también puede recibir un número entero. Cuando un número es pasado en lugar de un arreglo u objeto la plantilla será repetida tantas veces como el número dado.

```
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>Hola Vue</title>
6 </head>
7 <body>
8   <div class="container">
9     <h1>Tabla de multiplicar de 4.</h1>
10    <ul class="list-group">
11      <li v-for="i in 11" class="list-group-item">
12        {{ i-1 }} por 4 es igual a {{ (i-1) * 4 }}.
13      </li>
14    </ul>
15  </div>
16 </body>
17 <script src="https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js"></script>
18 <script type="text/javascript">
19   new Vue({
20     el: '.container'
21   })

```

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

El código de arriba muestra la tabla de multiplicar de 4.



Tabla de multiplicar por 4



Nota

Debido a que queremos mostrar toda la tabla de multiplicar de 4 (hasta 40) repetimos la plantilla 11 veces ya que el primer valor que `i` toma es 1.

Renderización de Arreglos

Iterar a Través de un Arreglo

En el siguiente ejemplo incluiremos un arreglo de historias dentro de nuestro objeto data y mostraremos todas una por una.

```
stories: [
    "¡Estrellé mi auto hoy!",
    "¡Ayer, alguien robó mi bolso!",
    "Alguien se comió mi chocolate...",
]
```

Lo que necesitamos hacer aquí es renderizar una lista, específicamente un arreglo de cadenas de texto.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" r\
4 el="stylesheet">
5     <title>Historias</title>
6 </head>
7 <body>
8     <div class="container">
9         <h1>Escuchemos algunas historias!</h1>
10        <div>
11            <ul class="list-group">
12                <li v-for="story in stories" class="list-group-item">
13                    Alguien dijo "{{ story }}"
14                </li>
15            </ul>
16        </div>
17        <pre>
18            {{ $data }}
19        </pre>
20    </div>
21 </body>
22 <script src="https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js"></script>
23 <script type="text/javascript">
24     new Vue({
25         el: '.container',
26         data: {
27             stories: [
28                 "¡Estrellé mi auto hoy!",
29                 "¡Ayer, alguien robó mi bolso!",
30                 "Alguien se comió mi chocolate...",
31             ]
32         }
33     })

```

```
34  </script>
35  </html>
```



Info

Tanto `list-group` como `list-group-item` son clases de Bootstrap. Aquí puedes encontrar más información sobre los estilos de listas de Bootstrap.³¹



Renderizando un arreglo usando v-for.

Usando `v-for` hemos conseguido mostrar nuestras historias en una lista no ordenada. ¡Es realmente así de fácil!

Iterar a Través de un Arreglo de Objetos

Ahora, modificaremos el arreglo `stories` para que contenga objetos `story`. Un objeto `story` tiene dos propiedades: `plot` y `writer`. Haremos lo mismo que hicimos anteriormente pero esta vez en lugar de mostrar `story` inmediatamente, mostraremos `story.plot` y `story.writer` respectivamente.

³¹<http://getbootstrap.com/css/#type-lists>

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
4 <title>Historias</title>
5 </head>
6 <body>
7   <div class="container">
8     <h1>¡Escuchemos algunas historias!</h1>
9     <div>
10       <ul class="list-group">
11         <li v-for="story in stories"
12           class="list-group-item">
13           >
14             {{ story.writer }} dijo "{{ story.plot }}"
15           </li>
16         </ul>
17       </div>
18       <pre>
19         {{ $data }}
20       </pre>
21     </div>
22   </body>
23 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
24 <script type="text/javascript">
25 new Vue({
26   el: '.container',
27   data: {
28     stories: [
29       {
30         plot: "¡Estrellé mi auto hoy!",
31         writer: "Alex"
32       },
33       {
34         plot: "¡Ayer, alguien robó mi bolso!",
35         writer: "John"
36       },
37       {
38         plot: "Alguien se comió mi chocolate...",
39         writer: "John"
40       },
41       {
42         plot: "¡Me comí el chocolate de alguien!",
```

```
44         writer: "Alex"
45     },
46 ]
47 }
48 })
49 </script>
50 </html>
```

Adicionalmente, cuando necesitas mostrar el índice del elemento actual, puedes usar la variable especial `index`. Funciona de la siguiente forma:

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

El `index` dentro de las llaves, claramente representa el índice del elemento iterado en el ejemplo.



Arreglo renderizado con índice

Objeto v-for

Puedes usar **v-for** para iterar a través de las propiedades de un objeto. Anteriormente mencionamos que puedes mostrar el **index** del arreglo, pero también puedes hacer lo mismo al iterar un objeto. Adicionalmente a **index** cada ámbito tendrá acceso a otra propiedad especial, llamada **key**.

Info

Al iterar un objeto, **index** está en el rango de **0 ... n-1** donde **n** es el número de propiedades del objeto.

Hemos reestructurado nuestros datos para que esta vez sea un solo objeto con tres atributos: **plot**, **writer** y **upvotes**.

```
<div class="container">
  <h1> iEscuchemos algunas historias!</h1>
  <ul class="list-group">
    <li v-for="value in story" class="list-group-item">
      {{ value }}
    </li>
  </ul>
</div>

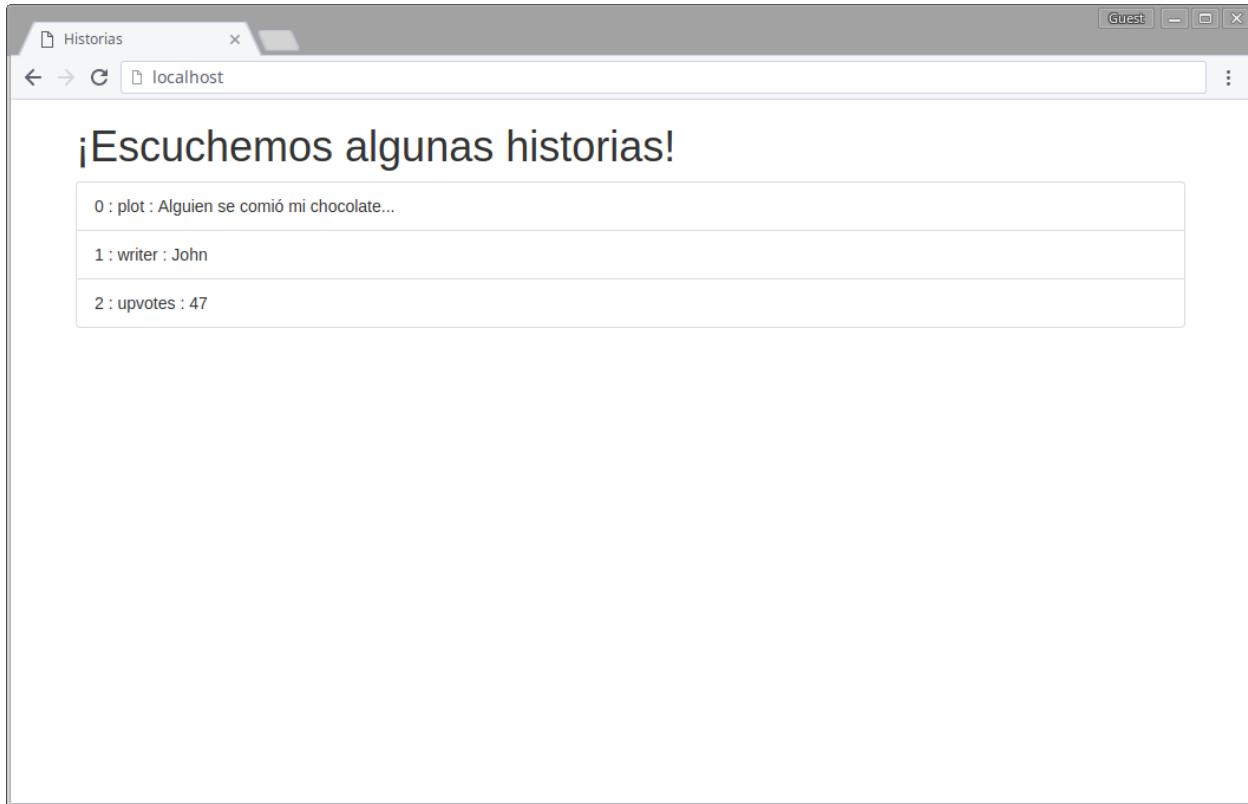
new Vue({
  el: '.container',
  data: {
    story: {
      plot: "Alguien se comió mi chocolate...",
      writer: 'John',
      upvotes: 47
    }
  }
})
```

Podemos suministrar un segundo y tercer argumento para **key** e **index** respectivamente.

```
1 <div class="container">
2   <h1> ¡Escuchemos algunas historias!</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>
```

Como puedes ver en el código del ejemplo anterior, usamos **key** e **index** para traer dentro de la lista los pares llave-valor (key-value) así como también el índice (**index**) de cada par.

El resultado será:



Iterar a través de las propiedades de un objeto.



Video

El listado de elementos con **v-for** es cubierto en la [lección 8](#) del curso de Vue 2 en Styde³².

³²<https://styde.net/listado-de-elementos-con-v-for-en-vue-2/>



Código de Ejemplo

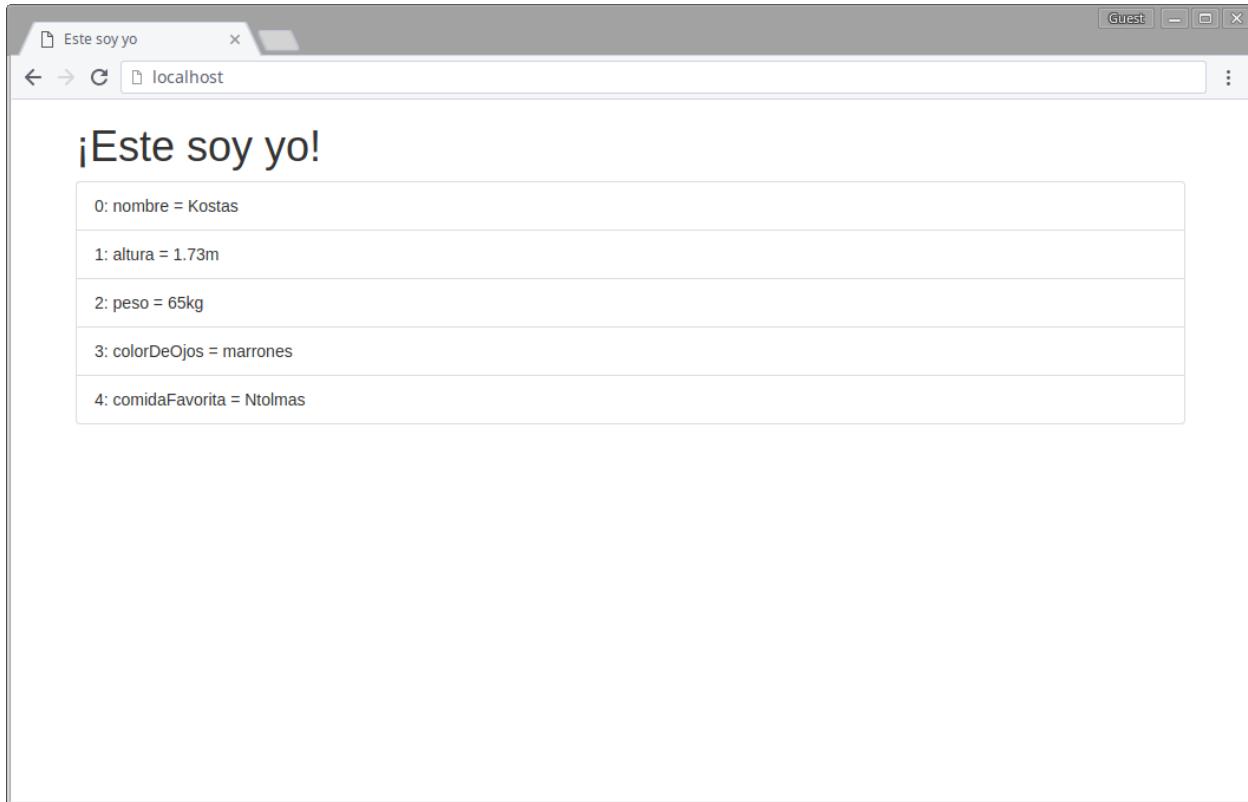
Puedes encontrar el código de ejemplo de este capítulo en [GitHub³³](#).

³³<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter4>

Ejercicio

Teniendo en cuenta lo que hemos visto en este capítulo, para este ejercicio crea un objeto con tus atributos personales. Por atributos personales quiero decir tu nombre, peso, altura, colorDeOjos y tu comidaFavorita.

Usando **v-for** itera a través de cada propiedad y muéstralala con el formato de: **index: key = value**.



Pantalla de ejemplo



Possible Solución

Puedes encontrar una posible solución a este ejercicio [aquí](#)³⁴.

³⁴<https://github.com/hoottlex/the-majesty-of-vuejs-2/blob/master/homework/chapter4.html>

Interactividad

En este capítulo vamos a crear y expandir ejemplos anteriores y aprender nuevas cosas respecto a ‘métodos’, ‘manejo de eventos’ y ‘propiedades computadas’. Desarrollaremos algunos ejemplos usando diferentes enfoques. Es momento de ver como podemos implementar la interactividad de Vue para obtener una pequeña aplicación, como una calculadora, funcionando bien y fácil.

Manejo de Eventos (#von)

Los eventos HTML son cosas que suceden a los elementos del DOM. Cuando Vue.js es usado en páginas HTML, puede **reaccionar** a estos eventos.

Los eventos pueden representar cualquier cosa, desde interacciones básicas del usuario, hasta cosas sucediendo en el modelo renderizado.

Estos son algunos ejemplos de eventos HTML:

- Una página web ha terminado de cargar.
- Un campo de formulario ha cambiado.
- Un botón ha sido presionado.
- Un formulario ha sido enviado.

El punto del manejo de eventos es que puedes hacer algo cuando un evento tiene lugar.

En Vue.js, para **escuchar** eventos del DOM puedes usar la directiva **v-on**.

La directiva **v-on** asigna un evento listener a un elemento. El tipo de evento es definido por el argumento, por ejemplo **v-on:keyup** escucha al evento **keyup**.



Info

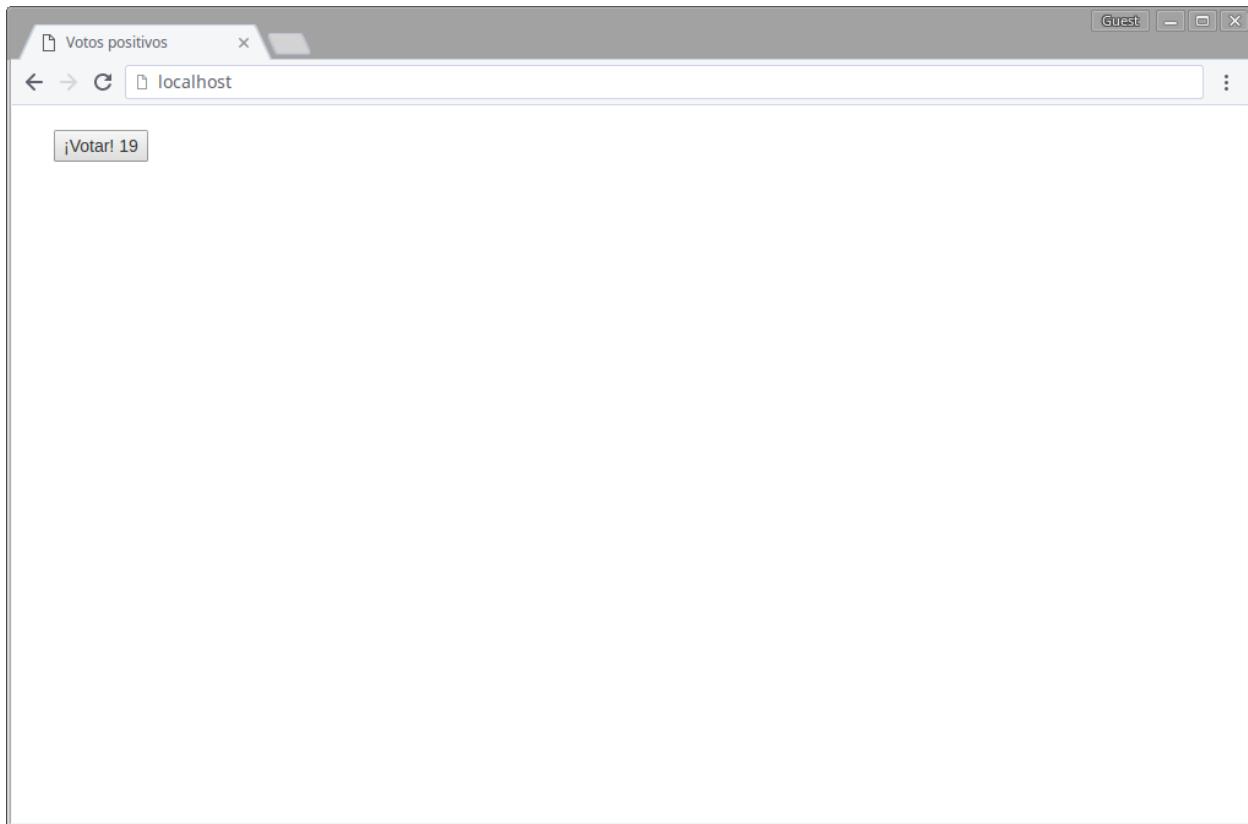
El evento **keyup** ocurre cuando el usuario libera una tecla. Puedes encontrar una lista completa de eventos HTML [aquí³⁵](#).

Manejo de Eventos en Línea

Suficiente con hablar, vamos a seguir adelante y ver el manejo de eventos en acción. Debajo, hay un botón ‘Votar!’ que incrementa el número de votos positivos cada vez que se hace clic sobre él.

³⁵http://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.css" r\
4 el="stylesheet">
5 <title>Votos positivos</title>
6 </head>
7 <body>
8     <div class="container">
9         <button v-on:click="upvotes++">
10            ¡Votar! {{upvotes}}
11        </button>
12    </div>
13 </body>
14 <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1\
15 /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>
```



Contador de votos a favor

Dentro de nuestros datos hay una variable `upvotes`. En este caso, enlazamos un evento listener para cada `click` con la declaración que está justo a su lado. Cada vez que el botón es presionado simplemente estamos incrementando el total de votos positivos por uno, usando el operador de incremento (`upvotes++`).

Manejo de Eventos usando Métodos

Ahora vamos a hacer exactamente lo mismo que antes, pero usando un método. Un método en Vue.js es un bloque de código diseñado para realizar una tarea en particular. Para ejecutar un método, primero tienes que definirlo y luego invocarlo.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" r\
4 el="stylesheet">
5 <title>Votos positivos</title>
6 </head>
7 <body>
8   <div class="container">
9     <button v-on:click="upvote">
10       ¡Votar! {{upvotes}}
11     </button>
12   </div>
13 </body>
14 <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1\>
15 /vue.js"></script>
16 <script type="text/javascript">
17 new Vue({
18   el: '.container',
19   data: {
20     upvotes: 0
21   },
22   // Define los métodos en el objeto **`methods`**
23   methods: {
24     upvote: function(){
25       // Dentro de los métodos, **`this`** hace referencia a la instancia de V\ue
26       this.upvotes++;
27     }
28   }
29 })
30 </script>
31 </html>
```

Estamos enlazando un evento click a un método llamado **upvote**. Funciona igual que antes, pero es más limpio y fácil de entender al leer tu código.



Advertencia

Los manejadores de eventos están restringidos a ejecutar solo una declaración.

Abreviatura para v-on

Cuando te encuentras usando v-on todo el tiempo en un proyecto, vas a encontrar que tu HTML rápidamente se ensuciará. Afortunadamente, hay una abreviatura para v-on, el simbolo @. El @ reemplaza a v-on: y al usarlo el código luce *mucho más limpio*. Usar la abreviatura es totalmente opcional.

Con el uso de @ el botón de nuestro ejemplo previo será:

Escuchando a ‘click’ usando v-on:

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

Escuchando a ‘click’ usando la abreviatura @

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

Modificadores de Eventos

Vamos a seguir adelante y crear una aplicación de Calculadora. Para hacerlo, usaremos un formulario con dos campos de texto y un lista desplegable para seleccionar la operación deseada.

Aunque el siguiente código se ve bien, nuestra calculadora no funciona como queremos.

```
1 <html>  
2   <head>  
3     <title>Calculadora</title>  
4     <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" \/  
5       rel="stylesheet">  
6   </head>  
7   <body>  
8     <div class="container">  
9       <h1>Ingresa 2 números y escoge una operación.</h1>  
10      <form class="form-inline">  
11        <!-- Nota aquí que el modificador especial 'number'  
12          es pasado para convertir el texto ingresado a números.-->  
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  <!-- Nota aquí que el modificador especial 'number'
21  es pasado para convertir el texto ingresado a números.-->
22  <input v-model.number="b" class="form-control">
23  <button type="submit" @click="calculate"
24  class="btn btn-primary">
25      Calcular
26  </button>
27 </form>
28 <h2>Resultado: {{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/2.0.1/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 })
```

```
59      }
60  }
61 },
62 });
63 </script>
64 </html>
```

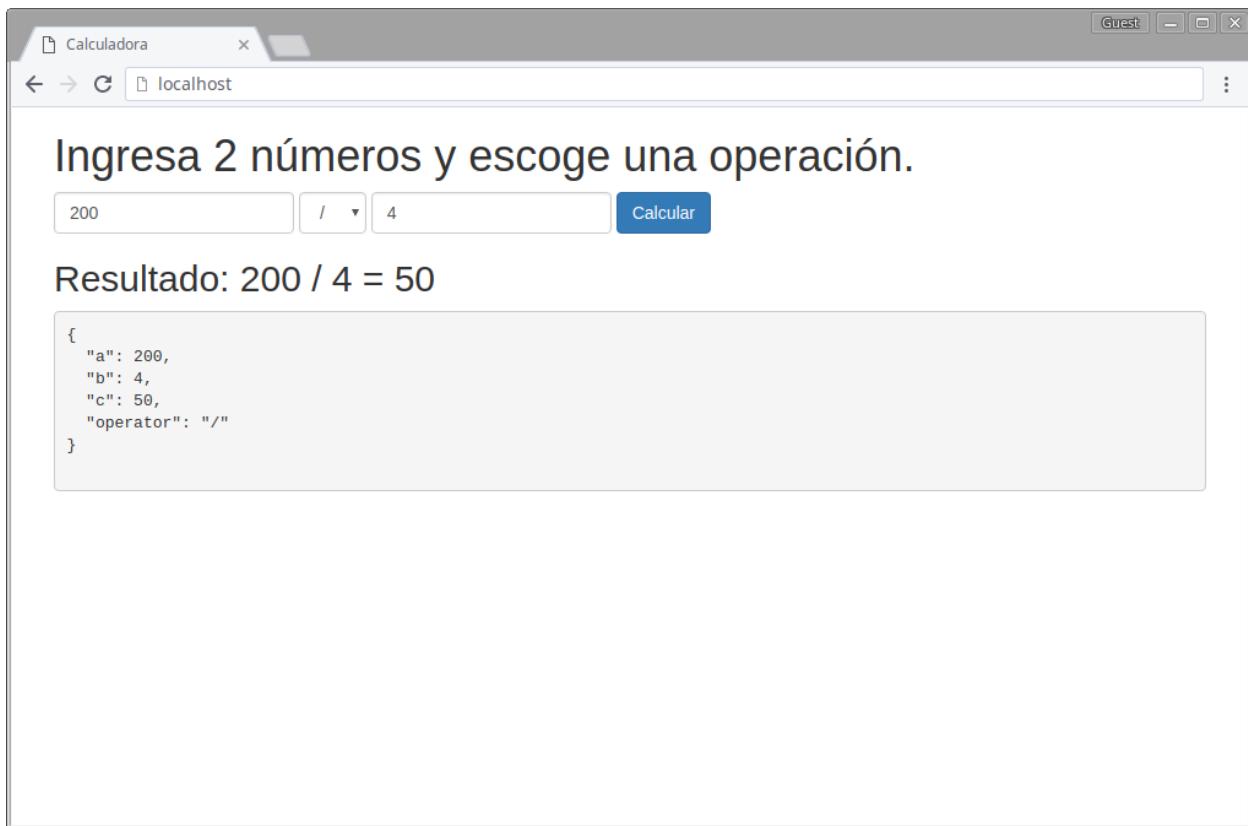
Si tratas de ejecutar este código tú mismo verás que cuando el botón “calcular” es presionado en lugar de calcular, recarga la página.

Esto tiene sentido, porque cuando presionas “calcular” en realidad estás enviando el formulario y por lo tanto la página se recarga.

Para evitar el envío del formulario tenemos que cancelar la acción por defecto del evento `onsubmit`. Es una necesidad muy común llamar a `event.preventDefault()` dentro de nuestro método de manejo de eventos. En nuestro caso el método de manejo de eventos es llamado `calculate`.

Así que, nuestro método se convertirá en:

```
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;
  }
}
```



Usando Modificadores de Evento para construir una calculadora.

Aunque podemos hacer esto fácilmente dentro de los métodos, sería mejor si los métodos pudieran concentrarse en la lógica de datos en lugar de tener que lidiar con detalles de los eventos del DOM.

Vue.js proporciona cuatro modificadores de eventos para `v-on` para prevenir el comportamiento por defecto del evento:

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

Así que, usando `.prevent`, nuestro botón de submit cambiará de:

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

a:

```
1 <!-- El evento submit no recargará la página -->
2 <button type="submit" @click.prevent="calculate">Calculate</button>
```

Ahora podemos eliminar sin problemas `event.preventDefault()` de nuestro método `calculate`.



Nota

`.capture` y `.self` son raramente usados, así que no nos detendremos a explicarlos. Si estás interesado en aprender más sobre *Orden de Eventos* echa un vistazo a este [tutorial](#)³⁶.

Modificadores de Teclas

Cuando haces foco en uno de los campos de texto y presionas enter te darás cuenta que el método `calculate` está siendo invocado. Si el botón no estuviera dentro del formulario, o si no hubiera ningún botón en absoluto, pudieras escuchar un evento del teclado en su lugar.

Al escuchar eventos del teclado, a menudo necesitamos verificar códigos de teclas. El código de tecla para el botón `Enter` es 13. Así que podríamos usarlo así:

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

Recordar todos los códigos de teclas es complicado, por lo que Vue proporciona alias para las teclas más comunes:

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

Así que, para ejecutar el método `calculate` cuando Enter es presionado en nuestro ejemplo, los campos serán así:

³⁶http://www.quirksmode.org/js/events_order.html

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



Tip

Cuando tienes un formulario con muchos campos/botones/etc y necesitas prevenir su comportamiento por defecto de envío, puedes modificar el evento **submit** del formulario.

Por ejemplo: `<form @submit.prevent="calculate">`

Finalmente, la calculadora está operativa.

Propiedades Computadas

Las expresiones en línea de Vue.js son muy convenientes, pero por la lógica más complicada, deberías usar propiedades computadas. Básicamente, las propiedades computadas son variables cuyo valor depende de otros factores.

Las propiedades computadas trabajan como funciones que pueden utilizarse como propiedades. Pero hay una diferencia significativa. Cada vez que una dependencia de una propiedad computada cambia, el valor de la propiedad computada se re-evalúa.

En Vue.js defines las propiedades computadas en el objeto **computed** dentro de tu instancia de **Vue**.

```
1 <html>
2 <head>
3   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" r\
4     el="stylesheet">
5   <title>Hola 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://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js"></script>
16 <script type="text/javascript">
17   new Vue({
18     el: '.container',
19     data: {
```

```
20      a: 1,
21    },
22  computed: {
23    // Un getter computado
24    b: function () {
25      // **`this`** hace referencia a la instancia de Vue
26      return this.a + 1
27    }
28  }
29 });
30 </script>
31 </html>
```

Hemos puesto dos variables. La primera, **a**, está establecida en 1 y la segunda, **b**, será establecida por el resultado devuelto por la función dentro del objeto computed. En este ejemplo el valor de **b** será establecido a 2.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" r\
4 el="stylesheet">
5 <title>Hola 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://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js"></script>
17 <script type="text/javascript">
18 new Vue({
19   el: '.container',
20   data: {
21     a: 1,
22   },
23   computed: {
24     // Un getter computado
25     b: function () {
26       // **`this`** hace referencia a la instancia de Vue
```

```
27     return this.a + 1
28   }
29 }
30 });
31 </script>
32 </html>
```

El ejemplo de arriba es el mismo que el previo, pero con una diferencia. Un campo de texto está enlazado a la variable **a**. El resultado deseado sería cambiar el valor del atributo enlazado e inmediatamente actualizar el resultado de **b**. Pero observa que no funciona como esperamos.

Si ejecutas este código y estableces la variable **a** a 5 esperas que **b** sea igual a 6. Sin embargo, **b** es establecido a 51.

¿Por qué está pasando esto? Bueno, como ya habrás pensado, **b** toma el valor dado desde el campo **a** como una cadena de texto y concatena el número 1 al final de esta.

Una posible solución es usar la función **parseFloat()** que analiza una cadena de texto y devuelve un número de punto flotante.

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

Otra posibilidad que viene a la mente es usar **<input type="number">** que es usado para campos que deben contener un valor numérico.

Pero hay una forma más limpia. Con Vue.js, cuando quieres que lo ingresado por el usuario sea automáticamente persistido como un número, puedes adjuntar el modificador especial **.number**.

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

El modificador `number` va a darnos el resultado deseado sin ningún esfuerzo adicional.

Para mostrar un panorama más amplio de las propiedades computadas, vamos a hacer uso de estas y construir la calculadora que ya hemos mostrado, pero esta vez usando propiedades computadas en lugar de métodos.

Empecemos con un ejemplo sencillo, donde una propiedad computada `c` contiene la suma de `a` más `b`.

```
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>Hola Vue</title>
6   </head>
7   <body>
8     <div class="container">
9       <h1>Ingresa 2 números para calcular su suma.</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>Resultado: {{a}} + {{b}} = {{c}}</h2>
16      <pre> {{ $data }} </pre>
17    </div>
18  </body>
19  <script src="https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/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>
```

El código inicial está listo, y en este punto el usuario puede escribir 2 números y obtener la suma de estos. Una calculadora que puede hacer las cuatro operaciones básicas es el objetivo ¡así que continuemos construyendo!.

Dado que el código HTML será el mismo que el de la [calculadora que construimos en la sección anterior de este capítulo](#) (excepto que ahora no necesitamos un botón) voy a mostrar solo el bloque de código JavaScript.

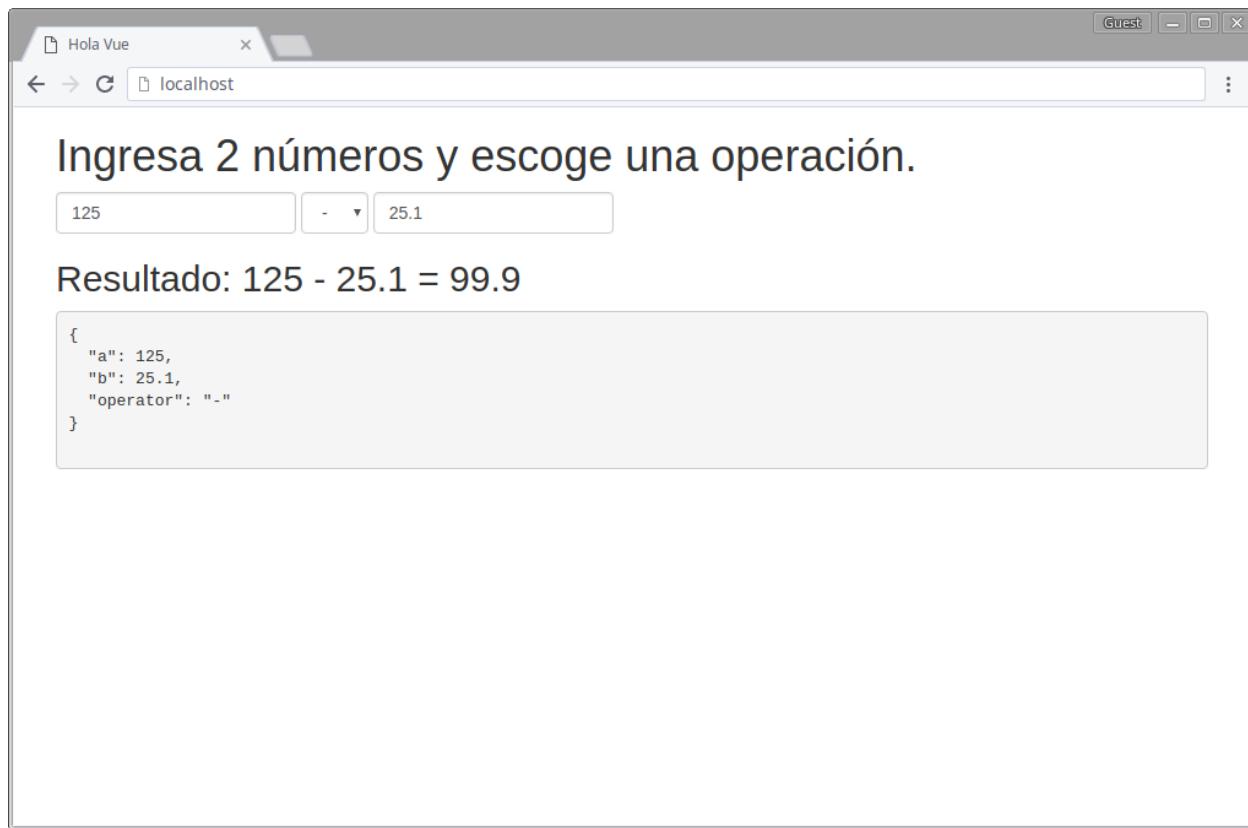
```
1 new Vue({
2   el: '.container',
3   data: {
4     a: 1,
5     b: 2,
6     operator: "+",
7   },
8   computed: {
9     c: function () {
10       switch (this.operator) {
11         case "+":
12           return this.a + this.b
13           break;
14         case "-":
15           return this.a -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

La calculadora está lista para usar. ¡La única cosa que tuvimos que hacer fue mover lo que estaba dentro del método `calculate` a la propiedad computada `c`! Cuando cambias el valor de `a` o `b` el resultado se actualiza en tiempo real. No necesitamos botones, eventos o cualquier cosa. ¡Cuán genial es eso?.



Nota

Nota aquí que un enfoque normal sería tener una declaración `if` para evitar errores de división. Pero ya hay una predicción para este tipo de defectos. Si el usuario escribe `1/0` el resultado se vuelve automáticamente infinito. Si el usuario ingresa un texto el resultado mostrado es “no es un número”.



Calculadora construida con propiedades computadas.



Video

El manejo de eventos es cubierto en la lección 9 del curso de Vue 2 en Styde³⁷.

³⁷<https://styde.net/manejo-de-eventos-en-vue-js-2/>



Video

Las propiedades computadas son cubiertas en la lección 11 del curso de Vue 2 en Styde³⁸.



Código de Ejemplo

Puedes encontrar el código de ejemplo de este capítulo en GitHub³⁹.

³⁸<https://styde.net/diferencias-entre-metodos-computed-properties-y-filtros-en-vue-js-2/>

³⁹<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter5>

Ejercicio

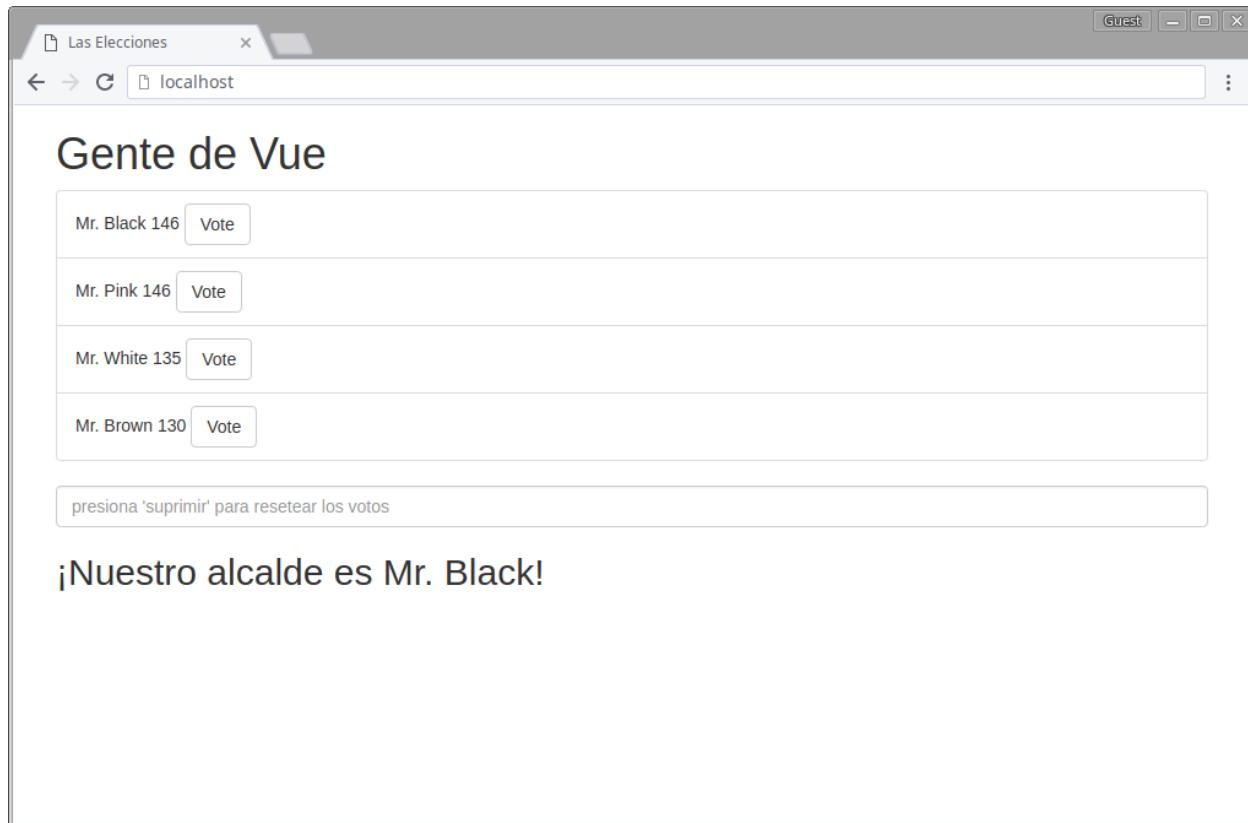
Ahora que tienes un conocimiento básico del manejo de eventos, métodos, propiedades computadas, etc. de Vue, deberías probar algo un poco más desafiante. Comienza por crear un arreglo de candidatos para “Alcalde”. Cada candidato tiene un “nombre” y un número de “votos”. Usa un botón para incrementar el número de votos de cada candidato. Usa una propiedad computada para determinar quién es el “Alcalde” actual y muestra su nombre.

Finalmente, agrega un campo de texto. Cuando este campo de texto esté enfocado y la tecla `delete` presionada, las elecciones comienzan desde el principio. Esto significa que todos los votos se convierten en 0.



Pista

Los métodos `sort()` y `map()` de JavaScript podrían resultar muy útiles y los modificadores de teclas te llevarán ahí.



Pantalla de Ejemplo



Possible Solución

Puedes encontrar una posible solución a este ejercicio [aquí](#)⁴⁰.

⁴⁰<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter5.html>

Filtros

En los dos capítulos anteriores vimos renderización de listas, métodos y propiedades computadas. Ahora es un buen momento para hacer algunos ejemplos usando toda la información anterior. En este capítulo, cubriremos como:

1. Filtrar un arreglo de elementos.
2. Ordenar un arreglo de elementos.
3. Aplicar un filtro personalizado.
4. Usar bibliotecas de utilidades.

El plan es ir a través de ejemplos similares a los anteriores, combinando algunas o todas las técnicas que vimos.

Resultados Filtrados

Algunas veces necesitamos mostrar una versión filtrada de un arreglo sin mutar o resetear realmente los datos originales. Continuando el ejemplo anterior, [Iterar a través de un arreglo de Objetos](#), nos gustaría mostrar una lista con las historias escritas por *Alex* y una lista con las historias escritas por *John*. Podemos lograr esto, creando un método que filtra nuestro arreglo y devuelve el resultado para ser renderizado.



Info

A partir de Vue 2.0 los filtros de Vue no pueden ser usados dentro de `v-for`. Los filtros ahora sólo pueden ser usados dentro de interpolaciones de texto (`{} {}`). El equipo de Vue sugiere mover la lógica de los filtros a JavaScript, para que puedan ser reutilizados a lo largo de tu componente.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
4     <title>Historias de los usuarios</title>
5 </head>
6 <body>
7     <div class="container">
8         <h1>Escuchemos algunas historias!</h1>
9         <div>
10             <h3>Historias de Alex</h3>
11             <ul class="list-group">
12                 <li v-for="story in storiesBy('Alex')"
13                     class="list-group-item"
14                     >
15                         {{ story.writer }} dijo "{{ story.plot }}"
16                     </li>
17             </ul>
18             <h3>Historias de John</h3>
19             <ul class="list-group">
20                 <li v-for="story in storiesBy('John')"
21                     class="list-group-item"
22                     >
23                         {{ story.writer }} dijo "{{ story.plot }}"
24                     </li>
25             </ul>
26         </div>
27         <pre>
28             {{ $data }}
29         </pre>
30     </div>
31 </body>
32 <script src="https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js"></script>
33 <script type="text/javascript">
34 new Vue({
35     el: '.container',
36     data: {
37         stories: [
38             {
39                 plot: "Estrellé mi auto hoy!",
40                 writer: "Alex"
41             },
42             {
43                 plot: "Me perdí en la montaña",
44                 writer: "John"
45             }
46         ]
47     }
48 })
```

```
44         plot: "¡Ayer, alguien robó mi bolso!",
45         writer: "John"
46     },
47     {
48         plot: "Alguien se comió mi chocolate...",
49         writer: "John"
50     },
51     {
52         plot: "¡Me comí el chocolate de alguien!",
53         writer: "Alex"
54     },
55 ]
56 },
57 methods:
58 {
59     // Un método que filtra las historias dependiendo del escritor
60     storiesBy: function (writer) {
61         return this.stories.filter(function (story) {
62             return story.writer === writer
63         })
64     },
65 }
66 })
67 </script>
68 </html>
```

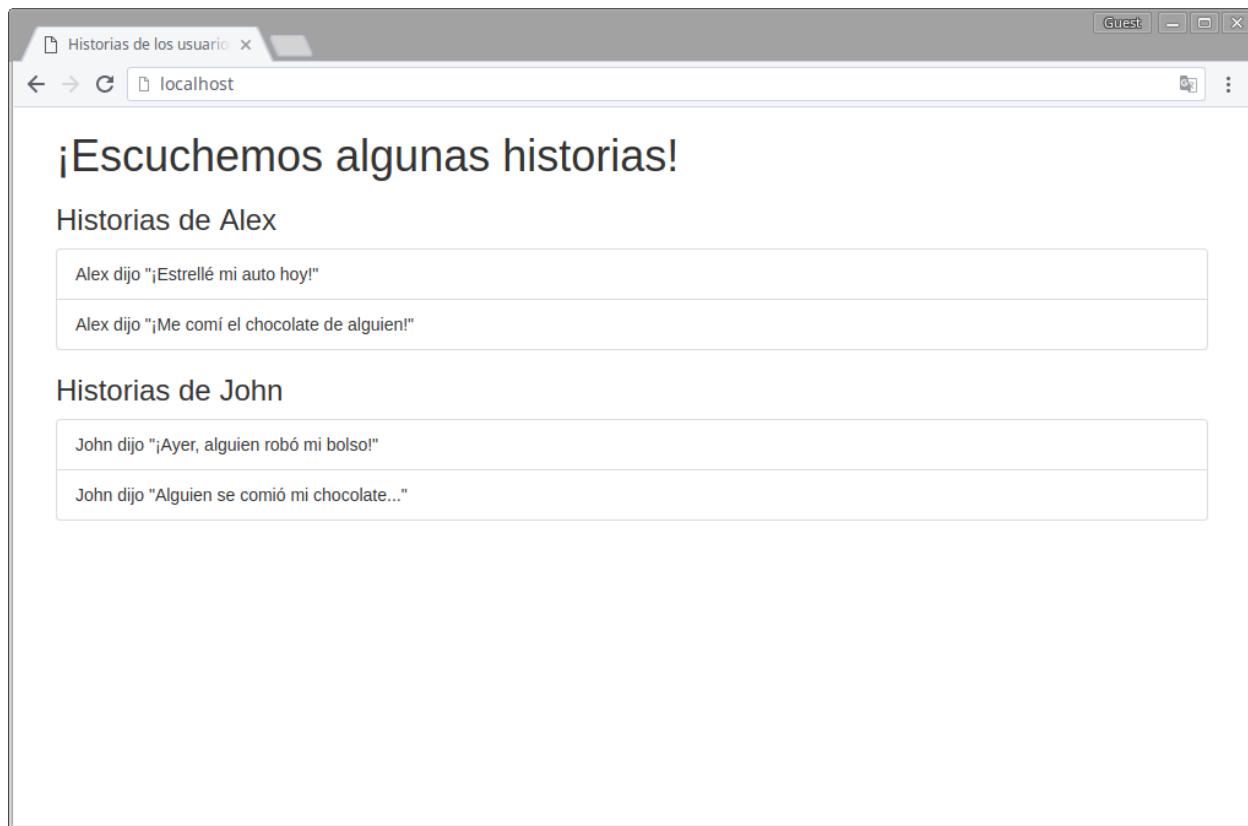


Info

Dentro del método `storiesBy`, usamos la función nativa de JavaScript `filter`.⁴¹ La función `filter()` crea un nuevo arreglo con todos los elementos que pasen la prueba, implementada por la función proporcionada.

Hemos creado un método llamado `storiesBy` que toma un `writer` como argumento y devuelve un arreglo filtrado con las historias del escritor. Podemos entonces usar esto para mostrar las historias de cada escritor usando la directiva `v-for` con el formato `story in storiesBy('Alex')`, como puedes ver en el ejemplo de arriba.

⁴¹https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Array/filter



Historias filtradas por escritor.



Nota

Como habrás podido notar, nuestra etiqueta `li` se está volviendo realmente grande, así que la hemos dividido en más líneas. El resultado sigue siendo el mismo que con el uso de filtros, introducido en Vue 1.x.

Suficientemente fácil, ¿no?

Usando Propiedades Computadas

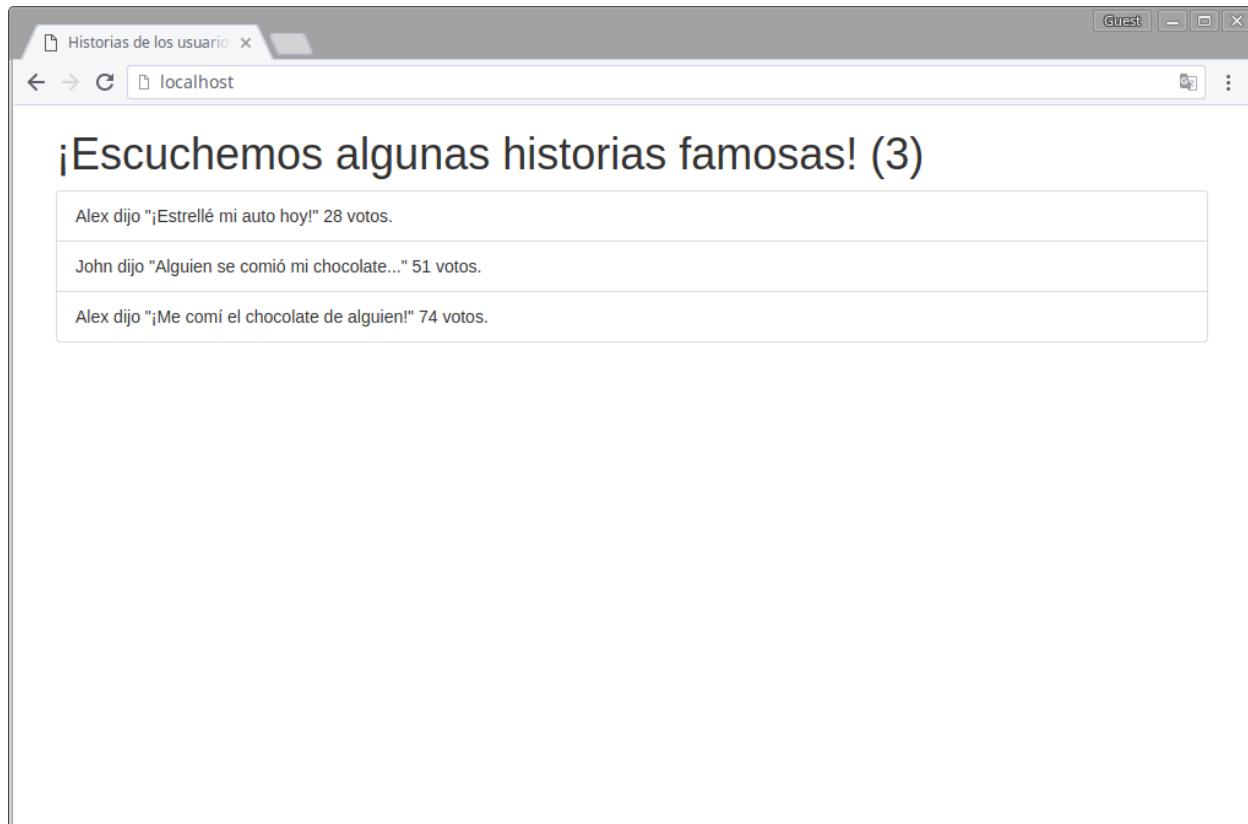
Una *computed property* también puede ser usada para filtrar un arreglo. Usar una propiedad computada para realizar el filtrado de arreglos te da mayor control y más flexibilidad, ya que es JavaScript completo, y te permite acceder al resultado filtrado en otra parte. Por ejemplo, puedes obtener el tamaño de un arreglo filtrado en cualquier parte de tu código.

Primero mejoraremos nuestras historias con una nueva propiedad llamada `upvotes`. Luego, filtraremos las historias `famosas`. Definimos como `famosas` las *historias* que tengan más de 25 votos positivos. Esta vez, crearemos una propiedad computada que devuelve el arreglo filtrado.

```
new Vue({
  el: '.container',
  data: {
    stories: [
      {
        plot: "¡Estrellé mi auto hoy!",
        writer: "Alex",
        upvotes: 28
      },
      {
        plot: "¡Ayer, alguien robó mi bolso!",
        writer: "John",
        upvotes: 8
      },
      {
        plot: "Alguien se comió mi chocolate...",
        writer: "John",
        upvotes: 51
      },
      {
        plot: "¡Me comí el chocolate de alguien!",
        writer: "Alex",
        upvotes: 74
      },
    ],
  },
  computed: {
    famous: function() {
      return this.stories.filter(function(item){
        return item.upvotes > 25;
      });
    }
  }
})
```

En nuestro código HTML en lugar del arreglo `stories` renderizaremos la propiedad computada `famous`.

```
<body>
  <div class="container">
    <h1> ¡Escuchemos algunas historias famosas! ({{famous.length}}) </h1>
    <ul class="list-group">
      <li v-for="story in famous"
          class="list-group-item">
        >
          {{ story.writer }} dijo "{{ story.plot }}"
          {{ story.upvotes }} votos.
      </li>
    </ul>
  </div>
</body>
```



Filtrar un arreglo usando una propiedad computada

Eso es todo. Hemos filtrado nuestro arreglo usando una propiedad computada. ¿Te diste cuenta con qué facilidad logramos mostrar el *número de historias famosas* junto a nuestro mensaje de encabezado usando `{{famous.length}}`?

Ahora vamos a implementar una búsqueda básica (pero genial). Cuando el usuario escribe una parte de una historia, podemos adivinar en tiempo real que historia es y quien la escribió.

Agregaremos un campo de texto `input`, enlazado a una variable vacía (`query`), para que podamos filtrar dinámicamente nuestro arreglo de historias.

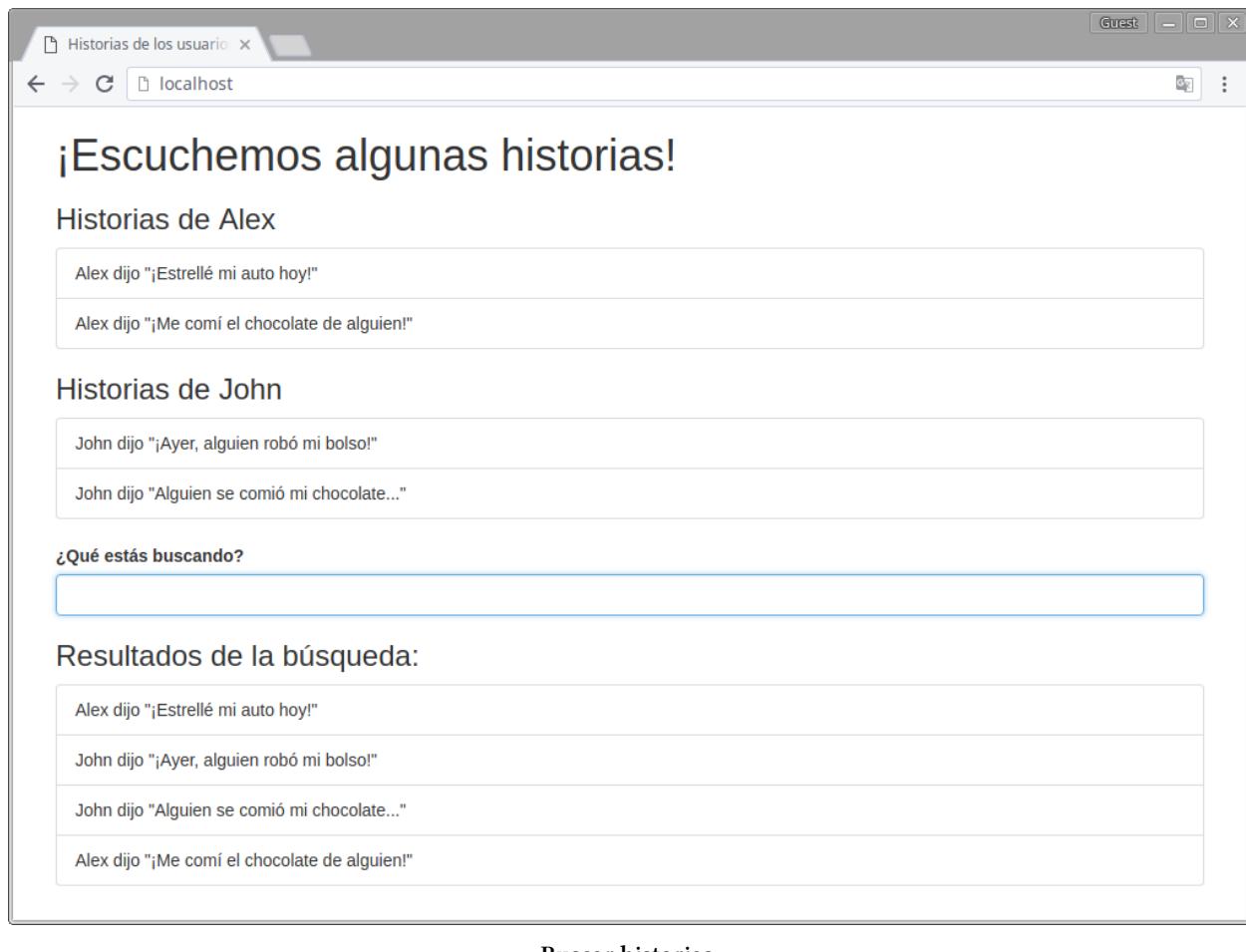
```
1 <div class="container">
2   <h1> ¡Escuchemos algunas historias! </h1>
3   <div>
4     ...
5     <div class="form-group">
6       <label for="query">
7         ¿Qué estás buscando?
8       </label>
9       <input v-model="query" class="form-control">
10      </div>
11      <h3>Resultados de la búsqueda: </h3>
12      <ul class="list-group">
13        <li v-for="story in search"
14          class="list-group-item">
15          >
16          {{ story.writer }} dijo "{{ story.plot }}"
17        </li>
18      </ul>
19    </div>
20 </div>
```

Luego crearemos una propiedad computada llamada `search`. Junto con la función nativa de JavaScript `filter`, vamos a usar la función de JavaScript `includes42` que determina si una cadena de texto puede ser encontrada dentro de otra cadena de texto.

```
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 })
```

⁴²https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/includes

```
14     computed: {
15       search: function () {
16         var query = this.query
17         return this.stories.filter(function (story) {
18           return story.plot.includes(query)
19         })
20       }
21     }
22   })
```



The screenshot shows a web browser window titled "Historias de los usuarios" at "localhost". The page displays a search interface for filtering user stories.

Section: Historias de Alex

- Alex dijo "¡Estrellé mi auto hoy!"
- Alex dijo "¡Me comí el chocolate de alguien!"

Section: Historias de John

- John dijo "¡Ayer, alguien robó mi bolso!"
- John dijo "Alguien se comió mi chocolate..."

Search Bar: ¿Qué estás buscando?

Results: Resultados de la búsqueda:

- John dijo "Alguien se comió mi chocolate..."
- Alex dijo "¡Me comí el chocolate de alguien!"

Buscando por 'choco'.

¿No es genial?

Resultados Ordenados

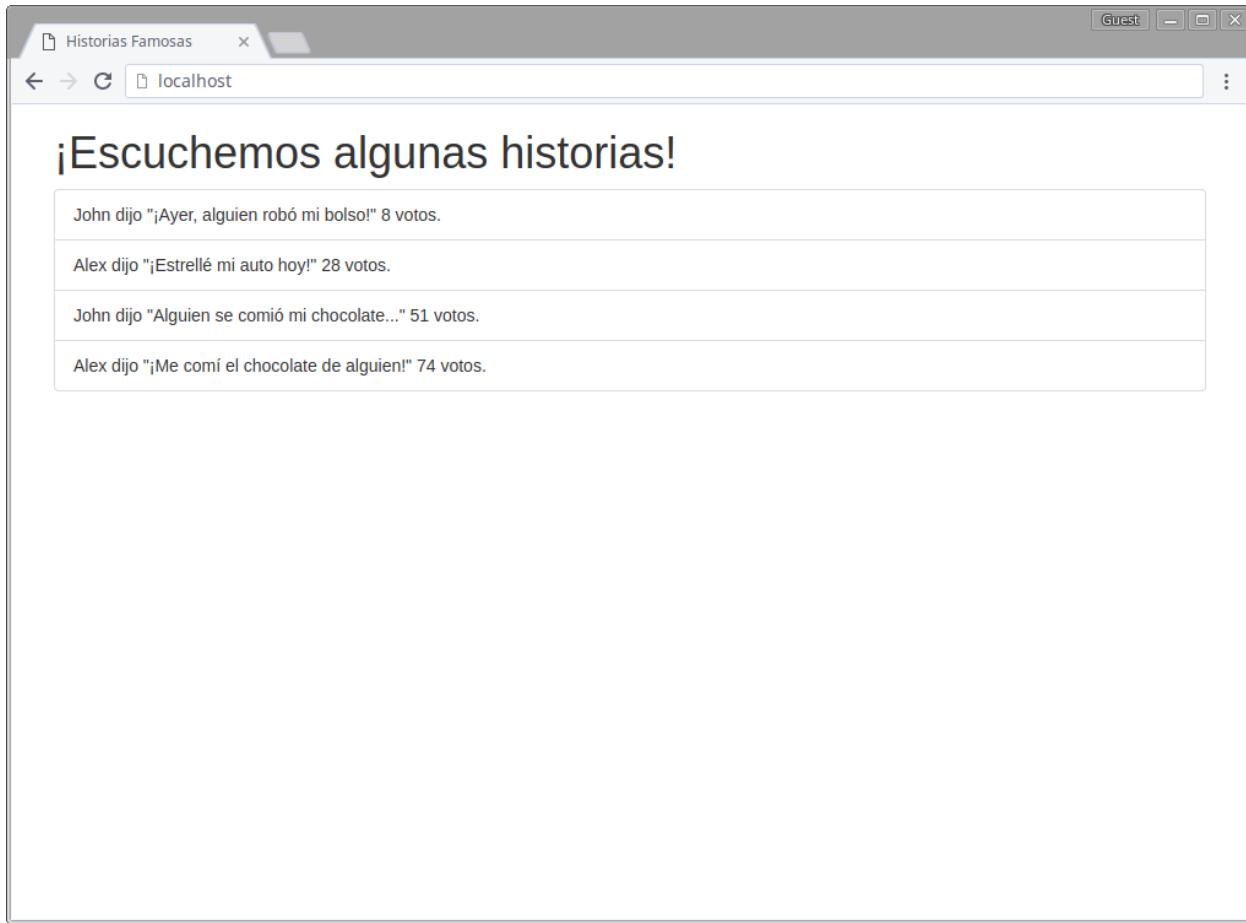
Algunas veces podemos querer mostrar los elementos de un arreglo ordenados en base a algún criterio. Podemos usar una *propiedad computada* para mostrar nuestro arreglo, ordenado por el número de *upvotes* de cada historia. Para ordenar el arreglo vamos a usar la función de JavaScript `sort`⁴³, la cual esta lista para ordenar los elementos de un arreglo y devuelve el arreglo.

Cuanto más famosa es una historia, más alta debe aparecer.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" r\
4 el="stylesheet">
5     <title>Historias Famosas</title>
6 </head>
7 <body>
8     <div class="container">
9         <h1>Escuchemos algunas historias!</h1>
10        <ul class="list-group">
11            <li v-for="story in orderedStories"
12                class="list-group-item"
13                >
14                {{ story.writer }} dijo "{{ story.plot }}"
15                {{ story.upvotes }} votos.
16            </li>
17        </ul>
18        <pre>
19            {{ $data }}
20        </pre>
21    </div>
22 </body>
23 <script src="https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/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){
```

⁴³https://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>
```



Arreglo de historias ordenado por votos a favor.

Hmmm, el arreglo está ordenado pero esto no es lo que esperábamos. Queríamos las **historias famosas primero**.

Para cambiar el orden del arreglo ordenado tenemos que echar un vistazo a la función **sort**. En JavaScript **sort(compareFunction)**, si **compareFunction** es suministrada, los elementos del arreglo son ordenados de acuerdo al valor devuelto por **compareFunction**. Si **a** y **b** son dos elementos siendo comparados, entonces:

- Si **compareFunction(a, b)** es menor a 0, mover **a** a un índice menor a **b**.

- Si `compareFunction(a, b)` es 0, dejar `a` y `b` sin cambios.
- Si `compareFunction(a, b)` es mayor a 0, mover `b` a un índice menor que `a`.

En nuestro caso, `compareFunction` será:

`compareFunction`

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

Por lo tanto, para cambiar el orden de ascendente a descendente podemos multiplicar por `-1` el valor devuelto (`return (a.upvotes - b.upvotes) * -1`).

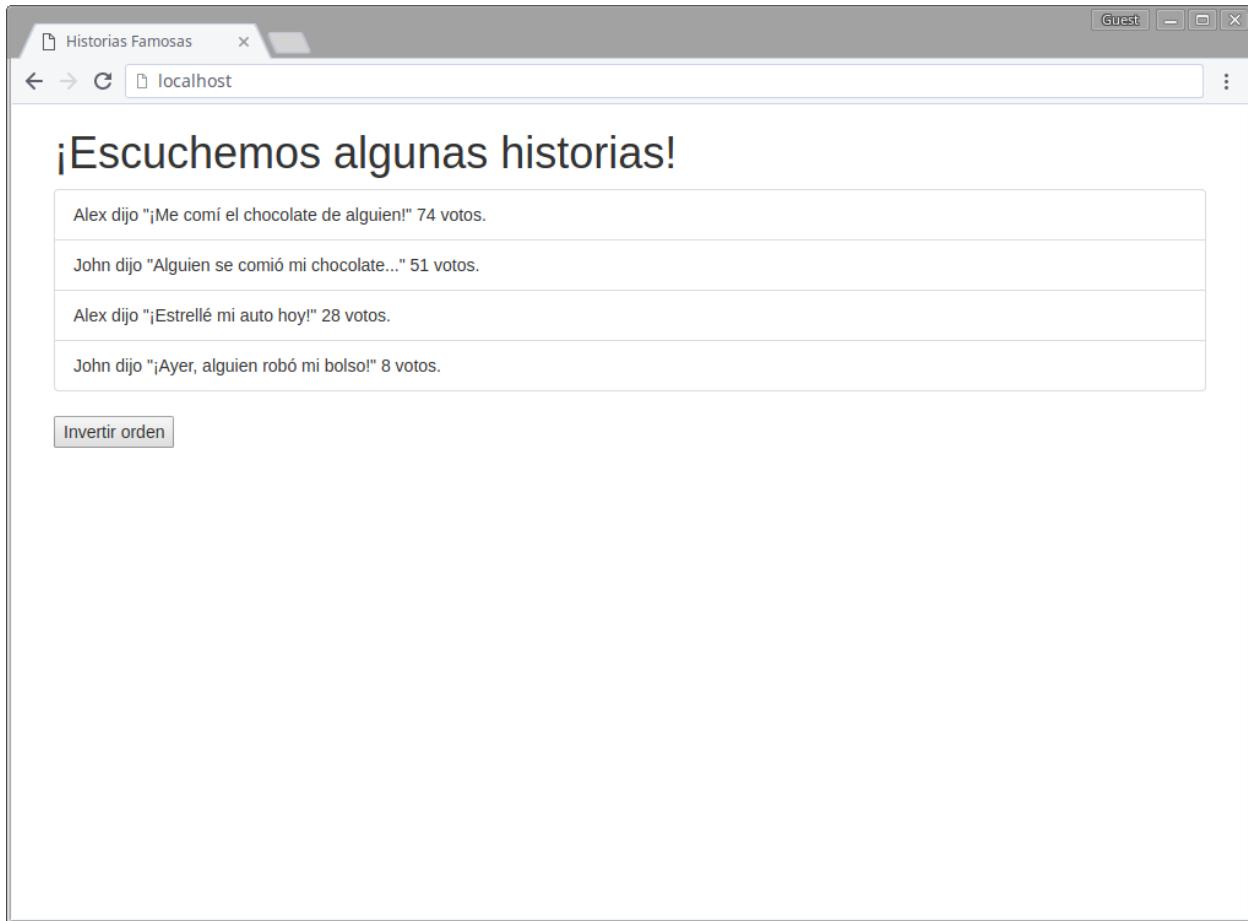
Podemos cambiar el orden dinámicamente usando una variable, `order`. Un botón `button` será usado, que alternará el valor de la nueva variable entre `-1` y `1`.

```
1 <div class="container">
2     <h1>iEscuchemos algunas historias!</h1>
3     <ul class="list-group">
4         <li v-for="story in orderedStories"
5             class="list-group-item"
6             >
7                 {{ story.writer }} dijo "{{ story.plot }}"
8                 {{ story.upvotes }} votos.
9             </li>
10        </ul>
11        <button @click="order = order * -1">Invertir orden</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 })
```

Inicializamos la variable **order** con el valor de **-1** y luego la pasamos a nuestra propiedad computada, así cada vez que el botón es presionado la variable cambia de valor y el arreglo cambia de orden.



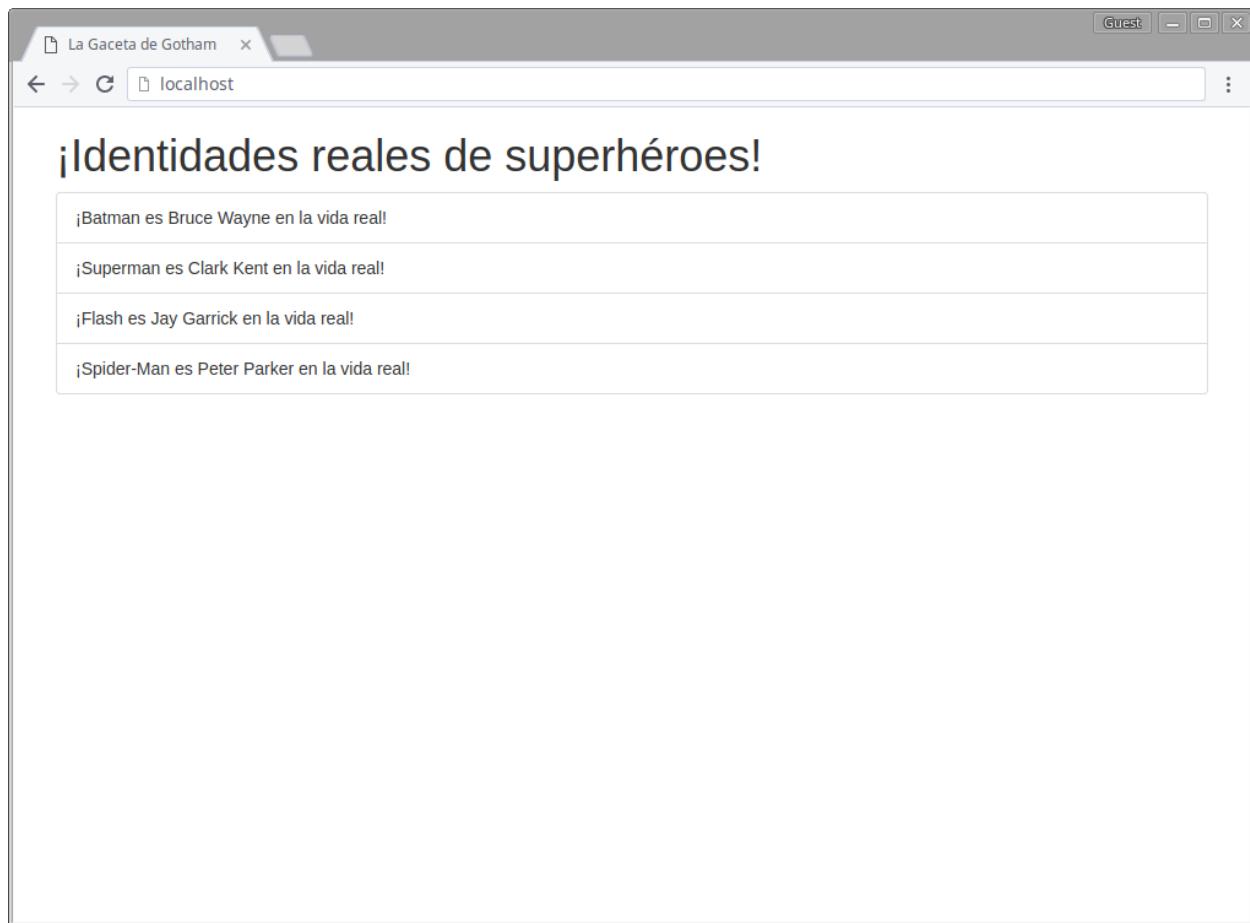
Arreglo en orden descendente

Filtros personalizados

Para demostrar los filtros personalizados realizaremos un nuevo ejemplo sencillo. Supongamos que ahora estamos a cargo del periódico de la ciudad de Gotham “The Gotham Gazette”. Nuestro primer trabajo es difundir la noticia de las identidades secretas de los héroes. Sabemos el nombre y apellido de ellos y queremos hacer una bonita lista donde cada identidad secreta será expuesta. Aquí es donde el método global `Vue.filter()` entra en juego, para crear un filtro que puede tomar un *héroe* y devolver toda su información para ser mostrada sin contaminar nuestro código HTML. Para registrar un filtro podemos pasar un `filterID` y una `filterFunction`, que devuelve un valor procesado. Luego usaremos el filtro dentro de interpolaciones de texto de la siguiente forma:

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" r\
4 el="stylesheet">
5     <title>La Gaceta de Gotham</title>
6 </head>
7 <body>
8     <div class="container">
9         <h1>¡Identidades reales de superhéroes!</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/2.0.1/vue.js">
20 </script>
21 <script>
22 Vue.filter('snitch', function (hero) {
23     return '¡' + hero.secretId + ' es '
24         + hero.firstname + ' '
25         + hero.lastname + ' en la vida real!'
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>
```



Filtro personalizado 'famous' en acción.

Bibliotecas de Utilidades

En este punto, nos gustaría señalar que cuando necesitas ordenar/filtrar/indexar datos de forma más avanzada, deberías considerar usar una biblioteca de utilidades de JavaScript. Hay algunas excelentes

bibliotecas de utilidades allí afuera como [Lodash⁴⁴](#), [Underscore⁴⁵](#), [Sugar⁴⁶](#), etc.

Para obtener una mejor comprensión, vamos a incluir *Lodash* y actualizar nuestro ejemplo anterior.

Si sigues el ejemplo, asegurate de incluir *Lodash* desde un CDN en tu archivo [HTML](#).

El método `orderBy` de Lodash devuelve un nuevo arreglo ordenado. Así que lo usaremos dentro de nuestra propiedad computada para ordenar el arreglo historias.

Sintaxis

La sintaxis de `orderBy` en Lodash es:

```
_orderBy(collection, [iteratees=[_.identity]], [orders])
```

No dejes que el segundo argumento te confunda. Es realmente simple. El primer argumento representa el arreglo que quieras ordenar. El segundo argumento espera un arreglo de claves en la que se basará el ordenamiento. El tercer argumento espera un arreglo de órdenes para cada clave.

Por ejemplo si tenemos un arreglo:

```
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}
]
```

Y ejecutamos:

```
_orderBy(kids, ['intelligence', 'strength'], ['desc', 'asc'])
```

Nuestro arreglo tendrá este orden:

```
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}
]
```

Debido a que el arreglo es primero ordenado por `intelligence` en orden descendente y luego por `strength` en orden ascendente.

⁴⁴<https://lodash.com>

⁴⁵<http://underscorejs.org/>

⁴⁶<https://sugarjs.com/>

Usaremos `_.orderBy` dentro de nuestra propiedad computada así:

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

Esto funciona, pero si ningún argumento `órdenes` es pasado, el arreglo será clasificado en orden ascendente. Adicionalmente, debería ser posible cambiar el orden del arreglo con un botón, justo como antes. Para hacerlo dinámicamente podemos establecer una propiedad de datos `order :desc` y crear un método para cambiar su valor:

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

Y el botón será casi el mismo, pero no del todo.

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

Eso es suficiente. Hemos logrado la misma funcionalidad usando *Lodash*.



Tip

Al usar una librería de utilidades para filtrar/ordenar datos puedes iterar el arreglo resultante sin usar ninguna propiedad computada.

Podemos actualizar este ejemplo para obtener la idea. Nuestro HTML que renderiza el arreglo ordenado sería:

```
<div class="container">
  <h1> ¡Escuchemos algunas historias! </h1>
  <ul class="list-group">
    <li v-for="story in _.orderBy(stories, ['upvotes'], ['desc'])">
      {{ story.writer }} dijo "{{ story.plot }}"
      {{ story.upvotes }} votos.
    </li>
  </ul>
</div>
```

Eso es todo. No hay necesidad de escribir *JavaScript*.



Video

El uso de filtros es cubierto en la [lección 10 del curso de Vue 2 en Styde](#)⁴⁷.



Código de Ejemplo

Puedes encontrar el código de ejemplo de este capítulo en [GitHub](#)⁴⁸.

⁴⁷<https://styde.net/uso-de-filtros-en-interpolaciones-de-texto-en-vue-js-2/>

⁴⁸<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter6>

Ejercicio

Para el ejercicio de este capítulo deberás hacer lo siguiente. Comienza por crear un arreglo de personas. Cada persona tiene un nombre y una edad. Usando lo que has aprendido hasta el momento, trata de renderizar el arreglo en una lista y ordénala por “edad”. Después de eso, crea una segunda lista debajo y crea una propiedad computada llamada “mayores”, que devolverá todas las personas mayores de 65 años.

Sientete libre de llenar el arreglo con tus propios datos. **Se cuidadoso** de añadir personas con una edad mayor y menor a 65 para asegurarte de que tu filtro está funcionando correctamente. ¡Adelante!.



Pista

La función nativa `.filter` es necesaria aquí.

The screenshot shows a web browser window titled "Gente de Gaul". The address bar says "localhost". The main content area has two sections:

- Gente de Gaul**: A list of five people with their names and ages:
 - Asterix 32
 - Obelix 33
 - Julius Caesar 66
 - Majestix 82
 - Methusalix 103
- Gente "mayor" de Gaul**: A list of three people who are 66 or older:
 - Majestix 82
 - Methusalix 103
 - Julius Caesar 66

Pantalla de Ejemplo



Possible Solución

Puedes encontrar una posible solución a este ejercicio [aquí](#)⁴⁹.

⁴⁹<https://github.com/hoottlex/the-majesty-of-vuejs-2/blob/master/homework/chapter6.html>

Componentes

¿Qué son los Componentes?

Los componentes son una de las características más poderosas de Vue.js. Ellos te ayudan a extender elementos básicos de HTML para encapsular código reusable. En un nivel alto, los componentes son elementos personalizados a los que el compilador de Vue.js asignará un comportamiento específico. En algunos casos, también pueden aparecer como un elemento HTML nativo extendido con el atributo especial `is`.

Es una manera realmente inteligente y poderosa de extender HTML con el fin de hacer nuevas cosas. En este capítulo vamos a comenzar con un ejemplo extremadamente sencillo. Luego, vamos a ver como los componentes nos pueden ayudar a mejorar el código que hemos creado en capítulos anteriores.

Usando Componentes

Vamos a comenzar con un componente sencillo. Para usar un componente, primero tenemos que registrararlo.

Una forma de registrar un componente es usar el método `Vue.component` y pasar la `etiqueta` y el `constructor`. Piensa en la `etiqueta` como el nombre del componente y el `constructor` como las opciones. En esta ocasión, nombraremos el componente `story` y definiremos la propiedad `story` (nuevamente). La opción `template` (cómo nos gustaría que se mostrara nuestra historia) está dentro del `constructor` donde otras opciones también serán añadidas.

Nuestro componente `story` será registrado así:

```
1 Vue.component('story', {  
2   template: '<h1> Mi caballo es genial! </h1>'  
3});
```

Ahora que hemos registrado el componente haremos uso de él. Agregaremos el elemento personalizado `<story>` dentro del HTML, para mostrar la historia.

```
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>Hola 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/2.0.1/vue.js"></script>
13 <script type="text/javascript">
14 Vue.component('story', {
15   template: '<h1> ¡Mi caballo es genial!</h1>'
16 });
17
18 new Vue({
19   el: '.container'
20 })
21 </script>
22 </html>
```

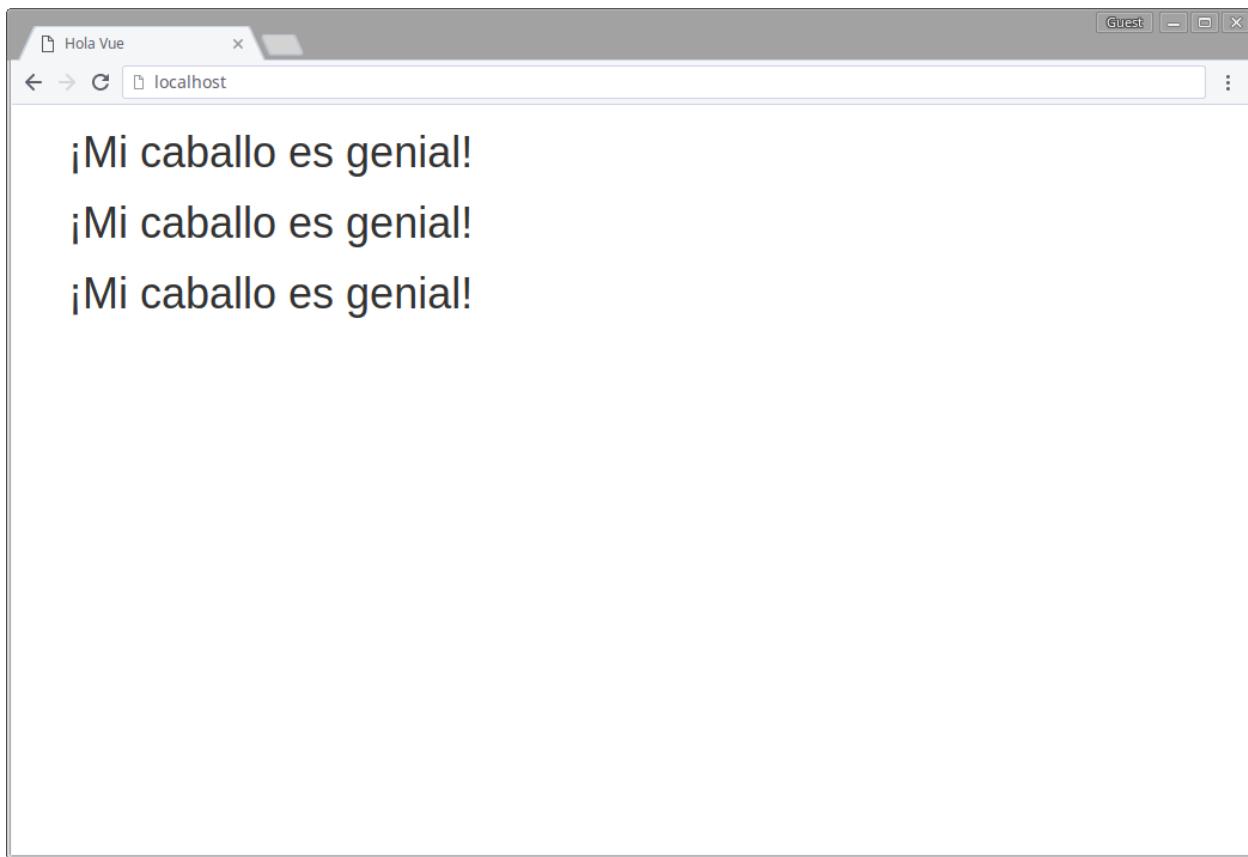


Nota

Observa aquí que puedes darle a tu componente personalizado cualquier nombre que quieras, pero generalmente se recomienda que uses un nombre único para evitar tener colisiones con etiquetas reales que podrían ser introducidas en algún punto en el futuro.

Como mencionamos en el comienzo del capítulo, los componentes son reutilizables. Lo que significa que puedes añadir tantos elementos `<story>` como quieras. El siguiente fragmento de HTML mostrará nuestra historia 3 veces.

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



Mostrando el componente story

Plantillas

Hay más de una forma de declarar una plantilla para un componente. La plantilla en línea que usamos anteriormente puede ensuciarse muy rápido.

Otra forma es crear una etiqueta **script** con la propiedad **type** establecido a **text/template** con un **id** de **story-template**. Para usar esta plantilla necesitamos hacer referencia a este script en la opción **template** de nuestro componente mediante un selector.

```
<script type="text/template" id="story-template">
  <h1> ¡Mi caballo es genial! </h1>
</script>

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



Info

"text/template" no es un script que el navegador puede entender, por lo que el navegador simplemente lo ignorará. Esto te permite poner cualquier cosa ahí, que luego pueden ser extraídas y generar fragmentos de HTML.

Mi manera favorita de definir una **plantilla** (y la que voy a usar en los ejemplos de este libro) es crear una etiqueta HTML **template** y darle un **id**. Entonces podemos hacer referencia a un selector como hicimos anteriormente. Utilizando esta técnica el componente anterior se verá así:

```
<template id="story-template">
  <h1> ¡Mi caballo es genial! </h1>
</template>

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

Propiedades

Veamos ahora como podemos usar múltiples instancias de nuestro componente **story** para mostrar una lista de historias. Tenemos que actualizar la plantilla **template** para no mostrar siempre la misma historia, sino la trama de la historia que queramos.

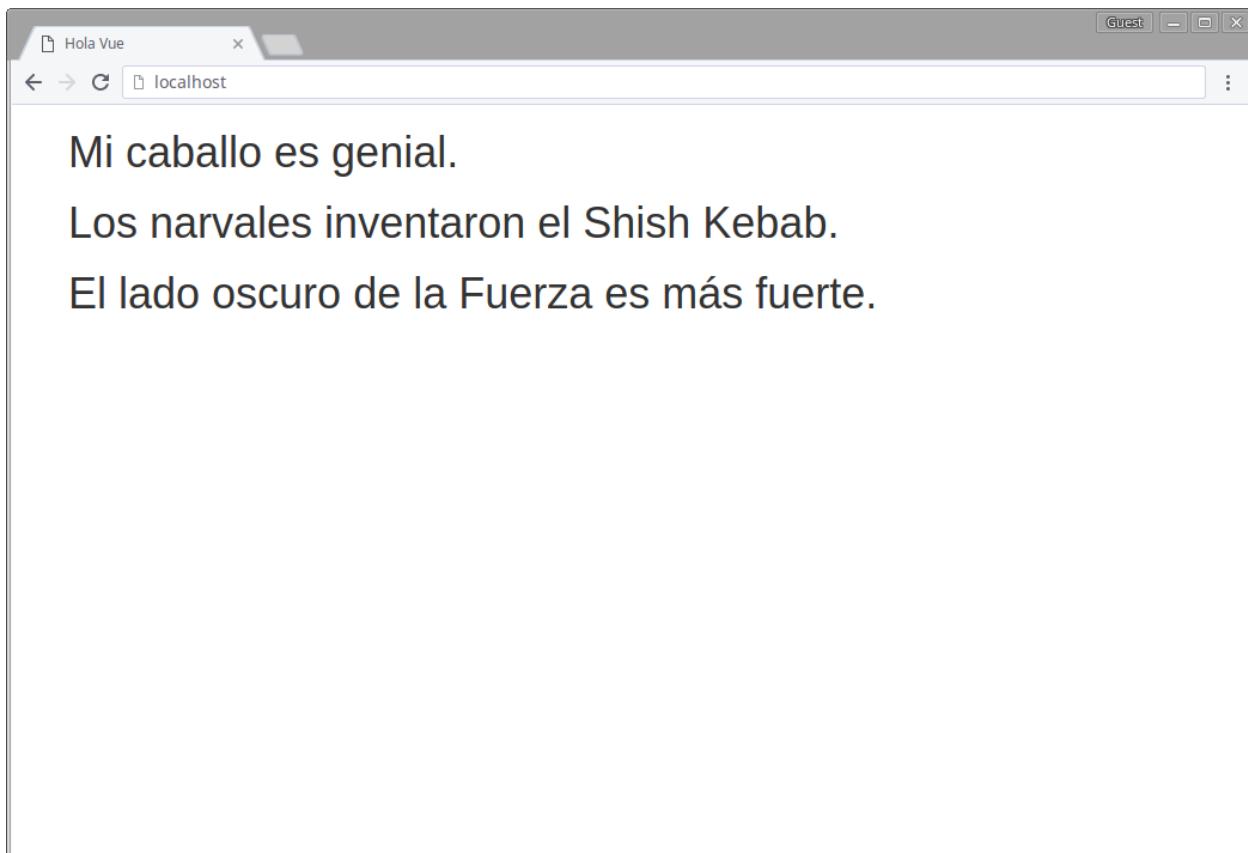
```
<template id="story-template">
  <h1>{{ plot }}</h1>
</template>
```

También tenemos que actualizar nuestro componente para usar esta propiedad. Para ello, añadiremos la nueva propiedad 'plot' al atributo `props` de nuestro componente.

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

Ahora podemos pasar la propiedad `plot` cada vez que usemos el elemento `<story>`.

```
<div class="container">
  <story plot="Mi caballo es genial."></story>
  <story plot="Los narvales inventaron el Shish Kebab."></story>
  <story plot="El lado oscuro de la Fuerza es más fuerte."></story>
</div>
```



Muestra diferentes 'historias'.



Advertencia

Los atributos HTML son insensibles a mayúsculas y minúsculas. Al usar nombres de propiedades CamelCase como atributos, necesitas usar su kebab-case equivalente (delimitado-por-guiones).

Así que, camelCase en JavaScript, kebab-case en HTML. Por ejemplo, para `props: ['isUser']`, el atributo HTML sería `<story is-user="true"></story>`.

Como probablemente has imaginado, un componente puede tener más de una propiedad. Por ejemplo, si queremos mostrar la propiedad writer junto con plot por cada historia tenemos que pasar `writer` también.

```
<story plot="Mi caballo es genial." writer="Mr. Weeb1"></story>
```

Si tienes muchas propiedades y tus elementos se están ensuciando puedes pasar un objeto y mostrar sus propiedades.

Vamos a terminar nuestro ejemplo refactorizando una vez más:

```
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>Historias Geniales</title>
6 </head>
7 <body>
8   <div class="container">
9     <story v-bind:story="{plot: 'Mi caballo es genial.', writer: 'Mr. Weeb1'}">
10    </story>
11    <story v-bind:story="{plot: 'Los narvales inventaron el Shish Kebab.', write\
12 r: 'Mr. Weeb1'}">
13    </story>
14    <story v-bind:story="{plot: 'El lado oscuro de la Fuerza es más fuerte.', wr\
15 iter: 'Darth Vader'}">
16    </story>
17  </div>
18  <template id="story-template">
19    <h1>{{ story.writer }} dijo "{{ story.plot }}"</h1>
20  </template>
21 </body>
22 <script src="https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js"></script>
23 <script type="text/javascript">
24 Vue.component('story', {
```

```
25     props: ['story'],
26     template: "#story-template"
27 });
28
29 new Vue({
30   el: '.container'
31 })
32 </script>
33 </html>
```



Info

`v-bind` es usado para enlazar dinámicamente uno o más atributos, o la propiedad de un componente, a una expresión.

Dado que la propiedad `story` no es una cadena de texto sino un objeto JavaScript en lugar de `story="..."` usamos `v-bind:story="..."` para enlazar la propiedad `story` con el objeto pasado.

La abreviatura para `v-bind` es `:`, así que de ahora en adelante vamos a usarlo así: `:story="..."`.

Reusabilidad

Echemos un vistazo nuevamente a nuestro ejemplo de [Resultados Filtrados](#). Supongamos que esta vez tomamos los datos de la variable `stories` de una API externa, a través de una llamada HTTP. Los desarrolladores de la API deciden renombrar la propiedad `plot` de `story` a `body`. Así que ahora tenemos que ir a través de nuestro código y hacer los cambios necesarios.



Info

Más adelante en este libro cubriremos como podemos usar `Vue` para realizar solicitudes web.

```
1 <div class="container">
2     <h1> ¡Escuchemos algunas historias! </h1>
3     <div>
4         <h3>Historias de Alex</h3>
5         <ul class="list-group">
6             <li v-for="story in storiesBy('Alex')"
7                 class="list-group-item">
8                 >
9                 {{ story.writer }} dijo "{{ story.plot }}"
10                {{ story.writer }} dijo "{{ story.body }}"
11            </li>
12        </ul>
13        <h3>Historias de John</h3>
14        <ul class="list-group">
15            <li v-for="story in storiesBy('John')"
16                class="list-group-item">
17                >
18                {{ story.writer }} dijo "{{ story.plot }}"
19                {{ story.writer }} dijo "{{ story.body }}"
20            </li>
21        </ul>
22        <div class="form-group">
23            <label for="query">
24                ¿Qué estás buscando?
25            </label>
26            <input v-model="query" class="form-control">
27        </div>
28        <h3>Resultados de la búsqueda:</h3>
29        <ul class="list-group">
30            <li v-for="story in search"
31                class="list-group-item">
32                >
33                {{ story.writer }} dijo "{{ story.plot }}"
34                {{ story.writer }} dijo "{{ story.body }}"
35            </li>
36        </ul>
37    </div>
38 </div>
```



Nota

En este ejemplo en particular el resultado de sintaxis está desactivado.

Como habrás notado, tuvimos que hacer exactamente el mismo cambio 3 veces y no sé tú, pero yo odio la repetición de código. Si no te parece nada del otro mundo, imagina que tuvieras que usar el mismo bloque de código en 100 lugares diferentes, ¿Qué harías entonces? Afortunadamente, *Vue* proporciona una solución para ese tipo de situaciones y esta solución tiene un nombre, **Componente**.

Tip

Cuando te encuentres repitiendo una pieza de funcionalidad, la manera mas eficiente de lidiar con ello es crear un componente dedicado.

Por suerte, hemos creado un Componente **story** en el ejemplo anterior, que muestra el escritor y el contenido para una historia específica. Podemos usar el elemento personalizado **<story>** dentro de nuestro **HTML** y pasar cada historia, como hicimos antes, con la etiqueta **:story**. Esta vez la usaremos dentro de una directiva **v-for**.

Así que nuestro código será:

```
<div class="container">
  <h1> iEscuchemos algunas historias!</h1>
  <div>
    <h3>Historias de Alex</h3>
    <ul class="list-group">
      <story v-for="story in storiesBy('Alex')"
        :story="story"></story>
    </ul>
    <h3>Historias de John</h3>
    <ul class="list-group">
      <story v-for="story in storiesBy('John')"
        :story="story"></story>
    </ul>
    <div class="form-group">
      <label for="query">¿Qué estás buscando?</label>
      <input v-model="query" class="form-control">
    </div>
    <h3>Resultados de la búsqueda:</h3>
    <ul class="list-group">
      <story v-for="story in search"
        :story="story"></story>
    </ul>
  </div>
</div>
```

Si tratas de ejecutar este código recibirás la siguiente advertencia:

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



Advertencia de Vue

Para arreglar esto necesitamos registrar el componente nuevamente. Esta vez tenemos que hacer algunos cambios a la plantilla del componente. Cambiaremos el atributo `plot` a `body` y la etiqueta `<h1>` a `` para satisfacer nuestras necesidades.

Así que la plantilla de la historia será:

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

El componente permanecerá igual.

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

Si ejecutas el código anterior, verás por ti mismo que todo funciona igual que antes, pero esta vez con el uso de un componente personalizado.

Muy genial ¿no?



Advertencia

Por favor se responsable. No bebas y conduzcas.

En conjunto

Usando nuestros recién adquiridos conocimientos deberíamos ser capaces de construir algo un poco más complejo. Basados en la estructura del ejemplo anterior, vamos a crear un sistema de votación para `stories`, y agregar una función de *marcar como favorito*. La forma de lograr esto es a través de métodos, directivas y por supuesto componentes.

Comencemos con la configuración de historias.

```
1 <html>
2 <head>
3 <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
4 <title>Hola Vue</title>
5 </head>
6 <body>
7 <div id="app">
8   <div class="container">
9     <h1>iEscuchemos algunas historias!</h1>
10    <ul class="list-group">
11      <story v-for="story in stories" :story="story"></story>
12    </ul>
13    <pre>{{ $data }}</pre>
14  </div>
15 </div>
16 <template id="story-template">
17   <li class="list-group-item">
18     {{ story.writer }} dijo "{{ story.plot }}"
19   </li>
20 </template>
21 </body>
22 <script src="https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js"></script>
23 <script type="text/javascript">
24   Vue.component('story', {
25     template: "#story-template",
26     props: ['story'],
27   });
28 );
29
30 new Vue({
31   el: '#app',
32   data: {
33     stories: [
34       {
35         plot: 'Mi caballo es genial.',
36         writer: 'Mr. Weebly',
37       },
38       {
39         plot: 'Los narvales inventaron el Shish Kebab.',
40         writer: 'Mr. Weebly',
41       },
42       {
43         plot: 'El lado oscuro de la Fuerza es más fuerte.',
```

```

44         writer: 'Darth Vader',
45     },
46     {
47         plot: 'Uno simplemente no camina hacia Mordor.',
48         writer: 'Boromir',
49     },
50 ]
51 }
52 })
53 </script>
54 </html>
```

El siguiente paso permite al usuario darle un voto a la historia que prefiera. Para aplicar este límite (1 voto por cada historia) mostraremos el botón ‘Votar’ sólo si el usuario no ha votado. Así que cada historia debe tener una propiedad `voted` que se convierte en verdadera cuando la función `upvote` es ejecutada.

```

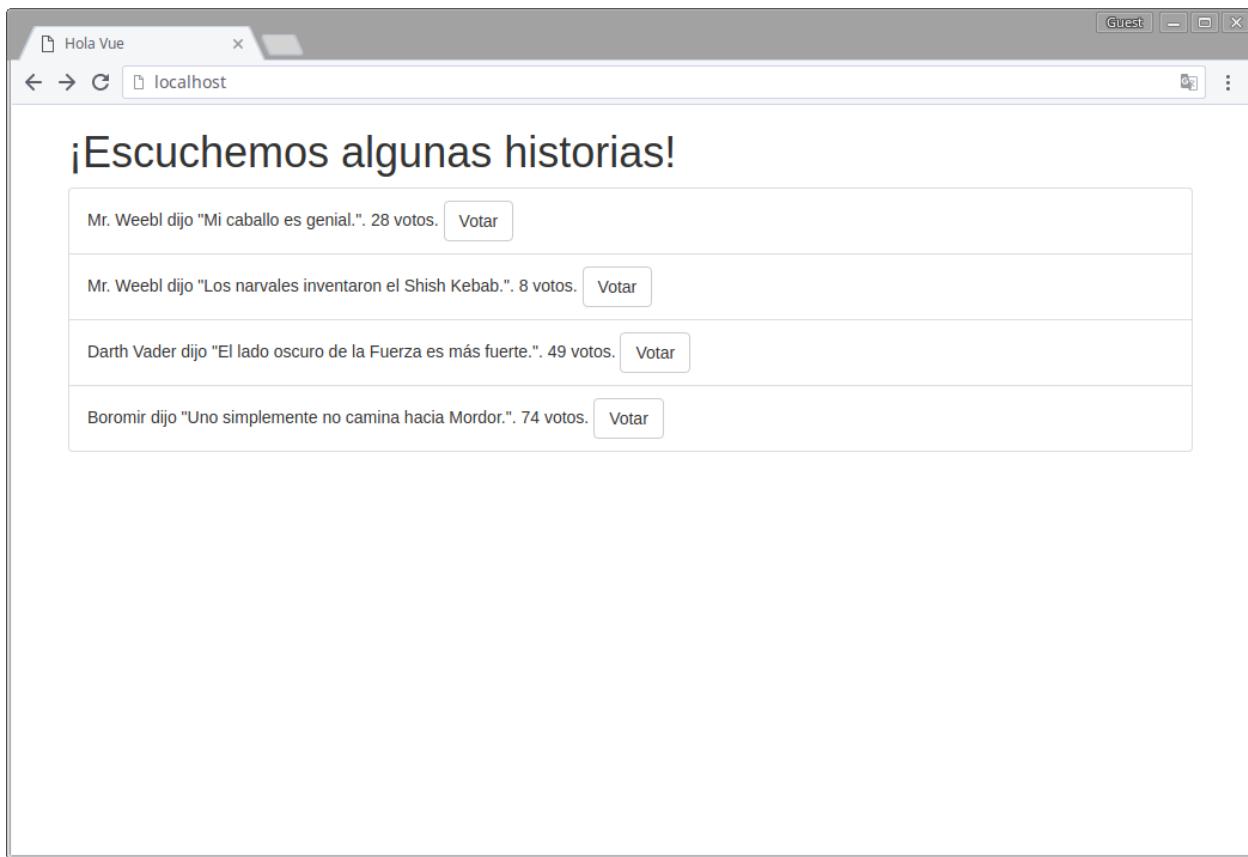
<template id="story-template">
  <li class="list-group-item">
    {{ story.writer }} dijo "{{ story.plot }}".
    {{ story.upvotes }} votos.
    <button v-show="!story.voted" @click="upvote"
      class="btn btn-default">
      Votar
    </button>
  </li>
</template>
```

```

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: 'Mi caballo es genial.',
      writer: 'Mr. Weebl',
      upvotes: 28,
      voted: false,
    },
    {
      plot: 'Los narvales inventaron el Shish Kebab.',
      writer: 'Mr. Weebl',
      upvotes: 8,
      voted: false,
    },
    {
      plot: 'El lado oscuro de la Fuerza es más fuerte.',
      writer: 'Darth Vader',
      upvotes: 49,
      voted: false,
    },
    {
      plot: 'Uno simplemente no camina hacia Mordor.',
      writer: 'Boromir',
      upvotes: 74,
      voted: false,
    },
  ],
}
})
```



Listo para votar!

Hemos implementado, con el uso de métodos, el sistema de votación. Creo que se ve bien, así que podemos continuar con la parte 'favorita de la historia'. Queremos que el usuario sea capaz de elegir sólamente una historia para que sea su favorita. Lo primera cosa que me viene a la cabeza es agregar un nuevo objeto vacío (`favorite`) y en cualquier momento que el usuario elija una historia para que sea su favorita, actualice la variable `favorite`. De esta forma podremos comprobar si una historia es igual a la historia favorita del usuario. Hagamos esto.

```
<template id="story-template">
  <li class="list-group-item">
    {{ story.writer }} dijo "{{ story.plot }}".
    {{ story.upvotes }} votos.
    <button v-show="!story.voted" @click="upvote"
      class="btn btn-default">
      Votar
    </button>
    <button v-show="!isFavorite" @click="setFavorite"
      class="btn btn-primary">
      Favorita
    </button>
```

```
<span v-show="isFavorite"
      class="glyphicon glyphicon-star pull-right" aria-hidden="true">
</span>
</li>
</template>

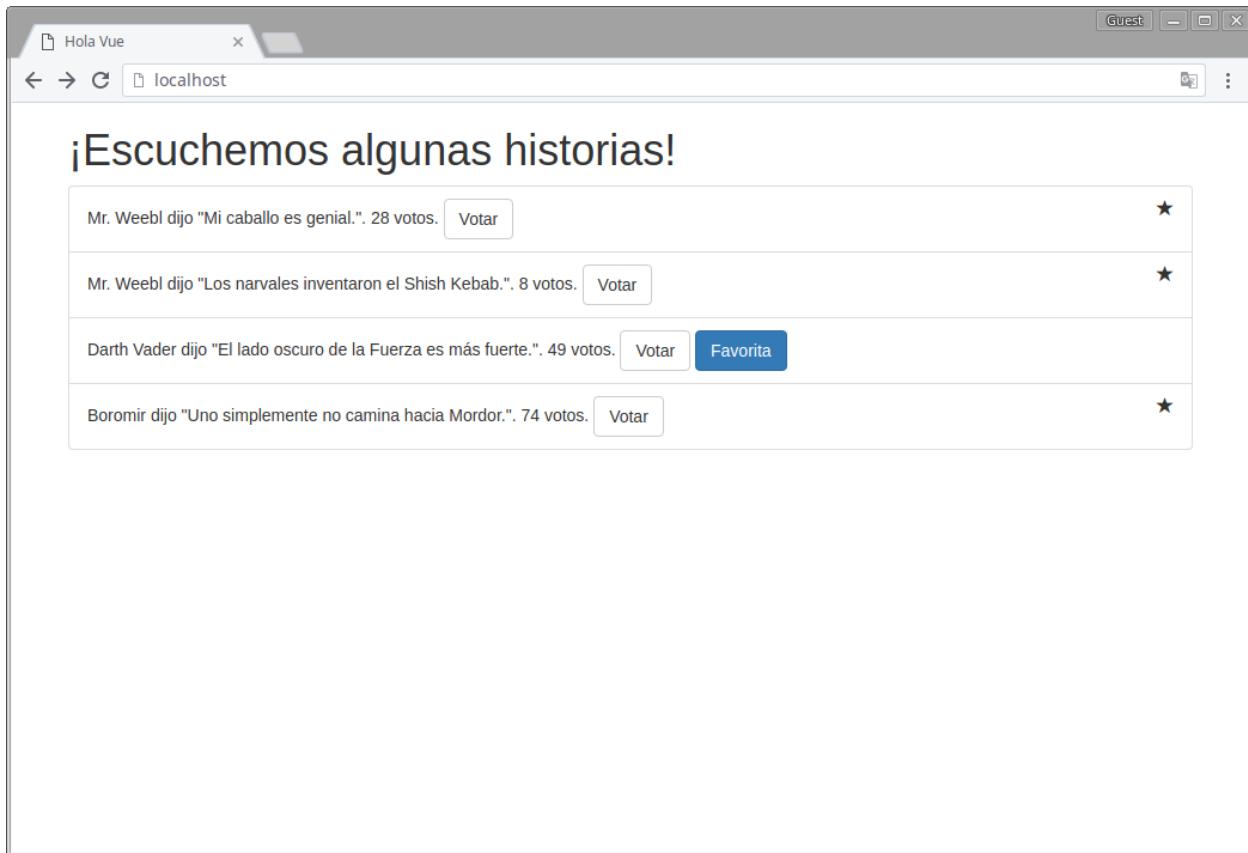
Vue.component('story', {
  template: "#story-template",
  props: ['story'],
  methods: {
    upvote: function(){
      this.story.upvotes += 1;
      this.story.voted = true;
    },
    setFavorite: function(){
      this.favorite = this.story;
    },
  },
  computed: {
    isFavorite: function(){
 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

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

Si tratas de ejecutar el código de arriba te darás cuenta de que no funciona como debería. Cuando intentas marcar una historia como favorita la variable `favorite` dentro de `$data` permanece en null. Parece que nuestro componente `story` es incapaz de actualizar el objeto `favorite`, así que vamos a pasarlo en cada historia y agregar `favorite` a las propiedades del componente.

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

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



Mal funcionamiento del método `setFavorite`

Hmmm, `favorite` todavía no se actualiza cuando `setFavorite` es ejecutado. El botón desaparece como se esperaba y un ícono de estrella aparece, pero la variable `favorite` es todavía null. Esto trae como consecuencia no deseable que el usuario pueda agregar a favoritos todas las historias.

El problema con este enfoque es que no mantenemos las cosas *sincronizadas*. Por defecto, todas

las propiedades forman un enlace de un solo sentido entre la propiedad hija y el padre. Cuando la propiedad padre se actualiza fluirá hasta el hijo pero no al revés.

No podemos sincronizar los datos del hijo con los del padre con lo que sabemos hasta ahora. Así que tomaremos un descanso y estudiaremos los **Eventos Personalizados** de Vue, antes de ir más lejos.



Video

El uso de componentes en Vue.js es cubierto a fondo a partir de la lección 17 del curso de [Vue 2 en Styde](#)⁵⁰.



Código de Ejemplo

Puedes encontrar el código de ejemplo de este capítulo en [GitHub](#)⁵¹.

⁵⁰<https://styde.net/creacion-y-uso-de-tu-primer-componente-con-vue-js-2/>

⁵¹<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter7>

Ejercicio

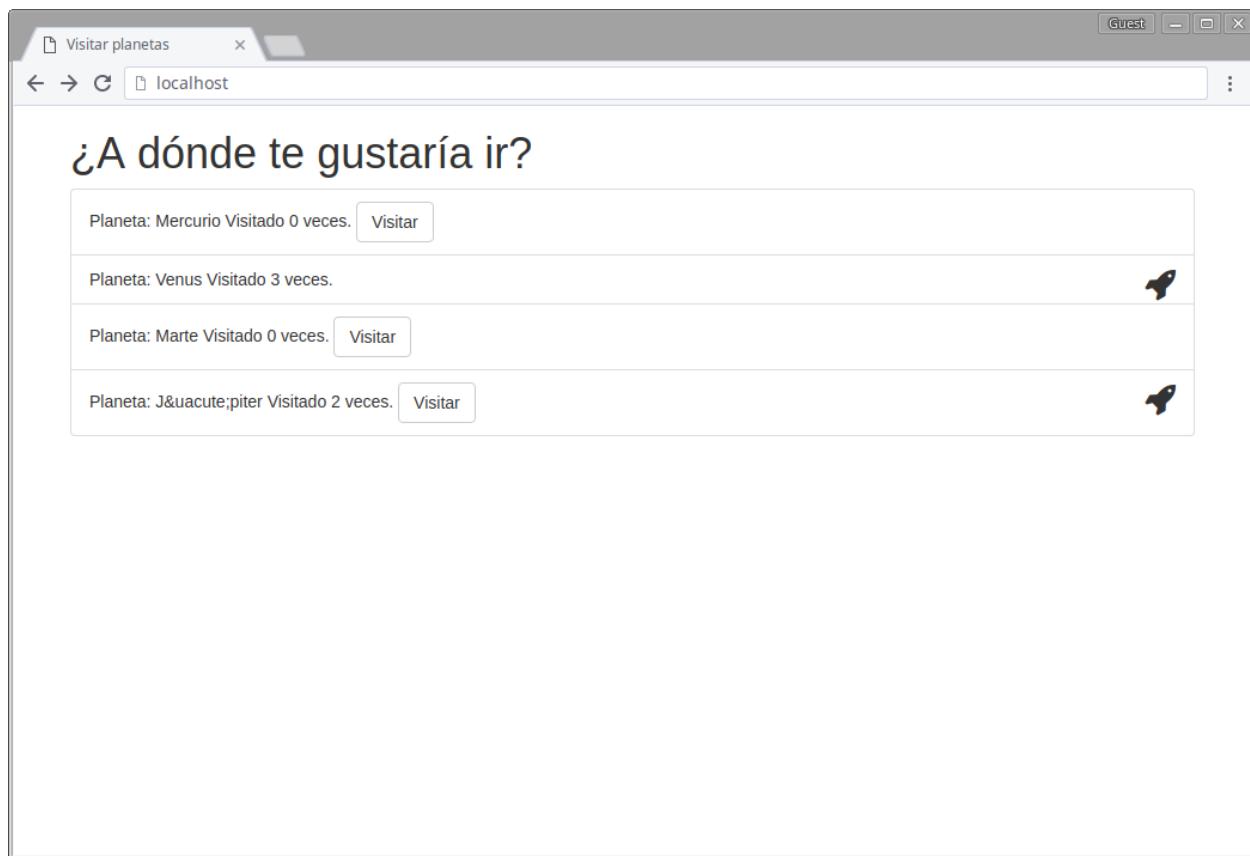
Crea un arreglo de planetas. Cada planeta debe tener un nombre y un número de visitas.

Puedes elegir viajar a cualquier planeta, pero estás limitado a 3 visitas por planeta debido a escasez de combustible.

Deberías tener un componente *Planeta* con los métodos y propiedades computadas apropiados.

Al renderizarse, cada planeta debería mostrar:

- Su nombre.
- El número de visitas.
- Un botón *Visitar* (si el número máximo de visitas no se ha alcanzado).
- Un ícono para indicar si el planeta ha sido visitado al menos una vez.



Pantalla de Ejemplo



Possible Solución

Puedes encontrar una posible solución para este ejercicio [aquí⁵²](#).

⁵²<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter7.html>

Eventos Personalizados

Algunas veces es necesario despachar un evento personalizado. Para hacerlo podemos usar métodos de la instancia de Vue. Cada instancia de Vue implementa la [interfaz de Eventos⁵³](#).

Esto significa que puede:

- Escuchar a un evento usando `$on(event)`.
- Despachar un evento usando `$emit(event)`.

También puede:

- Escuchar a un evento, pero sólo una vez, usando `$once(event)`.
- Eliminar *Listeners* de eventos usando `$off()`.

Emitir y Escuchar

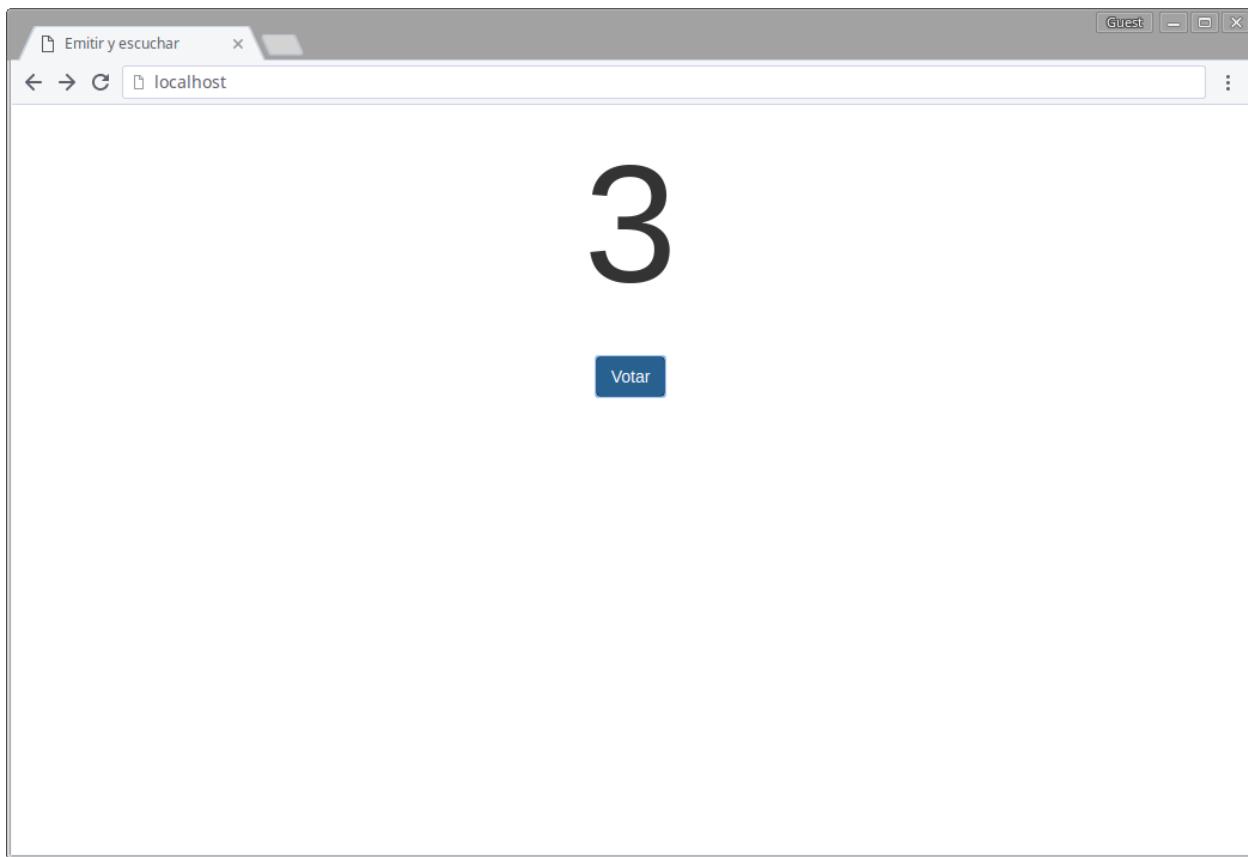
Comencemos con un ejemplo muy sencillo.

El siguiente bloque de código representa una página con un contador y un botón *Votar*. Cuando el botón es presionado, emite un evento, llamado ‘voted’. También hay un *listener* para el evento, que incrementa el número de votos cuando el evento es despachado.

```
1 <html>
2   <head>
3     <title>Emitir y escuchar</title>
4     <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" \
5       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">Votar</button>
13  </div>
14 </body>
```

⁵³<http://vuejs.org/api/#Instance-Methods-Events>

```
15 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/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>
```



Pantalla de Ejemplo

Registraremos el listener del evento dentro del *Lifecycle Hook* `created`. `this` está enlazado a la instancia de Vue dentro del método `vote` y el hook `created`. Así que podemos acceder a las funciones `$on` y `$emit` usando `this.$on` y `this.$emit`.

Lifecycle Hooks

Los lifecycle hooks son funciones que se ejecutan cuando suceden eventos relacionados a Vue.

En Vue 2 estos hooks son:

Hook	Se ejecuta
<code>beforeCreate</code>	Después de la inicialización de la instancia y antes de la observación de datos y la configuración de evento/observador.
<code>created</code>	Después de que la instancia es creada.
<code>beforeMount</code>	Justo antes de comenzar el montaje.
<code>mounted</code>	Después de que la instancia ha sido montada al DOM.
<code>beforeUpdate</code>	Cuando los datos cambian, antes de que el DOM virtual se vuelva a renderizar y parchear.
<code>updated</code>	Después de que un cambio en los datos causen que el DOM virtual se vuelva a renderizar y parchear.

Hook	Se ejecuta
<code>activated</code>	Cuando un componente kept-alive es activado.
<code>deactivated</code>	Cuando un componente kept-alive es desactivado.
<code>beforeDestroy</code>	Justo antes de que una instancia de Vue sea destruida.
<code>destroyed</code>	Después de que una instancia de Vue ha sido destruida.

No tienes que saber todo esto pero es bueno estar conscientes de su existencia. Si quieres aprender más sobre Lifecycle Hooks revisa la [API de Vue](#)⁵⁴.

Comunicación Padre-Hijo

Las cosas se están volviendo un poco diferentes cuando un componente padre necesita escuchar un evento de un componente *hijo*. No podemos usar `this.$on`/`this.$emit` dado que `this` estará enlazado a instancias diferentes.

¿Recuerdas al [listener de eventos `v-on (@)`]? (#v-on) Un componente padre puede escuchar los eventos emitidos desde un componente hijo usando `v-on` directamente en la plantilla, donde el componente hijo es usado.

Siguiendo el ejemplo anterior crearé un componente food que tendrá una propiedad *name*. En su plantilla, mostrará un botón con el valor de *name*. Cuando el botón es presionado queremos emitir el evento *vote*.

Componente Food

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

Plantilla del componente Food

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

En la instancia padre `<button>` será reemplazado por `<food @voted="countVote"></food>`.

`@voted="countVote"` significa que cuando el evento `voted` del hijo es emitido, el método `countVote` será ejecutado. También podemos deshacernos del listener `this.$on` dado que ya no lo necesitamos.

⁵⁴<http://vuejs.org/api/#Options-Lifecycle-Hooks>

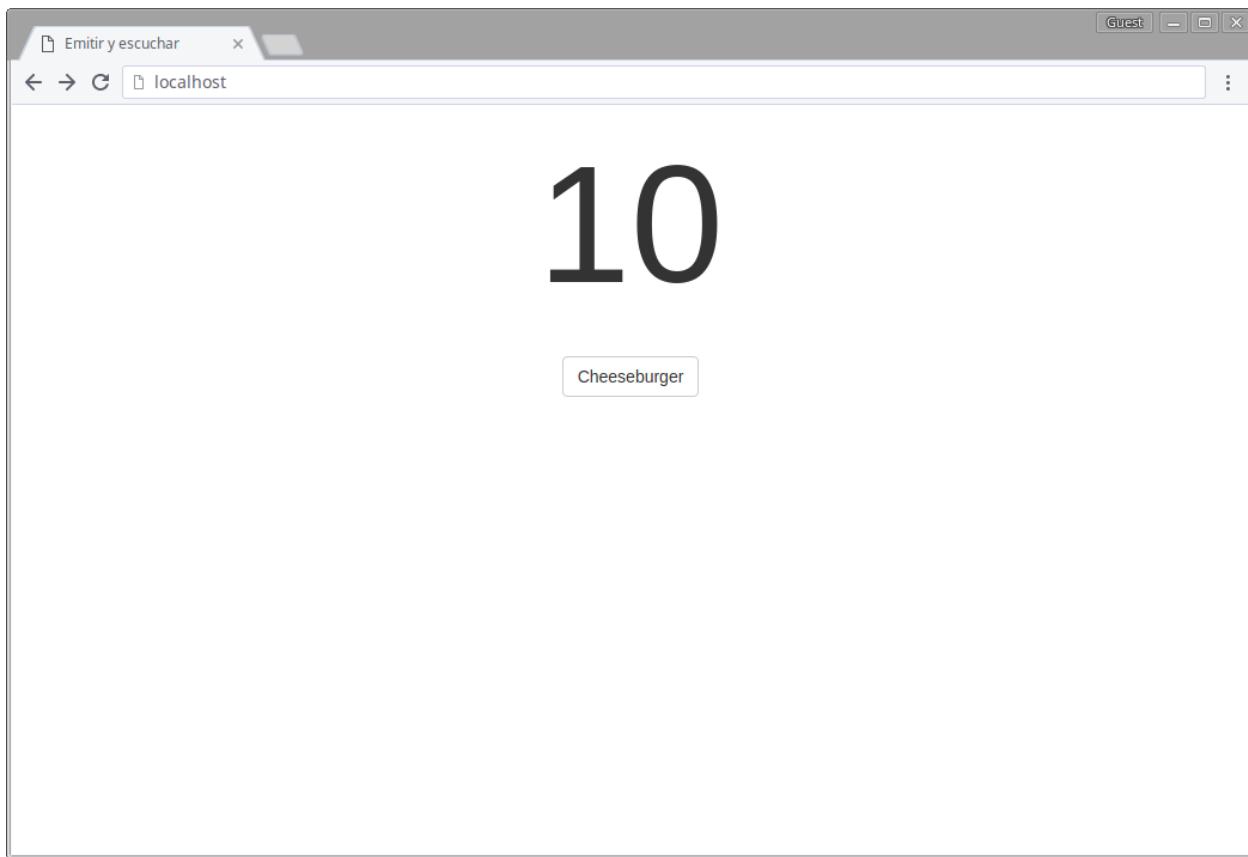
Componente padre

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

Plantilla del componente padre

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

Si ejecutas esto en tu navegador, verás que el resultado es el mismo.



Comunicación Padre-Hijo

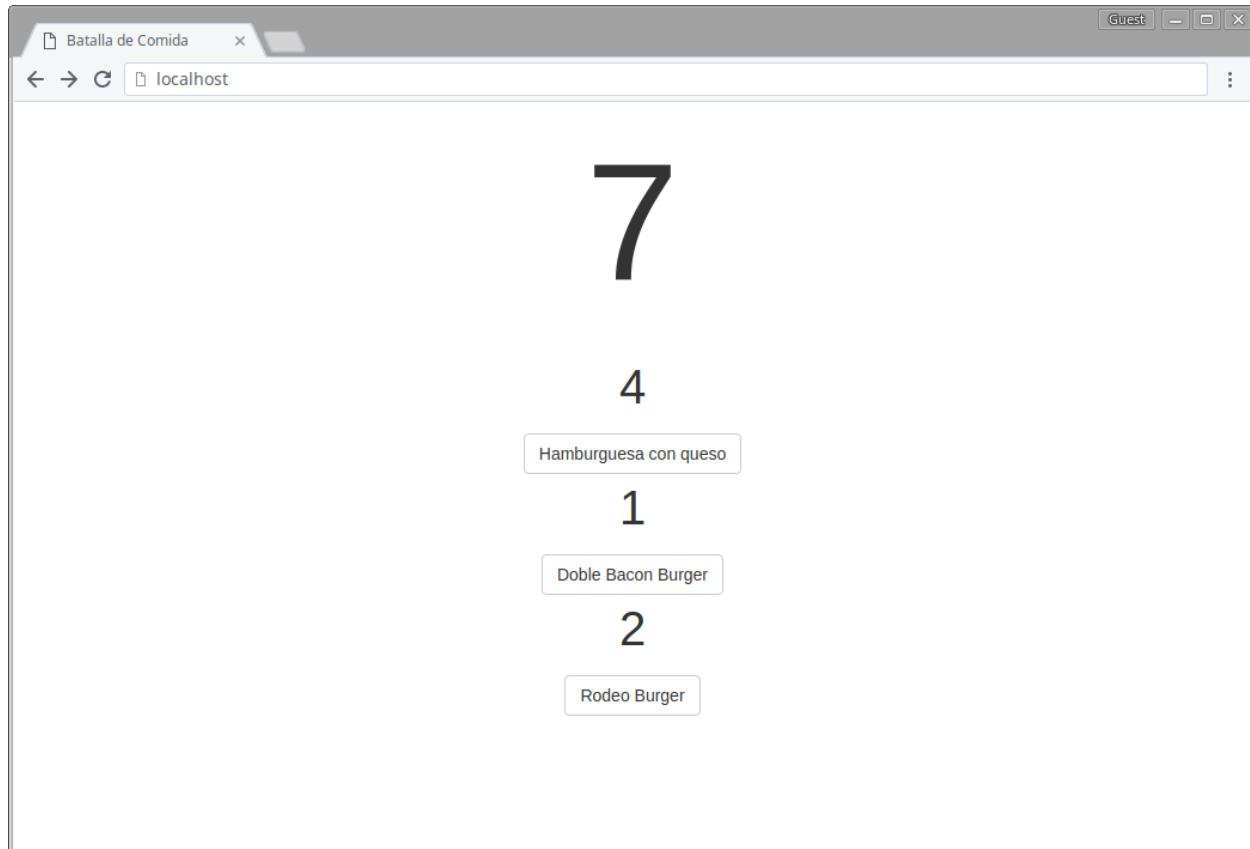
Pasando Argumentos

Vamos a crear 3 instancias del *componente food*. Cada instancia tendrá su propio número de votos. Cuando cualquiera de las comidas es votada, incrementará sus propios votos y emitirá un evento para actualizar el total de votos, también localizado en el componente padre.

```
1 <html>
2 <head>
3   <title>Batalla de Comida</title>
4   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" \
5     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="Hamburguesa con queso"></food>
15         <food @voted="countVote" name="Doble 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/2.0.1/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>
```



Múltiples instancias de componente

Nada nuevo hasta el momento. Para que la aplicación sea más sofisticada podemos añadir un *Registro de Votos*. El registro se actualizará cada vez que una comida es votada. Tenemos que actualizar el componente hijo, para pasar el nombre de la comida cuando emitimos el evento `voted`.

i Info

La función `$emit` junto con el nombre del evento como argumento pasa cualquier argumento adicional a la función de retorno del evento. Por ejemplo: `vm.$emit('voted', 'Alex', 'Sunday', 'Bob Ross')`

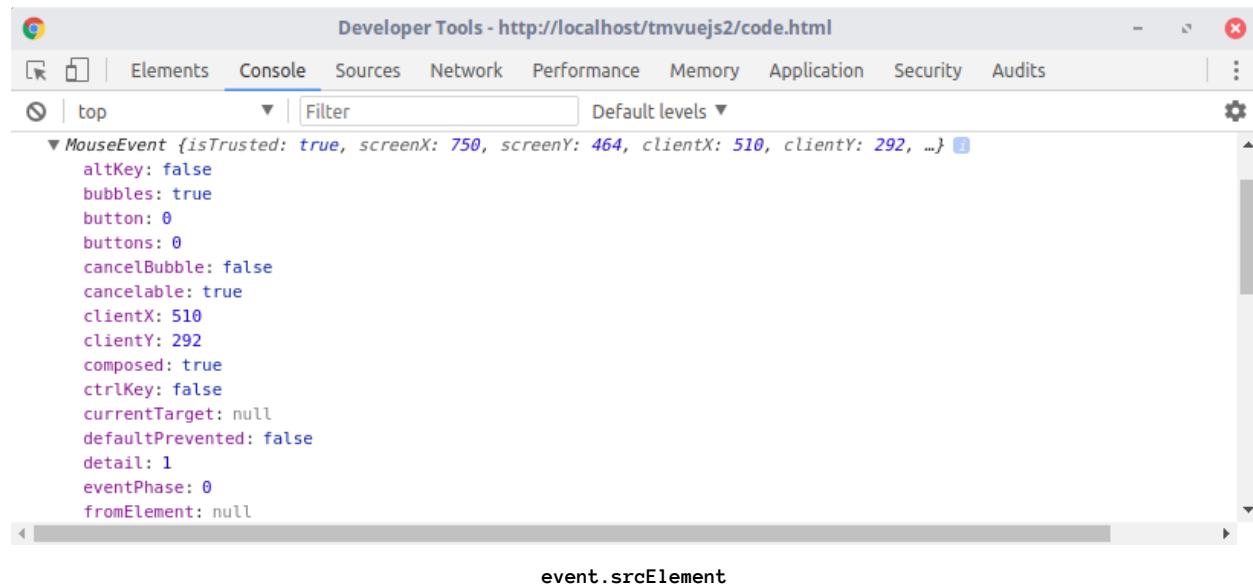
Tenemos dos opciones para acceder al nombre de la comida. Una es obviamente desde la propiedad

name del componente. La segunda es acceder al elemento que despachó el evento y encontrar su contenido de texto. Iremos con el segundo.

Dentro del método **vote** de **Food** podemos registrar la variable **event** en la consola para averiguar como podemos acceder al elemento presionado.

Componente Food

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



Si estás siguiendo los ejemplos verás que tenemos acceso al elemento presionado dentro del atributo **event.srcElement**. El nombre puede ser encontrado tanto en **event.srcElement.outerText** y **event.srcElement.textContent**.

Así que pasemos uno de esos a la función **\$emit**.

Componente Food

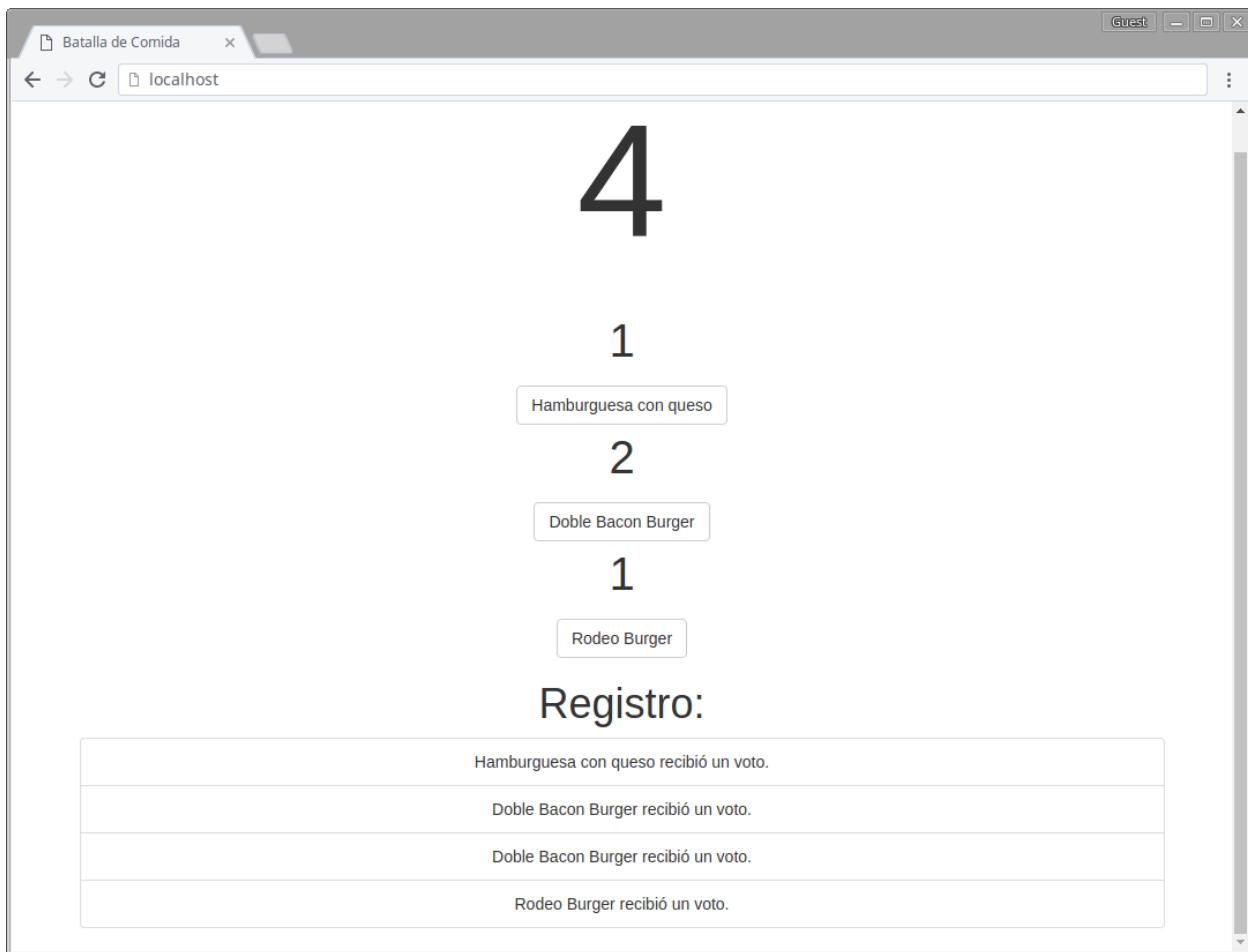
```
Vue.component('food', {
  ...
  methods: {
    vote: function () {
      this.votes++
      this.$emit('voted', event.srcElement.textContent)
    }
  }
})
```

Dentro del componente padre, añadiremos el voto entrante a un arreglo llamado log.

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

Para mostrar el registro, podemos añadir una lista a la plantilla HTML.

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



My fácil, ¿no?

Comunicación Entre Todos los Componentes

Llevaremos el ejemplo anterior un paso adelante, añadiendo un botón de resetear. Cuando el botón es presionado, restablecerá todos los contadores de votos. Como puedes imaginar, el botón de resetear emitirá un evento que debería ser manejado por todos los componentes, pero ¿Cómo podemos capturar este evento dentro de los componentes hijos?.

Si usamos `<food @voted="countVote">`, podemos escuchar al evento `voted`, pero no tenemos una forma de emitir eventos a los componentes hijos.

Para hacer que todos los componentes puedan comunicarse entre sí, usaremos una **instancia de Vue vacía** como un bus central de eventos. Entonces dentro del hook `created` de los componentes, registraremos los listeners del evento usando `bus.$on` en lugar de `this.$on`. En consecuencia, usaremos `bus.$emit` para despachar todos los eventos.

HTML

```
1 <body>
2   <div class="container text-center">
3     <h1>Batalla de Comida</h1>
4     <p style="font-size: 140px;">
5       {{ votes.count }}
6     </p>
7     <button class="btn btn-danger" @click="reset">Resetear votos</button>
8     <hr>
9
10    <div class="row">
11      <food name="Hamburguesa de queso"></food>
12      <food name="Doble Bacon Burger"></food>
13      <food name="Whooper"></food>
14    </div>
15    <hr>
16
17    <h1>Registro:</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       // En lugar de usar this.name
14       // podemos acceder al texto del elemento
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 })
```



Comunicación entre todos los componentes



Advertencia

Nota que estamos usando:

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

Si en su lugar estuviéramos tratando de usar:

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

habría ocurrido un error, ya que `this` estaría limitada a la *instancia bus* en lugar de a la instancia del componente actual.

Eliminando listeners de Eventos

Para eliminar uno o más listeners de eventos podemos usar `$off`. El método `$off([event, callback])` puede ser usado de diferentes formas.

1. `$off()` sin argumentos elimina todos los listeners de eventos.
2. `$off([event])` elimina todos los listeners de eventos para el evento especificado.
3. `$off([event, callback])` elimina el listener del evento para el callback especificado.

Para verlo en acción añadiremos un botón *Detener* para impedir que los votos sean contados o registrados. Colocaremos un método `stop` en la instancia de Vue.

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

Si vamos con `bus.$off(['voted'])`, verás que después de presionar el botón *Detener*, los próximos votos no son añadidos a la cuenta total. Tampoco aparecen en el registro. Aunque si pulsas el botón *Resetear Votos*, los votos del componente Food son restablecidos a 0.

Para inhabilitar también el *listener de reset*, podemos eliminar todos los listeners de eventos utilizando `bus.$off()`.

De Vuelta a Historias

¿Recuerdas el [Ejemplo historias](#) de los capítulos anteriores? Estabamos a punto de sincronizar *los datos del componente* con los *datos del padre*. La solución parece obvia ahora.

Usaremos el componente *story* así:

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

Dentro del componente Story, emitiremos el evento `update` cuando una historia es marcada como favorita. La historia siendo marcada como favorita, debería ser pasada como un argumento en emit.

Componente story

```
Vue.component('story', {
    ...
    methods: {
        ...
        updateFavorite: function(){
            // 'update' es solo el nombre del evento personalizado
            // puede ser cualquier cosa. Por ejemplo: fav-update
            this.$emit('update', this.story)
        }
    }
    ...
});
```

En la instancia padre agregaremos una variable `favorite` en data. También, crearemos un nuevo método, que cuando es llamado actualizará la variable `favorite`.

Instancia padre

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

```
{
  "stories": [
    {
      "plot": "Mi caballo es genial.",
      "writer": "Mr. Weeb1",
      "upvotes": 28,
      "voted": false
    },
    {
      "plot": "Los narvalos inventaron el Shish Kebab.",
      "writer": "Mr. Weeb1",
      "upvotes": 8,
      "voted": false
    },
    {
      "plot": "El lado oscuro de la Fuerza es más fuerte.",
      "writer": "Darth Vader",
      "upvotes": 49,
      "voted": false
    },
    {
      "plot": "Uno simplemente no camina hacia Mordor.",
      "writer": "Boromir",
      "upvotes": 74,
      "voted": false
    }
  ]
}
```

Establecer como favorita sólo una historia

Ahora el resultado deseado es conseguido y el usuario puede elegir solo una historia como su favorita, mientras que puede votar por tantas historias como quiera.



Info

En Vue 2 los enlaces son **siempre** en sentido único. Para mantener los datos sincronizados entre padre e hijo tienes que usar *eventos*.



Video

El manejo de eventos es cubierto en la lección 9 del curso de Vue 2 en Styde⁵⁵.



Código de Ejemplo

Puedes encontrar el código de ejemplo de este capítulo en [GitHub](#)⁵⁶.

⁵⁵<https://styde.net/manejo-de-eventos-en-vue-js-2/>

⁵⁶<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter8>

Ejercicio

Este es el ejercicio más difícil hasta el momento, así que asegurate de poner en uso todo lo que has aprendido en este libro. Crea un arreglo de 4 carruajes a caballo. Cada carruaje tiene un “nombre” y un número de “caballos” (de 1 a 4). Crea un componente llamado “carruaje”. El componente “carruaje” debe mostrar el nombre del carruaje y el número de caballos que tiene. También debe tener un botón de acción. El texto del botón depende del carruaje seleccionado actualmente.

Más específicamente, el texto del botón debe ser:

- ‘Escoger Carruaje’, antes de que el usuario haya elegido cualquier carruaje.
- ‘Descartar Caballos’, cuando el carruaje tiene menos caballos que el carruaje seleccionado.
- ‘Contratar Caballos’, cuando el carruaje tiene más caballos que el carruaje seleccionado.
- ‘¡Montando!’, cuando el carruaje es el seleccionado (este botón tiene que estar deshabilitado).

El usuario debe ser capaz de elegir un carruaje y luego escoger entre cualquier carruaje que quiera.

Escenario de Ejemplo: El usuario ha elegido un carruaje con 2 caballos y su botón dice ‘¡Montando!’. Un carruaje con 3 caballos tiene un caballo de más, así que su botón dice ‘Contratar Caballos’. Un carruaje con 1 caballo tiene un caballo menos que el carruaje del usuario, así que su botón dice ‘Descartar Caballos’. Creo que tienes la idea.



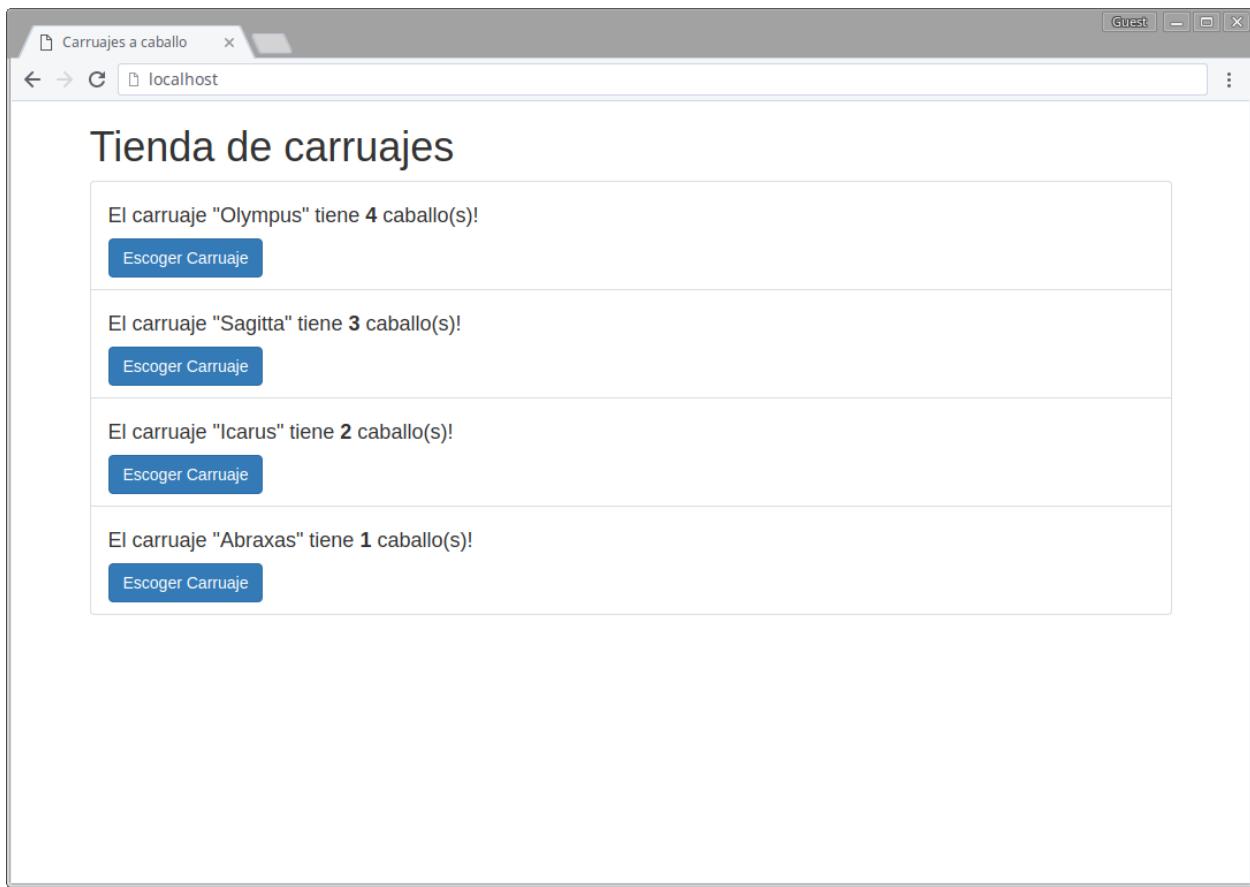
Pista

Tienes que mantener sincronizadas las propiedades `currentChariot` del padre y el hijo.



Pista

Para deshabilitar un botón usa el atributo `disabled="true"`. Tienes que descubrir como aplicarlo condicionalmente.



Pantalla de Ejemplo



Possible Solución

Puedes encontrar una posible solución para este ejercicio [aquí](#)⁵⁷.

⁵⁷<https://github.com/hoottlex/the-majesty-of-vuejs-2/blob/master/homework/chapter8.html>

Enlaces de Clase y Estilo

Enlace de Clases

Sintaxis de Objeto

Una necesidad común para el enlace de datos es la de manipular la clase de un elemento y sus estilos. Para tales casos, puedes usar `v-bind:class`. Esto puede ser usado para aplicar clases condicionalmente, alternarlas y/o aplicar muchas de ellas usando un objeto enlazado y otros.

La directiva `v-bind:class` recibe como argumento un objeto con el siguiente formato:

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

y aplica al elemento todas las clases con el valor `true`. Por ejemplo, las clases del siguiente elemento serán `classA` y `classC`.

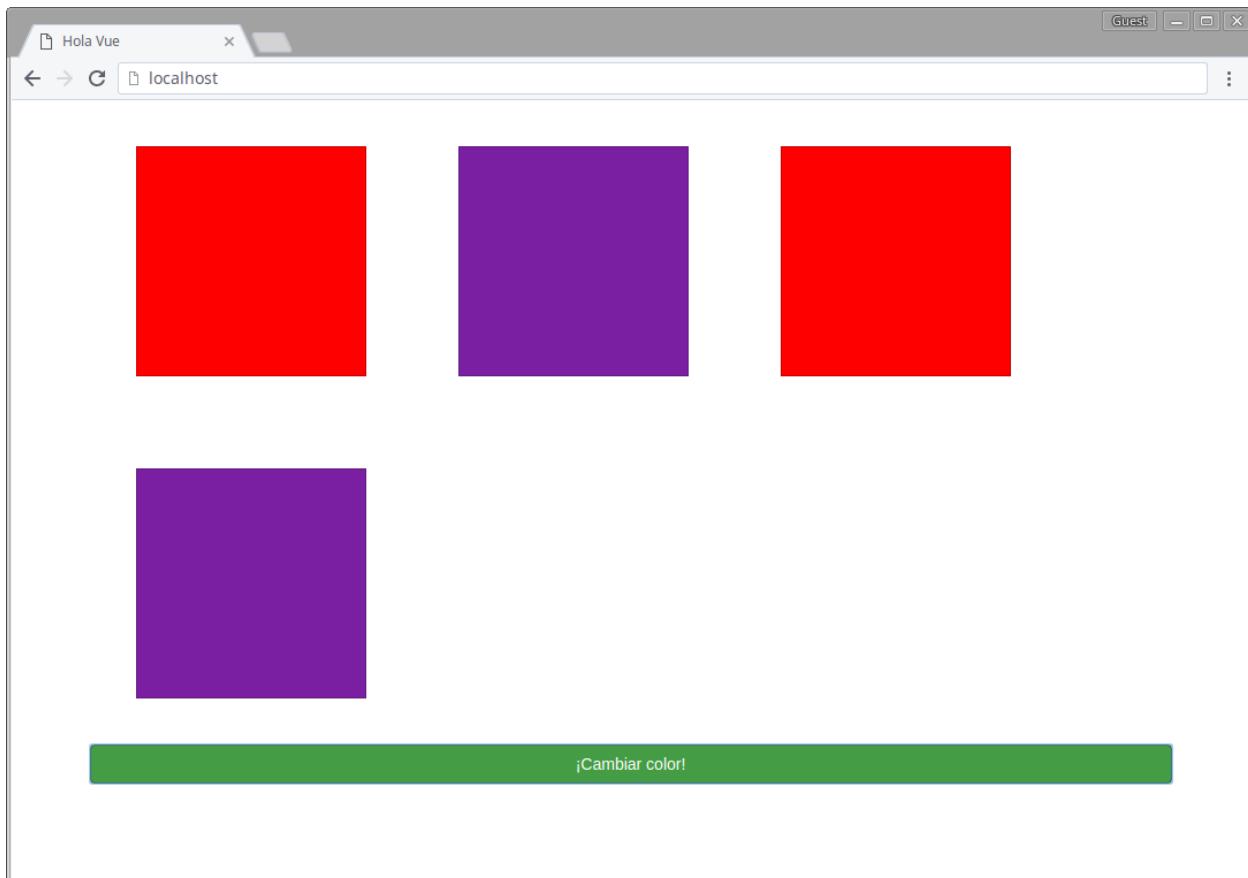
```
<div v-bind:class="elClasses"></div>
```

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

Para demostrar como `v-bind` es usado con atributos de clases, realizaremos un ejemplo de activación de clases. Usando la directiva `v-bind:class` vamos a alternar dinámicamente las clases de los elementos `div`.

```
1 <html>
2 <head>
3   <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" \
4   rel="stylesheet">
5   <title>Hola 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      Cambiar color!
15    </button>
16  </div>
17 </body>
18 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/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>
```



Alternar el color de las cajas



Alternar el color de las cajas

Hemos aplicado una clase `box` a cada `div`, para nuestra conveniencia. Lo que realmente hace este código es “ voltear” el color de las cajas al presionar el botón. Al presionarlo, este invoca la función `flipColor` que revierte el valor de `color` originalmente establecido en `true`. Luego `v-bind:class` cambiará condicionalmente el nombre de la clase de ‘red’ a ‘blue’ o de ‘purple’ a ‘green’ dependiendo de la veracidad del valor de `color`. Dado eso, el estilo se aplicará en cada clase y nos dará el resultado deseado.



Info

La directiva `v-bind:class` puede coexistir con el atributo `class`.

Así que, en nuestro ejemplo, los `divs` siempre tienen la clase `box` y condicionalmente `red`, `blue`, `purple` o `green`.

Sintaxis de Arreglo

También podemos aplicar una lista de clases a un elemento, usando un arreglo de nombres de clases.

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

Aplicar condicionalmente una clase, también puede ser logrado con el uso de un **if** en línea dentro del arreglo.

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



Info

El **if** en línea se conoce comúnmente como **operador ternario**, **operador condicional o if ternario**.

El operador condicional (ternario) es el único operador de JavaScript que toma tres operandos.

La sintaxis del **operador ternario** es **condición ? expresión1 : expresión2**. Si la condición es verdadera, el operador devuelve el valor de expresión1, de otra manera devuelve el valor de expresión2.

Usando el **if** en línea, el ejemplo de cambio de colores se verá así:

```
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     Cambiar 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

Para usar realmente un nombre de clase en lugar de una variable dentro de un arreglo de clases, usa comillas simples. `v-bind:class="[variable, 'classname']"`

Enlaces de Estilos

Sintaxis de Objeto

La sintaxis de objeto para `v-bind:style` es muy sencilla; parece casi CSS, excepto que es un objeto JavaScript. Usaremos la abreviatura que Vue.js proporciona para la directiva usada anteriormente, `v-bind(:)`.

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

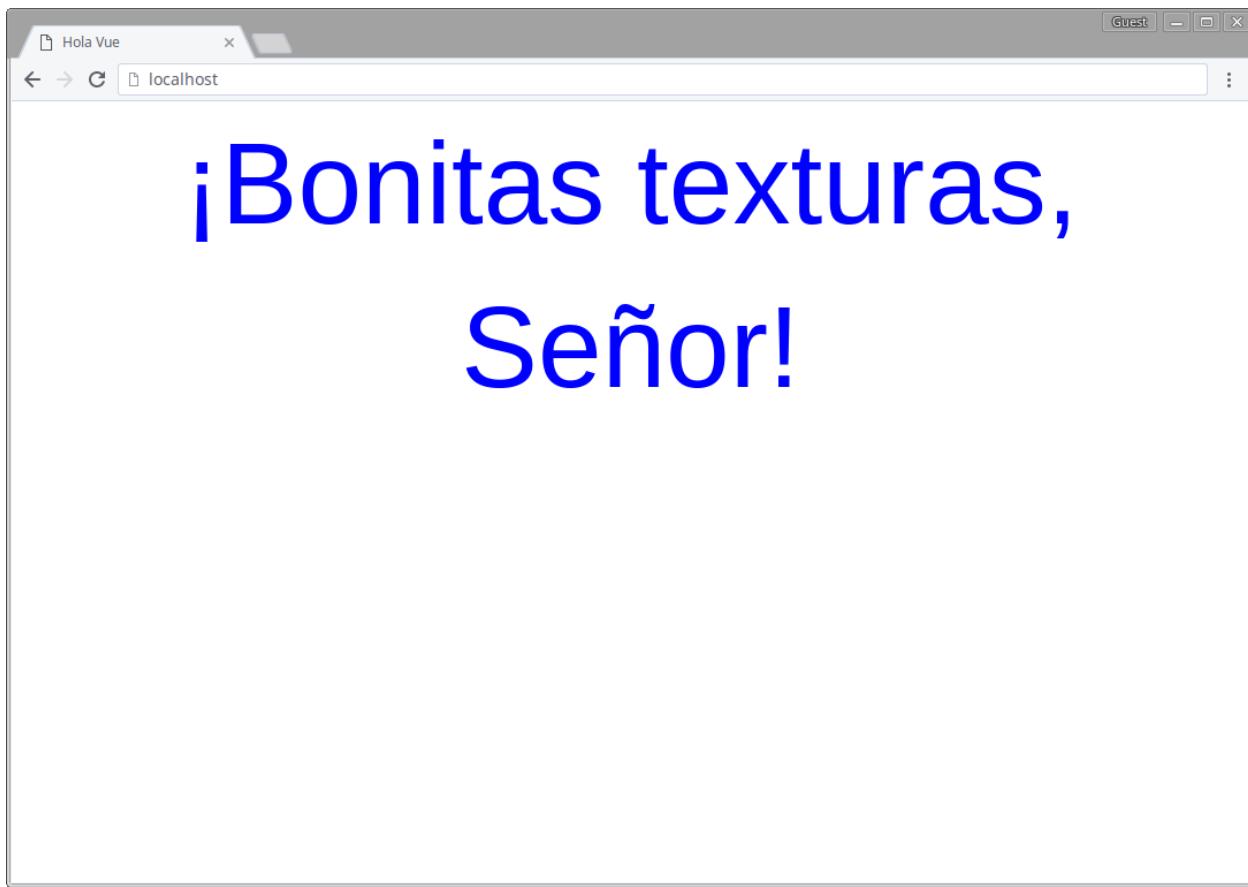
```
1 data: {
2     niceStyle:
3     {
4         color: 'blue',
5         fontSize: '20px'
6     }
7 }
```

También podemos declarar las propiedades de estilos dentro de un objeto `:style="..."` en línea.

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

Incluso podemos **hacer referencia a variables** dentro del objeto style:

```
<!-- La variable 'niceStyle' es la misma que usamos en el ejemplo anterior -->
<div :style="{ 'color': niceStyle.color, fontSize: niceStyle.fontSize }">
</div>
```



Enlace de objetos de estilos

A menudo es una buena idea usar un objeto de estilos y enlazarlo para que la plantilla sea más limpia.

Sintaxis de Arreglo

Usando la sintaxis de arreglo en línea para `v-bind:style` somos capaces de aplicar multiples objetos de estilos al mismo elemento, lo que significa que cada elemento de la lista va a tener el `color` y `font-size` de `niceStyle` y el estilo de fuente de `badStyle`.

```
1 <!-- abreviación -->
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 para el nivel Intermedio

Cuando en `v-bind:style` usas una propiedad CSS que requiere prefijos de navegadores, por ejemplo `transform`, Vue.js automáticamente detectará y agregará los prefijos apropiados para los estilos aplicados.

Puedes encontrar más información sobre prefijos de navegadores [aquí⁵⁸](#).

Enlaces en Acción

```

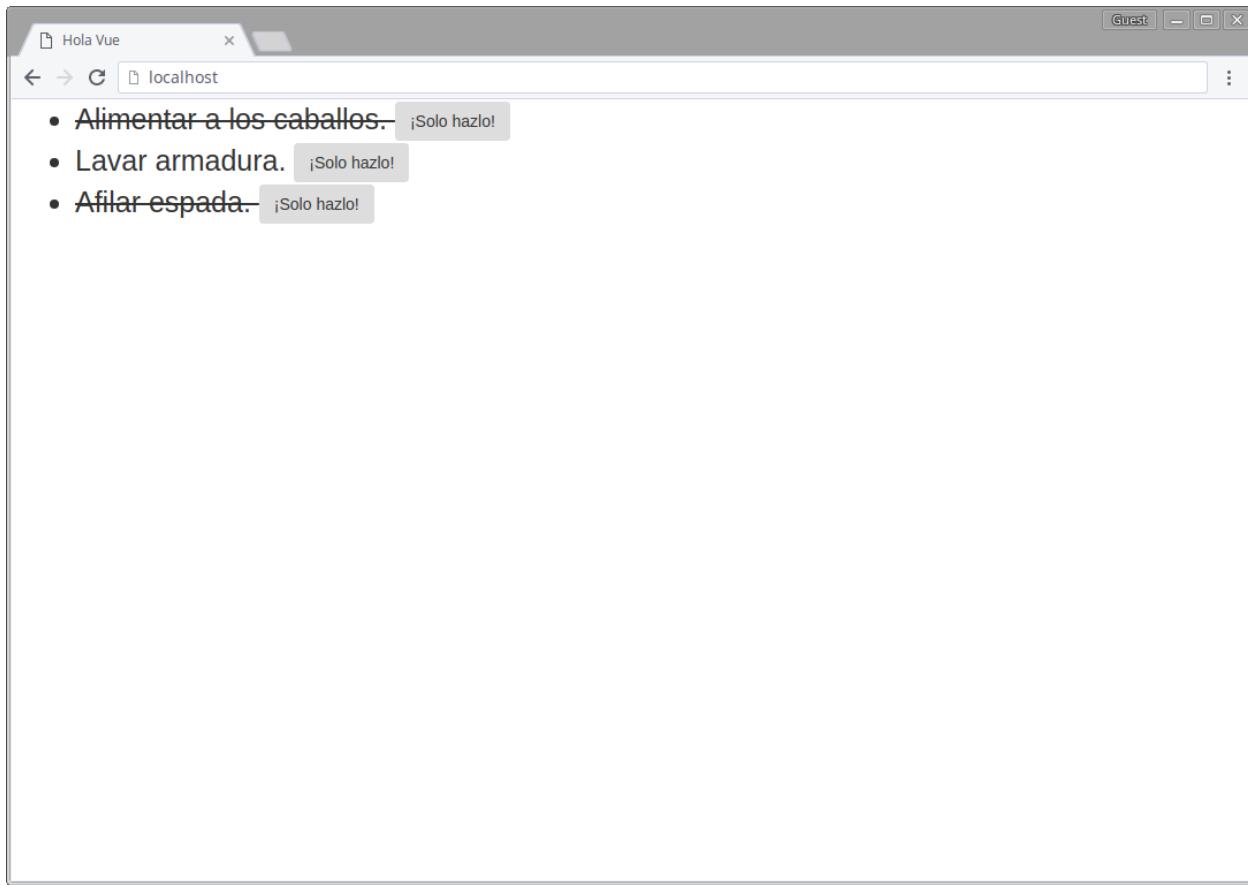
1  <html>
2  <head>
3  <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">
4  <title>Hola Vue</title>
5  </head>
6  <body class="container-fluid">
7      <div id="app">
8          <ul>
9              <li :class="{'completed' : task.done}" style="styleObject"
10                 v-for="task in tasks">
11                  {{task.body}}
12                  <button @click="completeTask(task)" class="btn">
13                      Solo hazlo!
14                  </button>
15              </li>
16          </ul>
17      </div>
18  </body>

```

⁵⁸https://developer.mozilla.org/en-US/docs/Glossary/Vendor_Prefix

```
19   </div>
20 </body>
21 <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1\
22 /vue.js"></script>
23 <script type="text/javascript">
24 new Vue({
25   el: '#app',
26   data: {
27     tasks: [
28       {body: "Alimentar a los caballos.", done: true},
29       {body: "Lavar armadura.", done: true},
30       {body: "Afilar espada.", done: false},
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>
```

El ejemplo anterior tiene un arreglo de objetos llamado `tasks` y un `styleObject` que contiene sólo una propiedad. Con el uso de `v-for` una lista de tareas es renderizada y cada tarea tiene una propiedad `done` con un valor booleano. Dependiendo del valor de `done`, una clase es aplicada condicionalmente como antes. Si una tarea ha sido completada entonces el estilo css se aplica y la tarea obtiene un `text-decoration` de `line-through`. Cada tarea es acompañada por un botón, esperando el evento `click`, que activa un método y cambia el estado de finalización de la tarea. El atributo `style` está enlazado a `styleObject`, resultando en el cambio de `font-size` de todas las tareas. Como puedes ver, el método `completeTasks` toma el parámetro `task`.



Estilo de tareas completadas



Video

El manejo de clases y estilos en Vue.js es cubierto en la lección 7 del curso de Vue 2 en [Styde⁵⁹](#).



Código de Ejemplo

Puedes encontrar el código de ejemplo de este capítulo en [GitHub⁶⁰](#).

⁵⁹<https://styde.net/estilos-y-clases-de-css-dinamicas-con-la-directiva-v-bind-en-vue-2/>

⁶⁰<https://github.com/hoottlex/the-majesty-of-vuejs-2/tree/master/codes/chapter9>

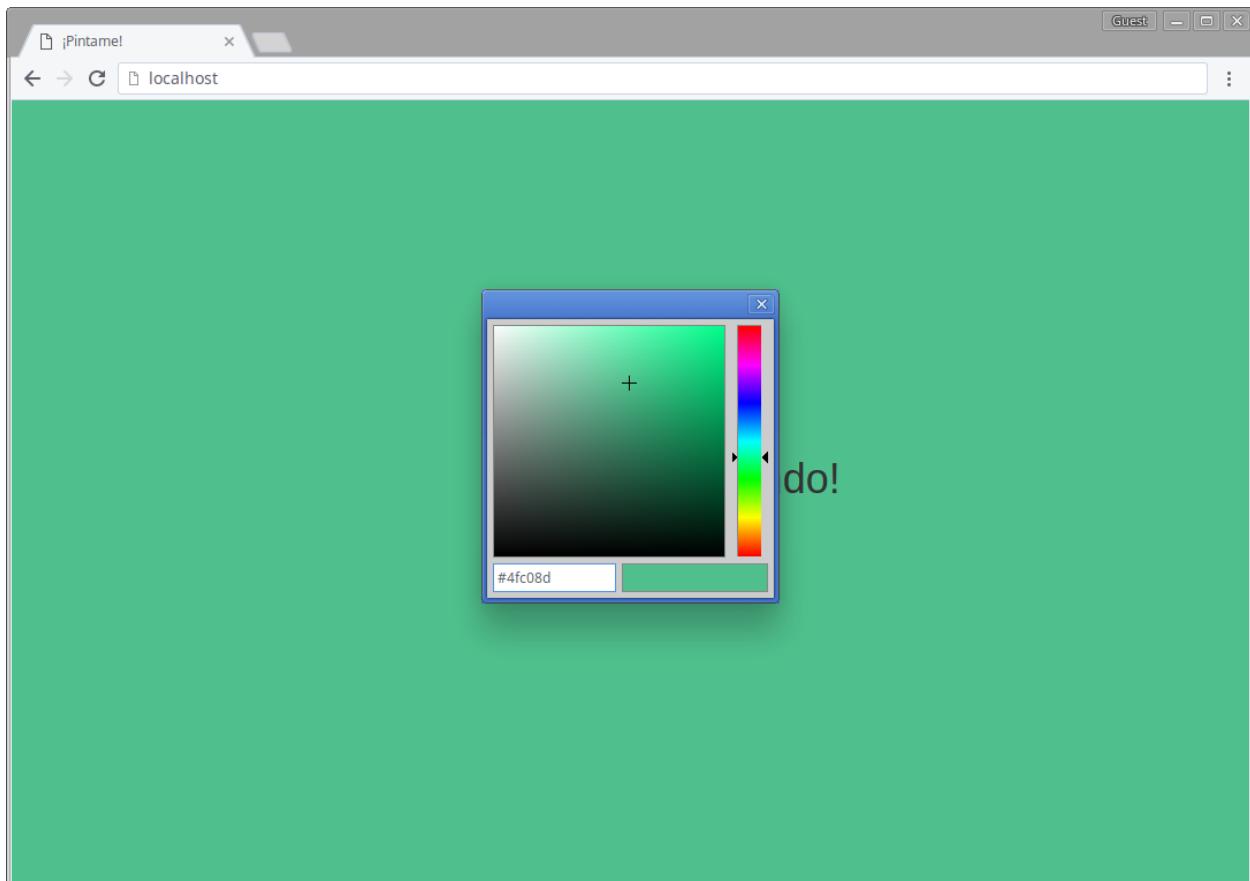
Ejercicio

Un divertido y quizás complicado ejercicio para este capítulo. Crea un campo de entrada donde el usuario puede seleccionar un color. Cuando un color es escogido, aplícalo a un elemento de tu elección. Eso es todo, ¡pintemos! :)



Pista

Para mayor facilidad podrías usar `input type="color"` (soportado en la mayoría de navegadores).



Pantalla de Ejemplo



Possible Solución

Puedes encontrar una posible solución para este ejercicio [aquí⁶¹](#).

⁶¹<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/chapter9.html>

Consumiendo una API

Introducción

En este capítulo, vamos a profundizar un poco y demostrar como podemos usar Vue.js para consumir una API.

Vamos a seguir los ejemplos de *historia* de capítulos anteriores, pero esta vez usando datos reales, procedentes de una fuente externa.

Para utilizar datos reales necesitamos hacer uso de una base de datos. Suponiendo que ya sabes como crear una base de datos, esto no será cubierto en este libro. Para trabajar junto a los ejemplos del libro, te tenemos cubierto, hemos creado una para ser puesta en uso.

CRUD

Imagina que tenemos una base datos y necesitamos realizar operaciones CRUD (Crear, Leer, Actualizar, Borrar). Particularmente, queremos:

- **Crear** nuevas historias en la base de datos
- **Leer** historias existentes
- **Actualizar** detalles existentes de la historia (como ‘upvotes’)
- **Borrar** historias que no nos gustan

Dado que Vue.js es un framework Front-end de JavaScript, no puede conectarse directamente a una base de datos. Para acceder a la base de datos necesitamos una capa entre Vue.js y la base de datos. Esta capa es la API (Interfaz de programación de aplicaciones).

API

Debido a que este libro es sobre Vue.js y no sobre diseño de APIs, te proporcionaremos una API de ejemplo construida con [Laravel⁶²](#). Laravel es uno de los frameworks de PHP más poderosos junto con Symfony2, Nette, CodeIgniter y Yii2. Eres libre de crear tu API usando *cualquier lenguaje o framework* que quieras. Yo uso Laravel porque es simple, tiene una gran comunidad y ¡Es genial! :)

Por lo tanto, te recomendamos ampliamente usar la *API de ejemplo* que hemos construido exclusivamente para los ejemplos de este libro.

⁶²<https://laravel.com/>

Descargar el Código del Libro

Para usar nuestra API tienes que descargar el código del libro e iniciar un servidor. Para ello, sigue las instrucciones debajo.

1. Abre tu terminal y crea un directorio (nosotros crearemos ‘~/themajestyofvuejs2’)

>_ mkdir ~/themajestyofvuejs2

1. Descarga el código fuente desde github

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

Alternativamente puedes visitar el repositorio en [github⁶³](#) y descargar el archivo zip. Luego, extrae su contenido dentro del directorio creado.

1. Navega hasta el capítulo actual dentro de ‘apis’ en el directorio recién creado.

>_ cd ~/themajestyofvuejs2/apis/stories

1. Ejecuta el script de instalación

>_ sh setup.sh

1. ¡Ahora tienes una base de datos llena de datos de ejemplo así como un servidor completamente funcional ejecutándose en [http://localhost:3000!](http://localhost:3000)

Si quieres personalizar el servidor (host, puerto, etc) puedes realizar la configuración manualmente. Debajo está el código fuente de nuestro script.

⁶³<https://github.com/hootlex/the-majesty-of-vuejs-2>

Archivo de Órdenes para Instalación: setup.sh

```
# Navega hasta el directorio del capítulo
$ cd ~/themajestyofvuejs2/apis/stories

# Instala las dependencias
$ composer install

# Crea la base de datos
$ touch database/database.sqlite;

# Migración y seed
$ php artisan migrate;
$ php artisan db:seed;

# Inicia el servidor
$ php artisan serve --port=3000 --host localhost;
```

¡Excelente! Ahora tienes una *API completamente funcional* y una base de datos llena de historias geniales.



Nota

Si estás usando Vagrant, tienes que ejecutar el servidor en **host ‘0.0.0.0’**. De esta forma, serás capaz de acceder a tu servidor en la IP de Vagrant.

Si, por ejemplo, la IP de Vagrant es **192.168.10.10** y tu ejecutas

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

puedes navegar a tu sitio web en 192.168.10.10:3000.

Si has descargado nuestra API de ejemplo, puedes pasar a la siguiente sección.

Si eliges crear tu propia API, tienes que crear una tabla en la base de datos para almacenar las historias. Las siguientes columnas deben estar presentes.

Nombre de Columna	Tipo de datos
id	Integer, Auto Increment
plot	String
writer	String
upvotes	Integer, Unsigned

Ingresa algunos datos de prueba antes de continuar con los próximos ejemplos.

Endpoints de la API

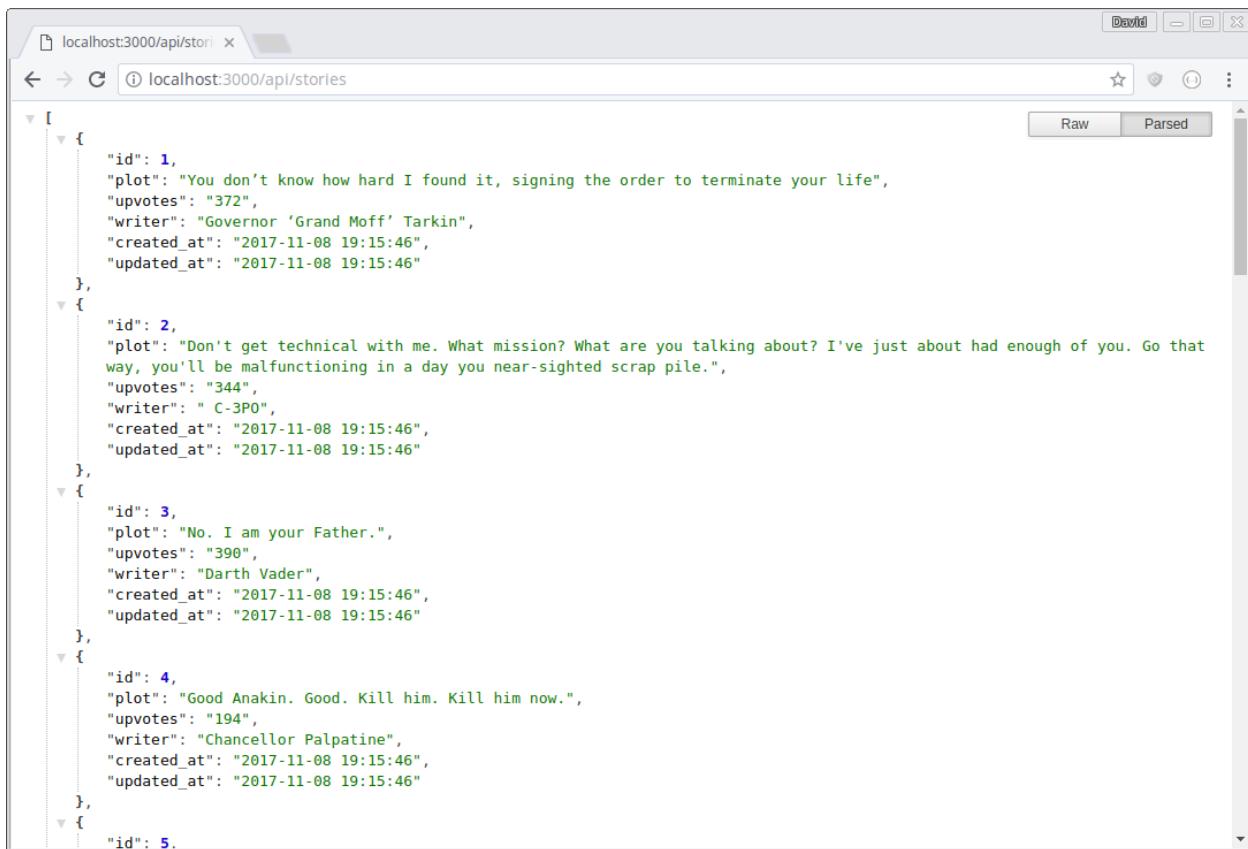
Un endpoint es simplemente una URL. Cuando vamos a `http://example.com/foo/bar`, es un endpoint y tú simplemente necesitas llamarlo /foo/bar porque el dominio será el mismo para todos los endpoints.

Para manipular el recurso **Story** necesitamos 5 endpoints. Cada endpoint corresponde a una acción específica.

Método HTTP	URI	Acción
GET/HEAD	api/stories	<i>Recupera</i> todas las historias
GET/HEAD	api/stories/{id}	<i>Recupera</i> la historia específica
POST	api/stories	<i>Crea</i> una nueva historia
PUT/PATCH	api/stories/{id}	<i>Actualiza</i> una historia existente
DELETE	api/stories/{id}	<i>Elimina</i> una historia específica

Como se indica en la tabla superior, para obtener un listado con todas las ‘historias’ tenemos que hacer una solicitud HTTP GET o HEAD a `api/stories`. Para actualizar una historia existente tenemos que hacer una solicitud HTTP PUT o PATCH a `api/stories/{storyID}` proporcionando los datos que queremos actualizar y reemplazando `{storyID}` con el id de la historia que queremos actualizar. La misma lógica aplica a todos los endpoints. Creo que tienes la idea.

Asumiendo que tu servidor se está ejecutando en `http://localhost:3000`, puedes ver un listado de todas las historias en formato JSON al visitar `http://localhost:3000/api/stories` en tu navegador web.



```
[{"id": 1, "plot": "You don't know how hard I found it, signing the order to terminate your life", "upvotes": "372", "writer": "Governor 'Grand Moff' Tarkin", "created_at": "2017-11-08 19:15:46", "updated_at": "2017-11-08 19:15:46"}, {"id": 2, "plot": "Don't get technical with me. What mission? What are you talking about? I've just about had enough of you. Go that way, you'll be malfunctioning in a day you near-sighted scrap pile.", "upvotes": "344", "writer": "C-3PO", "created_at": "2017-11-08 19:15:46", "updated_at": "2017-11-08 19:15:46"}, {"id": 3, "plot": "No. I am your Father.", "upvotes": "390", "writer": "Darth Vader", "created_at": "2017-11-08 19:15:46", "updated_at": "2017-11-08 19:15:46"}, {"id": 4, "plot": "Good Anakin. Good. Kill him. Kill him now.", "upvotes": "194", "writer": "Chancellor Palpatine", "created_at": "2017-11-08 19:15:46", "updated_at": "2017-11-08 19:15:46"}, {"id": 5}.
```

Respuesta JSON



Tip

Leer datos JSON puros en el navegador puede ser doloroso. Siempre es más fácil leer JSON bien formateado. Chrome tiene algunas buenas extensiones que pueden formatear datos JSON puros a formato de vista de árbol que puede ser leído fácilmente.

Yo uso [JSONFormatter⁶⁴](#) porque soporta resaltado de sintaxis y muestra JSON en vista de árbol, donde los nodos en el árbol pueden estar colapsados o expandidos haciendo clic en el ícono de triángulo a la izquierda de cada nodo. También proporciona un botón para cambiar a los datos originales (puros).

Puedes escoger cualquier extensión que quieras pero ¡definitivamente deberías usar una!

⁶⁴<https://chrome.google.com/webstore/detail/json-formatter/bcjindcccaagfapjjmafapmmgkkhgoa>

Trabajando con Datos Reales

Es tiempo de realmente poner en uso nuestra base de datos y llevar a cabo las operaciones que hemos mencionado (CRUD). Utilizaremos el [último ejemplo del capítulo de Componentes](#) pero esta vez, por supuesto, nuestros datos vendrán desde una fuente externa. Para intercambiar datos con el servidor, necesitamos llevar a cabo solicitudes HTTP asíncronas (Ajax).

Info

AJAX es una técnica que permite a las páginas web ser actualizadas de forma asíncrona al intercambiar pequeñas cantidades de datos con el servidor “detrás de cámara” (sin que se recargue el navegador).

Obtener Datos de Forma Asíncrona

Toma un momento para echar un vistazo al [último ejemplo del capítulo de Componentes](#). Como puedes ver, escribimos los datos del **arreglo stories** directamente en el código fuente, dentro del objeto data de la instancia de Vue.

Arreglo **stories** con los datos escritos directamente en el código fuente

```
new Vue({
  data: {
    stories: [
      {
        plot: 'Mi caballo es genial.',
        writer: 'Mr. Weebly',
      },
      {
        plot: 'Los narvales inventaron el Shish Kebab.',
        writer: 'Mr. Weebly',
      },
      ...
    ]
  }
})
```

Esta vez, queremos obtener las historias existentes desde el servidor.

Para ello, ejecutaremos una solicitud HTTP GET, al principio usando jQuery. Más adelante en este capítulo migraremos a [vue-resource⁶⁵](#), para ver las diferencias entre los dos.

Para realizar la llamada AJAX vamos a usar `$.get()`, una función de jQuery que carga datos desde el servidor usando una solicitud HTTP GET. La documentación completa sobre `$.get()` puede ser encontrada [aquí⁶⁶](#).

Info

`vue-resource` es un plugin para `Vue.js` que proporciona servicios para realizar solicitudes web y manejar respuestas.

La sintaxis del método `$.get()` es:

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

que en realidad es una abreviatura para:

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

Entonces, ¿qué hacemos ahora?. Queremos obtener las historias desde el servidor, usando `$.get('/api/stories')` y almacenar los datos de la respuesta dentro del arreglo `stories`.

Hay un problema común aquí, tenemos que hacer la llamada **luego de que la Instancia esté lista**. ¿Recuerdas los [Lifecycle Hooks de Vue](#)?

Hay un *hook* llamado `mounted`, que es ejecutado justo después de que la instancia ha sido montada.

⁶⁵<https://github.com/vuejs/vue-resource>

⁶⁶<https://api.jquery.com/jquery.get/>



Advertencia

El hook `mounted` no es equivalente a `$(document).ready()` de jQuery. Al usar `mounted`, **no hay garantía de estar en el documento**. Si necesitas ejecutar algo una vez que el Document Object Model (DOM) de la página está listo, puedes usar:

```
mounted: function () {
  this.$nextTick(function () {
    // código que asume que this.$el está en el documento
  })
}
```

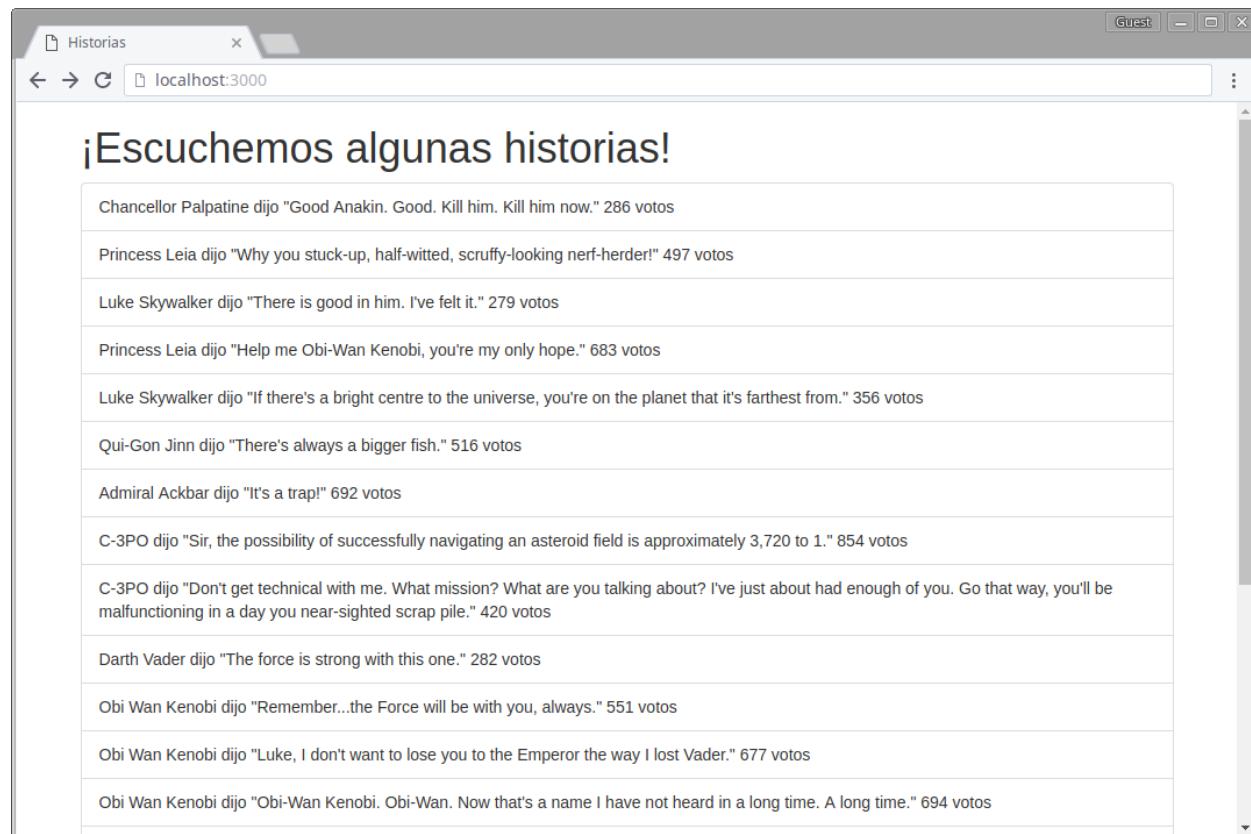
Veamos esto en acción.

```
1 <div id="app">
2   <div class="container">
3     <h1> iEscuchemos algunas historias! </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-raw">
12   <li class="list-group-item">
13     {{ story.writer }} dijo "{{ story.plot }}"
14     <span>{{story.upvotes}}</span> votos
15   </li>
16 </template>

1 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
2 <script src="https://code.jquery.com/jquery-2.1.4.min.js"></script>
3 <script type="text/javascript">
4 Vue.component('story', {
5   template: "#template-story-raw",
6   props: ['story'],
7 });
8
9 var vm = new Vue({
10   el: '#app',
```

```
11     data: {
12         stories: []
13     },
14     mounted: function(){
15         \$.get('api/stories', function(data){
16             vm.stories = data;
17         })
18     }
19 })
20 </script>
```

Comenzamos enlazando jQuery desde [cdnjs⁶⁷](#). Luego, dentro del hook `mounted`, realizamos una solicitud GET. Después de que la solicitud ha finalizado con éxito, almacenamos los datos de la respuesta (que se encuentran dentro de la función callback) en el arreglo `stories`.



Obtener historias

⁶⁷<https://cdnjs.com/libraries/jquery/>



Nota aquí, dentro de la función callback nos estamos refiriendo a la variable `stories` usando `vm.stories` en lugar de `this.stories`. Hacemos eso porque la variable `this` no está enlazada a la instancia de `Vue` dentro de la función callback. Así que guardamos toda la instancia `Vue` en una variable llamada `vm`, con el propósito de tener acceso a ella desde cualquier lugar de nuestro código. Para aprender más sobre `this`, echa un vistazo a: [la documentación⁶⁸](#).

Refactorizando

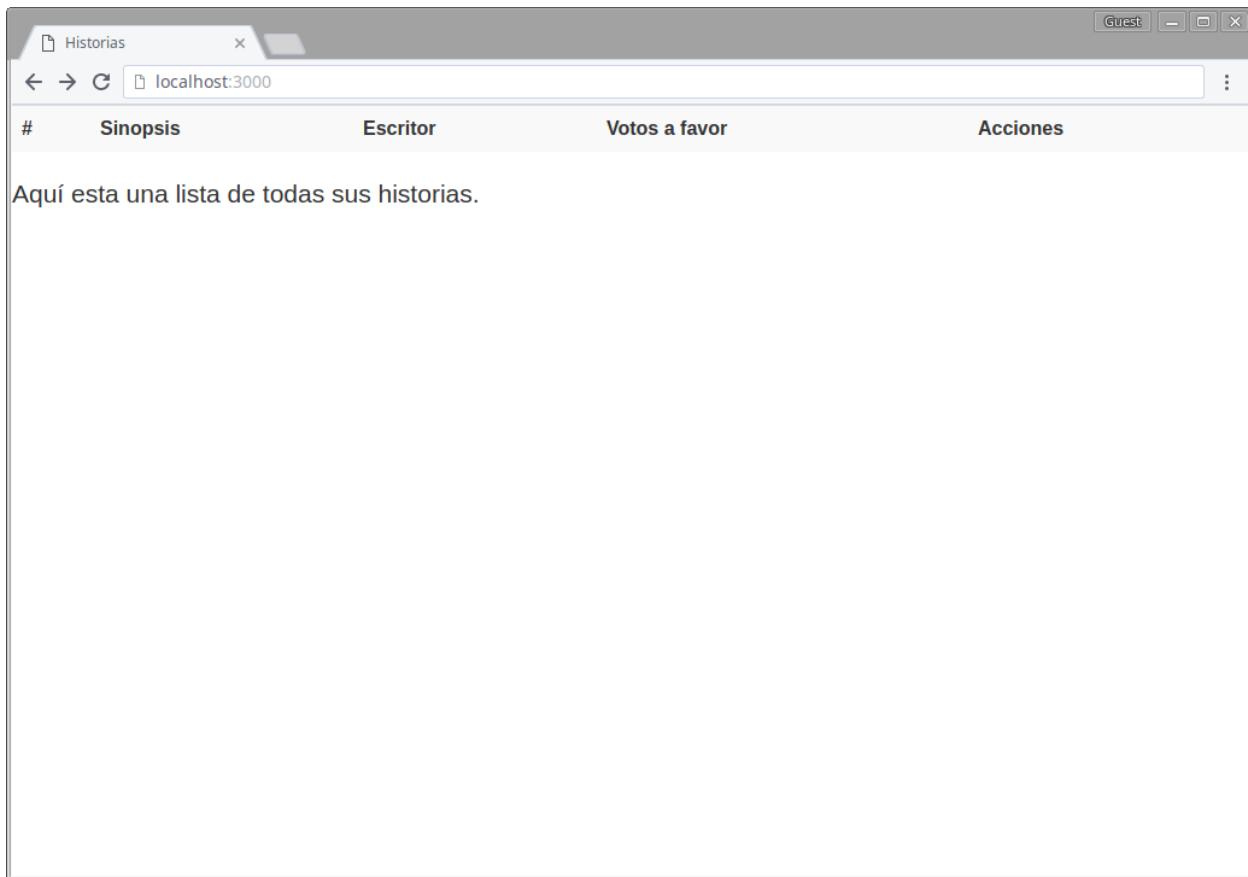
Tener grandes cantidades de código en nuestro editor, así como en el navegador, puede ser confuso si no se muestra de forma adecuada. Por esa razón vamos a refactorizar nuestro código de ejemplo, para renderizar la lista de historias usando un elemento `<table>` en lugar de un ``.

```
1 <div id="app">
2   <table class="table table-striped">
3     <tr>
4       <th>#</th>
5       <th>Sinopsis</th>
6       <th>Escritor</th>
7       <th>Votos a favor</th>
8       <th>Acciones</th>
9     </tr>
10    <tr v-for="story in stories" is="story" :story="story"></tr>
11  </table>
12 </div>
13 <template id="template-story-raw">
14   <tr>
15     <td>
16       {{story.id}}
17     </td>
18     <td>
19       <span>
20         {{story.plot}}
21       </span>
22     </td>
23     <td>
24       <span>
25         {{story.writer}}
26       </span>
27     </td>
28     <td>
```

⁶⁸<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this>

```
29          {{story.upvotes}}
30      </td>
31  </tr>
32</template>
33< p class="lead">Aquí está una lista de todas sus historias.
34</p>
35< pre>{{ $data }}</pre>
```

Pero hay un problema importante.



Los problemas de la renderización

Nuestra tabla no se renderiza correctamente, pero ¿por qué?⁶⁹

*Algunos elementos HTML, por ejemplo <table>, tienen restricciones en los elementos que pueden aparecer dentro de ellos. Los elementos personalizados que no están en la lista blanca se eliminarán y por lo tanto no se renderizarán correctamente. En esos casos deberías usar el atributo especial **is** para indicar un elemento personalizado.*

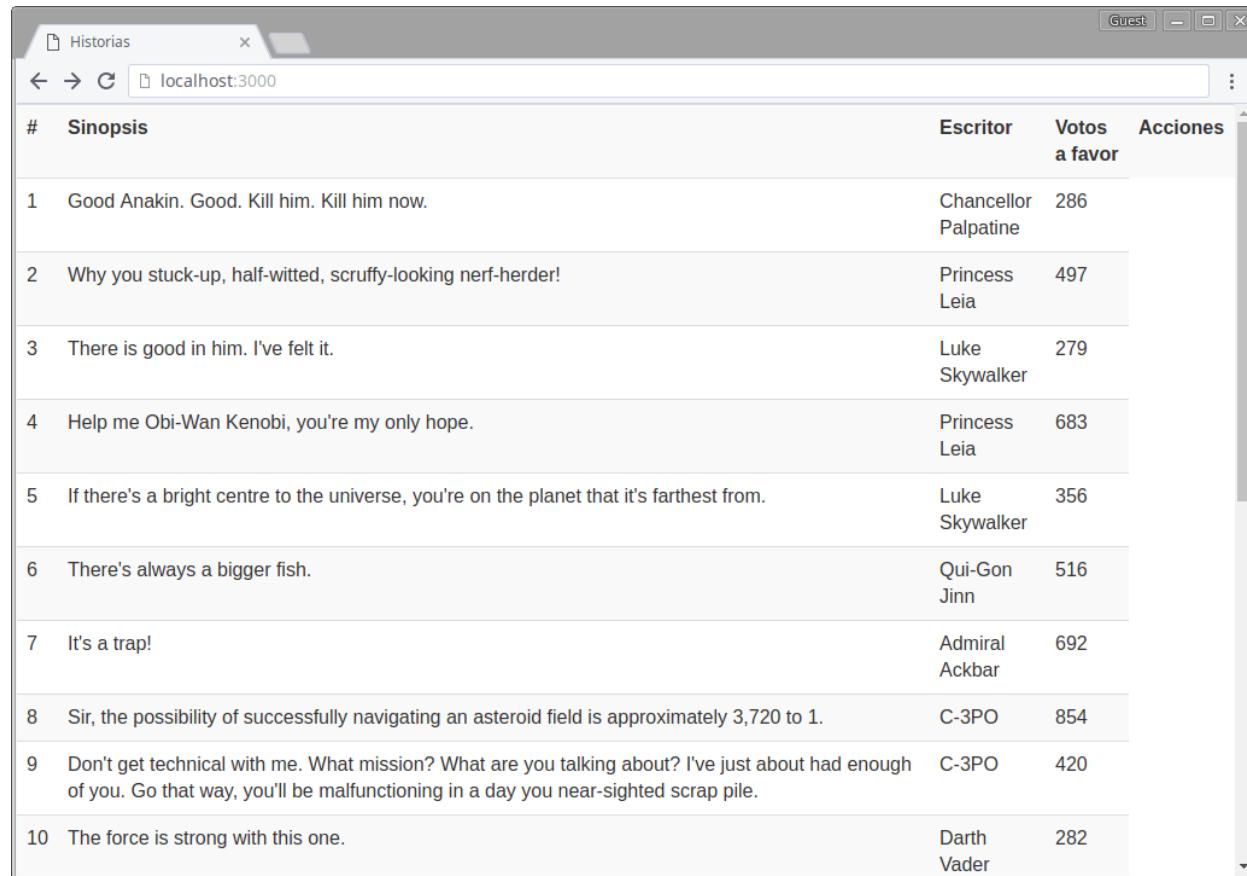
Por lo tanto, para resolver este problema tenemos que usar el **atributo especial is** de Vue.

⁶⁹<http://goo.gl/Xr9RoQ>

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

Así que nuestro ejemplo se convertirá en:

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



The screenshot shows a web browser window with the title 'Historias' and the URL 'localhost:3000'. The page displays a table with the following data:

#	Sinopsis	Escritor	Votos	Acciones a favor
1	Good Anakin. Good. Kill him. Kill him now.	Chancellor Palpatine	286	
2	Why you stuck-up, half-witted, scruffy-looking nerf-herder!	Princess Leia	497	
3	There is good in him. I've felt it.	Luke Skywalker	279	
4	Help me Obi-Wan Kenobi, you're my only hope.	Princess Leia	683	
5	If there's a bright centre to the universe, you're on the planet that it's farthest from.	Luke Skywalker	356	
6	There's always a bigger fish.	Qui-Gon Jinn	516	
7	It's a trap!	Admiral Ackbar	692	
8	Sir, the possibility of successfully navigating an asteroid field is approximately 3,720 to 1.	C-3PO	854	
9	Don't get technical with me. What mission? What are you talking about? I've just about had enough of you. Go that way, you'll be malfunctioning in a day you near-sighted scrap pile.	C-3PO	420	
10	The force is strong with this one.	Darth Vader	282	

La tabla se renderiza adecuadamente

Bueno, ¡esto se ve mejor!

Actualizar Datos

Solíamos tener una función que permitía al usuario votar a favor de cualquier historia que quisiera. Pero ahora queremos algo más. Queremos que el servidor sea informado cada vez que una historia es votada a favor, asegurando que los votos a favor de la historia también sean actualizados en la base de datos.

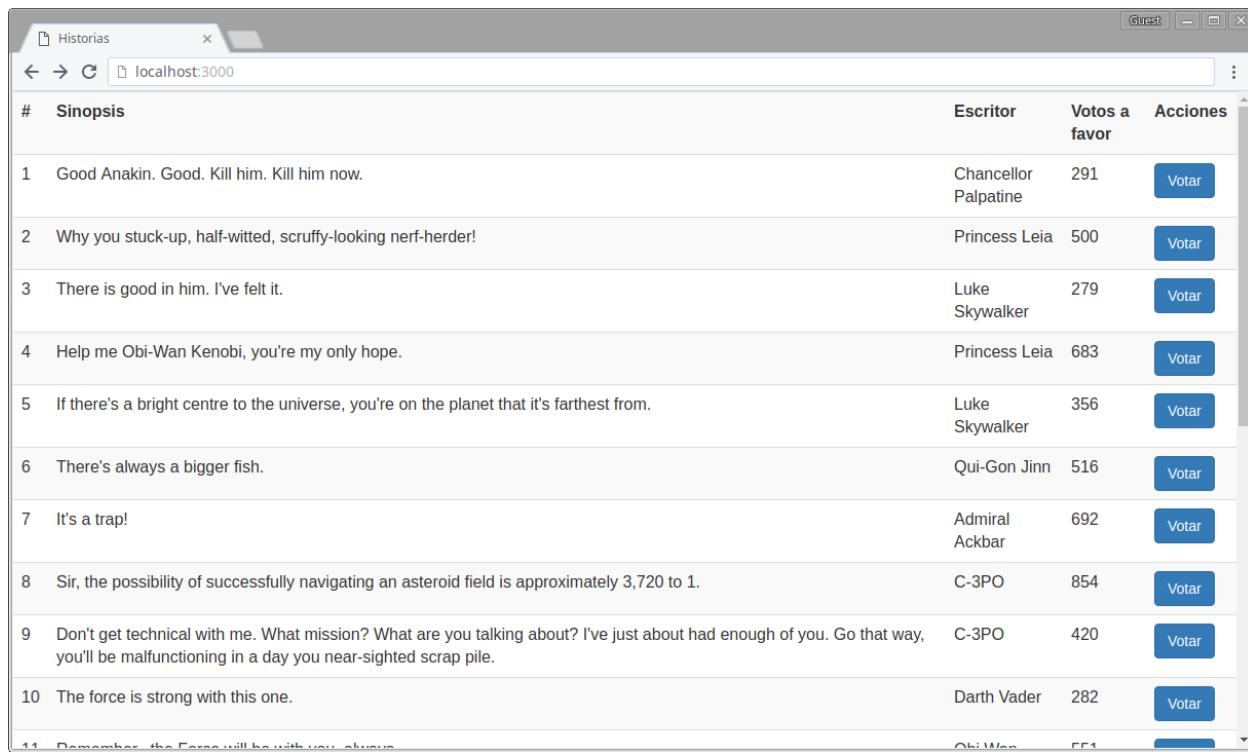
Para actualizar una historia existente, tenemos que hacer una solicitud HTTP PATCH o PUT a `api/stories/{storyID}`.

Dentro de la función `upvoteStory`, que está por ser creada, vamos a hacer una llamada HTTP después de haber incrementado los **votos a favor de la historia**. Pasaremos la variable `story` recién actualizada en la parte correspondiente a los datos dentro de la solicitud HTTP, con el fin de actualizar los datos en nuestro servidor.

```
1 <td>
2   <div class="btn-group">
3     <button @click="upvoteStory(story)" class="btn btn-primary">
4       Votar a favor
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 })
```

Trajimos de vuelta el método `upvote` y lo situamos dentro del componente `story`. Ahora al hacer una solicitud PATCH, proporcionando los datos nuevos, el servidor actualiza el conteo de `upvotes`.



The screenshot shows a table titled "Historias" with 11 rows. Each row contains a quote, its author, the number of votes, and a blue "Votar" button. The table has columns for "#", "Sinopsis", "Escritor", "Votos a favor", and "Acciones". The last row is a footer with the text "Remember, the Force will be with you, always".

#	Sinopsis	Escritor	Votos a favor	Acciones
1	Good Anakin. Good. Kill him. Kill him now.	Chancellor Palpatine	291	<button>Votar</button>
2	Why you stuck-up, half-witted, scruffy-looking nerf-herder!	Princess Leia	500	<button>Votar</button>
3	There is good in him. I've felt it.	Luke Skywalker	279	<button>Votar</button>
4	Help me Obi-Wan Kenobi, you're my only hope.	Princess Leia	683	<button>Votar</button>
5	If there's a bright centre to the universe, you're on the planet that it's farthest from.	Luke Skywalker	356	<button>Votar</button>
6	There's always a bigger fish.	Qui-Gon Jinn	516	<button>Votar</button>
7	It's a trap!	Admiral Ackbar	692	<button>Votar</button>
8	Sir, the possibility of successfully navigating an asteroid field is approximately 3,720 to 1.	C-3PO	854	<button>Votar</button>
9	Don't get technical with me. What mission? What are you talking about? I've just about had enough of you. Go that way, you'll be malfunctioning in a day you near-sighted scrap pile.	C-3PO	420	<button>Votar</button>
10	The force is strong with this one.	Darth Vader	282	<button>Votar</button>
11	Remember, the Force will be with you, always.	Obi-Wan Kenobi	554	<button>Votar</button>

Votos a favor de las historias

Eliminar datos

Vamos a proceder con otra pieza de funcionalidad, nuestra lista de **historias** debería tener: Eliminar una historia que no nos gusta. Para eliminar una **historia** del arreglo y el DOM, tenemos que buscarla y eliminarla del arreglo **stories**.

```

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

```

Agregamos un botón *Eliminar* a la columna *acciones*, enlazado a un método para eliminar la historia. El método **deleteStory** será:

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

Pero por supuesto, de esta forma sólo removeremos la historia temporalmente. Con el fin de eliminar la historia de la base de datos, tenemos que ejecutar una solicitud DELETE de AJAX.

```
1 Vue.component('story',{
2   ...
3   methods: {
4     ...
5     deleteStory: function(story){
6       // encontrar la historia
7       var index = vm.stories.indexOf(story);
8
9       // eliminarla
10      vm.stories.splice(index, 1)
11
12      // ejecutar la solicitud DELETE
13      $.ajax({
14        url: '/api/stories/' + story.id,
15        type: 'DELETE'
16      });
17    },
18  }
19  ...
20 })
```

Estamos pasando la URL, como hicimos antes. El tipo (Type) aquí debe ser igual a **DELETE**. Nuestro método ahora está listo y podemos eliminar la historia de la base de datos así como del DOM.

#	Sinopsis	Escritor	Votos a favor	Acciones
1	One thing's for sure, we're all going to be a lot thinner.	Han Solo	166	<button>Votar</button> <button>Eliminar</button>
2	The Force is strong with you. A powerful Sith you will become. Henceforth, you shall be known as Darth... Vader.	Darth Sidious	421	<button>Votar</button> <button>Eliminar</button>
3	Great, kid. Don't get cocky	Han Solo	176	<button>Votar</button> <button>Eliminar</button>
4	Don't call me a mindless philosopher, you overweight glob of grease.	C-3PO	294	<button>Votar</button> <button>Eliminar</button>
5	No. I am your Father.	Darth Vader	561	<button>Votar</button> <button>Eliminar</button>
6	The force is strong with this one.	Darth Vader	160	<button>Votar</button> <button>Eliminar</button>
7	You don't know how hard I found it, signing the order to terminate your life	Governor 'Grand Moff' Tarkin	237	<button>Votar</button> <button>Eliminar</button>
8	Oh my. Space travel sound rather perilous. I can assure you they will never get me on one of those dreadful Star Ships.	C-3PO	676	<button>Votar</button> <button>Eliminar</button>
9	I suggest a new strategy, R2: let the Wookiee win.	C-3PO	477	<button>Votar</button> <button>Eliminar</button>
10	You can't win, Darth. Strike me down, and I will become more powerful than you could possibly imagine.	Obi Wan Kenobi	320	<button>Votar</button> <button>Eliminar</button>

Votar a favor y Eliminar las historias

Eso es todo por ahora. Continuaremos nuestro ejemplo en el próximo capítulo, al mejorar la funcionalidad con **Crear nuevas historias**, **Editando historias existentes** y más. Pero antes que nada, reemplazaremos **jQuery** con **vue-resource**.

Integrando Vue-resource

Resumen

Vue-resource es un plugin de recursos para Vue.js.

Este plugin proporciona servicios para hacer solicitudes web y maneja respuestas usando XMLHttpRequest o JSONP.

Vamos a hacer de nuevo todas las solicitudes web que hicimos en el capítulo previo, usando este plugin en su lugar. De esta forma puedes ver las diferencias y decidir cuál es mejor para tus necesidades. *jQuery* es bueno, pero si estás usándolo sólo para realizar llamadas AJAX podrías considerar eliminarlo.

Aquí puedes encontrar instrucciones para la instalación y documentación sobre [vue-resource⁷⁰](#). Como siempre, vamos a enlazarlo desde una página [cdn⁷¹](#).

Para obtener datos desde un servidor podemos usar el método `$http` de vue-resource con la siguiente sintaxis:

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



Info

Una instancia de Vue proporciona la función `this.$http(options)` que recibe un objeto de opciones para generar una solicitud HTTP y devuelve una “promesa” (JavaScript). También la instancia de Vue será automáticamente enlazada a `this` en todos los callbacks de las funciones.

En lugar de pasar la opción del método, hay abreviaciones de métodos disponibles para todos los tipos de solicitudes.

⁷⁰<https://github.com/vuejs/vue-resource>

⁷¹<https://cdnjs.com/libraries/vue-resource>

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);
```

Migración

Es hora de usar `vue-resource` en nuestro ejemplo. Primero que todo, tenemos que incluirlo. Añadiremos esta línea a nuestro archivo HTML:

```
1 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue-resource/1.0.3/vue-resource.\
2 js"></script>
```

Para recibir las historias realizaremos una solicitud GET en el formato correspondiente:

```
mounted: function() {
  // Solicitud GET
  this.$http({url: '/api/stories', method: 'GET'})
  .then(function (response) {
    Vue.set(vm, 'stories', response.data)
    // O como hicimos anteriormente
    // vm.stories = response.data
  })
}
```

Al usar la sintaxis de arriba, nuestra lista de historias viene sin ningún problema.

Continuemos ahora con las solicitudes DELETE y PATCH usando los métodos abreviados.

Solicitud PATCH

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

Solicitud DELETE

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

Hemos reemplazado los métodos de AJAX con estos en un abrir y cerrar de ojos.

Info

Dado que el componente *story* no tiene acceso al arreglo *stories*, accedemos al arreglo utilizando `this.$parent.stories`. También podríamos usar `vm.stories` o emitir un evento para actualizar el arreglo dentro de la instancia padre de Vue.

Mejorando la Funcionalidad

Deberíamos agregar un par de características más, para hacer que nuestra lista de historias sea más ordenada. Podemos darle al usuario la habilidad de **cambiar la trama de una historia, su autor y también crear nuevas historias**.

Editar Historias

Comencemos con la primera tarea y demos al usuario algunos campos para manipular los atributos de la historia. Dos campos enlazados deberían hacer el trabajo, pero debemos mostrarlos **sólo** cuando el usuario está **editando** una historia. Se parece al tipo de trabajo que hicimos en capítulos anteriores.

Para definir si una *historia* está en **estado de edición** usaremos una propiedad, `editing`, que se convertirá en verdadera cuando el usuario presione el botón *Editar*.

```
1 <td>  
2        
3     <input v-if="story.editing" v-model="story.plot" class="form-control">  
4     </input>  
5     <!--en otras ocasiones, muestra plot--&gt;<br/>6     <span v-else>  
7         {{story.plot}}  
8     </span>  
9 </td>  
10 <td>
```

```

11      <!--Si estás editando la historia, muestra el campo de texto para writer -->
12      <input v-if="story.editing" v-model="story.writer" class="form-control">
13      </input>
14      <!--en otras ocasiones, muestra 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              Votar
26          </button>
27          <button @click="editStory(story)" class="btn btn-default">
28              Editar
29          </button>
30          <button @click="deleteStory(story)" class="btn btn-danger">
31              Eliminar
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 })

```

Esta es nuestra tabla actualizada, con dos campos de entrada nuevos y un botón. Usamos la función `editStory` para establecer `story.editing` a true, así `v-if` traerá los campos de entrada para editar la historia y ocultará los botones *Votar* y *Eliminar*. Sin embargo, esta forma no funcionará. Parece que el DOM no se está actualizando luego de establecer `story.editing` a true. Pero, ¿por qué sucede esto?

Resulta, de acuerdo a [este post en el blog de Vue.js⁷²](#), que cuando estás añadiendo una nueva propiedad que no estaba presente cuando los datos fueron observados el DOM no se actualizará. La mejor práctica es siempre declarar por adelantado las propiedades que deben ser reactivas. En casos donde absolutamente necesitas agregar o eliminar propiedades en tiempo de ejecución, usa los métodos globales `Vue.set` o `Vue.delete`.

Por esta razón tenemos que inicializar el atributo `story.editing` a `false` en cada historia, justo después de recibir las historias desde el servidor.

Para hacer esto, vamos a usar el método `.map()` de Javascript, dentro de la función callback de la solicitud GET.

```
mounted: function() {
  var vm = this;

  // solicitud GET
  this.$http({url: '/api/stories', method: 'GET'})
    .then(function (response) {
      var storiesReady = response.data.map(function(story){
        story.editing = false;
        return story
      })

      Vue.set(vm, 'stories', storiesReady)
    })
}
```



Info

El método `.map()` llama, en cada elemento de un arreglo, a una función callback definida y devuelve un arreglo que contiene los resultados. Puedes encontrar más información sobre el método `.map()` y su sintaxis [aquí⁷³](#).

Esta función agrega el atributo `editing` a cada objeto de la historia y luego retorna la historia actualizada. La nueva variable, `storiesReady`, es un arreglo que contiene nuestro arreglo actualizado con el nuevo atributo en `true`.

Cuando la historia está bajo edición le daremos al usuario dos opciones: actualizar la historia con los nuevos valores o cancelar la edición.

⁷²<http://vuejs.org/2016/02/06/common-gotchas/>

⁷³[https://msdn.microsoft.com/en-us/library/ff679976\(v=vs.94\).aspx](https://msdn.microsoft.com/en-us/library/ff679976(v=vs.94).aspx)

	Historia	Creador	Votos	Votar	Editar	Eliminar
9	I suggest a new strategy, R2: let the wookiee win.	C-3PO	411	Votar	Editar	Eliminar
10	You can't win, Darth. Strike me down, and I will become more powerful than you could possibly imagine.	Obi Wan Kenobi	320	Votar	Editar	Eliminar
11	Great shot kid, that was one in a million.	Han Solo	11	Votar	Editar	Eliminar
12	You can't win, Darth. Strike me down, and I will become more powerful than you could possibly imagine.	Obi Wan Kenobi	9	Votar	Editar	Eliminar
13	Aren't you a little short for a storm trooper?	Princess Leia	509	Votar	Editar	Eliminar
14	Master Kenobi, you disappoint me. Yoda holds you in such high esteem. Surely you can do better!	Count Dooku	622	Votar	Editar	Eliminar
15	Obi-Wan has taught you well.	Darth Vader	18			
16	Traveling through hyperspace ain't like dusting crops, farm boy.	Han Solo	166	Votar	Editar	Eliminar
17	There's always a bigger fish.	Qui-Gon Jinn	584	Votar	Editar	Eliminar
18	Oh my. Space travel sound rather perilous. I can assure you they will never get me on one of those dreadful Star Ships.	C-3PO	742	Votar	Editar	Eliminar
19	Luke, I don't want to lose you to the Emperor the way I lost Vader.	Obi Wan Kenobi	746	Votar	Editar	Eliminar
20	Remember...the Force will be with you, always.	Obi Wan Kenobi	146	Votar	Editar	Eliminar

Campos de entrada de formularios para la edición de la historia

Así que, continuemos y agreguemos dos botones, que deberían ser mostrados sólo cuando el usuario esté editando una historia. Adicionalmente, un nuevo método llamado `updateStory` será creado. Este va a actualizar la historia que se está editando actualmente, luego de que el botón *Actualizar Historia* sea presionado.

```
<!-- Si la historia está bajo edición, muestra este grupo de botones -->
<div class="btn-group" v-else>
  <button @click="updateStory(story)" class="btn btn-primary">
    Actualizar Historia
  </button>
  <button @click="story.editing=false" class="btn btn-default">
    Cancelar
  </button>
</div>
```

```

1  Vue.component('story',{
2      ...
3      methods: {
4          ...
5              updateStory: function(story){
6                  this.$http.patch('/api/stories/'+story.id , story)
7                      // Establecer editing a false para mostrar las acciones nuevamente y ocu\
8                      ltar los campos
9                      story.editing = false;
10                 },
11             }
12         ...
13     })

```

#	Sinopsis	Escritor	Votos a favor	Acciones
1	One thing's for sure, we're all going to be a lot thinner.	Han Solo	166	<button>Votar</button> <button>Editar</button> <button>Eliminar</button>
2	The Force is strong with you. A powerful Sith you will become. Henceforth, you shall be known as Darth... Vader.	Darth Sidious	421	<button>Votar</button> <button>Editar</button> <button>Eliminar</button>
3	Great, kid. Don't get cocky	Han Solo	176	<button>Actualizar Historia</button> <button>Cancelar</button>
4	Don't call me a mindless philosopher, you overweight glob of grease.	C-3PO	294	<button>Votar</button> <button>Editar</button> <button>Eliminar</button>
5	No. I am your Father.	Darth Vader	561	<button>Votar</button> <button>Editar</button> <button>Eliminar</button>
6	The force is strong with this one.	Darth Vader	160	<button>Votar</button> <button>Editar</button> <button>Eliminar</button>
7	You don't know how hard I found it, signing the order to terminate your life	Governor 'Grand Moff' Tarkin	237	<button>Votar</button> <button>Editar</button> <button>Eliminar</button>
8	Oh my. Space travel sound rather perilous. I can assure you they will never get me on one of those dreadful Star Ships.	C-3PO	676	<button>Votar</button> <button>Editar</button> <button>Eliminar</button>
9	I suggest a new strategy, R2: let the Wookiee win.	C-3PO	477	<button>Votar</button> <button>Editar</button> <button>Eliminar</button>
10	You can't win, Darth. Strike me down, and I will become more powerful than you could possibly imagine.	Obi Wan Kenobi	320	<button>Votar</button> <button>Editar</button> <button>Eliminar</button>

Acciones de la actualización de historia

Aquí está, ejecutándose y funcionando. Una vez que la solicitud PATCH es finalizada con éxito, tenemos que establecer `story.editing` de nuevo a `false`, con el fin de ocultar los campos de entrada y traer de vuelta los botones de acciones.

Crear Nuevas Historias

Ahora, para una tarea un poco más complicada, vamos a darle al usuario la habilidad de crear una nueva historia y guardarla en nuestro servidor. Primero debemos proporcionar campos de entrada, para que la nueva historia pueda ser escrita. Para esto, crearemos una historia vacía y la añadiremos al arreglo `stories` usando el método `push()` de JavaScript. Inicializaremos todos los atributos de la historia a *un valor nulo apropiado* excepto por `editing`. Dado que, queremos manipular inmediatamente la nueva historia, la propiedad `editing` será establecida a `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">Aquí está una lista de todas tus historias.
2   <button @click="createStory()" class="btn btn-primary">
3     ¿Añadir una nueva?
4   </button>
5 </p>

```



Info

El método `push()` agrega nuevos elementos al final de un arreglo, y retorna el nuevo tamaño. Puedes encontrar más información sobre el método `push()` y su sintaxis [aquí⁷⁴](#).

Nombramos a la nueva función `createStory` y la colocamos en nuestra instancia de Vue.

Justo debajo de nuestra lista hemos añadido un botón. Cuando el botón es presionado, el método `createStory` es invocado. Dado que `newStory.editing` está establecida a `true`, los campos enlazados para `plot` y `writer` junto con los *botones de edición* están siendo renderizados al instante.

También, el nuevo objeto `story` debe ser enviado al servidor para ser almacenado en la base de datos. Vamos a realizar una solicitud POST dentro de un método llamado `storeStory`.

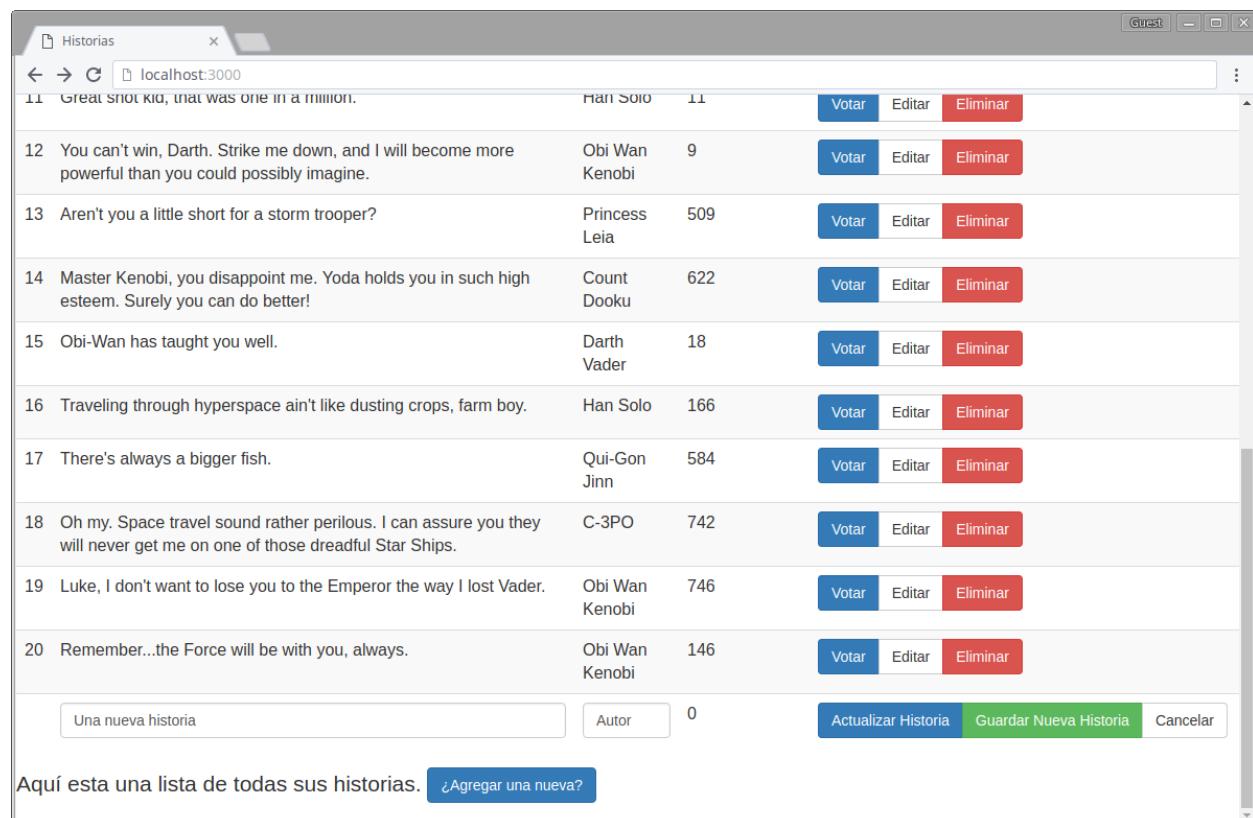
⁷⁴https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/push

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

Hemos usado el método abreviado para realizar la solicitud POST. En la función callback hemos establecido editing a false con el fin de mostrar los botones *de acción* nuevamente y ocultar los campos de entrada del formulario y los botones de *edición*. Debajo, vamos a actualizar los *grupos* de botones, de acuerdo con el nuevo método.

```
1 <td>
2 <div class="btn-group" v-if="!story.editing">
3     <button @click="upvoteStory(story)" class="btn btn-primary">
4         Votar a favor
5     </button>
6     <button @click="editStory(story)" class="btn btn-default">
7         Editar
8     </button>
9     <button @click="deleteStory(story)" class="btn btn-danger">
10        Eliminar
11    </button>
12 </div>
13 <div class="btn-group" v-else>
14     <button class="btn btn-primary" @click="updateStory(story)">
15         Actualizar Historia
16     </button>
17     <button class="btn btn-success" @click="storeStory(story)">
18         Guardar Nueva Historia
19     </button>
20     <button @click="story.editing=false" class="btn btn-default">
21         Cancelar
22     </button>
23 </div>
24 </td>
```

Podemos observar un pequeño error en este bloque de código. Cuando estamos en modo de *edición* (bloque `v-else`) vemos que los botones para actualizar y guardar están siendo mostrados juntos, pero sólo necesitamos uno por cada historia, ya que cada historia será **Almacenada ó Actualizada**. No puede hacer ambas cosas a la vez. Así que, si la historia es antigua y el usuario está a punto de editarla, necesitamos el botón de actualizar. Por otra parte, si la historia es nueva, necesitamos el botón de guardar.



The screenshot shows a web browser window titled "Historias" at "localhost:3000". The page displays a list of 20 Star Wars quotes, each with an ID, the quote text, the author, the number of votes, and three buttons: "Votar" (blue), "Editar" (light blue), and "Eliminar" (red). At the bottom of the list, there is a form for adding a new quote, with fields for "Uma nueva historia" (text input), "Autor" (button), and "0" (number input). Below the form are three buttons: "Actualizar Historia" (blue), "Guardar Nueva Historia" (green), and "Cancelar" (gray).

ID	Historia	Autor	Votos	Opciones
11	Great shot kid, that was one in a million.	Han Solo	11	Votar Editar Eliminar
12	You can't win, Darth. Strike me down, and I will become more powerful than you could possibly imagine.	Obi Wan Kenobi	9	Votar Editar Eliminar
13	Aren't you a little short for a storm trooper?	Princess Leia	509	Votar Editar Eliminar
14	Master Kenobi, you disappoint me. Yoda holds you in such high esteem. Surely you can do better!	Count Dooku	622	Votar Editar Eliminar
15	Obi-Wan has taught you well.	Darth Vader	18	Votar Editar Eliminar
16	Traveling through hyperspace ain't like dusting crops, farm boy.	Han Solo	166	Votar Editar Eliminar
17	There's always a bigger fish.	Qui-Gon Jinn	584	Votar Editar Eliminar
18	Oh my. Space travel sound rather perilous. I can assure you they will never get me on one of those dreadful Star Ships.	C-3PO	742	Votar Editar Eliminar
19	Luke, I don't want to lose you to the Emperor the way I lost Vader.	Obi Wan Kenobi	746	Votar Editar Eliminar
20	Remember...the Force will be with you, always.	Obi Wan Kenobi	146	Votar Editar Eliminar

Aquí esta una lista de todas sus historias. [¿Agregar una nueva?](#)

Un pequeño error

Para evitar este problema, vamos a reestructurar nuestros botones. El botón **Actualizar** será mostrado **solo** cuando la **historia es antigua**. En consecuencia, el botón **Guardar nueva Historia** será mostrado cuando la **historia es nueva**.

Te habrás dado cuenta que todas las historias obtenidas desde el servidor tienen un atributo `id`. Vamos a usar esta observación para **definir si una historia es nueva o no**.

```
1 <div class="btn-group" v-else>
2     <!-- Si la historia es antigua entonces queremos actualizarla
3     TIP: si la historia es tomada de la base de datos entonces tendrá un id -->
4     <button v-if="story.id" class="btn btn-primary" @click="updateStory(story)">
5         Actualizar Historia
6     </button>
7     <!-- Si la historia es nueva queremos guardarla -->
8     <button v-else class="btn btn-success" @click="storeStory(story)">
9         Guardar Nueva Historia
10    </button>
11    <!-- Siempre mostramos cancelar -->
12    <button @click="story.editing=false" class="btn btn-default">
13        Cancelar
14    </button>
15 </div>
```



Tip

Si la historia es tomada de la base de datos, tendrá un id.

Historias				
				Guest
localhost:3000				...
12	You can't win, Darth. Strike me down, and I will become more powerful than you could possibly imagine.	Obi Wan Kenobi	9	Votar Editar Eliminar
13	Aren't you a little short for a storm trooper?	Princess Leia	509	Votar Editar Eliminar
14	Master Kenobi, you disappoint me. Yoda holds you in such high esteem. Surely you can do better!	Count Dooku	622	Votar Editar Eliminar
15	Obi-Wan has taught you well.	Darth Vader	18	Votar Editar Eliminar
16	Traveling through hyperspace ain't like dusting crops, farm boy.	Han Solo	166	Votar Editar Eliminar
17	There's always a bigger fish.	Qui-Gon Jinn	584	Votar Editar Eliminar
18	Oh my. Space travel sound rather perilous. I can assure you they will never get me on one of those dreadful Star Ships.	C-3PO	742	Votar Editar Eliminar
19	Luke, I don't want to lose you to the Emperor the way I lost Vader.	Obi Wan Kenobi	746	Votar Editar Eliminar
20	Remember...the Force will be with you, always.	Obi Wan Kenobi	146	Votar Editar Eliminar
Una nueva historia		Autor	0	Guardar Nueva Historia Cancelar
Aquí está una lista de todas sus historias. ¿Aregar una nueva?				

añadiendo una nueva historia

Ahí lo tenemos. No fue tan difícil, ¿verdad?.

Luego de finalizar esta parte, al probar nuestra aplicación encontramos otro error. Luego de crear, guardar y tratar de editar una historia, vemos que el botón dice “Guardar nueva Historia” en lugar de ¡“Actualizar Historia”!. Eso es por que luego de enviarla no estamos obteniendo la historia recién creada desde el servidor, después que la enviamos, y por tanto aún no tiene un id. Para solucionar este problema, podemos obtener las historias nuevamente desde el servidor, justo después de almacenar una nueva en la base de datos.

Dado que no me gusta repetir mi código, extraeré el procedimiento de obtención a un método llamado `fetchStories()`. Luego de eso, puedo usar este método para obtener las historias en cualquier momento.

El método fetchStories

```
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       this.$http.get('/api/stories')
20         .then(function (response) {
21           var storiesReady = response.data.map(function(story){
22             story.editing = false
23             return story
24           })
25           Vue.set(vm, 'stories', storiesReady)
26           // or: vm.stories = storiesReady
27         });
28       },
29     }
30   }
31});
```

En nuestro caso, llamaremos a `fetchStories()` dentro de la función callback de la solicitud POST.

```

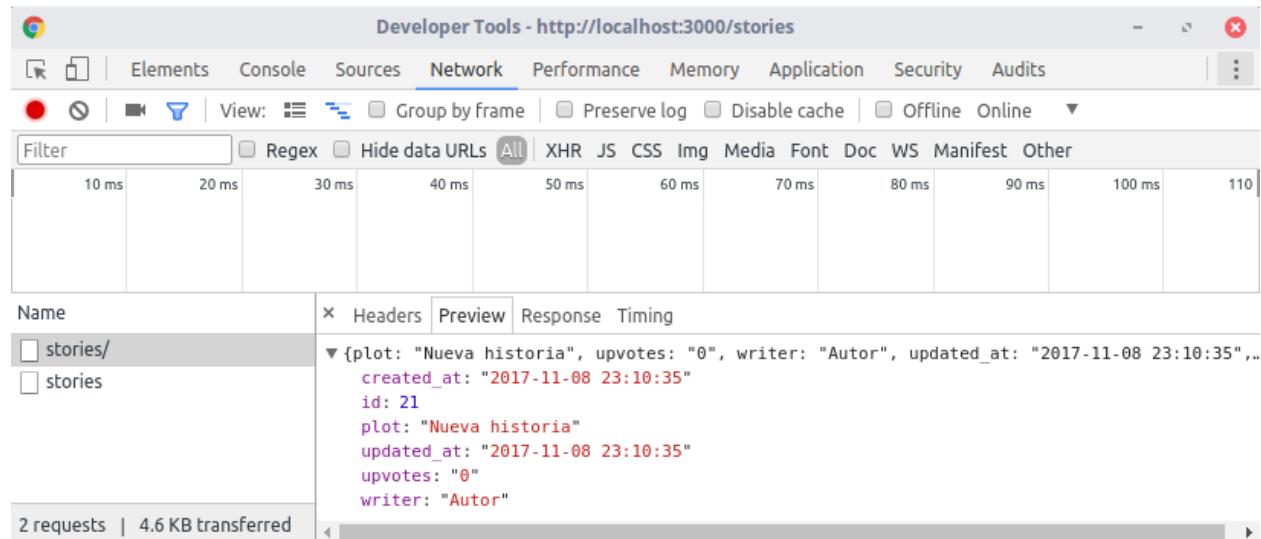
1 Vue.component('story', {
2   ...
3   methods: {
4     ...
5     storeStory: function(story){
6       this.$http.post('/api/stories/', story).then(function() {
7         story.editing = false;
8         vm.fetchStories();
9       });
10      },
11    }
12   ...
13 })

```

¡Eso es todo! Ahora podemos crear y editar cualquier historia que queramos.

Guardar y Actualizar

Una mejor forma de solucionar el problema anterior, es obtener solo la *historia* recién creada desde la base de datos, en lugar de obtener y reescribir todas las *historias*. Si revisas la respuesta del servidor, para la solicitud POST, verás que devuelve la *historia* creada junto con su *id*.



The screenshot shows the Network tab of the Google Chrome Developer Tools. A single request to `/api/stories/` is listed, showing a response time of 110ms. The response body is expanded to show a JSON object representing a new story:

```

{
  "plot": "Nueva historia",
  "upvotes": "0",
  "writer": "Autor",
  "updated_at": "2017-11-08 23:10:35",
  "created_at": "2017-11-08 23:10:35",
  "id": 21,
  "plot": "Nueva historia",
  "updated_at": "2017-11-08 23:10:35",
  "upvotes": "0",
  "writer": "Autor"
}

```

Below the response, it says "2 requests | 4.6 KB transferred".

Respuesta del servidor después de crear una nueva historia

Lo único que tenemos que hacer, es actualizar nuestra *historia* para que coincida con la del servidor. Así que, estableceremos el id de la historia con el valor del atributo id de la respuesta. Haremos esto dentro de la función callback de POST.

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

Yo uso `Vue.set(story, 'id', response.data.id)` en lugar de `story.id = response.data.id` porque dentro de nuestra tabla mostramos el `id` de cada historia. Dado que la nueva historia no tenía `id`, cuando es insertada al arreglo `stories` el DOM no se actualizará cuando el `id` cambie, así que no seremos capaces de ver el nuevo `id`.



Tip

Cuando estás añadiendo una **nueva propiedad que no estaba presente cuando los datos fueron observados**, Vue.js no puede detectar la adición de la propiedad. Así que, si necesitas agregar o eliminar propiedades en tiempo de ejecución usa los métodos globales `Vue.set` o `Vue.delete`.

Archivo JavaScript

Como habrás notado, nuestro código está empezando a crecer considerablemente. A medida que nuestro proyecto crece se está volviendo más difícil de mantener. Para empezar, separaremos el código *JavaScript* del *HTML*. Crearé un archivo llamado `app.js` y lo guardaré en el directorio `js`.

Todo el código JavaScript, debería residir dentro de ese archivo de ahora en adelante. Para incluir el script recién creado en cualquier página HTML simplemente tienes que agregar esta etiqueta:

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

y estás listo para continuar.

Código Fuente

Deabajo está todo el código fuente del ejemplo anterior *Administrando Historias*. Si has descargado nuestro repositorio, te sugiero que abras tus archivos locales con tu editor de texto favorito, porque el código es bastante grande. Los archivos están localizados en `~/themajestyofvuejs2/apis/stories/public`.

Si no has descargado el repositorio todavía puedes ver los archivos `stories.html`⁷⁵ y `app.js`⁷⁶ en *github*.

`stories.html`

```
1 <html lang="en">
2   <head>
3     <title>Historias</title>
4     <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.2/css\bootstrap.min.css">
5   </head>
6
7
8   <body>
9     <main>
10       <div class="container">
11         <h1>Historias</h1>
12         <div id="v-app">
13           <table class="table table-striped">
14             <tr>
15               <th>#</th>
16               <th>Sinopsis</th>
17               <th>Escritor</th>
18               <th>Votos a favor</th>
19               <th>Acciones</th>
20             </tr>
21             <tr v-for="story in stories" is="story" :story="story"></tr>
22           </table>
23           <p class="lead">Aquí está una lista de todas sus historias.
24             <button @click="createStory()" class="btn btn-primary">
25               ¿Añadir una nueva?
26             </button>
27           </p>
28           <pre>{{ $data }}</pre>
29         </div>
30       </div>
31     </main>
```

⁷⁵<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/apis/stories/public/stories.html>

⁷⁶<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/apis/stories/public/js/app.js>

```
32 <template id="template-story-raw">
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     <!-- En otras ocasiones mostrar 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     <!-- En otras ocasiones mostrar 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         Votar
64       </button>
65       <button @click="editStory(story)" class="btn btn-default">
66         Editar
67       </button>
68       <button @click="deleteStory(story)"
69         class="btn btn-danger">
70         Eliminar
71       </button>
72     </div>
73     <div class="btn-group" v-else>
74       <!-- Si la historia es tomada desde la base de datos entonces tendrá \
```

```
75  un id -->
76      <button v-if="story.id"
77          class="btn btn-primary"
78          @click="updateStory(story)">
79              Actualizar historia
80      </button>
81
82      <!-- Si la historia es nueva queremos guardarla -->
83      <button v-else class="btn btn-success"
84          @click="storeStory(story)">
85          Guardar nueva historia
86      </button>
87
88      <!-- Siempre mostrar cancelar -->
89      <button @click="story.editing=false"
90          class="btn btn-default">
91          Cancelar
92      </button>
93  </div>
94  </td>
95 </tr>
96 </template>
97 <script src="http://cdnjs.cloudflare.com/ajax/libs/vue/2.0.1/vue.js"></script>
98 <script src="https://cdnjs.cloudflare.com/ajax/libs/vue-resource/0.7.0/vue-resource.\
99 js"></script>
100 <script src='/js/app.js' type="text/javascript"></script>
101 </body>
102 </html>
```

```
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             this.$http.delete('/api/stories/' + story.id)
9         },
10        upvoteStory: function (story) {
11            story.upvotes++;
12            this.$http.patch('/api/stories/' + story.id, story)
13        },
14    }
15 })
```

```
14     editStory: function (story) {
15         story.editing = true;
16     },
17     updateStory: function (story) {
18         this.$http.patch('/api/stories/' + story.id, story)
19         // Establecer editing a false para mostrar las acciones de nuevo y ocultar los campos
20         story.editing = false;
21     },
22     storeStory: function (story) {
23         this.$http.post('/api/stories/', story)
24             .then(function (response) {
25                 // Luego de que la nueva es guardada en la base de datos
26                 // obtener todas las historias de nuevo con
27                 // vm.fetchStories();
28                 // O mejor, actualizar el id de la historia creada
29                 Vue.set(story, 'id', response.data.id);
30
31                 // Establecer editing a false para mostrar las acciones de nuevo y ocultar los campos
32                 story.editing = false;
33             });
34     },
35 },
36 },
37 }
38 })
39
40 new Vue({
41     el: '#v-app',
42     data: {
43         stories: [],
44         story: {}
45     },
46     mounted: function () {
47         this.fetchStories()
48     },
49     methods: {
50         createStory: function () {
51             var newStory = {
52                 plot: "",
53                 upvotes: 0,
54                 editing: true
55             };
56             this.stories.push(newStory);
57         }
58     }
59 })
```

```
57      },
58      fetchStories: function () {
59          console.log('fetsi')
60          var vm = this;
61          this.$http.get('/api/stories')
62              .then(function (response) {
63                  // Establecer los datos en vm
64                  var storiesReady = response.data.map(function (story) {
65                      story.editing = false;
66                      return story
67                  })
68                  Vue.set(vm, 'stories', storiesReady)
69              });
70      },
71  });
72});
```



Código de Ejemplo

Puedes encontrar el código de ejemplo de este capítulo en [GitHub⁷⁷](#)

Ejercicios

Para familiarizarte con hacer solicitudes web y manejar respuestas, deberías repetir lo que hicimos en este capítulo.

Lo que tienes que hacer es consumir una API para:

- crear una tabla y **mostrar** películas existentes
- **modificar** películas existentes
- **guardar** nuevas películas en la base de datos
- **eliminar** películas de la base de datos

He preparado la **base de datos** y la **API** para ti. Solo tienes que escribir HTML y JavaScript.

Prefacio

Si has seguido las instrucciones del [Capítulo 10](#), abre tu terminal y ejecuta:

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

Si no lo has hecho todavía, debes ejecutar esto:

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

Ahora tienes una **base de datos llena de buenas películas** junto con un servidor completamente funcional ejecutándose en ***http://localhost:3000***.

Para asegurarte que todo está funcionando bien navega a ***http://localhost:3000/api/movies***, deberías ver un arreglo de películas en formato JSON.

⁷⁷<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter12>

Los Endpoints de la API

Los Endpoints de la API que vas a necesitar son:

Método HTTP	URI	Acción
GET/HEAD	api/movies	Obtiene todas las películas
GET/HEAD	api/movies/{id}	Obtiene la película especificada
POST	api/movies	Crea una nueva película
PUT/PATCH	api/movies/{id}	Actualiza una película existente
DELETE	api/movies/{id}	Elimina la película especificada

Tu Código

Pon tu código HTML dentro del archivo `~/themajestyofvuejs2/apis/movies/public/movies.html` que hemos creado. También puedes colocar tu código JavaScript ahí o dentro de `js/app.js`.

Para ver tu trabajo visita `http://localhost:3000/movies.html` en tu navegador.

Espero que disfrutes este ejercicio, ¡Buena suerte!

#	Titulo	Director	Acciones
1	One Flew Over the Cuckoo's Nest	Milos Forman	Editar Eliminar
2	The Exorcist	William Friedkin	Editar Eliminar
3	What's Eating Gilbert Grape	Lasse Hallström	Editar Eliminar
4	Spirited Away	Hayao Miyazaki	Editar Eliminar
5	Aladdin	Ron Clements	Editar Eliminar
6	Star Wars: Episode VI - Return of the Jedi	George Lucas	Editar Eliminar
7	The Bourne Ultimatum	Paul Greengrass	Editar Eliminar
8	Scarface	Brian De Palma	Editar Eliminar
20	The Lord of the Rings: The Fellowship of the Ring	Peter Jackson	Editar Eliminar

Aquí hay una lista de todas tus películas. [¿Agregar una nueva?](#)

Pantalla de Ejemplo



Possible Solución

Puedes encontrar una posible solución para este ejercicio [aquí⁷⁸](#).

⁷⁸<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/homework/Chapter12>

Paginación

En el capítulo anterior logramos obtener todos los registros de la base de datos y mostrarlos dentro de una tabla. Esa implementación está bien para cuando tenemos solo algunos registros, pero en el mundo real, cuando tienes que trabajar con miles o millones de registros, simplemente no puedes obtenerlos y colocarlos dentro de un *array*. Si haces eso, tu navegador no va a estar feliz de tener que cargar esa cantidad de datos, e incluso si logras hacerlo te aseguro que a ningún usuario le gusta lidiar con una tabla que contiene 100.000 filas.

Info

La paginación es usada de alguna forma en casi todas las aplicaciones web para dividir datos retornados y mostrarlos en múltiples páginas. La paginación también incluye la lógica para preparar y mostrar los enlaces a las diversas páginas, y puede ser manejada en el lado del cliente o servidor. La paginación en el lado del servidor es más común.

En situaciones como esta, (esperamos que) los desarrolladores que diseñaron la API dividen los datos retornados en varias páginas.

La respuesta HTTP contendrá algunos **metadatos** sencillos junto con los datos principales para infomarte sobre el *total* de elementos, elementos *por página* etc. Para ayudarte a ir a través de las páginas, proporcionará información como **la página actual**, **la página siguiente** y **la página anterior**.

Respuesta de Ejemplo con datos Paginados

```
{  
  "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": [...]}
```

Los **metadatos** de la paginación también pueden estar dentro de un objeto junto a **data** o en cualquier lugar que los desarrolladores de la API hayan decidido.

Ejemplo de respuesta con datos paginados

```
{  
    "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,  
    }  
}
```

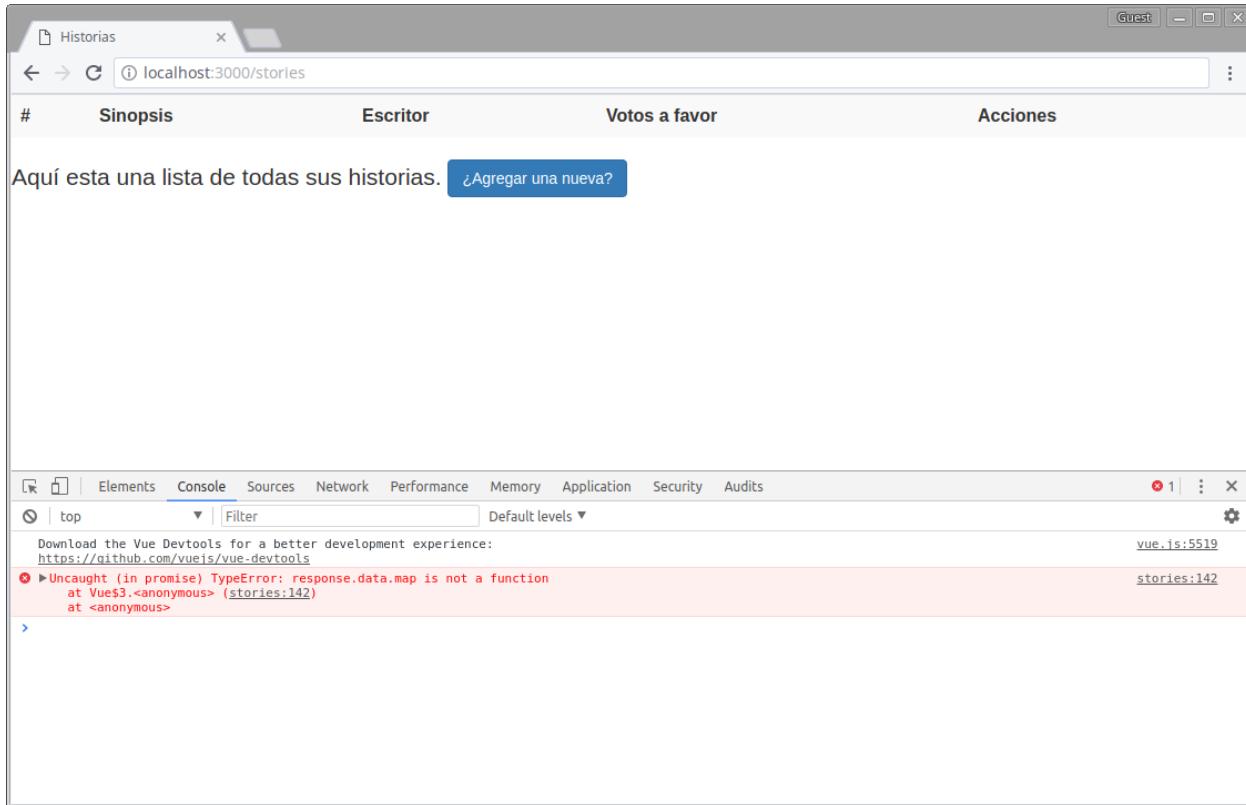
Implementación

Vamos a continuar trabajando con los ejemplos de *story* del capítulo anterior usando la API paginada mejorada ligeramente. Así que, vamos a modificar el código para ser capaces de acceder y usar esos datos.

Si echas un vistazo al código del [ejemplo anterior](#) verás que nuestro método **fetchStories** es similar a esto:

```
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)  
                });  
        },  
        ...  
    }  
});
```

Si abrimos nuestro archivo HTML en el navegador, como habrás podido adivinar, nuestra tabla no se renderiza correctamente.

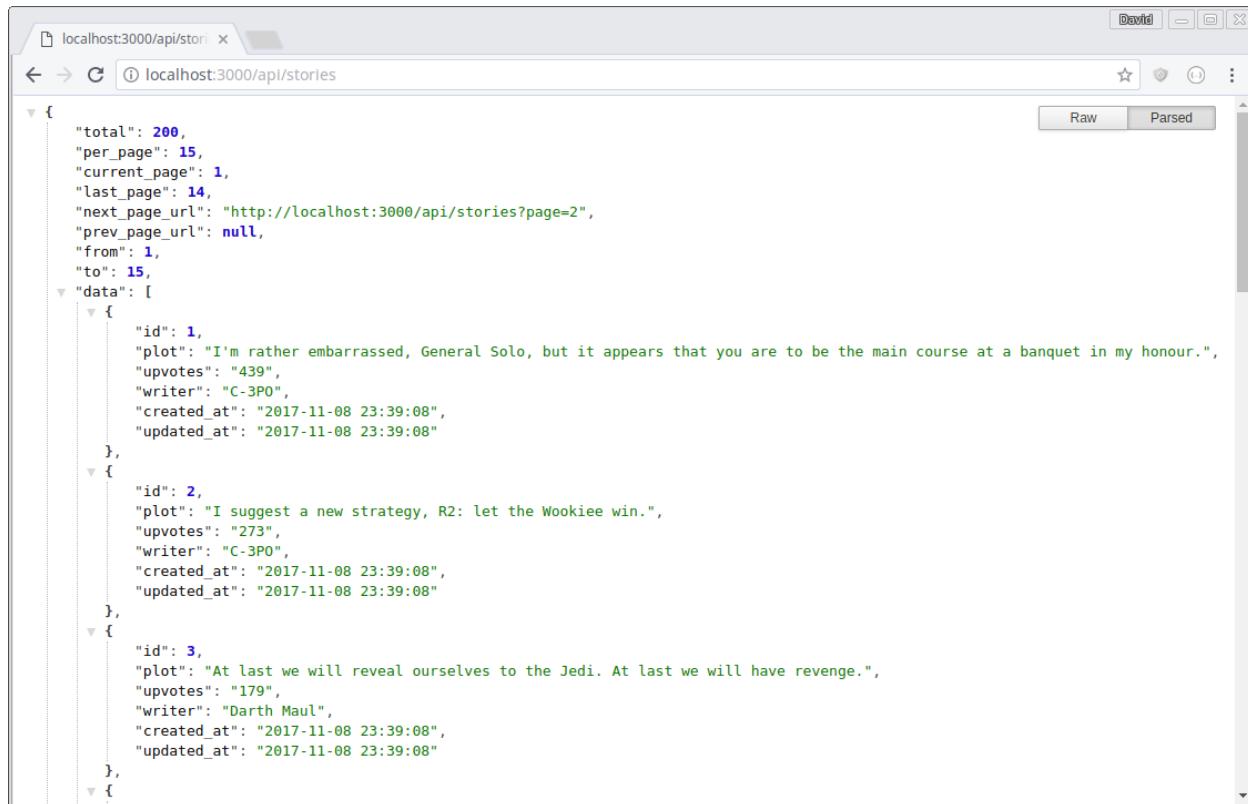


Las historias del ejemplo no se están mostrando

Esto sucede porque las historias ahora son retornadas dentro de un arreglo llamado `data`. Para arreglar esto, tenemos que cambiar `response.data` a `response.data.data` (Sé que esto es un poco raro, pero...).

Excepto por el arreglo `stories`, también queremos guardar los datos de la paginación dentro de un objeto a fin de implementar fácilmente la funcionalidad de paginación.

Para encontrar como podemos acceder a esos datos, echemos un vistazo a la respuesta del servidor.



```
{  
  "total": 200,  
  "per_page": 15,  
  "current_page": 15,  
  "last_page": 14,  
  "next_page_url": "http://localhost:3000/api/stories?page=2",  
  "prev_page_url": null,  
  "from": 1,  
  "to": 15,  
  "data": [  
    {  
      "id": 1,  
      "plot": "I'm rather embarrassed, General Solo, but it appears that you are to be the main course at a banquet in my honour.",  
      "upvotes": "439",  
      "writer": "C-3PO",  
      "created_at": "2017-11-08 23:39:08",  
      "updated_at": "2017-11-08 23:39:08"  
    },  
    {  
      "id": 2,  
      "plot": "I suggest a new strategy, R2: let the Wookiee win.",  
      "upvotes": "273",  
      "writer": "C-3PO",  
      "created_at": "2017-11-08 23:39:08",  
      "updated_at": "2017-11-08 23:39:08"  
    },  
    {  
      "id": 3,  
      "plot": "At last we will reveal ourselves to the Jedi. At last we will have revenge.",  
      "upvotes": "179",  
      "writer": "Darth Maul",  
      "created_at": "2017-11-08 23:39:08",  
      "updated_at": "2017-11-08 23:39:08"  
    }  
  ]  
}
```

Respuesta del servidor

Para empezar, no necesitamos todos esos datos. Así que nos quedaremos con `current_page`, `last_page`, `next_page_url` y `prev_page_url`.

Nuestro objeto de paginación será algo como esto:

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

Modifiquemos nuestro método `fetchStories` para actualizar el objeto `pagination` cada vez que obtiene historias de la base de datos.

```
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  
          })  
          // aquí usamos 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)  
        });  
    },  
    ...  
  }  
});
```

Enlaces de Paginación

Por ahora tenemos nuestro objeto `pagination` pero siempre obtenemos la primera página de `stories` ya que estamos haciendo una solicitud HTTP GET a `/api/stories`. Tenemos que cambiar la página solicitada en base a la interacción del usuario (página anterior, página siguiente).

Primero actualizaremos el método `fetchStories` para aceptar un argumento con la página deseada. Si ningún argumento es pasado obtendrá la primera página. También crearé un nuevo método, `makePagination`, para hacer el código más limpio.

```
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){  
      // aquí usamos 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)  
    }  
    ...  
  }  
}
```

Ahora que nuestro método está listo, necesitamos una forma de llamarlo apropiadamente. Agregaremos 2 botones, uno para la página siguiente y otro para la página anterior, en la parte superior de nuestro `div #app`. Cada botón, al ser presionado, llamará al método `fetchStories` pasando la url de la página correspondiente.

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

¡Ceremonial! Si tratas de presionar los botones verás que funcionan como esperamos. Tenemos nuestra paginación en un abrir y cerrar de ojos. Aunque sería útil informar al usuario sobre qué página esta viendo actualmente, y el número total de páginas. También, podemos inhabilitar el botón **Anterior** cuando el usuario está en la primera página, y el botón **Siguiente** en la última, correspondientemente.

```
1 <div class="pagination">
2   <button @click="fetchStories(pagination.prev_page_url)"
3     :disabled="!pagination.prev_page_url"
4   >
5     Anterior
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    Siguiente
12  </button>
13 </div>
```

revenge.				
8	It's a trap!	Admiral Ackbar	451	Votar Editar Eliminar
9	I sense great fear in you, Skywalker. You have hate... you have anger... but you don't use them.	Count Dooku	435	Votar Editar Eliminar
10	Mmm. Lost a planet, Master Obi-Wan has. How embarrassing.	Yoda	195	Votar Editar Eliminar
11	Laugh it up, Fuzz ball.	Han Solo	372	Votar Editar Eliminar
12	Luke, you can destroy the Emperor. He has foreseen this. It is your destiny. Join me, and together we can rule the galaxy as father and son.	Darth Vader	400	Votar Editar Eliminar
13	Luke, you can destroy the Emperor. He has foreseen this. It is your destiny. Join me, and together we can rule the galaxy as father and son.	Darth Vader	378	Votar Editar Eliminar
14	You may dispense with the pleasantries, Commander. I am here to put you back on schedule.	Darth Vader	403	Votar Editar Eliminar
15	Now, young Skywalker... you will die.	The Emperor	779	Votar Editar Eliminar

Aquí está una lista de todas sus historias. [¿Aregar una nueva?](#)

Botón de previa página inhabilitado



Código de Ejemplo

Puedes encontrar el código de ejemplo para este capítulo en [GitHub⁷⁹](#).

Ejercicios

No hay nada en particular para hacer como ejercicio en este capítulo. Si realmente deseas trabajar en este ejemplo, te proporcionaré la API paginada.

Si has resuelto el ejercicio del capítulo anterior (descargas el código e iniciates el servidor) estás sólo a unos clics de distancia. Si no lo has hecho, sólo sigue [estas instrucciones](#).

La API paginada se encuentra dentro del directorio `'~/themajestyofvuejs2/apis/pagination/stories'`.

El archivo HTML está en el directorio `'~/themajestyofvuejs2/apis/pagination/stories/public'`.

Si sólo quieres ver el código final puedes echar un vistazo a los archivos en [GitHub⁸⁰](#).

⁷⁹<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter13>

⁸⁰<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/apis/pagination/stories/public>

Resumen de axios

Retiro de vue-resource

Como probablemente sepas, Vue tenía su propio cliente HTTP, pero el equipo principal decidió retirarlo de su status de recomendación oficial, anunciándolo mediante el post de Evan You “[Retiring vue-resource⁸¹](#)”, explicando las diversas razones detrás de esto.

Vamos a hacer de nuevo todas las solicitudes web que hicimos arriba, esta vez usando [axios⁸²](#), pero no sin antes mostrar un ejemplo del plugin vue-resource. *Este plugin proporciona servicios para realizar solicitudes web y manejar respuestas usando XMLHttpRequest o JSONP.* De esta forma puedes ver las diferencias y decidir cual es mejor para tus necesidades. También *jQuery* es útil, pero si lo estás utilizando sólo para realizar solicitudes AJAX, puedes considerar removerlo.

Aquí puedes encontrar instrucciones para la instalación y documentación sobre [vue-resource⁸³](#). Como siempre, vamos a enlazarlo desde una página [cdn⁸⁴](#).

Para obtener datos desde un servidor, podemos usar el método `$http` de vue-resource con la siguiente sintaxis:

```
mounted: function() {
  // Solicitud GET
  this.$http({url: '/someUrl', method: 'GET'})
    .then(function (response) {
      // función de retorno para éxito
    }, function (response) {
      // función de retorno para error
    });
}
```



Info

Una instancia de Vue provee la función `this.$http(options)` que recibe un objeto options para generar una solicitud HTTP y devuelve una promesa. También, la instancia de Vue será automáticamente enlazada a `this` en todos las funciones de retorno.

En lugar de pasar el método como opción, hay abreviaciones de métodos disponibles para todos los tipos de solicitudes.

⁸¹<https://medium.com/the-vue-point/retiring-vue-resource-871a82880af4#.lew43e17f>

⁸²<https://github.com/mzabriskie/axios>

⁸³<https://github.com/vuejs/vue-resource>

⁸⁴<https://cdnjs.com/libraries/vue-resource>

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);
```

De acuerdo al post de Evan *Está totalmente bien seguir usándolo si estás feliz con él y eres libre de elegir lo que quieras (incluso solo \$.ajax)*, pero ya que axios es la última recomendación, podemos seguir adelante con la integración de axios.

Integrando axios

Axios⁸⁵ es un cliente HTTP basado en promesas para el navegador y Node.js. En adición, cubre casi todo lo que vue-resource proporciona con una API muy similar, es universal, soporta cancelación y tiene definiciones TypeScript⁸⁶.

Realizando una solicitud GET con axios:

```
// Hacer una solicitud para un usuario con un ID dado axios.get('/user?ID=12345') .then(function (response) { console.log(response); }) .catch(function (error) { console.log(error); });
```



Tip

Si te gustaría seguir utilizando `this.$http` como en vue-resource, puedes simplemente establecer `Vue.prototype.$http = axios` y tendrás axios sin cambiar mucho más.

Migración

Es hora de usar axios en nuestro ejemplo. Primero que nada, tenemos que incluirlo. Añadiremos esta línea a nuestro archivo HTML.

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

Para obtener las historias, realizaremos una solicitud GET en el formato correspondiente:

⁸⁵<https://github.com/mzabriskie/axios>

⁸⁶<http://www.typescriptlang.org/>

```
mounted: function() {
  // Solicitud GET
  axios.get('/api/stories')
    .then(function (response) {
      Vue.set(vm, 'stories', response.data)
      // O como hicimos anteriormente
      // vm.stories = response.data
    })
}
```

Nuestra lista de historias viene sin ningún problema, al usar la sintaxis de arriba.

Por conveniencia han sido proporcionados alias para todos los métodos de solicitud soportados.

Alias para los métodos de solicitudes

```
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]])
```



Al usar los alias de métodos, las propiedades url, method, y data no necesitan ser especificadas en la configuración.

Sigamos adelante, con las solicitudes DELETE y PATCH, usando los alias de métodos.

PATCH request

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

DELETE request

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

Hemos reemplazado los métodos AJAX con estos en un abrir y cerrar de ojos.

 **Info**

Dado que el componente *story* no tiene acceso al array *stories*, accedemos al arreglo usando `this.$parent.stories`. También podríamos usar `vm.stories` o emitir un evento para actualizar el arreglo dentro de la instancia padre de Vue.

Mejorando la Funcionalidad

Deberíamos agregar un par de características más, para hacer nuestra lista de historias más limpia. Podemos darle al usuario la habilidad de **cambiar la trama de una historia, su autor y también crear nuevas historias**.

Editar Historias

Comencemos con la primera tarea y demos al usuario algunos campos para manipular los atributos de la historia. Dos campos enlazados deben hacer el trabajo, pero debemos mostrarlos **sólo** cuando el usuario está **editando** una historia. Se parece al tipo de trabajo que hicimos en capítulos anteriores.

Para definir si una historia está en **estado de edición** usaremos una propiedad, `editing`, que se volverá verdadera cuando el usuario presione el botón *Editar*.

```
1 <td>  
2   <!-- si estás editando la historia, muestra el campo de texto para plot -->  
3   <input v-if="story.editing" v-model="story.plot" class="form-control">  
4   </input>  
5   <!-- en otras ocasiones, muestra plot -->  
6   <span v-else>  
7     {{story.plot}}  
8   </span>  
9 </td>  
10 <td>
```

```

11      <!-- si estás editando la historia, muestra el campo de texto para writer -->
12      <input v-if="story.editing" v-model="story.writer" class="form-control">
13      </input>
14      <!-- en otras ocasiones, muestra 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              Votar
26          </button>
27          <button @click="editStory(story)" class="btn btn-default">
28              Editar
29          </button>
30          <button @click="deleteStory(story)" class="btn btn-danger">
31              Eliminar
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 })

```

Esta es nuestra tabla actualizada, con dos campos nuevos y un botón. Usamos la función `editStory` para establecer `story.editing` a `true`, así `v-if` traerá los campos para editar la historia y ocultará los botones *Votar* y *Eliminar*. Sin embargo, de esta forma no funcionará. Parece que el DOM no se está actualizando luego de establecer `story.editing` a `true`. Pero, ¿por qué sucede esto?

Resulta que, de acuerdo a [este post en el blog de Vue.js⁸⁷](#), que cuando estás añadiendo una nueva

⁸⁷<http://vuejs.org/2016/02/06/common-gotchas/>

propiedad que no estaba presente cuando los datos fueron observados el DOM no se actualizará. La mejor práctica es siempre declarar por adelantado las propiedades que deben ser reactivas. En casos donde absolutamente necesitas agregar o eliminar propiedades en tiempo de ejecución, usa los métodos globales `Vue.set` o `Vue.delete`.

Por esta razón tenemos que inicializar el atributo `story.editing` a `false` en cada historia, justo después de recibir las historias desde el servidor.

Para hacer esto vamos a usar el método `.map()` de Javascript dentro de la función de retorno para éxito de la solicitud GET.

```
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

El método `.map()` llama, a una función de retorno definida sobre cada elemento de un arreglo, y devuelve un arreglo que contiene los resultados. Puedes encontrar más información sobre el método `.map()` y su sintaxis [aqui⁸⁸](#).

Esta función agrega el atributo `editing` a cada objeto de la historia y luego devuelve la historia actualizada. La nueva variable, `storiesReady`, es un arreglo que contiene nuestro arreglo actualizado con el nuevo atributo en `true`.

Cuando la historia está bajo edición le daremos al usuario dos opciones: actualizar la historia con los nuevos valores o cancelar la edición.

⁸⁸[https://msdn.microsoft.com/en-us/library/ff679976\(v=vs.94\).aspx](https://msdn.microsoft.com/en-us/library/ff679976(v=vs.94).aspx)

9	I suggest a new strategy, R2: let me wookiee win.	C-3PO	411	Votar	Editar	Eliminar
10	You can't win, Darth. Strike me down, and I will become more powerful than you could possibly imagine.	Obi Wan Kenobi	320	Votar	Editar	Eliminar
11	Great shot kid, that was one in a million.	Han Solo	11	Votar	Editar	Eliminar
12	You can't win, Darth. Strike me down, and I will become more powerful than you could possibly imagine.	Obi Wan Kenobi	9	Votar	Editar	Eliminar
13	Aren't you a little short for a storm trooper?	Princess Leia	509	Votar	Editar	Eliminar
14	Master Kenobi, you disappoint me. Yoda holds you in such high esteem. Surely you can do better!	Count Dooku	622	Votar	Editar	Eliminar
15	Obi-Wan has taught you well.	Darth Vader	18			
16	Traveling through hyperspace ain't like dusting crops, farm boy.	Han Solo	166	Votar	Editar	Eliminar
17	There's always a bigger fish.	Qui-Gon Jinn	584	Votar	Editar	Eliminar
18	Oh my. Space travel sound rather perilous. I can assure you they will never get me on one of those dreadful Star Ships.	C-3PO	742	Votar	Editar	Eliminar
19	Luke, I don't want to lose you to the Emperor the way I lost Vader.	Obi Wan Kenobi	746	Votar	Editar	Eliminar
20	Remember...the Force will be with you, always.	Obi Wan Kenobi	146	Votar	Editar	Eliminar

Campos de entrada de formularios para la edición de la historia

Así que continuemos y agreguemos dos botones, que deberían ser mostrados sólo cuando el usuario esté editando la historia. Adicionalmente, un nuevo método llamado `updateStory` será creado. Este método va a actualizar la historia que se está editando actualmente, luego de que el botón *Actualizar Historia* sea presionado.

```
<!-- Si la historia esta bajo edición, muestra este grupo de botones -->
<div class="btn-group" v-else>
  <button @click="updateStory(story)" class="btn btn-primary">
    Actualizar Historia
  </button>
  <button @click="story.editing=false" class="btn btn-default">
    Cancelar
  </button>
</div>
```

```

1 Vue.component('story', {
2   ...
3   methods: {
4     ...
5       updateStory: function (story) {
6         axios.patch('/api/stories/' + story.id, story)
7           //Establecer editing a false para mostrar las acciones nuevamente y ocultar los campos de entrada
8         story.editing = false;
9       },
10    }
11  }
12  ...
13})

```

#	Sinopsis	Escritor	Votos a favor	Acciones
1	One thing's for sure, we're all going to be a lot thinner.	Han Solo	166	<button>Votar</button> <button>Editar</button> <button>Eliminar</button>
2	The Force is strong with you. A powerful Sith you will become. Henceforth, you shall be known as Darth... Vader.	Darth Sidious	421	<button>Votar</button> <button>Editar</button> <button>Eliminar</button>
3	Great, kid. Don't get cocky	Han Solo	176	<button>Actualizar Historia</button> <button>Cancelar</button>
4	Don't call me a mindless philosopher, you overweight glob of grease.	C-3PO	294	<button>Votar</button> <button>Editar</button> <button>Eliminar</button>
5	No. I am your Father.	Darth Vader	561	<button>Votar</button> <button>Editar</button> <button>Eliminar</button>
6	The force is strong with this one.	Darth Vader	160	<button>Votar</button> <button>Editar</button> <button>Eliminar</button>
7	You don't know how hard I found it, signing the order to terminate your life	Governor 'Grand Moff' Tarkin	237	<button>Votar</button> <button>Editar</button> <button>Eliminar</button>
8	Oh my. Space travel sound rather perilous. I can assure you they will never get me on one of those dreadful Star Ships.	C-3PO	676	<button>Votar</button> <button>Editar</button> <button>Eliminar</button>
9	I suggest a new strategy, R2: let the Wookiee win.	C-3PO	477	<button>Votar</button> <button>Editar</button> <button>Eliminar</button>
10	You can't win, Darth. Strike me down, and I will become more powerful than you could possibly imagine.	Obi Wan Kenobi	320	<button>Votar</button> <button>Editar</button> <button>Eliminar</button>

Acciones de la actualización de historia

Aquí está, funcionando. Luego de que la solicitud PATCH es finalizada exitosamente tenemos que establecer `story.editing` de nuevo a `false`, con el fin de ocultar los campos y traer de vuelta los botones de acciones.

Crear Nuevas Historias

Ahora para una tarea un poco más complicada, vamos a darle al usuario la habilidad de crear una nueva historia y guardarla en nuestro servidor. Primero, debemos proporcionar campos para que la nueva historia pueda ser escrita. Para hacer que esto suceda, crearemos una historia vacía y la agregaremos al arreglo `stories` usando el método `push()` de JavaScript. Inicializaremos todos los atributos de la historia a *un valor nulo apropiado*, excepto por `editing`. Dado que queremos manipular inmediatamente la nueva historia, la propiedad `editing` será establecida a `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">Aquí está una lista de todas tus historias.
2   <button @click="createStory()" class="btn btn-primary">
3     ¿Añadir una nueva?
4   </button>
5 </p>

```



Info

El método `push()` agrega nuevos elementos al final del arreglo, y devuelve el nuevo tamaño. Puedes encontrar más información sobre el método `push()` y su sintaxis [aquí⁸⁹](#).

Nombramos a la nueva función `createStory` y la colocamos en nuestra instancia de Vue.

Justo debajo de nuestra lista hemos añadido un botón. Cuando el botón es presionado el método `createStory` es invocado. Dado que `newStory.editing` está establecida a `true`, los campos enlazados para *trama* y *escritor* junto con los *botones de edición* están siendo renderizados al instante.

También, el nuevo objeto *story* debe ser enviado al servidor para ser almacenado en la base de datos. Vamos a realizar una solicitud POST dentro de un método llamado `storeStory`.

⁸⁹https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/push

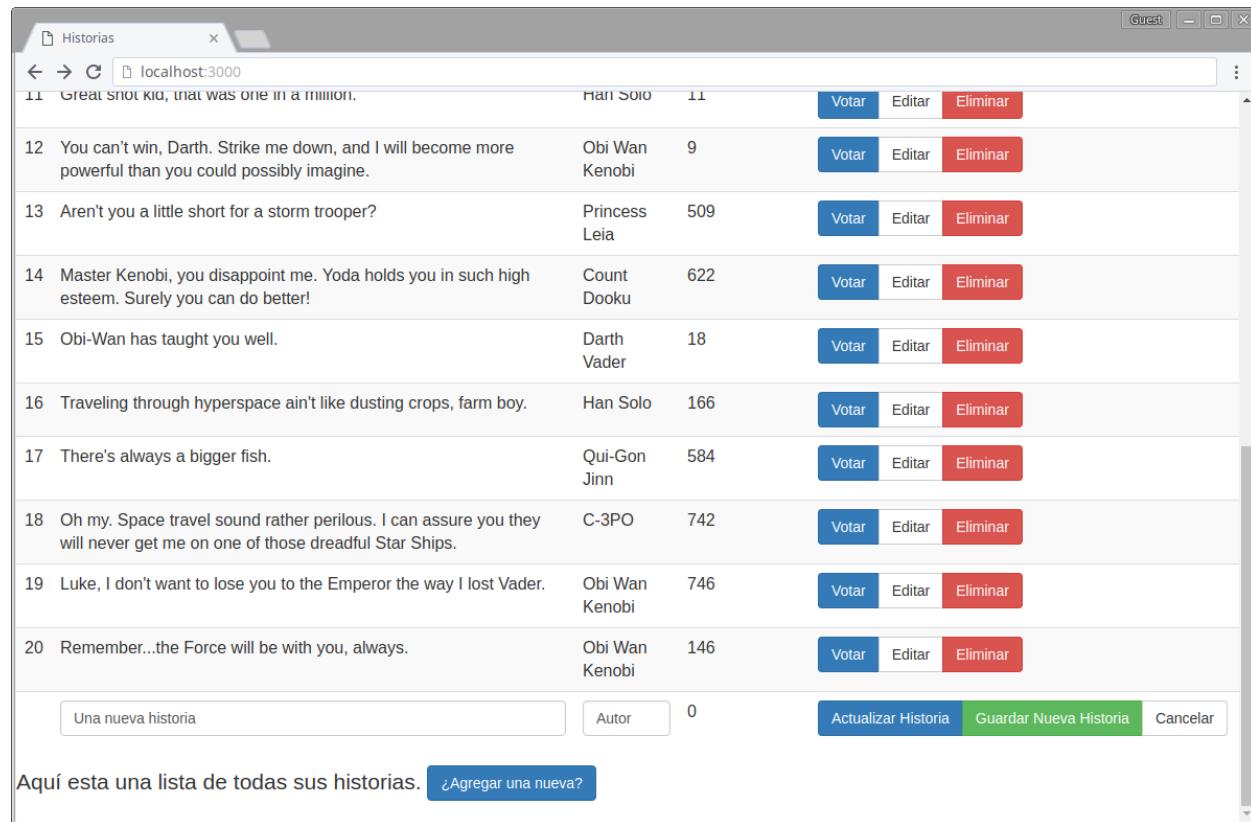
```
1 Vue.component('story', {
2     ...
3     methods: {
4         ...
5         storeStory: function(story){
6             axios.post('/api/stories/', story).then(function () {
7                 story.editing = false;
8             });
9         },
10    }
11    ...
12 })
```

En la función de retorno para éxito hemos establecido editing a `false` a fin de mostrar los botones de *acción* nuevamente y ocultar los campos del formulario y los botones de *edición*. Debajo, vamos a actualizar los *grupos* de botones, de acuerdo con el nuevo método.

```
1 <td>
2 <div class="btn-group" v-if="!story.editing">
3     <button @click="upvoteStory(story)" class="btn btn-primary">
4         Votar
5     </button>
6     <button @click="editStory(story)" class="btn btn-default">
7         Editar
8     </button>
9     <button @click="deleteStory(story)" class="btn btn-danger">
10        Eliminar
11    </button>
12 </div>
13 <div class="btn-group" v-else>
14     <button class="btn btn-primary" @click="updateStory(story)">
15         Actualizar Historia
16     </button>
17     <button class="btn btn-success" @click="storeStory(story)">
18         Guardar Nueva Historia
19     </button>
20     <button @click="story.editing=false" class="btn btn-default">
21         Cancelar
22     </button>
23 </div>
24 </td>
```

Podemos observar un pequeño error en este bloque de código. Cuando estamos en modo de *edición*

(bloque `v-else`) vemos que los botones para actualizar y guardar están siendo mostrados juntos, pero sólo necesitamos uno por cada historia, dado que cada historia será Guardada o Actualizada. No puede hacer ambas cosas. Así que, si la historia es antigua y el usuario está a punto de editarla, necesitamos el botón de actualizar. Por otra parte, si la historia es nueva, necesitamos el botón de guardar.



The screenshot shows a web browser window titled "Historias" with the URL "localhost:3000". The page displays a list of 20 Star Wars quotes, each with its author, upvote count, and three buttons: "Votar" (blue), "Editar" (light blue), and "Eliminar" (red). The quotes are numbered 11 to 20. At the bottom of the list, there is a form with fields for "Una nueva historia" (with placeholder "Autor" and value "0"), and buttons for "Actualizar Historia" (blue), "Guardar Nueva Historia" (green), and "Cancelar" (gray).

Historia	Autor	Votos	Opciones
11 Great shot kid, that was one in a million.	Han Solo	11	Votar Editar Eliminar
12 You can't win, Darth. Strike me down, and I will become more powerful than you could possibly imagine.	Obi Wan Kenobi	9	Votar Editar Eliminar
13 Aren't you a little short for a storm trooper?	Princess Leia	509	Votar Editar Eliminar
14 Master Kenobi, you disappoint me. Yoda holds you in such high esteem. Surely you can do better!	Count Dooku	622	Votar Editar Eliminar
15 Obi-Wan has taught you well.	Darth Vader	18	Votar Editar Eliminar
16 Traveling through hyperspace ain't like dusting crops, farm boy.	Han Solo	166	Votar Editar Eliminar
17 There's always a bigger fish.	Qui-Gon Jinn	584	Votar Editar Eliminar
18 Oh my. Space travel sound rather perilous. I can assure you they will never get me on one of those dreadful Star Ships.	C-3PO	742	Votar Editar Eliminar
19 Luke, I don't want to lose you to the Emperor the way I lost Vader.	Obi Wan Kenobi	746	Votar Editar Eliminar
20 Remember...the Force will be with you, always.	Obi Wan Kenobi	146	Votar Editar Eliminar

Un pequeño error

Para evitar este problema, vamos a reestructurar nuestros botones. El botón Actualizar será mostrado sólo cuando la historia es antigua. En consecuencia, el botón Guardar Nueva Historia será mostrado cuando la historia es nueva.

Te habrás dado cuenta que todas las historias obtenidas desde el servidor tienen un atributo id. Vamos a usar esta observación para definir si una historia es nueva o no.

```
1 <div class="btn-group" v-else>
2     <!-- Si la historia es antigua entonces queremos actualizarla
3     TIP: si la historia es tomada de la base de datos entonces tendrá un id -->
4     <button v-if="story.id" class="btn btn-primary" @click="updateStory(story)">
5         Actualizar Historia
6     </button>
7     <!-- Si la historia es nueva queremos guardarla-->
8     <button v-else class="btn btn-success" @click="storeStory(story)">
9         Guardar Nueva Historia
10    </button>
11    <!-- Siempre mostramos cancelar-->
12    <button @click="story.editing=false" class="btn btn-default">
13        Cancelar
14    </button>
15 </div>
```



Tip

Si la historia es tomada de la base de datos, tendrá un id.

The screenshot shows a web application window titled "Historias" with the URL "localhost:3000". The page displays a table of 20 stories, each with a number, a quote, the author's name, the number of votes, and three buttons: "Votar" (blue), "Editar" (light blue), and "Eliminar" (red). A message at the bottom left says "Aquí está una lista de todas sus historias." and a button "¿Aregar una nueva?" is visible.

12	You can't win, Darth. Strike me down, and I will become more powerful than you could possibly imagine.	Obi Wan Kenobi	9	Votar Editar Eliminar
13	Aren't you a little short for a storm trooper?	Princess Leia	509	Votar Editar Eliminar
14	Master Kenobi, you disappoint me. Yoda holds you in such high esteem. Surely you can do better!	Count Dooku	622	Votar Editar Eliminar
15	Obi-Wan has taught you well.	Darth Vader	18	Votar Editar Eliminar
16	Traveling through hyperspace ain't like dusting crops, farm boy.	Han Solo	166	Votar Editar Eliminar
17	There's always a bigger fish.	Qui-Gon Jinn	584	Votar Editar Eliminar
18	Oh my. Space travel sound rather perilous. I can assure you they will never get me on one of those dreadful Star Ships.	C-3PO	742	Votar Editar Eliminar
19	Luke, I don't want to lose you to the Emperor the way I lost Vader.	Obi Wan Kenobi	746	Votar Editar Eliminar
20	Remember...the Force will be with you, always.	Obi Wan Kenobi	146	Votar Editar Eliminar

añadiendo una nueva historia

Ahí lo tenemos. No fue tan difícil, ¿verdad?

Luego de finalizar esta parte, al probar nuestra aplicación encontramos otro error. Luego de crear, guardar y tratar de editar una historia, vemos qué ¡el botón dice “Guardar Nueva Historia” en lugar de “Actualizar Historia”! Eso es por que no estamos obteniendo la historia recién creada desde el servidor, luego de enviarla, y no tiene un id todavía. Para solucionar este problema, podemos obtener las historias nuevamente desde el servidor, justo después de almacenar una nueva en la base de datos.

Dado que no me gusta repetir mi código, voy a extraer el procedimiento de obtención a un método llamado `fetchStories()`. Luego de eso, puedo usar este método para obtener las historias en cualquier momento.

El método fetchStories

```
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       };
17       this.stories.push(newStory);
18     },
19     fetchStories: function () {
20       var vm = this;
21       axios.get('/api/stories')
22         .then(function (response) {
23           var storiesReady = response.data.map(function (story) {
24             story.editing = false;
25             return story
26           })
27           // vm.stories = storiesReady
28           Vue.set(vm, 'stories', storiesReady)
29           // o: vm.stories = storiesReady
30         });
31     },
32   }
33});
```

En nuestro caso, llamamos a **fetchStories()** dentro de la función de retorno para éxito de la solicitud POST.

```

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 })

```

¡Eso es todo! Ahora podemos crear y editar cualquier historia que queremos.

Guardar y Actualizar

Una mejor forma de solucionar el problema anterior, es obtener solo la *historia* recién creada desde la base de datos, en lugar de obtener y reescribir todas las *historias*. Si revisas la respuesta del servidor, para la solicitud POST, verás que devuelve la *historia* creada junto con su *id*.

The screenshot shows the Network tab of the Chrome DevTools. A single request to '/api/stories/' is listed, showing a response time of 110ms. The response body is expanded to show the following JSON:

```

{
  "plot": "Nueva historia",
  "upvotes": "0",
  "writer": "Autor",
  "updated_at": "2017-11-08 23:10:35",
  "created_at": "2017-11-08 23:10:35",
  "id": 21
}

```

Below the main interface, the status bar indicates "2 requests | 4.6 KB transferred".

Respuesta del servidor después de crear una nueva historia

Lo único que tenemos que hacer, es actualizar nuestra *historia* para que coincida con la del servidor. Así que, estableceremos el *id* de los datos de respuesta al atributo *id* de la historia. Haremos esto dentro de la función de retorno para éxito del POST.

```
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 })
```

Yo uso `Vue.set(story, 'id', response.data.id)` en lugar de `story.id = response.data.id` porque dentro de nuestra tabla mostramos el `id` de cada historia. Dado que la nueva historia no tenía `id`, cuando es insertada al arreglo `stories` el DOM no se actualizará cuando el `id` cambie, así que no seremos capaces de ver el nuevo `id`.



Tip

Cuando estás añadiendo una **nueva propiedad que no estaba presente cuando los datos fueron observados**, Vue.js no puede detectar la adición de la propiedad. Así que, si necesitas agregar o eliminar propiedades en tiempo de ejecución, usa los métodos globales `Vue.set` o `Vue.delete`.

Archivo JavaScript

Como habrás notado, nuestro código está empezando a crecer considerablemente. A medida que nuestro proyecto crece, se vuelve más difícil de mantener. Para empezar, separaremos el código *JavaScript* del *HTML*. Crearé un archivo llamado `app.js` y lo guardaré en el directorio `js`.

Todo el código JavaScript, debería residir dentro de ese archivo de ahora en adelante. Para incluir el script recién creado a cualquier página HTML simplemente tienes que agregar esta etiqueta:

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

¡y estás listo para continuar!

Código Fuente

Debajo está todo el código fuente del ejemplo anterior *Administrando Historias*. Si has descargado nuestro repositorio, te sugiero que abras tus archivos locales con tu editor de texto favorito, porque el código es bastante grande. Los archivos están localizados en `~/themajestyofvuejs2/apis/stories/public`.

Si no has descargado el repositorio, puedes ver los archivos [stories.html⁹⁰](#) y [app.js⁹¹](#) en *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/css\bootstrap.min.css">
5 </head>
6
7
8 <body>
9 <main>
10    <div class="container">
11        <h1>Historias</h1>
12        <div id="v-app">
13            <table class="table table-striped">
14                <tr>
15                    <th>#</th>
16                    <th>Trama</th>
17                    <th>Escritor</th>
18                    <th>Votos</th>
19                    <th>Acciones</th>
20                </tr>
21                <tr v-for="story in stories" is="story" :story="story"></tr>
22            </table>
23            <p class="lead">Aquí está una lista de todas tus historias.
24                <button @click="createStory()" class="btn btn-primary">
25                    ¿Añadir una nueva?
26                </button>
27            </p>
28            <pre>{{ $data }}</pre>
29        </div>
30    </div>
31 </main>
32 <template id="template-story-raw">
```

⁹⁰<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/apis/stories/public/stories.html>

⁹¹<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/apis/stories/public/js/app.js>

```
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              <!--En otras ocasiones muestra 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              <!--En otras ocasiones muestra 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                      Votar
64                  </button>
65                  <button @click="editStory(story)" class="btn btn-default">
66                      Editar
67                  </button>
68                  <button @click="deleteStory(story)"
69                      class="btn btn-danger">
70                      Eliminar
71                  </button>
72              </div>
73              <div class="btn-group" v-else>
74                  <!--Si la historia es tomada desde la base de datos entonces tendrá \
75 un id-->
```

```
76      <button v-if="story.id"
77        class="btn btn-primary"
78        @click="updateStory(story)">
79          Actualizar historia
80      </button>
81
82      <!--Si la historia es nueva queremos guardarla-->
83      <button v-else class="btn btn-success"
84        @click="storeStory(story)">
85          Guardar nueva historia
86      </button>
87
88      <!--Siempre muestra cancelar-->
89      <button @click="story.editing=false"
90        class="btn btn-default">
91          Cancelar
92      </button>
93    </div>
94  </td>
95 </tr>
96 </template>
97 <script src="https://unpkg.com/vue@2.1.10/dist/vue.js"></script>
98 <script src="https://unpkg.com/axios/dist/axios.min.js"></script>
99 <script src='/js/app.js' type="text/javascript"></script>
100 </body>
101 </html>
```

```
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              //Poner editing a falso para mostrar las acciones de nuevo y ocultar los\
20      campos de entrada
21          story.editing = false;
22      },
23      storeStory: function (story) {
24          axios.post('/api/stories/', story).then(function (response) {
25              //Luego de que la nueva es guardada en la base de datos, obtener tod\as las historias de nuevo con
26              vm.fetchStories();
27              //O mejor, actualizar el id de la historia creada
28              Vue.set(story, 'id', response.data.id);
29              //Establecer editing a false para mostrar las acciones de nuevo y ocultar los campos
30              story.editing = false;
31          });
32      },
33  });
34 },
35 }
36 })
37
38 // Vue.prototype.$http = axios
39
40 new Vue({
41     el: '#v-app',
42     data: {
43         stories: [],
44         story: {}
45     },
46     mounted: function () {
47         this.fetchStories()
48     },
49     methods: {
50         createStory: function () {
51             var newStory = {
52                 plot: "",
53                 upvotes: 0,
54                 editing: true
55             };
56             this.stories.push(newStory);
57         },
58         fetchStories: function () {
```

```
59         var vm = this;
60         axios.get('/api/stories')
61             .then(function (response) {
62                 // Establecer los datos en vm
63                 var storiesReady = response.data.map(function (story) {
64                     story.editing = false;
65                     return story
66                 })
67                 // vm.stories = storiesReady
68                 Vue.set(vm, 'stories', storiesReady)
69             });
70     },
71 }
72});
```



Código de Ejemplo

Puedes encontrar el código de ejemplo de este capítulo en [GitHub](#)⁹²

Ejercicios

Para que te sientas más cómodo con hacer solicitudes web y manejar respuestas, deberías replicar lo que hicimos en este capítulo.

Lo que tienes que hacer es consumir una API, a fin de:

- Crear una tabla y **mostrar** películas existentes.
- **Modificar** películas existentes.
- **Guardar** nuevas películas en la base de datos.
- **Eliminar** películas de la base de datos.

He preparado la **base de datos** y la **API** para ti. Sólo tienes que escribir HTML y JavaScript.

Prefacio

Si has seguido las instrucciones del [Capítulo 10](#), abre tu terminal y ejecuta:

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

Si no lo has hecho, debes ejecutar esto:

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

¡Ahora tienes una **base de datos llena de buenas películas** junto con un servidor completamente funcional ejecutándose en [http://localhost:3000!](http://localhost:3000)

Para asegurarte de que todo está funcionando bien navega a <http://localhost:3000/api/movies>, deberías ver un arreglo de películas en formato JSON.

⁹²<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter12>

Los Endpoints de la API

Los Endpoints de la API que vas a necesitar son:

Método HTTP	URI	Acción
GET/HEAD	api/movies	Obtiene todas las películas
GET/HEAD	api/movies/{id}	Obtiene la película especificada
POST	api/movies	Crea una nueva película
PUT/PATCH	api/movies/{id}	Actualiza una película existente
DELETE	api/movies/{id}	Elimina la película especificada

Tu Código

Pon tu código HTML dentro del archivo `~/themajestyofvuejs2/apis/movies/public/movies.html` que hemos creado. Puedes colocar tu código JavaScript ahí también o dentro de `js/app.js`.

Para revisar tu trabajo visita `http://localhost:3000/movies.html` en tu navegador.

Espero que disfrutes este ejercicio. ¡Buena suerte!

#	Titulo	Director	Acciones
1	One Flew Over the Cuckoo's Nest	Milos Forman	Editar Eliminar
2	The Exorcist	William Friedkin	Editar Eliminar
3	What's Eating Gilbert Grape	Lasse Hallström	Editar Eliminar
4	Spirited Away	Hayao Miyazaki	Editar Eliminar
5	Aladdin	Ron Clements	Editar Eliminar
6	Star Wars: Episode VI - Return of the Jedi	George Lucas	Editar Eliminar
7	The Bourne Ultimatum	Paul Greengrass	Editar Eliminar
8	Scarface	Brian De Palma	Editar Eliminar
20	The Lord of the Rings: The Fellowship of the Ring	Peter Jackson	Editar Eliminar

Aquí hay una lista de todas tus películas. [¿Agregar una nueva?](#)

Pantalla de Ejemplo



Possible Solución

Puedes encontrar una posible solución para este ejercicio [aquí⁹³](#).

⁹³<https://github.com/hoottlex/the-majesty-of-vuejs-2/tree/master/homework/Chapter12>

Construyendo Aplicaciones Complejas

ECMAScript 6

Antes de llevar las cosas un paso más adelante y ver como podemos construir aplicaciones complejas, me gustaría familiarizarte con ECMAScript 6.

Info

ECMAScript es una especificación de lenguajes basados en script de lado del cliente, que es la base de varios de lenguajes de programación que incluyen a JavaScript, ActionScript y JScript.

ECMAScript 6 (ES6), también conocido como ES2015, es la última versión del estándar ECMAScript. La especificación ES6 fue finalizada en Junio de 2015. Es una actualización significativa para el lenguaje y la primera mayor actualización desde que ES5 fue estandarizado en 2009. La implementación de las características de ES6 en los principales motores de JavaScript está [en progreso⁹⁴](#).

Introducción

ES6 tiene muchas características nuevas. Vamos a revisar esas que utilizaremos en los próximos capítulos. Si estás interesado en aprender más sobre lo nuevo en ES6, te recomiendo sumamente el libro “Understanding ECMAScript 6” por Nicholas C. Zakas, disponible en [leanpub⁹⁵](#). También hay una [versión online⁹⁶](#) del libro, la cual puedes leer gratis.

También, hay otros recursos y tutoriales útiles como el de [Babel⁹⁷](#), un artículo en [tutsplus⁹⁸](#), un [post en el blog⁹⁹](#) de Nicholas C. Zakas y ¡muchas otras cosas alrededor de la web!

⁹⁴<http://kangax.github.io/compat-table/es6/>

⁹⁵<https://leanpub.com/understandinges6/>

⁹⁶<https://leanpub.com/understandinges6/read>

⁹⁷<https://babeljs.io/docs/learn-es2015/>

⁹⁸<http://code.tutsplus.com/articles/use-ecmascript-6-today--net-31582>

⁹⁹<https://www.nczonline.net/blog/2013/09/10/understanding-ecmascript-6-arrow-functions/>

Compatibilidad

Como era de esperarse, el soporte varía enormemente de motor a motor, con Mozilla tendiendo a liderar el camino. La [tabla de compatibilidad de ES6¹⁰⁰](#) es un comienzo útil para definir qué características de ECMAScript 6 tu navegador soporta o no.



Nota

Si estás utilizando Chrome la mayoría de las características de ES6 están deshabilitadas. Navega a *chrome://flags*, encuentra la sección titulada “Habilitar JavaScript Experimental” y habilitala para activar el soporte.

De ahora en adelante desarrollaremos nuestros ejemplos usando características de ES6.

Declaraciones de Variables

Declaraciones Let

`let` es el nuevo `var`. Puedes básicamente reemplazar `var` con `let` para declarar una variable, pero limitas el alcance de la variable sólo al bloque de código actual. Dado que no es obligatorio que las declaraciones `let` estén ubicadas en la parte superior del bloque delimitado, es mejor que siempre coloques las declaraciones `let` al principio, para que estén disponibles para todo el bloque. Por ejemplo:

Let dentro de un if

```
1 let age = 22
2 if (age >= 18) {
3     let adult = true;
4     console.log(adult); //muestra true
5 }
6 // adult no es accesible aquí
7 console.log(adult);
8 //ERROR: Uncaught ReferenceError: adult no está definido
```

¹⁰⁰<https://kangax.github.io/compat-table/es6/>

Let al principio del bloque

```
1 let age = 22
2 let adult
3 if (age >= 18) {
4     adult = true;
5     console.log(adult); //muestra true
6 }
7 // ahora adult es accesible aquí
8 console.log(adult); //muestra true
```

Declaraciones de Constantes

Las constantes, como las declaraciones `let`, son declaraciones a nivel de bloque. Hay una gran diferencia entre `let` y `const`. Una vez que declares una variable usando `const`, esta es definida como una constante, lo que significa que **no puedes cambiar su valor**.

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



Info

Similar a las constantes en otros lenguajes, su valor no puede cambiar más adelante. Sin embargo, a diferencia de las constantes en otros lenguajes, el valor que una constante posee **puede ser modificado si es un objeto**.

Funciones Flecha

Una de las partes nuevas más interesantes de ECMAScript 6 son las funciones flecha. Las funciones flecha son funciones definidas con una nueva sintaxis que usa una “flecha” (\Rightarrow). Soportan tanto bloque de instrucciones así como expresiones. A diferencia de las funciones normales, las funciones flechas comparten el mismo `this` léxico que su código alrededor.

Por ejemplo, la siguiente función flecha toma un solo argumento y retorna su valor incrementado en 1:

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

// equivalente a:

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

Otro ejemplo con una función flecha que toma 2 argumentos y retorna su suma:

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

// equivalente a:

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

Una función flecha de ejemplo que no toma ningún argumento y usa un bloque de instrucciones:

```
var sayHiAndBye = () => {
    console.log('¡Hola!');
    console.log('¡Adios!');
};

sayHiAndBye()
```

// equivalente a:

```
var sayHiAndBye = function() {
    console.log('¡Hola!');
    console.log('¡Adiós!');
};
```

Módulos

Esta es, para mí, la mejora más importante al lenguaje. ES6 ahora soporta la exportación e importación de módulos a través de diferentes archivos.

El ejemplo más simple es crear un archivo `.js` con una variable, y usarlo dentro de otro archivo así:

module.js

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

main.js

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

También puedes exportar variables junto con funciones **una a una**:

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, 'tiene', getAge())
```

O dentro de un objeto:

module.js

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

main.js

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

Clases

Las clases de JavaScript son introducidas en ECMAScript 6 y son azúcar sintáctica sobre la herencia basada en prototipos existente de JavaScript. La sintaxis de clase **no** está introduciendo un nuevo modelo de herencia orientado a objetos a JavaScript. Las clases de JavaScript proporcionan una sintaxis mucho más simple y clara para crear objetos y tratar con la herencia.

Ejemplo de Clase

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

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

    // Para crear un getter, usa la palabra clave get.
    get area() {
        return this.calcArea();
    }
    // Para crear un setter haces lo mismo pero usando la palabra clave set.
}

// clase hija
class Square extends Rectangle{
    constructor(side) {
        // llama al constructor del padre
        super(side, side)
    }
}

var square = new Square(5);

console.log(square.area); // Muestra "25"
```

Valores de Parámetros Predeterminados

Con ES6 puedes definir valores de parámetros predeterminados.

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

// equivalente a:

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

Plantillas de literales

Las plantillas de literales son cadenas literales que permiten expresiones embebidas. Con ellas, puedes usar cadenas de texto de varias líneas e interpolación de cadenas de texto. Eran llamadas “plantillas de cadena de texto” en ediciones anteriores de la especificación ES2015/ES6.

Las plantillas de literales están delimitadas por caracteres de acento invertido (`) en lugar de comillas dobles o simples. Dentro de los acentos invertidos puedes usar \${expression} donde expression puede ser una función, variable o una expresión.

Plantilla de literales - Usando Variables

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

// equivalente a:
console.log('Hello ' + name )
```

Plantillas de literales - Usando Funciones

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

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

console.log(`Si tienes ${a} huevos y compras ${b}
más, tendrás ${add(a,b)} huevos!`)

// equivalente a:
console.log('Si tienes ' + a + ' huevos y compras ' + b +
'\nmás, tendrás ' + add(a,b) + ' huevos!')
```

Plantillas de literales - Usando Expresiones

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

console.log(`Si tienes ${a} huevos y compras ${b}
más, tendrás ${a + b} huevos!`)

// equivalente a:
console.log('Si tienes ' + a + ' huevos y compras ' + b +
'\nmás, tendrás ' + (a + b)*1 + ' huevos!')
```

También es posible dividir el mensaje en múltiples líneas utilizando '\n'.



Video

El uso de Promises en ES6 es cubierto en la [lección 51 del curso de Vue 2 en Styde](#)¹⁰¹.

¹⁰¹<https://styde.net/uso-de-promesas-en-javascript/>

Flujo de Trabajo Avanzado

Todas estas características de ES6 (y muchas más) pueden entusiasmarte, pero hay un inconveniente. Como mencionamos anteriormente, **no** todos los navegadores soportan completamente las características de ES6/ES2015.

Para poder escribir esta nueva sintaxis de JavaScript hoy mismo, necesitamos tener un intermediario que tomará nuestro código y lo transpilará a [JavaScript Vanilla¹⁰²](#), es decir JavaScript, al cual *todos los navegadores entienden*. Aunque puede que no lo pienses así, este procedimiento es realmente **importante** en producción.

Dejame que te cuente una historia. Hace unos años atrás, un compañero de trabajo mío comenzó a usar algunas características geniales de JS que no eran totalmente soportadas por todos los navegadores. Unos días después nuestros usuarios comenzaron a quejarse sobre algunas páginas de nuestro sitio web que no se mostraban correctamente, pero no podíamos saber por qué. Lo probamos en diferentes PCs, teléfonos Android, iPhones, etc. y era 100% funcional en todos nuestros navegadores. Mas tarde, este compañero descubrió que versiones antiguas de Safari móvil no soportaban su código. *¡No seas ese chico!*.

Algunas veces es **realmente difícil** saber si el código que escribes va a funcionar bien en **todos los navegadores**, incluyendo el navegador móvil de Facebook, que es mi peor miedo.

Compilando ES6 con Babel

Babel será nuestro *intermediario*. Babel es un compilador source-to-source de JavaScript, que nos permite usar JavaScript de nueva generación hoy mismo.



Info

Un compilador source-to-source, transcompilador o transpilador es un tipo de compilador que toma el código fuente de un programa escrito en un lenguaje de programación y produce el código fuente equivalente en otro lenguaje de programación.

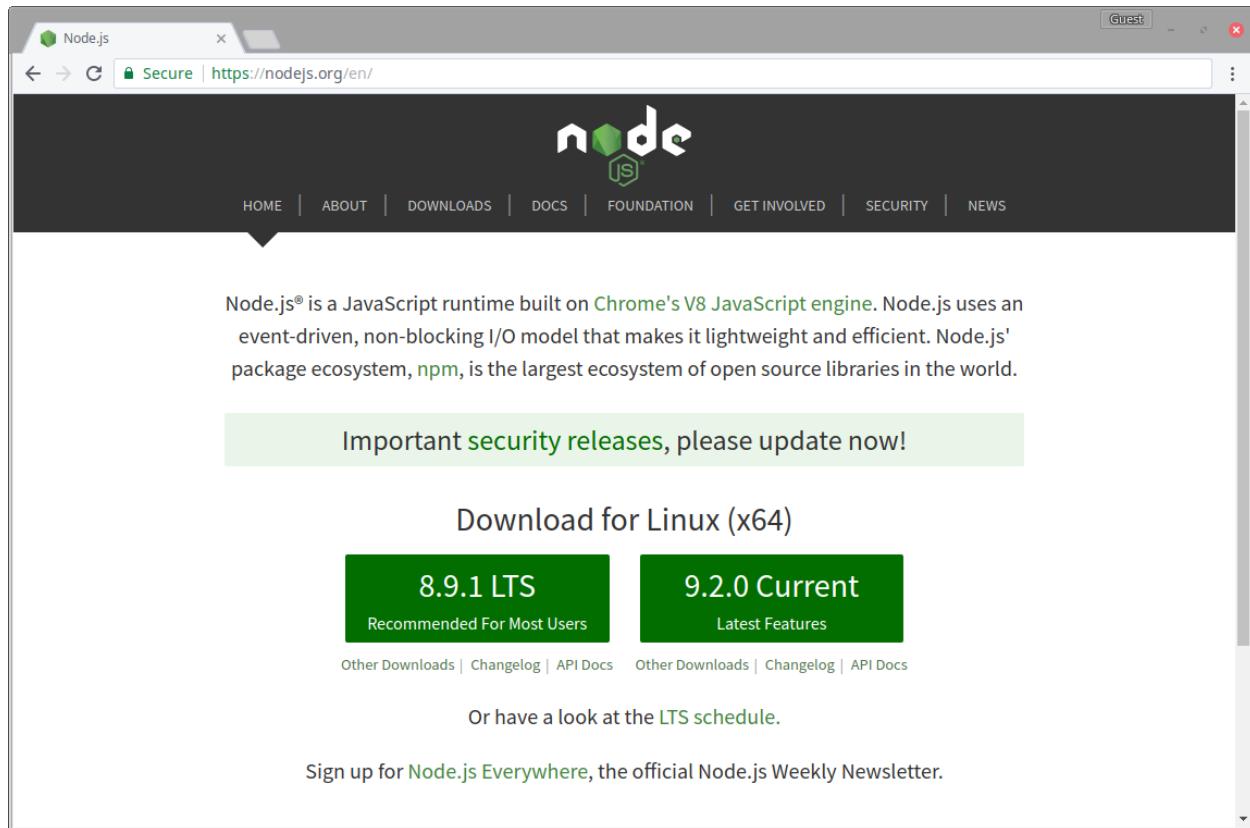
¹⁰²<http://vanilla-js.com/>

The screenshot shows the Babel website at <https://babeljs.io>. The main heading is "Babel is a JavaScript compiler." with the subtitle "Use next generation JavaScript, today.". Below this is a code editor interface with two panes: "Put in next-gen JavaScript" and "Get browser-compatible JavaScript out". Both panes contain the same code snippet:var obj = {
 shorthand,
 method() {
 return "⊕";
 }
};A callout at the bottom of the editor says "Check out our REPL to experiment more with Babel!". At the bottom of the page is a button labeled "Latest From Our Blog: Babel Turns Three".

<https://shop.nets.eu/nb/web/partners/test-cards1>

Antes de instalar Babel, tienes que instalar **Node.js**. Para hacerlo, dirígete al [sitio web de Node](https://nodejs.org/en/)¹⁰³ y presiona el botón de descargar para la Última Versión Estable. Te va a dar un archivo *.pkg* (o *.msi* si estás en Windows). Cuando la descarga ha finalizado, abre el archivo y sigue las instrucciones. Luego, has el reinicio requerido y estás listo.

¹⁰³<https://nodejs.org/en/>



Node.js

Instalación

Crea un nuevo directorio y coloca un archivo llamado **package.json** dentro, conteniendo un objeto JSON vacío ({}). Puedes hacerlo manualmente o ejecutando los siguientes comandos en tu terminal.

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

Luego ejecuta esto para instalar 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, caseless@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)

~ »
```

Resultados por Terminal

Cuando esté listo, tu archivo package.json debería ser algo así:

package.js

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



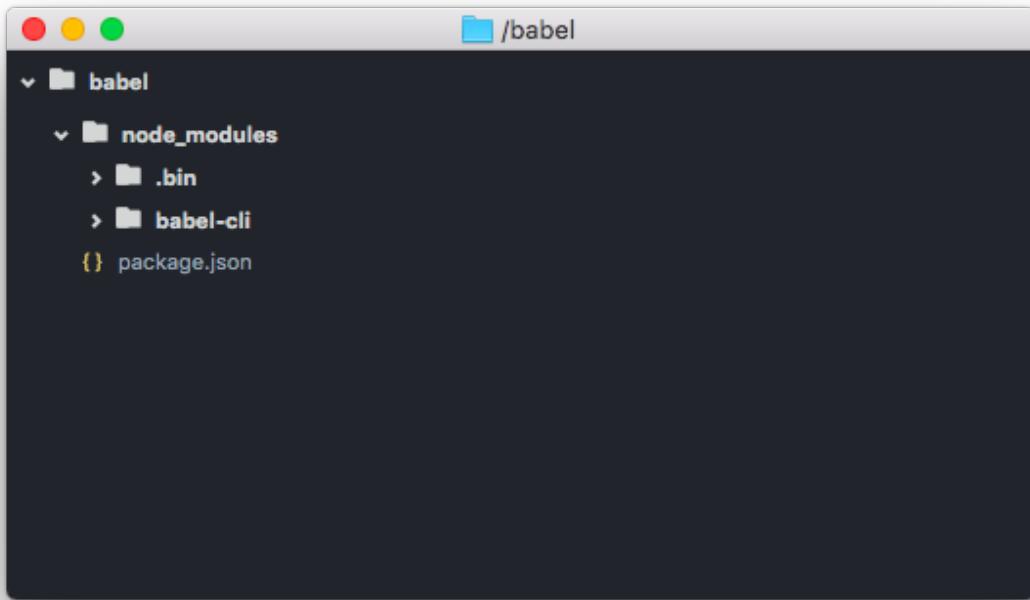
¿Qué es package.json?

Un archivo package.json contiene metadatos sobre tu aplicación o módulo. Principalmente, contiene la lista de dependencias a instalar desde npm al ejecutar `npm install`. Si estás familiarizado con Composer, es similar al archivo composer.json.

Para aprender más sobre package.json echa un vistazo a la [documentación de npm](#)¹⁰⁴.

El directorio de tu proyecto debería verse así:

¹⁰⁴<https://docs.npmjs.com/files/package.json>



Directorio del Proyecto

Configuración

Ahora que hemos instalado Babel, necesitamos indicarle explícitamente que transformaciones ejecutar al compilar. Dado que queremos transformar código ES2015, instalaremos [ES2015-Preset¹⁰⁵](#). También crearemos un archivo de configuración (`.babelrc`) para habilitar nuestro preset.

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



Tip

Si el segundo comando falla, encierra el contenido del archivo dentro de comillas, así:

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

¹⁰⁵<https://babeljs.io/docs/plugins/preset-es2015/>

Alias para Compilar

En lugar de ejecutar Babel directamente desde la línea de comandos, vamos a colocar nuestros comandos dentro de **scripts** de **npm**.

Agregaremos un campo **scripts** a nuestro archivo **package.json** y ahí registraremos el comando **babel** como **build**.

Nuestro **package.json** se verá de la siguiente manera:

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

Esto funciona como un alias. Lo que significa que cuando ejecutamos **npm run build** realmente estamos ejecutando **babel src -d assets/js**. Este comando le dice a Babel que transpile el código del directorio **src** al directorio **assets/js**.

Antes de que ejecutemos el comando **build** tenemos que hacer algunas cosas más. Para comenzar, ve y crea los directorios anteriormente mencionados (**src** y **assets/js**).

Uso

Sigamos adelante y pongamos algunos archivos dentro de la carpeta **src**. Crearé un archivo con una sencilla función llamada **sum** y llamaré a este archivo **sum.js**.

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

Eso es todo. Ahora podemos ejecutar:

```
>_ npm run build
```

Cuando lo ejecutas, puedes ver en tu terminal que el archivo **src\sum.js** ha sido compilado a **assets\js\sum.js** y se ve así:

assets/js/sum.js

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

De ahora en adelante, cada vez que quieras compilar tu código ES6 puedes hacerlo ejecutando el comando **build**. Bastante grandioso, ¿verdad?.

Es hora de ver el archivo **sum.js** resultante en el navegador. Crearé **sum.html** e incluiré nuestro **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>
```



Resultado en el Navegador

Como puedes ver, el resultado de la función `sum` se imprime con éxito en la consola.



Info

Cuando quieras probar un archivo `.js`, pero no quieras entrar en el proceso para verlo en el navegador, puedes ejecutarlo con Node.js.

En el ejemplo `sum.js` hay una línea `console.log(sum(5, 3))`, así que si escribes en tu terminal `node sum.js` verás el resultado (8) aparecer inmediatamente!.

Ejercicios

Estos ejercicios pretenden ayudarte a recordar lo que has aprendido mientras reproduces el ejemplo que hemos construido. En lugar de `sum.js` ve y usa *clases ES6* para crear un archivo `Ninja.js` que contendrá una clase `Ninja`.

Un `Ninja` debería tener una propiedad `name` y un método `announce` que alertará de la presencia de un ninja.

Por ejemplo

```
new Ninja('Leonardo').announce()  
// muestra "El Ninja Leonardo está aquí!"
```

No te olvides de compilar tu `JavaScript` utilizando Babel antes de incluirlo en tu `HTML`.



Pista

Puedes encontrar un ejemplo para crear clases en el capítulo anterior.



Pista 2

No te olvides de ejecutar `npm run build` cada vez que haces un cambio en tu archivo `JavaScript`, de lo contrario no se actualizará.



Possible Solución

Puedes encontrar una posible solución para este ejercicio [aquí¹⁰⁶](#).

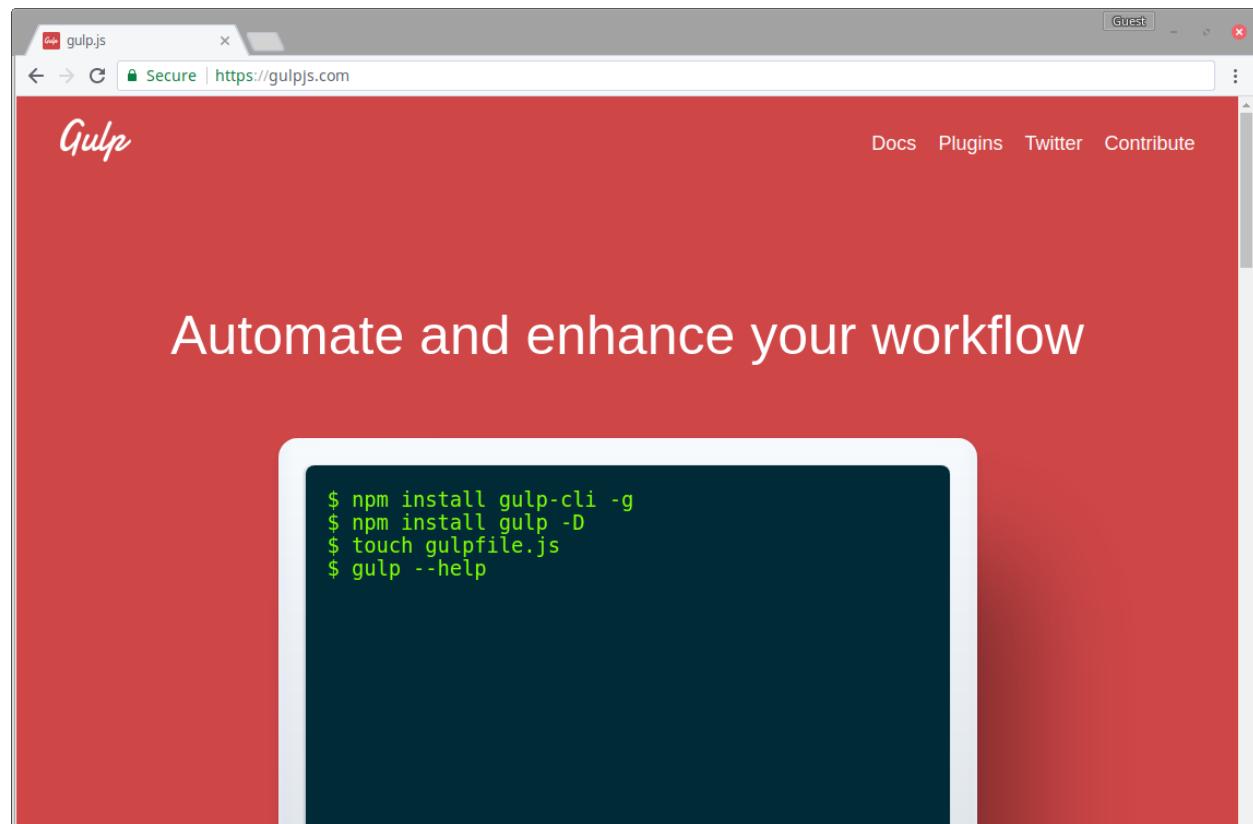
¹⁰⁶<https://github.com/hoottlex/the-majesty-of-vuejs-2/tree/master/homework/Chapter15/chapter15.1>

Automatización del Flujo de Trabajo con Gulp

Automatizadores de Tareas

Si has dedicado algún tiempo a desarrollar la aplicación del ejercicio de la sección anterior, probablemente has encontrado que es un poco molesto tener que ejecutar `npm run build` cada vez que haces un cambio en tu código.

Aquí es donde los **automatizadores de tareas** como [Gulp¹⁰⁷](#) o [Grunt¹⁰⁸](#) son útiles. Los automatizadores de tareas te permiten automatizar y mejorar tu flujo de trabajo.



Gulp

¿Por qué usar un automatizador de tareas?

En una palabra: **Automatización**. Cuanto menos tiempo tengas que dedicar a realizar tareas repetitivas como minificación, compilación, pruebas unitarias, linting etc. más fácil se vuelve tu trabajo.

¹⁰⁷<http://gulpjs.com/>

¹⁰⁸<http://gruntjs.com/>



Gulp vs Grunt

Grunt, como Gulp, es una herramienta para definir y ejecutar tareas. La mayor diferencia entre Grunt y Gulp es que Grunt define las tareas utilizando **objetos de configuración** mientras que Gulp define las tareas como **funciones de JavaScript**. Dado que Gulp ejecuta JavaScript, proporciona más flexibilidad en la escritura de tus tareas.

Ambos tienen una librería masiva de plugins, donde puedes encontrar alguno que implemente una tarea que necesites.

Instalación

Te mostraré un ejemplo de como puedes usar Gulp para observar los cambios en tus archivos JavaScript y automáticamente ejecutar el comando **build**.

Primero tenemos que instalar Gulp de forma global:

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

Luego instalaremos Gulp en el devDependencies de nuestro proyecto:

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

Ahora que tenemos gulp instalado, crearemos un **gulpfile.js** en el directorio raíz de nuestro proyecto:

gulpfile.js

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

gulp.task('default', function() {
  // coloca el código para tu tarea predeterminada aquí
});
```

Uso

Ahora cuando ejecutamos **gulp** en nuestra consola, se inicia, pero no hace nada aún. Tenemos que configurar una tarea predeterminada.

Para ejecutar `babel` directamente, instalaré un plugin de npm llamado `gulp-babel`¹⁰⁹.

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

Agregaré una nueva *tarea de gulp* llamada `babel` y la estableceré como la tarea predeterminada. Mi `gulpfile` se verá así:

`gulpfile.js`

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

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

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

Esta tarea básicamente le dice a Babel que transforme, utilizando el preset `es2015`, todos los archivos `js` que se encuentran en el directorio `src` y los coloque dentro del directorio `assets/js`.

Observar

Actualmente ejecutar `gulp` en tu consola tiene el mismo efecto que con `npm run build`. Lo que queremos lograr aquí es ejecutar esta tarea cada vez que un archivo `js` cambie. Para ello, configuraremos un *observador* dentro de nuestro `gulpfile`, así:

¹⁰⁹<https://www.npmjs.com/package/gulp-babel>

gulpfile.js

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

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

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

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

Cuando ejecutamos `gulp watch` en nuestra consola, gulp esta observando los cambios en todos nuestros archivos `.js` bajo el directorio especificado. Cada vez que hacemos un cambio, gulp ejecuta nuestra tarea de `babel` y los archivos bajo `assets/js` son actualizados. ¡Cuán genial es eso?.

Ejercicios

Este ejercicio sigue al anterior. Si no has hecho el anterior, ¡nunca es demasiado tarde para comenzar!.

Dado que esta parte del capítulo está dedicada a automatizadores de tareas, tienes que configurar un observador con Gulp y compilar tu código con `Babel`, cuando un cambio es detectado.



Nota

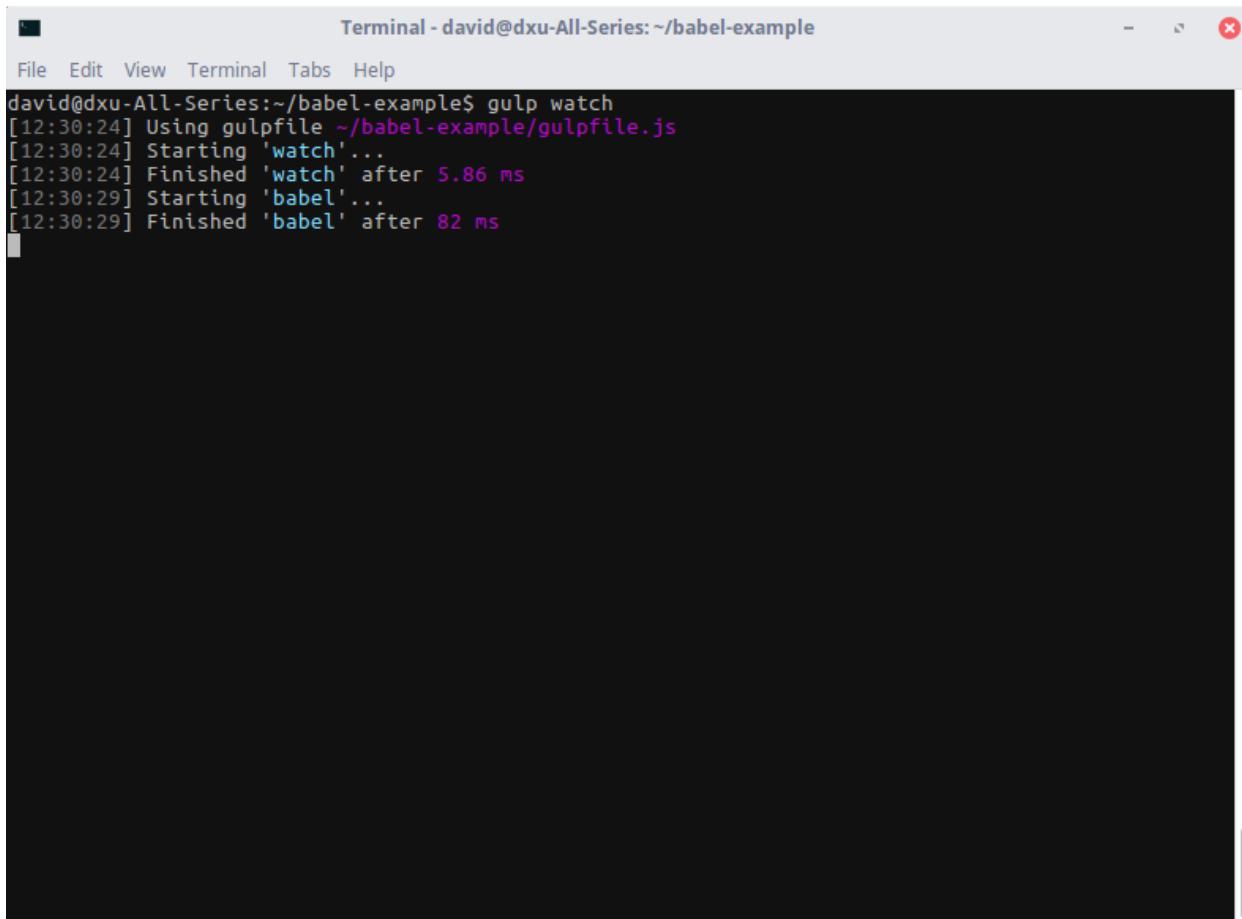
Es probable que ya hayas notado que al ejecutar Gulp este imprime mensajes en la terminal (*“Starting”* - *“Finished”*), así que no seas impaciente y espera a que los cambios sean aplicados.



Possible Solución

Puedes encontrar una posible solución para este ejercicio [aquí](#)¹¹⁰.

¹¹⁰<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/homework/Chapter15/chapter15.2>

A screenshot of a terminal window titled "Terminal - david@dxdxu-All-Series: ~/babel-example". The window shows the command "gulp watch" being run, followed by several log messages indicating the process: "[12:30:24] Using gulpfile ~/babel-example/gulpfile.js", "[12:30:24] Starting 'watch'...", "[12:30:24] Finished 'watch' after 5.86 ms", "[12:30:29] Starting 'babel'...", and "[12:30:29] Finished 'babel' after 82 ms".

```
Terminal - david@dxdxu-All-Series: ~/babel-example
File Edit View Terminal Tabs Help
david@dxdxu-All-Series:~/babel-example$ gulp watch
[12:30:24] Using gulpfile ~/babel-example/gulpfile.js
[12:30:24] Starting 'watch'...
[12:30:24] Finished 'watch' after 5.86 ms
[12:30:29] Starting 'babel'...
[12:30:29] Finished 'babel' after 82 ms
```

¡Gulp está observando!

Empaquetado de Módulos con Webpack

Empaquetadores de Módulos

Nuestro flujo de trabajo está bien con el código actual de `sum.js`. Extenderemos sus características para calcular el costo de una pizza, una cerveza y dejarle saber al cliente.

`src/sum.js`

```
const pizza = 10
const beer = 5

const sum = (a, b) => a + b + '$';
console.log(`Alex, tienes que pagar ${sum(pizza, beer)})`)
```

Este código se ve bien, pero asumiendo que no todo el mundo se llama *Alex* crearemos un nuevo archivo, `client.js`, que proporcionará el nombre del cliente.

src/client.js

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

Importaremos el nombre desde ahí.

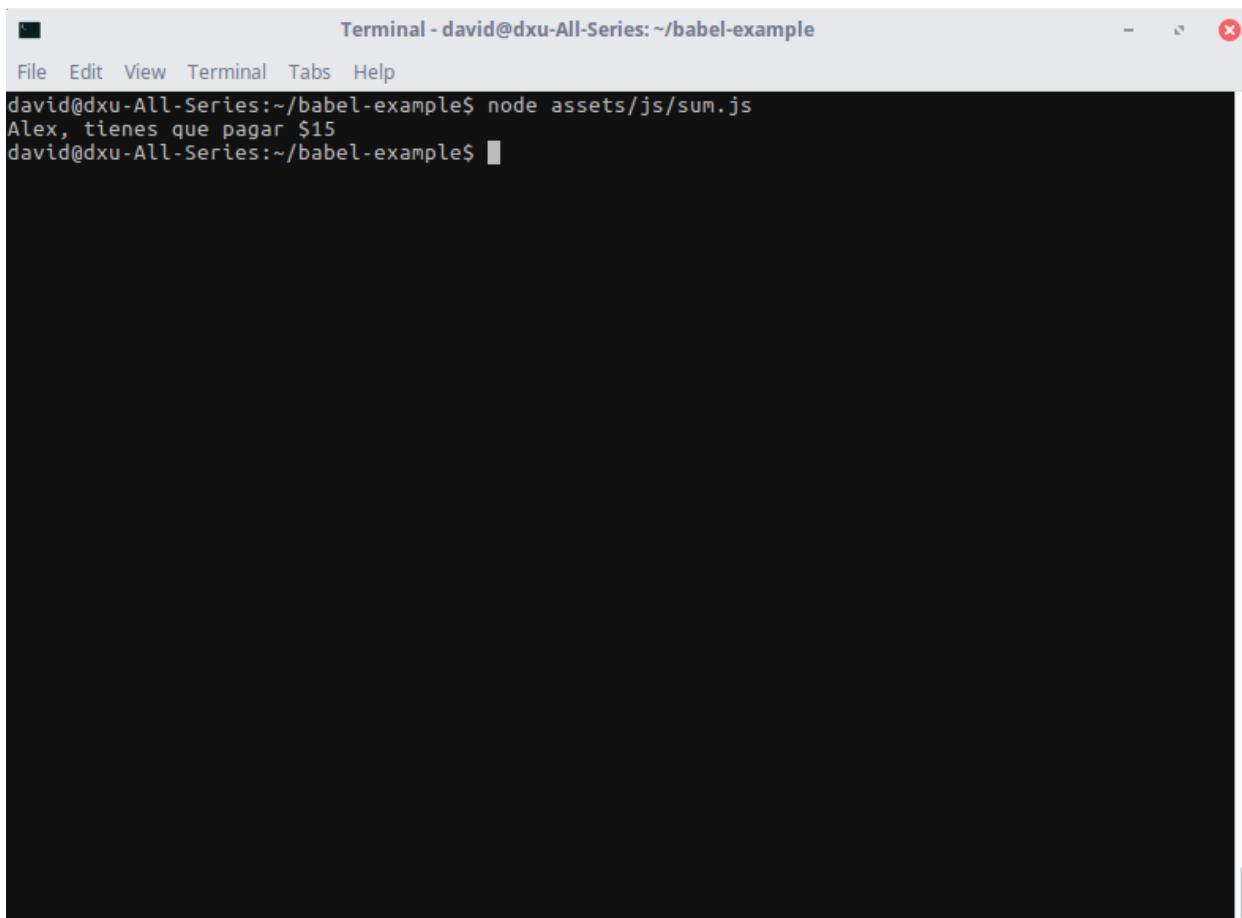
src/sum.js

```
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)} `)
```

¡Genial! Si ejecutamos `node assets/js/sum.js` obtenemos el resultado deseado.

A screenshot of a terminal window titled "Terminal - david@dxdxu-All-Series: ~/babel-example". The window has a standard OS X-style title bar with icons for close, minimize, and maximize. The menu bar shows "File Edit View Terminal Tabs Help". The main terminal area contains the following text:

```
File Edit View Terminal Tabs Help
david@dxdxu-All-Series:~/babel-example$ node assets/js/sum.js
Alex, tienes que pagar $15
david@dxdxu-All-Series:~/babel-example$
```

The terminal window is set against a dark background.

Salida de sum.js

Esperarías que el mismo comportamiento se aplicará cuando abramos el archivo *html* en el navegador, ¡pero no ocurre!. Obtenemos un error en su lugar.



Require no está definido

Verifica `node assets/js/sum.js` y observa el `var _client = require('./client');` en la parte superior. La razón por la que obtenemos el error en el navegador es debido a que `require()` no existe en el navegador/JavaScript del lado del cliente.. Lo que tenemos que hacer es *empaquetar* los módulos en un archivo para que pueda ser incluido dentro de una etiqueta `<script>`.

The screenshot shows a Mac OS X desktop environment. On the left is a file browser window titled 'babel-example' showing a directory structure:

- assets/ja
- node_modules
- src

Under 'src', there are files: client.js, sum.js, .babelrc, .DS_Store, gulpfile.js, package.json, and sum.html.

On the right is a code editor window titled 'sum.js — assets/ja — /babel-example'. It contains the following JavaScript code:

```
'use strict';

var _client = require('./client');

var pizza = 10;
var beer = 5;

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

console.log(_client.name, ' have to pay', sum(pizza, beer));
```

The status bar at the bottom of the code editor shows: File 0 Project 0 ✓ No Issues assets/ja/sum.js LF UTF-8 2 Spaces JavaScript

assets/ja/sum.js

Aquí es donde necesitamos **empaquetadores de módulos** como [Webpack¹¹¹](#) o [Browserify¹¹²](#).

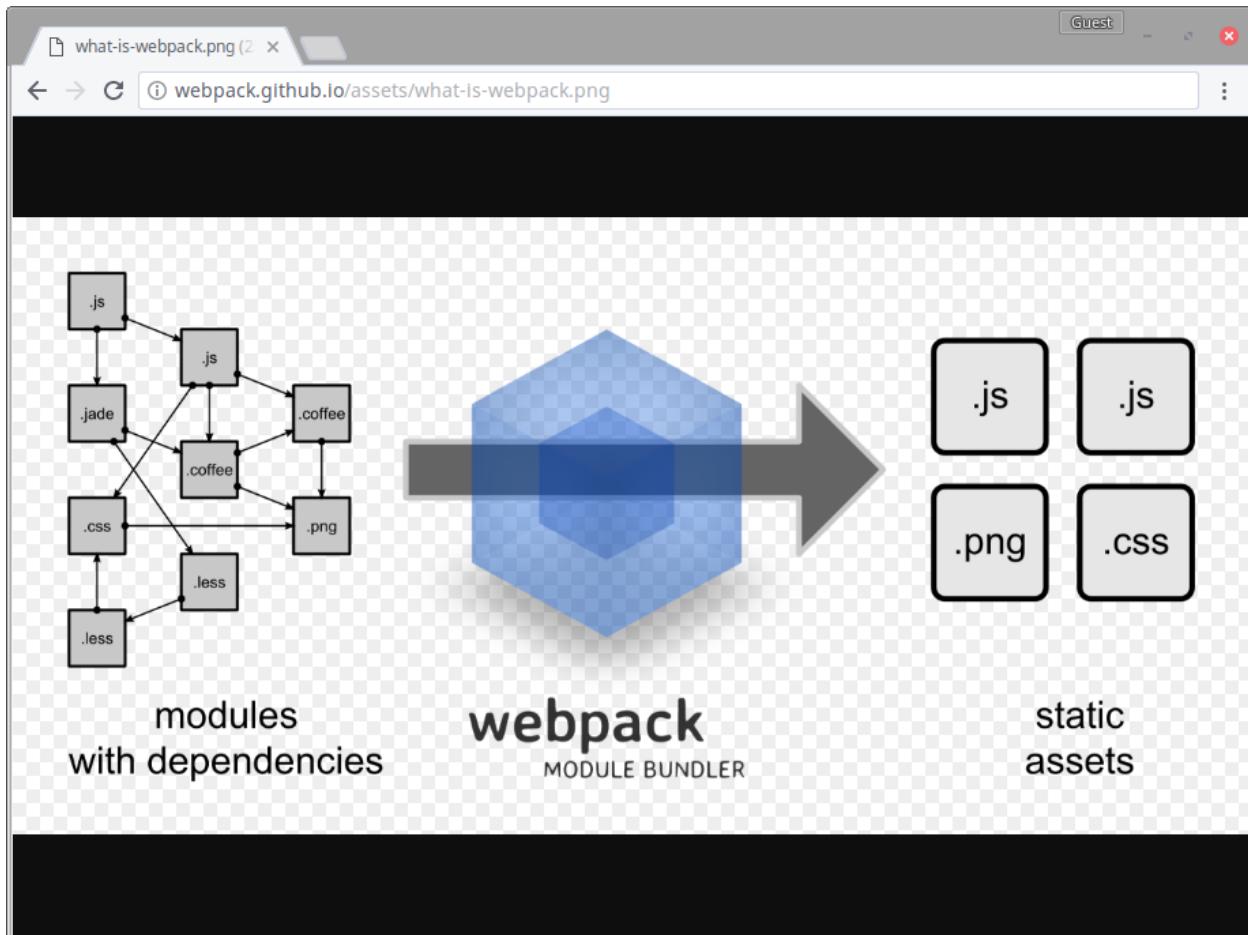
Webpack

Webpack es un empaquetador de módulos. Toma módulos de JavaScript, reconoce sus dependencias, y luego las concatena y produce recursos estáticos que representan esos módulos.

Yo usaré Webpack en los siguientes ejemplos. Eso es debido a que creo que será útil en el futuro para otros usos, ya que puede hacer más que empaquetar modulos. Usando *loaders* le podemos enseñar a Webpack a transformar todo tipo de archivos en cualquier forma que queramos, antes de producir el paquete final.

¹¹¹<https://webpack.github.io/>

¹¹²<http://browserify.org/>



Lo que Webpack hace

Instalación

Primero voy a instalar Webpack de forma global, y luego lo agregaré como una dependencia en nuestro proyecto.

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



Tip

Al momento de escribir esto, hay un bug conocido cuando estás usando Vagrant¹¹³ en Windows, por lo que ejecutar `npm install` puede fallar. Para resolver este problema sal de Vagrant, haz `cd` a tu proyecto en la terminal de Windows y ejecuta `npm install` desde ahí.

¹¹³<https://www.vagrantup.com/>

Uso

A fin de compilar nuestro JavaScript tenemos que darle a Webpack un punto de origen y uno de salida. En nuestro caso el origen es el archivo `assets/js/sum.js` “producido por Babel” y como salida estableceré `assets/webpacked/app.js`.

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

```
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 »
```

Salida de Webpack

Luego de que el empaquetado ha sido completado podemos usar el `JavaScript` generado, `assets/webpacked/app.js`, dentro de `sum.html`. También podemos ejecutarlo en la terminal usando `node assets/webpacked/app.js`.

Automatización

Si eres perezoso, como yo, y no te gusta tener que ejecutar `webpack` cada vez que haces un cambio, puedes automatizar su proceso. Puedes configurar Webpack para observar tu origen y reconstruir tu paquete cuando cualquiera de tus archivos cambie. Yo no lo haré aquí. En su lugar lo integraré en Gulp, con el fin de demostrar como puedes combinar múltiples herramientas.



Aprendizaje Adicional

Si quieres aprender más sobre como funciona Webpack y como puedes configurarlo, ve y lee “[Beginner’s guide to Webpack](#)¹¹⁴” por [Nader Dabit](#)¹¹⁵.

¹¹⁴<https://medium.com/@dabit3/beginner-s-guide-to-webpack-b1f1a3638460>

¹¹⁵<https://twitter.com/dabit3>

Para integrar Webpack en Gulp usará un plugin llamado `webpack-stream`¹¹⁶.

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

Luego de instalarlo, crearé una nueva tarea con el nombre de ‘webpack’ y le diré a Gulp que la ejecute inmediatamente después de ejecutar la tarea ‘babel’, cada vez que detecte un cambio.

`gulpfile.js`

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

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

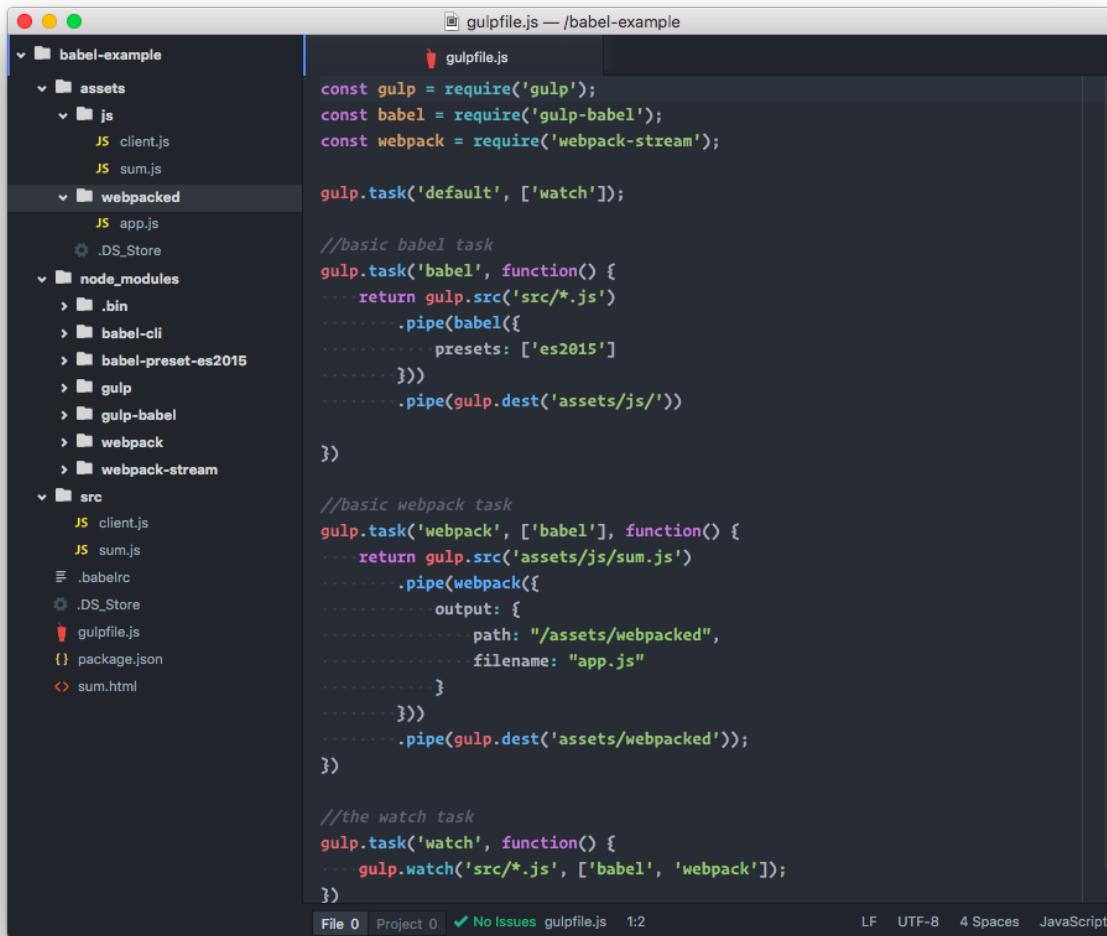
//tarea básica de babel
gulp.task('babel', function() {
    return gulp.src('src/*.js')
        .pipe(babel({
            presets: ['es2015']
        }))
        .pipe(gulp.dest('assets/js/'))
})

//tarea básica de webpack
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'));
})

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

¹¹⁶<https://www.npmjs.com/package/webpack-stream>

Esta solución no es ideal. Es sólo una demostración de como puedes enlazar en combinación cualquier cosa que hayas aprendido. En producción, hay muchas mejores maneras de automatizar tus tareas de webpack.



```

babel-example
├── assets
│   ├── js
│   │   ├── client.js
│   │   └── sum.js
│   └── webpacked
│       ├── app.js
│       └── .DS_Store
└── node_modules
    ├── .bin
    ├── babel-cli
    ├── babel-preset-es2015
    ├── gulp
    ├── gulp-babel
    ├── webpack
    └── webpack-stream
├── src
│   ├── client.js
│   ├── sum.js
│   ├── .babelrc
│   └── .DS_Store
└── gulpfile.js
{} package.json
<-- sum.html

gulpfile.js — /babel-example
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']);
})

```

File 0 | Project 0 | ✓ No Issues | gulpfile.js | 1:2 | LF | UTF-8 | 4 Spaces | JavaScript

Webpack en Gulp

Resumen

Cuando quieres compilar ES6 puedes usar **Babel**¹¹⁷.

Para automatizar operaciones como esta y muchas otras (como minificar, compilar SASS/LESS, etc) necesitas automatizadores de tareas como **Gulp**¹¹⁸ o **Grunt**¹¹⁹.

¹¹⁷<http://babeljs.io/>

¹¹⁸<http://gulpjs.com/>

¹¹⁹<http://gruntjs.com/>

Para empaquetar cosas puedes usar [Webpack¹²⁰](#) o [Browserify¹²¹](#).

En el siguiente capítulo nos adentraremos en los Componentes de un Solo Archivo de Vue y usaremos con varias herramientas que Vue te proporciona, junto con las herramientas que hemos aprendido.



Video

El uso de Webpack y Babel es cubierto en la [lección 24 del curso de Vue 2 en Styde¹²²](#).



Nota

Si encontraste este capítulo difícil de entender, no te preocupes. No necesitas recordar todas esas cosas. Esto fue sólo una demostración para darte una mejor idea de como funcionan las cosas. En el siguiente capítulo usaremos *plantillas de proyecto*. Ahí, cosas como **empaquetamiento de modulos, automatización, compilación al cambiar** y mucho más ya están implementadas y tomaremos ventaja de ello.

¹²⁰<https://webpack.github.io/>

¹²¹<http://browserify.org/>

¹²²<https://styde.net/uso-de-es6-en-vue-con-webpack-y-babel/>

Trabajando con Componentes de un Solo Archivo

Como prometimos, en este capítulo vamos a revisar los componentes de un solo archivo. Para usar esos componentes de un solo archivo de Vue necesitamos herramientas como **webpack** con **vue-loader** o **Browserify** con **vueify**. Para nuestros ejemplos, vamos a usar webpack, que ya hemos visto como funciona. Si prefieres Browserify o algo diferente, sientete libre de usarlo.

Los componentes de un solo archivo o componentes de vue encapsulan sus estilos CSS, plantilla y código JavaScript en un solo archivo usando la extensión `.vue`. Y ahí es donde webpack entra, para empaquetar este nuevo tipo de archivo con los otros archivos.

webpack usa [vue-loader¹²³](#) para transformar los componentes de Vue en módulos de JavaScript plano. *vue-loader* también proporciona una muy buena serie de características tales como ES2015 habilitado por defecto, CSS accesible para cada componente y más.

vue-cli

Para evitar configurar webpack y crear un nuevo flujo de trabajo desde cero usaremos **vue-cli**.



Info

`vue-cli124` es una simple *interfaz de línea de comandos* para estructurar proyectos de Vue.js.

Esta genial herramienta es la forma más rápida de levantar un entorno preconfigurado. Ofrece plantillas con recarga automática, análisis sintáctico al guardar, pruebas unitarias y mucho más. Actualmente, ofrece plantillas preconfiguradas para webpack y browserify, pero si lo necesitas, puedes [crear tus propias plantillas¹²⁵](#).

Plantillas de Vue

Lo que CLI hace, es descargar plantillas desde [el repositorio oficial de plantillas de Vue.js¹²⁶](#) donde hay 5 plantillas disponibles. Creo que este número crecerá en un futuro próximo. Puedes comprobar lo que incluye cada plantilla en GitHub.

¹²³<https://github.com/vuejs/vue-loader>

¹²⁴<https://github.com/vuejs/vue-cli>

¹²⁵<https://github.com/vuejs/vue-cli#custom-templates>

¹²⁶<https://github.com/vuejs-templates>

Todas las plantillas contienen un archivo `package.json`, que maneja las dependencias del proyecto y viene con algunos scripts de NPM preconfigurados.

Usando las plantillas de proyecto de Vue, obtienes muchas características en conjunto. Por ejemplo, la descripción de la plantilla “webpack” dice que es “*Una completa configuración de webpack más vue-loader con recarga automática, análisis sintáctico, pruebas y extracción de css.*”

Instalación

Nos vamos a quedar con el método de configuración de webpack e instalar `vue-cli` globalmente usando el siguiente comando.

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

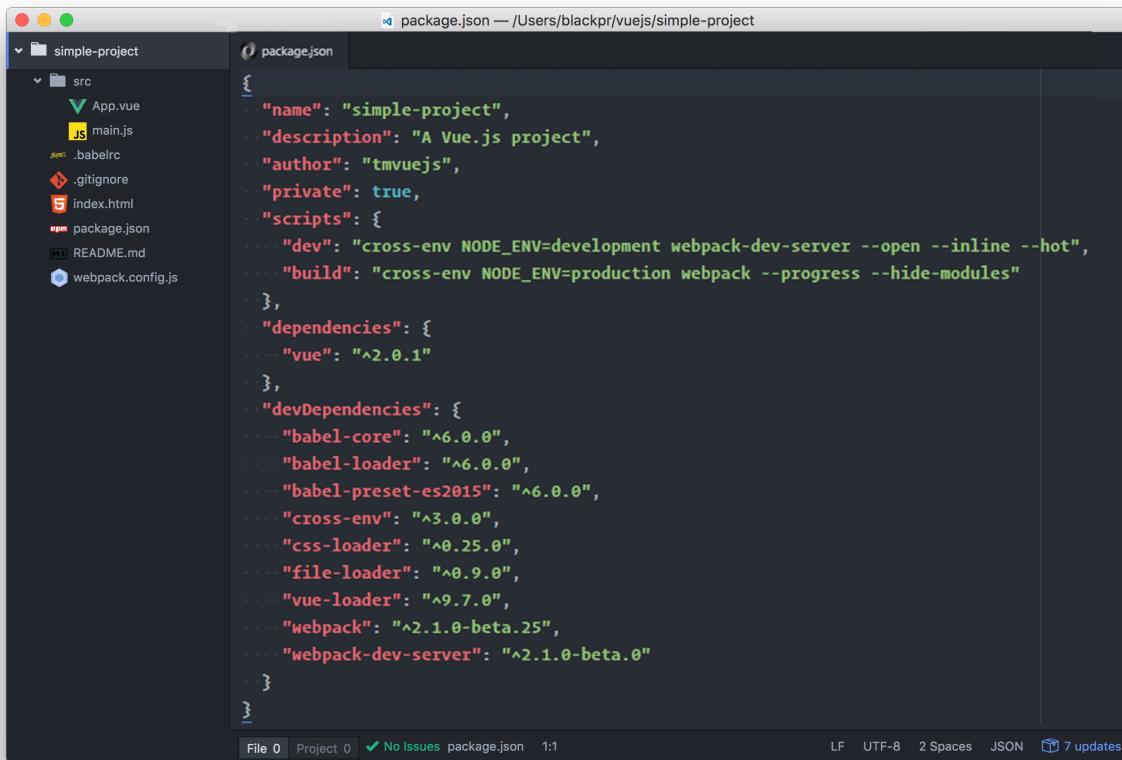
Uso

Usando la CLI puedes ejecutar `vue init <nombre-de-plantilla> <nombre-del-proyecto>` donde `<nombre-de-plantilla>` es el nombre de la plantilla (ya sea oficial o personalizada) y `<nombre-del-proyecto>` es el nombre del directorio/proyecto que vas a crear.

Así que, si ejecutas:

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

vas a tener un directorio llamado `simple-project` con la siguiente estructura:



The screenshot shows a code editor window with the title "package.json — /Users/blackpr/vuejs/simple-project". The left sidebar shows a file tree with a "simple-project" folder containing "src", ".babelrc", ".gitignore", "index.html", "package.json", "README.md", and "webpack.config.js". The main editor area displays the contents of the package.json file:

```
{  
  "name": "simple-project",  
  "description": "A Vue.js project",  
  "author": "tmvuejs",  
  "private": true,  
  "scripts": {  
    "dev": "cross-env NODE_ENV=development webpack-dev-server --open --inline --hot",  
    "build": "cross-env NODE_ENV=production webpack --progress --hide-modules"  
  },  
  "dependencies": {  
    "vue": "^2.0.1"  
  },  
  "devDependencies": {  
    "babel-core": "^6.0.0",  
    "babel-loader": "^6.0.0",  
    "babel-preset-es2015": "^6.0.0",  
    "cross-env": "^3.0.0",  
    "css-loader": "^0.25.0",  
    "file-loader": "^0.9.0",  
    "vue-loader": "^9.7.0",  
    "webpack": "^2.1.0-beta.25",  
    "webpack-dev-server": "^2.1.0-beta.0"  
  }  
}
```

At the bottom of the editor, there are tabs for "File 0", "Project 0", "No Issues", "package.json", and "1:1". On the right, there are buttons for "LF", "UTF-8", "2 Spaces", "JSON", and "7 updates".

Estructura de webpack-simple

Para nuestro ejemplo usaremos la plantilla de webpack completa, por lo que nuestro comando será:

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



Tip

Usa `vue list` para ver todas las plantillas oficiales disponibles.



Info

Cuando estás inicializando un nuevo proyecto se te pedirá que rellenes algunos detalles como el nombre, versión, autor, entorno, etc. Cada vez que usemos la cli para crear un nuevo proyecto seleccionaremos el entorno Runtime + Compiler, porque necesitamos compilar plantillas en el momento (p.ej. pasando una cadena de texto a la opción de la plantilla o montando a un elemento usando su DOM HTML como la plantilla.) Puedes encontrar explicaciones detalladas sobre los diferentes entornos en la guía¹²⁷.

En algún punto se te pedirá **escoger una preconfiguración de ESLint**. Las opciones disponibles son [feross/standard](#)¹²⁸ y [airbnb/javascript](#)¹²⁹.

Creé una tabla para comparar los dos estilos y tengas una mejor idea de que reglas aplica a cada uno.

Reglas	feross/standard	airbnb/javascript
Sangría	2 espacios	2 espacios
Punto y coma	No	Sí
VARIABLES no utilizadas	No permitido	No permitido
Comillas	Simples	Simples
Usar === en lugar de ==	Sí	Sí
Número de líneas vacías permitidas	1	2
Espacio luego del nombre de la función	Sí	No
Comenzar una línea con [No	Sí
Finalizar archivos con un carácter de nueva línea	Sí	Sí
Comas finales permitidas	No	No



Standard vs Airbnb

Las reglas de la tabla son algunas de muchas aplicadas en cada estilo. Para considerar y decidir que se ajusta mejor a ti, comprueba sus repositorios de Github.

Luego de que hayas seleccionado un estilo recibirás algunas indicaciones sobre instalar varias herramientas como [Karma-Mocha](#)¹³⁰ y [Nightwatch](#)¹³¹. No vamos a necesitar esas herramientas ahora, así que responde las preguntas de forma negativa y continúa.

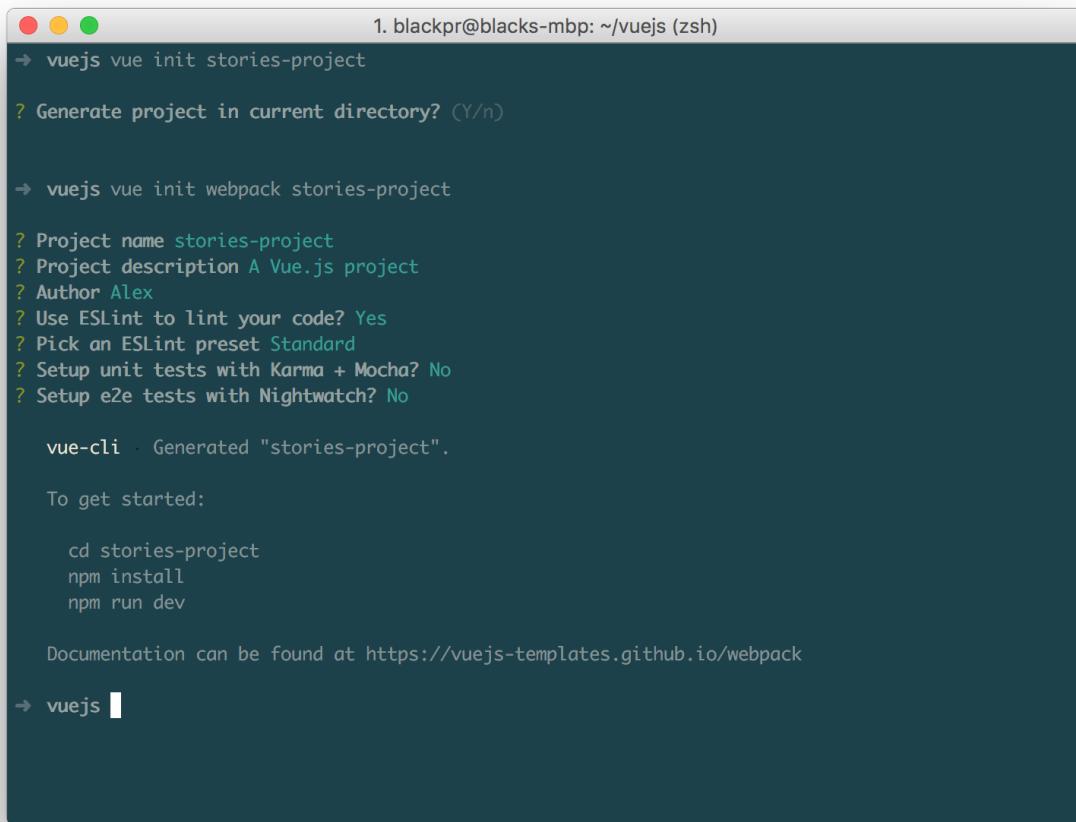
¹²⁷<https://vuejs.org/v2/guide/installation.html#Explanation-of-Different-Builds>

¹²⁸<https://github.com/feross/standard>

¹²⁹<https://github.com/airbnb/javascript>

¹³⁰<https://github.com/karma-runner/karma-mocha>

¹³¹<http://nightwatchjs.org/>



```
1. blackpr@blackpr-mbp: ~/vuejs (zsh)
→ vuejs vue init webpack 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 █
```

Instalación de la Plantilla de Vue



Info

Karma es un plugin y adaptador para el framework de pruebas [Mocha](#)¹³².

Nightwatch permite escribir pruebas automatizadas para el navegador, que se ejecutan sobre un servidor [Selenium](#)¹³³.

Plantilla de webpack

Para completar la configuración de nuestro proyecto necesitamos instalar sus dependencias. Sigamos adelante y ejecutemos:

¹³²<https://mochajs.org/>

¹³³<http://www.seleniumhq.org/>

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

La terminal muestra *Listening at http://localhost:8080*. Deberías esperar hasta que el mensaje `webpack: bundle is now VALID` es mostrado. ¡Entonces estás listo!

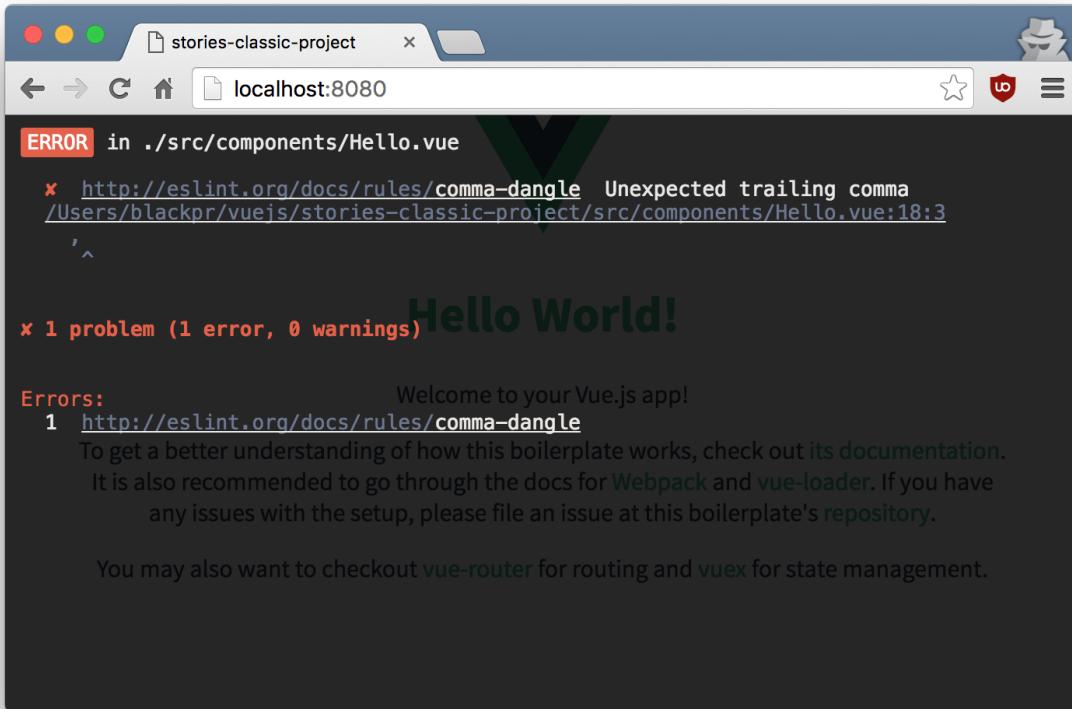
```
1. npm run dev (node)  
  └─ lodash._arraycopy@3.0.0  
    ├─ lodash._arrayeach@3.0.0  
    ├─ lodash._createassigner@3.1.1  
    | ├─ lodash._isiterateeceil@3.0.9  
    | └─ lodash.restparam@3.6.1  
    └─ lodash._getnative@3.9.1  
      └─ lodash.istypedarray@3.0.6  
        └─ lodash.toplainobject@3.0.0  
          └─ 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          [emitted]  
Child html-webpack-plugin for "index.html":  
  Asset      Size  Chunks      Chunk Names  
  index.html 21.4 kB      0  
webpack: bundle is now VALID.
```

Servidor en Ejecución...



Advertencia

Se cuidadoso, tienes que ser explícito en tu código. De otra manera recibirás errores por líneas extras vacías entre bloques, espacios finales, sangría de más de 2 espacios y otras cosas que no siguen las [reglas del estilo seleccionado](#).



Capa de Error



Nota

Si usas `webpack-simple` todavía tendrás las características básicas, pero la capa de error no se mostrará en el navegador, así que comprueba la terminal para saber si hay errores.

Estructura del Proyecto

Luego de completar los pasos anteriores deberías tener un directorio del proyecto completo con todos los archivos necesarios.

The screenshot shows a code editor window titled "main.js — /Users/blackpr/vuejs/stories-classic-project". The left sidebar displays the project structure:

- stories-classic-project
 - build
 - config
 - node_modules
 - src
 - assets
 - components
 - Hello.vue
 - App.vue
 - static
 - .babelrc
 - .editorconfig
 - .eslintrc.js
 - .gitignore
 - index.html
 - package.json
 - README.md

The main editor area shows the content of the `main.js` file:

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

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

At the bottom of the editor, there are status indicators: "File 0", "Project 0", "✓ No Issues", "src/main.js", "LF", "UTF-8", "2 Spaces", "JavaScript", and "7 updates".

Estructura de webpack

Los archivos, con los que usualmente trabajarás, son:

1. `index.html`
2. `main.js`
3. archivos dentro de los directorios `src` y `src/components`

index.html

Comenzemos con `index.html`. Debería verse así:

index.html

```
<html>
  <head>
    <meta charset="utf-8">
    <title>stories-classic-project</title>
  </head>
  <body>
    <app></app>
    <!-- los archivos compilados serán auto inyectados -->
  </body>
</html>
```

Como puedes ver, es una configuración muy básica con un componente ya incluido. El comentario hace referencia al archivo, `app.js`, que es generado por webpack. Básicamente quiere decir que luego de que webpack ha empaquetado el código, automáticamente se incluirá el archivo generado aquí, para que no tengas que incluirlo manualmente.

Hello.vue

Si estás siguiendo los pasos, dirígete a `src/components` y abre el archivo `Hello.vue` para ver como es un archivo `.vue`.

src/components/Hello.vue

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

<script>
export default {
  name: 'hello',
  data () {
    return {
      msg: 'Bienvenido a tu App de Vue.js'
    }
  }
}
</script>
```

```
<!-- Añade el atributo "scoped" para limitar el CSS a este componente únicamente -->
<style scoped>
h1, h2 {
  font-weight: normal;
}
...
</style>
```

Dentro de la etiqueta `<script>` el componente contiene sólo sus *datos*. No hay necesidad de definir una plantilla. Automáticamente será enlazada si el bloque `<template>` existe. El bloque `<template>` define la plantilla del componente, por supuesto. Piensa en `<template>` como una etiqueta `<template-hello>` en tu HTML.

Podríamos tener sólo el bloque `<script>`, pero de esta manera, no aprovecharíamos los beneficios de los componentes de un solo archivo.

Recuerda que `export`, una característica de ES6, es manejada por esas geniales herramientas (transpiladores y empaquetadores de módulos) que hemos instalado.



Advertencia

Cada archivo `.vue` no debe contener más de un bloque `<script>`. Cada plantilla debe contener exactamente **un elemento root**, como `<div id=hello>...</div>`, que encapsula a todos los demás elementos.

El script debe exportar un *objeto de opciones del componente de Vue.js*. Exportar un constructor extendido creado por `Vue.extend()` también es posible, pero un objeto plano es preferible.

ES6 nos permite tener cualquier número de exportaciones, pero en componentes de un solo archivo ese no es el caso.

Si tratas de hacer exportaciones adicionales recibirás una advertencia similar a esta: `[vue-loader] src/components/Hello.vue: named exports in /*.vue files are ignored.`

El bloque `<style>` define los estilos CSS, como era de esperar.

App.vue

El archivo `App.vue` está localizado en el directorio `src` y es el que contiene la plantilla principal de la aplicación. Este componente es usualmente responsable de incluir los otros componentes.

`App.vue` tiene algunas líneas más de texto y estilos, pero ya que nos estamos centrando en la estructura, lo hemos acortado un poco.

src/App.vue

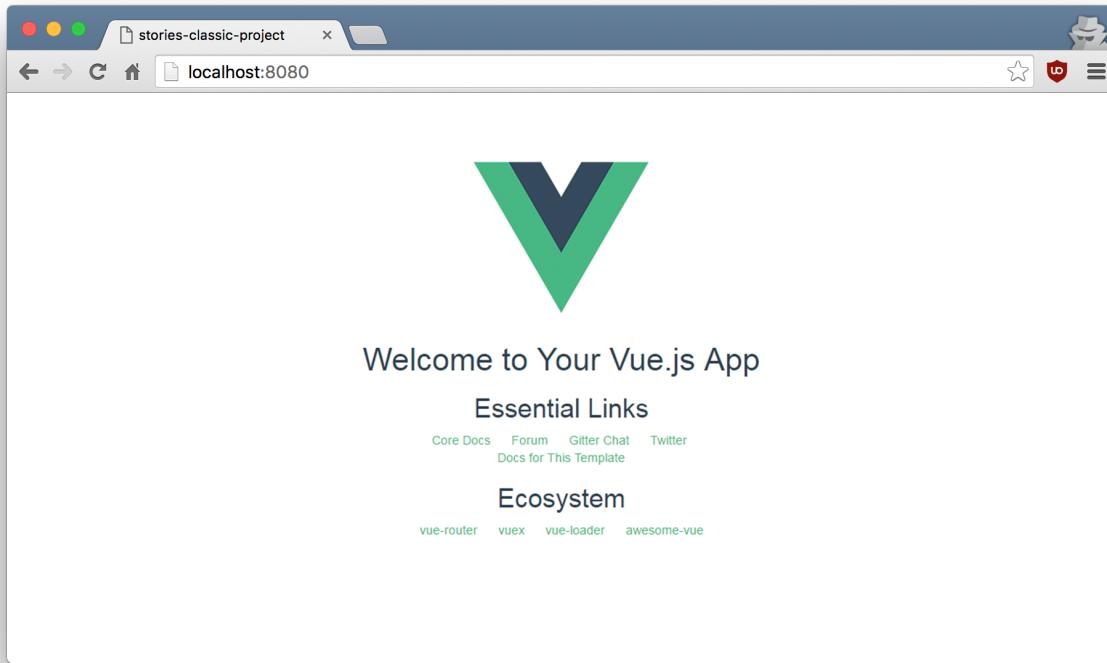
```
<template>
  <div id="app">
    
    <hello></hello>
  </div>
</template>

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

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

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

Tiene la misma estructura que el archivo *Hello.vue* que vimos arriba. Por defecto, hay un objeto **components** que contiene el componente `Hello`. Dentro de este objeto importaremos cualquier nuevo componente. En la plantilla está la etiqueta `<hello></hello>` y por lo tanto la plantilla del componente `Hello` será mostrada.



Página Principal del Proyecto

main.js

El archivo `main.js` dentro de `src`, como puedes imaginar, es nuestro script principal.

`src/main.js`

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

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

Este archivo importa `Vue` como un módulo desde `node_modules` y al componente `App` desde el directorio `src` también. Debajo está nuestra instancia de `Vue` y en el objeto de componentes se encuentra `App`.

Cuando necesites importar un script o un componente global puedes ponerlo dentro de `main.js`.



Nota

La propiedad `template: '<App/>'` representa la plantilla del componente, que va a ser inyectada al `index.html` que hemos mencionado anteriormente.

`<App/>` funciona igual que `<App></App>`* y **`<app></app>`.



Info

Puedes encontrar más información sobre la estructura del proyecto de la plantilla webpack en su [documentación](#)¹³⁴

Formando Archivos .vue

Hemos visto como se ve un componente de un solo archivo y como es usado en un proyecto. Es hora de crear algunos componentes en un escenario de la vida real. Supongamos que queremos crear algun tipo de red social o foro, donde los usuarios publican sus historias y experiencias. Para comenzar, vamos a necesitar 2 formularios, uno para el registro e inicio de sesión, y una página para mostrar las historias de los usuarios.

Antes de comenzar, incluiremos Bootstrap globalmente con el fin de poder utilizar sus estilos dentro de todos nuestros componentes. Para hacer eso, tenemos que actualizar nuestro `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>
    <!-- los archivos compilados serán auto inyectados -->
  </body>
</html>
```

Vamos a crear el formulario de inicio de sesión en un nuevo archivo, `Login.vue`.

¹³⁴<http://vuejs-templates.github.io/webpack/structure.html>

src/components/Login.vue

```
<template>
  <div id="login">
    <h2>Iniciar sesión</h2>
    <input type="email" placeholder="Correo electrónico">
    <input type="password" placeholder="Contraseña">
    <button class="btn">Ingresar</button>
  </div>
</template>

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

Y ahí está. Para poder ver el archivo en el navegador tenemos que incluir nuestro componente *Login* en alguna parte. Así que lo importaremos dentro del componente principal *App* y lo agregaremos a su objeto de componentes.

src/App.vue

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

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

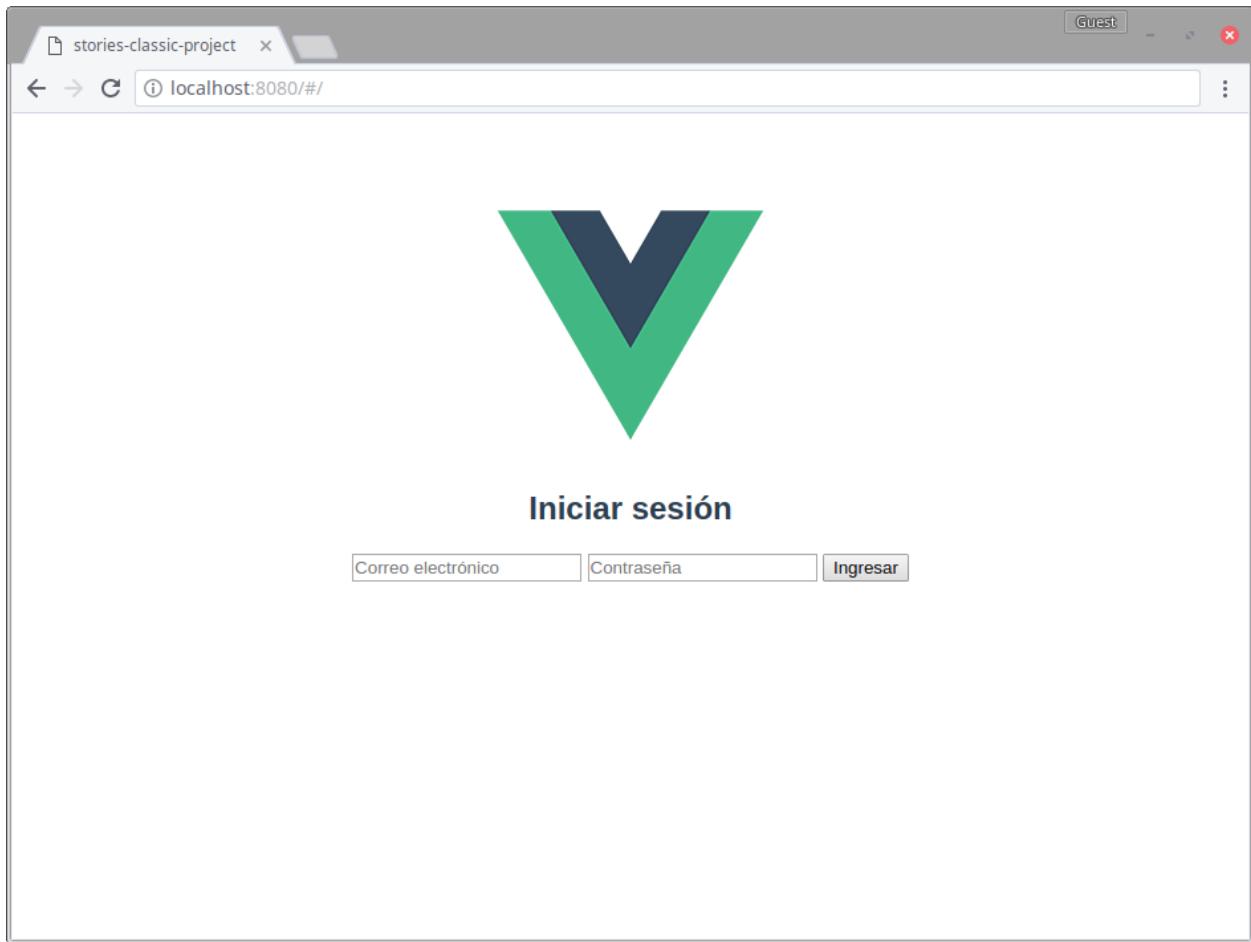
export default {
  name: 'app',
  components: {
    Hello,
    Login
  }
}
</script>
```

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

Si presionas recargar en tu navegador no verás el componente *Login* aún, ya que primero necesitamos hacerle referencia. Colocalo debajo de `<hello></hello>` ¡Y tendrás un genial formulario de login!

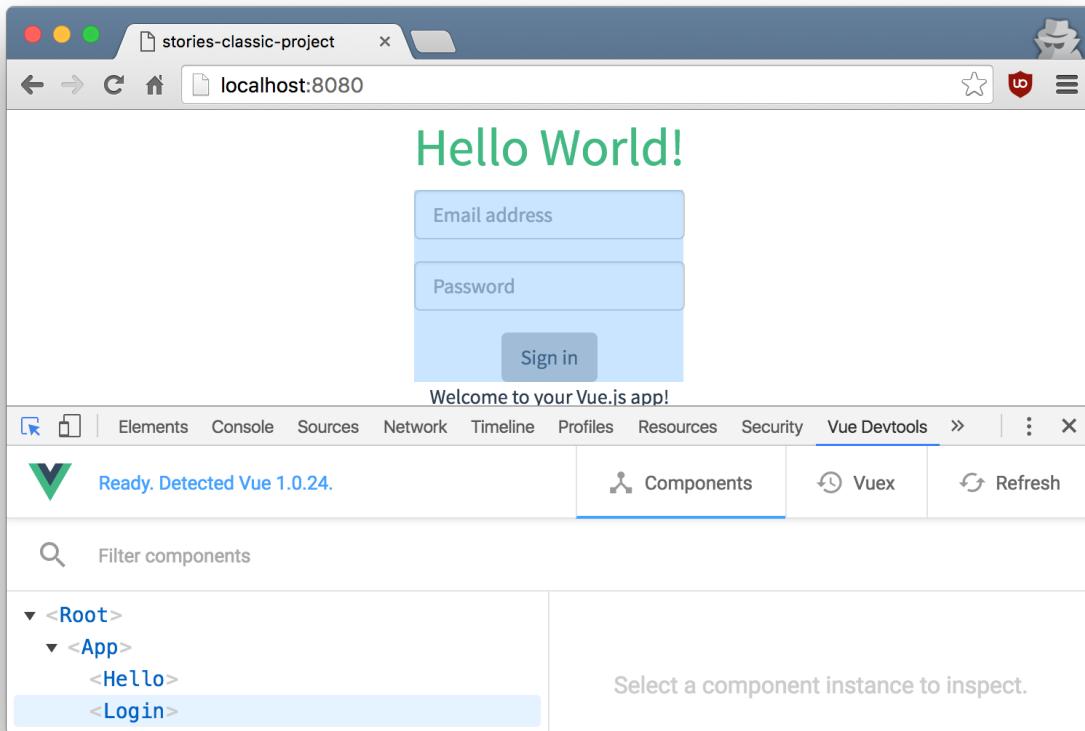
src/App.vue

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



Componente Login

Si abres la consola del navegador, deberías ver el mensaje `login` que estamos registrando cuando el componente es creado. Si estás usando vue-devtools, lo cual es altamente recomendable, también deberías verlo en la vista de árbol de los componentes.



Vista de Árbol

Vamos a crear otro componente, esta vez para el registro.

src/components/Register.vue

```
<template>
<div id="register">
  <h2>Formulario de Registro</h2>
  <input placeholder="Nombre" class="form-control">
  <input placeholder="Apellido" class="form-control">
  <input placeholder="Correo electrónico" class="form-control">
  <input placeholder="Escoge una contraseña" class="form-control">
  <input placeholder="Confirmar contraseña" class="form-control">
  <button class="btn">Registrarme</button>
</div>
</template>

<script>
export default {
  created () {
```

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

Entonces podemos importarlo en el archivo *App.vue*.

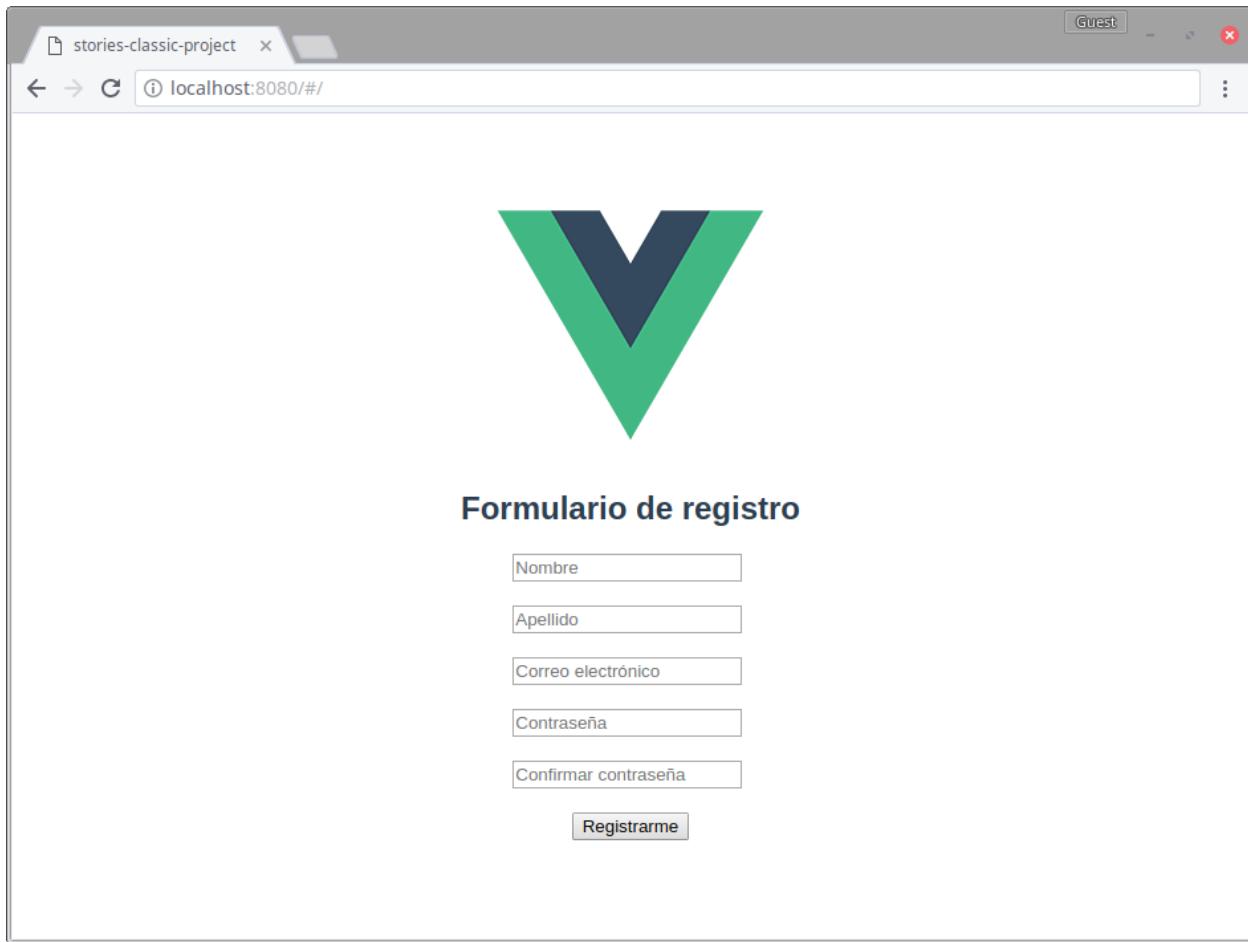
```
src/App.vue

<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>
...
...
```

La plantilla del componente Register aparece, cuándo comprobamos el navegador.



Componente Register



Nota

Los otros componentes están comentados porque no queremos mostrarlos uno debajo del otro. El componente `Hello` está ahí por defecto, pero no lo vamos a utilizar en ningún otro ejemplo, así que lo eliminaremos.

Dijimos que estamos trabajando en una red social (o algo similar), así que queremos un lugar para mostrar las historias. Por tanto, vamos a crear un componente `Stories` que cuando es renderizado, traerá todas las historias contadas por los usuarios.

src/components/Stories.vue

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

<script>
  export default {
    data () {
      return {
        stories: [
          {
            plot: 'Mi caballo es maravilloso.',
            writer: 'Mr. Weeb1',
            upvotes: 28,
            voted: false
          },
          {
            plot: 'Los narvales inventaron el Shish kebab.',
            writer: 'Mr. Weeb1',
            upvotes: 8,
            voted: false
          },
          {
            plot: 'El lado oscuro de la Fuerza es más fuerte.',
            writer: 'Darth Vader',
            upvotes: 52,
            voted: false
          },
          {
            plot: 'Uno simplemente no camina hacia Mordor.',
            writer: 'Boromir',
            upvotes: 74,
            voted: false
          }
        ]
      }
    }
  }
</script>
```

```
    }
</script>
```

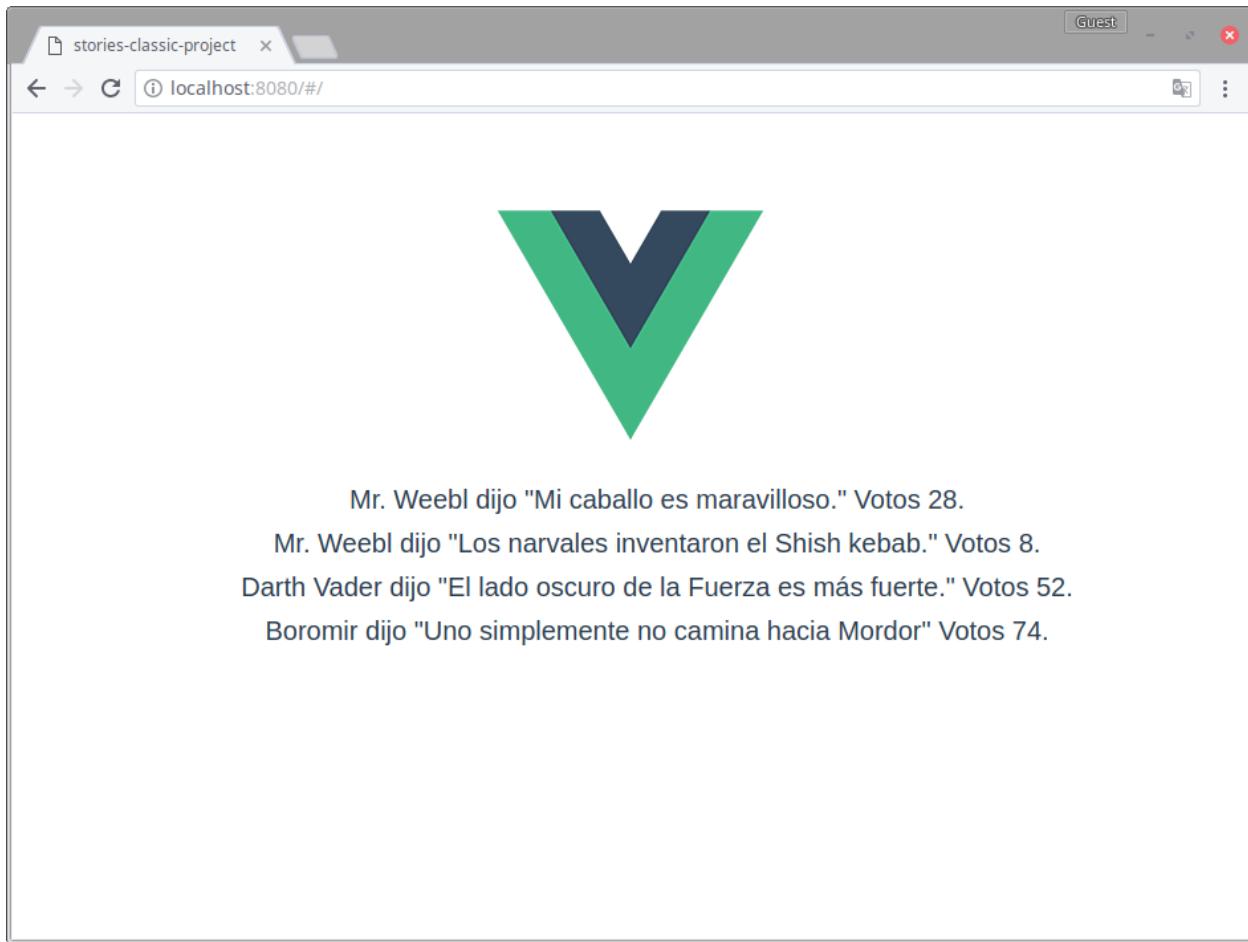
Este es el archivo *Stories.vue*. Podemos usarlo en nuestro archivo principal *App.vue*. En este punto las historias están escritas directamente en el código por simplicidad. Es hora de importarlo como los otros componentes.

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>
```



Componente Stories

¡Genial! Ahora tenemos una página para mostrar todos los listados.

Componentes Anidados

Nos gustaría ser capaces de mostrar las historias más “famosas”, en cualquier lugar que queramos. Así que luego de la creación del componente *Famous*, deberíamos ser capaces de usarlo en cualquier parte.

src/components/Famous.vue

```
<template>
  <div id="famous">
    <h2>Historias populares<strong>({{ famous.length }})</strong></h2>
    <ul class="list-group">
      <li v-for="story in famous" class="list-group-item">
        {{ story.writer }} dijo "{{ story.plot }}".
        Votos {{ story.upvotes }}.
      </li>
    </ul>
  </div>
</template>

<script>
export default {
  computed: {
    famous () {
      return this.stories.filter(function (item) {
        return item.upvotes > 50
      })
    }
  },
  data () {
    return {
      stories: [
        {
          plot: 'Mi caballo es maravilloso.',
          writer: 'Mr. Weebly',
          upvotes: 28,
          voted: false
        },
        {
          plot: 'Los narvales inventaron el Shish kebab.',
          writer: 'Mr. Weebly',
          upvotes: 8,
          voted: false
        },
        {
          plot: 'El lado oscuro de la Fuerza es más fuerte.',
          writer: 'Darth Vader',
          upvotes: 52,
          voted: false
        },
      ],
    }
  }
}
```

```

    {
      plot: 'Uno simplemente no camina hacia Mordor.',
      writer: 'Boromir',
      upvotes: 74,
      voted: false
    }
  ]
}
}
}

</script>

```

Este es todo el archivo `Famous.vue`. Hemos **filtrado** el arreglo `stories` usando computed properties, como vimos en capítulos anteriores, y creamos una **plantilla** para mostrarlas.



Note

El arreglo `stories` está escrito directamente en el código otra vez aquí y los datos son los mismos de antes. Esta es una mala práctica, encontraremos más adelante una forma de definir el arreglo `stories` una vez y compartirlo entre todos los componentes.

¿Pero donde podríamos usar este componente? Una idea es tenerlo dentro de la página de registro, así el usuario podría leer las historias más populares y estar intrigado. Esto significa - en el proyecto actual - que necesitamos tener el componente `Famous` dentro del componente `Register`. Bueno, esto se puede hacer de la misma forma que lo hicimos dentro de `App.vue`.

Así que abre `Register.vue`, importa el componente allí, y hazle referencia dentro de la plantilla.

`src/components/Register.vue`

```

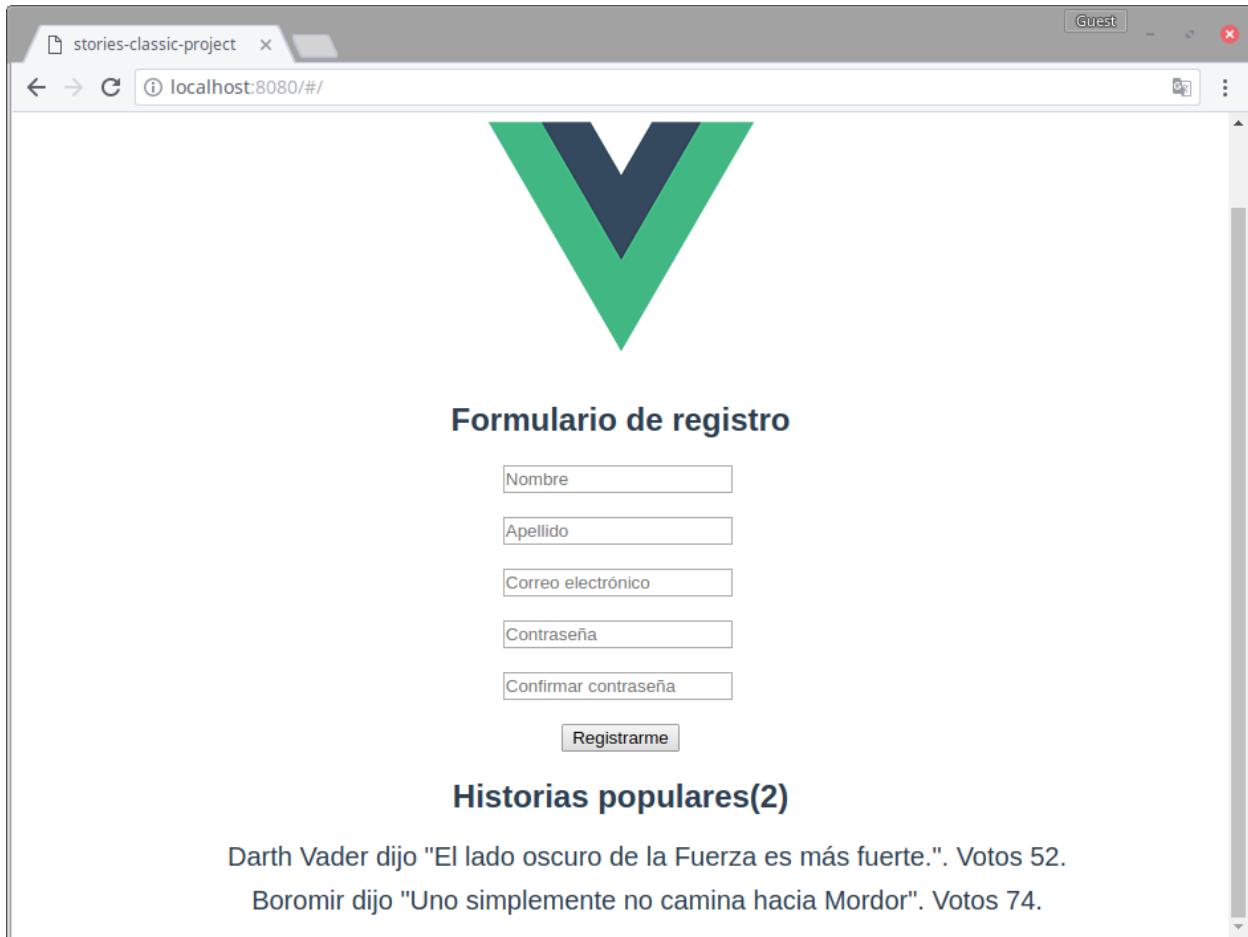
<template>
  <div id="register">
    <h2>Formulario de registro</h2>
    ...
    <famous></famous>
  </div>
</template>

<script>
  import Famous from './Famous.vue'

  export default {
    components: {
      Famous
    }
  }
</script>

```

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



Página de registro con las historias famosas

Presta atención a la ruta del archivo importado. Ahora que los dos archivos se encuentran en el mismo directorio tienes que usar `./Famous` en lugar de la ruta completa. Este es un error fácil de cometer, especialmente si no estás familiarizado con el.



Video

vue-cli y webpack son cubiertos a partir de la [lección 22 del curso de Vue 2 en Stye¹³⁵](#).

¹³⁵<https://stye.net/introduccion-a-vue-cli-y-webpack/>



Código de Ejemplo

Puedes encontrar el código de ejemplo de este capítulo en GitHub¹³⁶.

¹³⁶<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter16/16.3>

Eliminando Estado Duplicado

En los ejemplos anteriores escribimos los datos - el arreglo de historias - directamente en el código dentro de cada componente. **Esta no es una forma apropiada de trabajar con datos.**

Cuando más de un componente utiliza los mismos datos, es una buena práctica crear/obtener el arreglo una vez y luego encontrar una forma de compartirlo entre los componentes de la aplicación.

Stories.vue y *Famous.vue* están usando el mismo arreglo `stories`. Veremos dos formas de compartir los datos:

1. Usando las propiedades de los componentes.
2. Usando un almacenamiento global.

Compartiendo con Propiedades

Lo primero que vamos a hacer es mover el arreglo `stories` al componente App.

src/App.vue

```
1 <script>
2 ...
3
4 export default {
5   components: {
6     ...
7   },
8   data () {
9     // Aquí colocamos el arreglo stories
10    return {
11      stories: [
12        {
13          plot: 'Mi caballo es maravilloso.',
14          writer: 'Mr. Weebly',
15          upvotes: 28,
16          voted: false
17        },
18        {
19          plot: 'Los narvales inventaron el Shish kebab.',
20          writer: 'Mr. Weebly',
```

```
21      upvotes: 8,
22      voted: false
23    },
24    {
25      plot: 'El lado oscuro de la Fuerza es más fuerte.',
26      writer: 'Darth Vader',
27      upvotes: 52,
28      voted: false
29    },
30    {
31      plot: 'Uno simplemente no camina hacia Mordor',
32      writer: 'Boromir',
33      upvotes: 74,
34      voted: false
35    }
36  ]
37 }
38 }
39 }
40 </script>
```

El siguiente paso es eliminar `data()` de los componentes `Stories` y `Famous`, y declarar la propiedad `stories`.

Vamos a hacerlo para el primer componente.

`src/components/Stories.vue`

```
1 <script>
2   export default {
3     props: ['stories']
4   }
5 </script>
```

Tenemos que actualizar la forma en la que hacemos referencia a nuestro componente dentro de `App.vue`.

src/App.vue

```

1 <template>
2   <div id="app">
3     ...
4     <stories :stories="stories"></stories>
5     ...
6     <p>
7       ¡Bienvenido a tu app de Vue.js!
8     </p>
9   </div>
10 </template>

```

Aquí enlazamos la propiedad stories al arreglo stories.

Mr. Weebl dijo "Mi caballo es maravilloso." Votos 28.
 Mr. Weebl dijo "Los narvales inventaron el Shish kebab." Votos 8.
 Darth Vader dijo "El lado oscuro de la Fuerza es más fuerte." Votos 52.
 Boromir dijo "Uno simplemente no camina hacia Mordor" Votos 74.

The screenshot shows a browser window with the URL `localhost:8080/#`. Below the address bar, there's a large green and blue logo. Underneath it, four story cards are displayed, each containing a quote and its corresponding vote count. At the bottom of the browser window, there's a developer tools panel titled "Vue". The "Components" tab is selected, showing a tree view of the component hierarchy: <Root> -> <App> -> <Stories>. The <Stories> component is highlighted with a green background. To the right of the component tree, there are sections for "data" and "props". The "data" section shows a single object named "\$route". The "props" section shows an array named "stories" containing four objects, each representing a story with properties like "id", "title", "content", and "votes".

Respuesta identica, usando las propiedades

Bien, ¡tenemos nuestras historias otra vez, obtenidas desde el componente padre!

No podemos hacer lo mismo para el componente Famous porque no le estamos haciendo referencia dentro de `App.vue`. Tendremos que pasar nuestro arreglo al componente Register para poder pasarlo

a 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>Formulario de registro</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>
```

Esta implementación funciona, pero no es eficiente, porque el componente Famous **no es independiente**. Esto significa que no podemos usarlo en cualquier lugar que queramos, a no ser que pasemos los datos desde el componente principal *App.vue*.

En un escenario en el que un componente que no es independiente está profundamente anidado, vas a tener que pasar una propiedad inútil, de componente a componente, sólo para ser capaz de usarla. En nuestro caso, si queremos usar Famous dentro de los widgets de la barra lateral de Register, tendríamos que pasar el arreglo *stories* todo el tiempo.

App -> Register -> Sidebar -> WidgetX -> Famous

Almacenamiento Global

La “forma usando propiedades” se veía bien al principio, pero como se vio en el componente Famous, mientras un proyecto crece y los componentes se anidan dentro de otros, el manejo y distribución de datos entre ellos se vuelve realmente difícil de seguir.

Así que, hagamos los datos de nuestros ejemplos un poco más fáciles de manejar. Podemos extraer los datos de *stories* a un archivo *.js*, almacenarlos en una **constante** y luego importarlos a los lugares deseados.

Nombraré nuestro archivo *JavaScript store.js* y lo pondré en el directorio */src*.

src/store.js

```
1 export const store = {
2   stories: [
3     {
4       plot: 'Mi caballo es maravilloso.',
5       writer: 'Mr. Weebly',
6       upvotes: 28,
7       voted: false
8     },
9     {
10       plot: 'Los narvales inventaron el Shish kebab.',
11       writer: 'Mr. Weebly',
12       upvotes: 8,
13       voted: false
14     },
15     {
16       plot: 'El lado oscuro de la Fuerza es más fuerte.',
17       writer: 'Darth Vader',
```

```
18     upvotes: 52,
19     voted: false
20   },
21   {
22     plot: 'Uno simplemente no camina hacia Mordor',
23     writer: 'Boromir',
24     upvotes: 74,
25     voted: false
26   }
27 ]
28 }
```



Advertencia

La propiedad `stories` debe ser eliminada de todos los archivos, ya que hemos cambiado la forma en la que almacenamos los datos y pueden haber conflictos, que podrían romper nuestro código.

Luego de que hayamos almacenado todos los datos dentro de `store.js` podemos importarlo dentro de `Stories.vue` usando la sintaxis de módulos de ES6.

src/components/Stories.vue

```
1 <script>
2   import {store} from '../store.js'
3
4   export default {
5     data () {
6       return {
7         // nos dará acceso a store.stories
8         store
9       }
10    },
11    created () {
12      console.log('stories')
13    }
14  }
15 </script>
```

Debido a que estamos importando el objeto `store` también tenemos que cambiar la plantilla del componente.

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 }} dijo "{{ story.plot }}"
5       Votos {{ story.upvotes }}.
6     </li>
7   </ul>
8 </template>
```

Estamos usando `v-for` para renderizar los elementos del arreglo (`store.stories`). Nuestra lista de historias se muestra como antes.

Podríamos hacer lo mismo sin tener que cambiar la plantilla, enlazando el atributo `stories` del componente a `store.stories` directamente.

src/components/Stories.vue

```
1 <script>
2   data () {
3     return {
4       // Enlazar directamente a stories
5       stories: store.stories,
6     }
7   }
8 </script>
```

Lo mismo aplica para `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>
```

```
14      })
15    }
16  }
17 }
18 </script>
```

Si no hubieramos enlazado a stories, la propiedad computada `famous()` habría tenido que ser actualizada para filtrar `this.store.stories`.

Una vez que te acostumbras a trabajar con objetos globales creo que ¡Te va a encantar! :)



Código de Ejemplo

Puedes encontrar el código de ejemplo de este capítulo en [GitHub](#)¹³⁷.

¹³⁷<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter17>

Intercambiando Componentes

Usar componentes de un solo archivo es la forma más fácil de construir una Aplicación de Página Única con Vue. Hasta el momento hemos visto como configurar un nuevo proyecto, crear archivos `.vue` y gestionar estado duplicado. Ahora es momento de ver una forma de intercambiar componentes específicos de la vista.

Como referencia, en ejemplos anteriores, teníamos 3 componentes dentro de `App.vue` y algunos otros anidados dentro. Necesitamos encontrar una forma de intercambiar componentes dinámicamente, para que no sean renderizados en la página de forma simultanea.

Componentes Dinámicos

El atributo especial `is`

Podemos usar el elemento reservado `<component>` y usar el mismo punto de montaje para cambiar dinámicamente entre multiples componentes, usando el atributo especial `is`.

`src/App.vue`

```
<template>
  <div id="app">
    <component is="hello"></component>
    <p>
      Esto es muy útil...
    </p>
  </div>
</template>

<script>
import Hello from './components/Hello'
// El componente Hello retorna una plantilla que contiene un propiedad de datos "msg"
export default {
  components: {
    Hello
  }
}
</script>
```

Hemos creado un nuevo proyecto y modificado el archivo `Hello.vue`. Tenemos exactamente el mismo resultado de antes, pero aquí estamos usando el elemento `<component is="hello">`. El componente `Hello` está enlazado al atributo `is`. Para ver como esto funcionaría dinámicamente mira el siguiente ejemplo donde estamos cambiando entre 2 componentes diferentes al presionar en sus enlaces.

Primero, crea un componente similar con un mensaje diferente, llamado `Greet.vue`.

src/Greet.vue

```
<template>
  <div class="greet">
    <h1>{{ msg }}</h1>
  </div>
</template>

<script>
export default {
  data () {
    return {
      msg: '¡No! ¡Quiero usar el elemento <component>!'
    }
  }
}
</script>
```

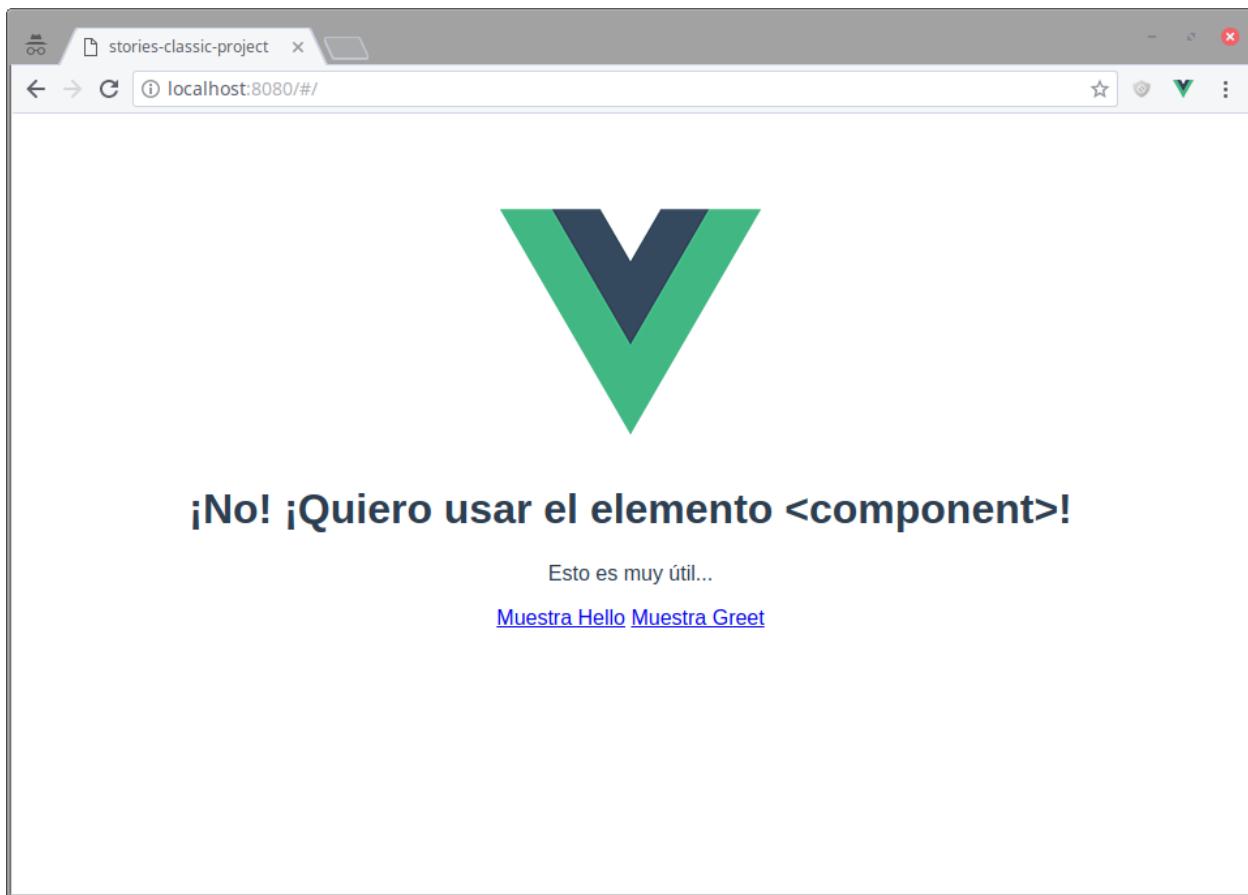
¡Hicimos que `Greet` muestre un mensaje para manifestar su presencia! Vamos a importarlo dentro de `App` y demos al usuario la posibilidad de intercambiar entre los 2 componentes.

src/App.vue

```
<template>
  <div id="app">
    
    <component :is="currentComponent">
      <!-- El componente cambia cuando this.currentComponent cambia -->
    </component>
    <p>
      Esto es muy útil...
    </p>
    <a href="#" @click="currentComponent = 'hello'">Muestra Hello</a>
    <a href="#" @click="currentComponent = 'greet'">Muestra 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

Bien, como puedes ver, estamos enlazando el atributo especial `is` a `currentComponent`, por lo que cuando su valor cambie, el componente mostrado también cambiará. Para intercambiar la vista, el usuario sólo tiene que presionar en cualquier enlace para cambiar el valor de `currentComponent`.

Esta forma dinámica de cambiar entre multiples componentes puede resultar útil.

Navegación

En los [ejemplos anteriores](#) usamos archivos `.vue` para simular una red social, donde teníamos componentes como `Login`, `Registration`, etc. Ahora podemos usar un sistema de pestañas para navegar con estilo entre los componentes.

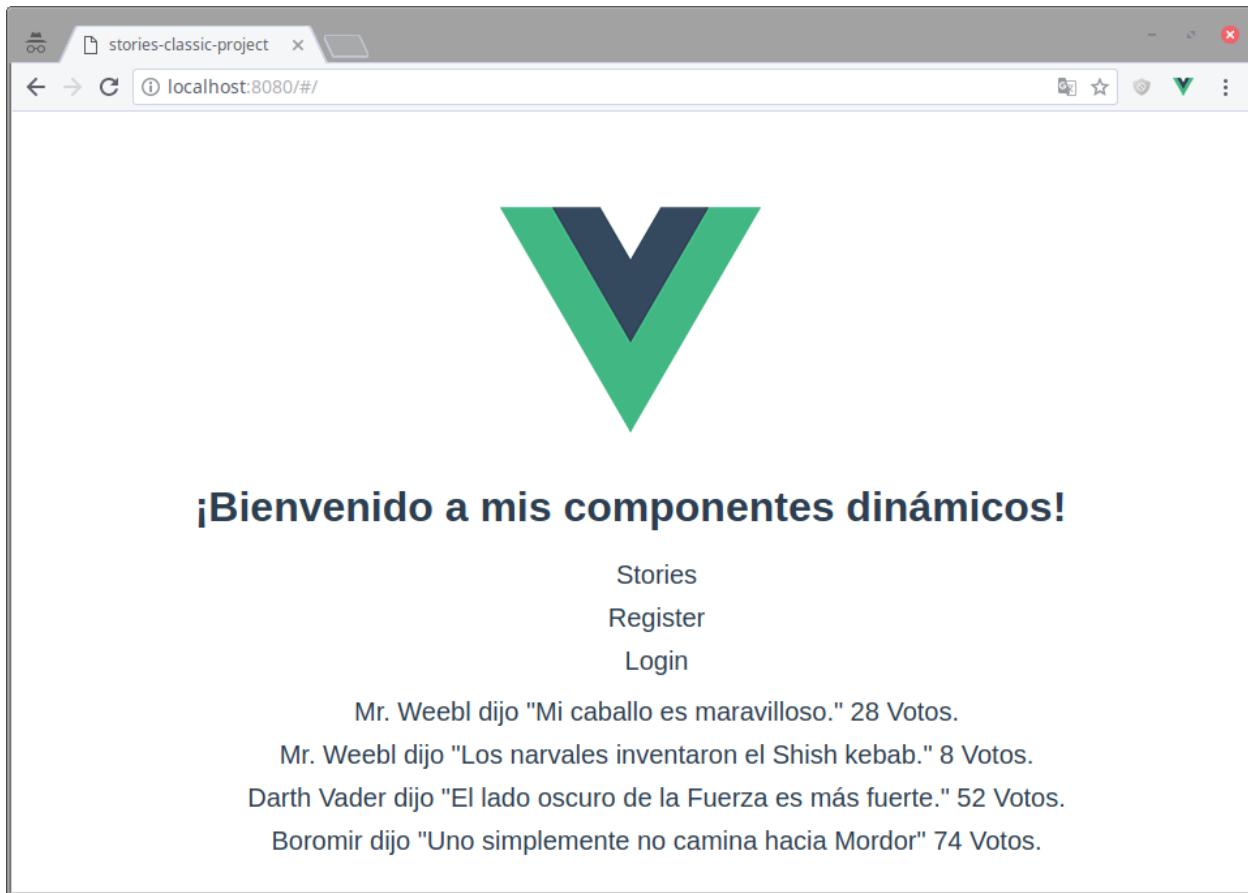
Vamos a tener `Stories.vue` en una pestaña, `Register.vue` en otra y `Login.vue` en una tercera. No olvides que `Register` contiene el componente `Famous`, que devuelve las historias más populares.

Lee cuidadosamente el siguiente ejemplo.

src/App.vue

```
1 <template>
2   <div id="app">
3     
4     <h1>iBienvenido a mis componentes dinámicos!</h1>
5     <ul class="nav nav-tabs">
6       <!-- Establecer la clase 'activa' condicionalmente --&gt;
7       &lt;li v-for="page in pages" :class="isActivePage(page) ? 'active' : ''"&gt;
8         <!-- Usar enlaces para cambiar entre pestañas --&gt;
9         &lt;a @click="setPage(page)"&gt;{{page | capitalize}}&lt;/a&gt;
10        &lt;/li&gt;
11      &lt;/ul&gt;
12      &lt;component :is="activePage"&gt;&lt;/component&gt;
13    &lt;/div&gt;
14  &lt;/template&gt;
15
16  &lt;script&gt;
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 {</pre>
```

```
27 components: {
28   Login,
29   Register,
30   Stories
31 },
32 data () {
33   return {
34     // Las páginas que queremos renderizar siempre
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>
```



Una página para cada componente

Vayamos por parte.

El arreglo llamado `pages` contiene los componentes que nos gustaría renderizar. Estamos usando la directiva `v-for` para crear una pestaña para cada uno.

Para navegar entre pestañas hemos creado un método llamado `setPage`.

La propiedad `activePage` está inicialmente establecida a '`stories`'. Cuando una pestaña es presionada, `activePage` cambia para reflejar el nombre del componente que queremos mostrar.

Para determinar que pestaña debe estar activa un `if` en línea es aplicado, que establece la clase `active` si la propiedad `activePage` coincide con el nombre del componente actual.

Para hacer que la primera letra de cada pestaña sea mayúscula hemos creado un `Vue.filter()` llamado `capitalize` que es usado dentro de las interpolaciones de texto.

Con estas pocas y simples líneas de código hemos logrado un sencillo sistema de navegación, intercambiando entre nuestros componentes.



Video

El uso de componentes es cubierto a partir de la lección 17 del curso de Vue 2 en Styde¹³⁸.



Código de Ejemplo

Puedes encontrar el código de ejemplo para este capítulo en [GitHub](#)¹³⁹.

¹³⁸<https://styde.net/creacion-y-uso-de-tu-primer-componente-con-vue-js-2/>

¹³⁹<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter18>

Vue Router

El enrutamiento, en general, se refiere a determinar cómo tu aplicación responde a la solicitud de un cliente. La solicitud de un navegador web no sería dirigida a tu aplicación sin algún tipo de enrutamiento. Un enrutador o “router” ayuda a un servidor web a obtener información apropiada y exacta para el usuario. Es como el oficial a cargo en una estación de trenes, que informa al operador del tren cuando cambiar de vía.

La forma de intercambiar vistas que acabamos de estudiar, prepara el camino para el enrutamiento. El enrutador oficial de Vue.js es llamado **vue-router**. Está profundamente integrado con el núcleo de Vue.js para hacer fácil la construcción de Aplicaciones de una Sola Página. Este plugin es relativamente fácil de entender, instalar y usar.

Las características principales son:

- Mapeo de ruta/vista anidado
- Configuración del enrutador modular y basado en componentes
- Parámetros de rutas, consultas, comodines
- Efectos de transiciones de vistas proporcionado por el sistema de transiciones de Vue.js
- Control de navegación detallado
- Enlaces con clases de CSS activas de forma automática
- Modo de historial de HTML5 o modo hash, con compatibilidad automática para IE9
- Restaura la posición del desplazamiento (scroll) al ir atrás en modo de historial

Info

Estas son sólo algunas de las características proporcionadas, puedes ver más en [Github¹⁴⁰](#). También, aquí está la [documentación oficial¹⁴¹](#).

Instalación

Este plugin se puede instalar de las maneras usuales; usando un CDN, NPM y Bower. Vamos a usar la terminal para instalarlo vía NPM.

```
>_ npm install vue-router
```

¹⁴⁰<https://router.vuejs.org/es/>

¹⁴¹<http://router.vuejs.org/en/index.html>

Escribe este comando en tu terminal para instalarlo en tu carpeta `node_modules` dentro del directorio de tu proyecto. Luego de completada la instalación, ve al archivo `main.js` y agrega las siguientes líneas.

src/main.js

```
import Vue from 'vue'  
import VueRouter from 'vue-router'  
Vue.use(VueRouter)
```

Puedes instalar un plugin de Vue.js usando `Vue.use()` como en el ejemplo. Para más información sobre `Vue.use` revisa la guía¹⁴².

Uso

El primer paso es crear una instancia del enrutador, pasaremos más opciones luego, pero vamos a mantenerlo simple por ahora:

src/main.js

```
...  
const router = new VueRouter({  
  routes // abreviatura para routes: routes  
})
```

Ahora, tenemos que definir algunas rutas. Cada ruta debería estar mapeada a un componente, lo que significa que vamos a crear rutas para los archivos `*.vue` que hemos estado utilizando todo este tiempo en nuestros ejemplos.

El método principal para definir mapeos de ruta para el enrutador, es crear un array `routes` donde pasaremos los objetos de ruta.

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 }  
]
```

¹⁴²<http://vuejs.org/api/#Vue-use>

Dentro del array definimos 2 rutas.

1. Cuando `http://localhost:3000/` (o cualquier puerto que podrías utilizar) se visita, el componente por defecto `Hello.vue` será renderizado
2. Cuando `http://localhost:3000/login` se visita, el componente `Login.vue` será renderizado.

Info

Estamos utilizando `{ path: '/login', component: Login }` porque el componente `Login` es importado en la parte superior del código. Alternativamente podrías usar la función `require()` así: `{ path: '/login', component: require('Login') }`

El siguiente paso es crear una salida para el enrutador. La guía indica que el enrutador necesita un componente raíz para renderizar. En nuestro caso, usaremos el componente `App.vue` como la raíz.

Vamos a crear y montar la instancia raíz. Ten en cuenta que tenemos que injectar el enrutador con la propiedad `router` para hacer que toda la aplicación sea *consciente de las rutas*.

`src/main.js`

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

Establecemos la plantilla a `<app> </app>` para que Vue reemplace el `div` con el id de `app` con la plantilla del componente `App.vue`. Dado que ya estamos importando `App` en nuestro archivo `main.js`, estamos listos aquí.

Si pasas por alto incluir el comentario `/* eslint-disable no-new */` obtendrás un error de `eslint` indicando:

<http://eslint.org/docs/rules/no-new> No use ‘new’ por efectos secundarios

Lo que tenemos que hacer aquí es desactivar esa regla.

Tip

Cada vez que te encuentras forcejeando con una regla de `eslint`, puedes desactivarla agregando un comentario como el anterior. Por ejemplo `/* eslint-disable eqeqeq */`. Puedes encontrar una lista completa con las reglas [aquí¹⁴³](#).

¹⁴³<http://eslint.org/docs/rules/>

Ahora, necesitamos definir algunos enlaces para la navegación. Dirígete a `App.vue`, para realizar los cambios requeridos.

src/App.vue

```
<template>
  <div id="app">
    
    <h1>iBienvenido al Enrutamiento!</h1>
    <router-link to="/">Página Principal</router-link>
    <router-link to="/login">Inicio de Sesión</router-link>
    <!-- Salida de ruta -->
    <router-view></router-view>
  </div>
</template>
```

`<router-view></router-view>` es el lugar donde toda la magia sucede y los componentes son renderizados para nosotros.

`router-link` es el componente que permite a los usuarios navegar en una aplicación habilitada para rutas. La propiedad `to` define la ubicación de destino, correspondiente a una ruta definida en `main.js`. Por defecto, `router-link` renderizará una etiqueta `<a>` con el atributo `href` establecido a la URI deseada. Pueden necesitarse más argumentos para navegaciones más complejas, como veremos pronto.



Nota

Si estás utilizando la plantilla de `vue-cli` con el enrutador incluido, tienes que definir tu array de rutas dentro de `router/index.js`.

Rutas con nombres

Mientras que las opciones de enrutamiento que vimos cubren nuestras necesidades en un proyecto pequeño, como nuestro ejemplo, probablemente necesitarás más opciones a medida que tu proyecto crece. Por ejemplo, si decidimos más adelante cambiar `/login` a `/signin`, necesitaremos actualizar todos los enlaces que dirigen a la página de inicio de sesión. Para evitar que esto suceda, podemos darle a cada ruta un nombre.

Para agregarle un nombre a una ruta tenemos que cambiar su configuración.

src/main.js

```
...
const routes = [
  {
    path: '/',
    name: 'home',
    component: Hello
  },
  {
    path: '/login',
    name: 'login',
    component: Login
  }
]
```

Podemos darle un nombre a una ruta agregando una propiedad `name` y usar esa propiedad como un identificador para enlazarla más tarde.

src/App.vue

```
<template>
  <div id="app">
    ...
    <router-link :to="{ name: 'home' }">Página Principal</router-link>
    <router-link :to="{ name: 'login' }">Inicio de Sesión</router-link>
    <!-- route outlet -->
    <router-view></router-view>
  </div>
</template>
```

Observa aquí que en lugar de usar un `string` para definir el destino del enlace (`to="/home"`), estamos usando un objeto (`:to="{ name: 'home' }"`). Explicaremos esto con más detalle más adelante.

Modo de historial

Antes de seguir adelante, me gustaría señalar algo relacionado a la URL del navegador, cuando usamos vue-router. Como has visto, cuando una ruta cambia, un símbolo '#' es anexado a la URL. Por ejemplo, la URL es `/*/login`, cuando navegamos a la página de inicio de sesión.

Esto es ocasionado por el modo predeterminado de vue-router, el modo hash, que usa el hash de la URL para simular una URL completa para que la página no sea recargada cuando la URL cambia. Para deshacernos del hash, podemos cambiar a `modo de historial`. También estableceremos otra

opción, **base**, que define una ruta raíz para todas las navegaciones del router. Cambiando esto a algo diferente a su valor inicial, que es **default**, resultará en rutas que siempre incluirán el nuevo valor en la URL actual del navegador.

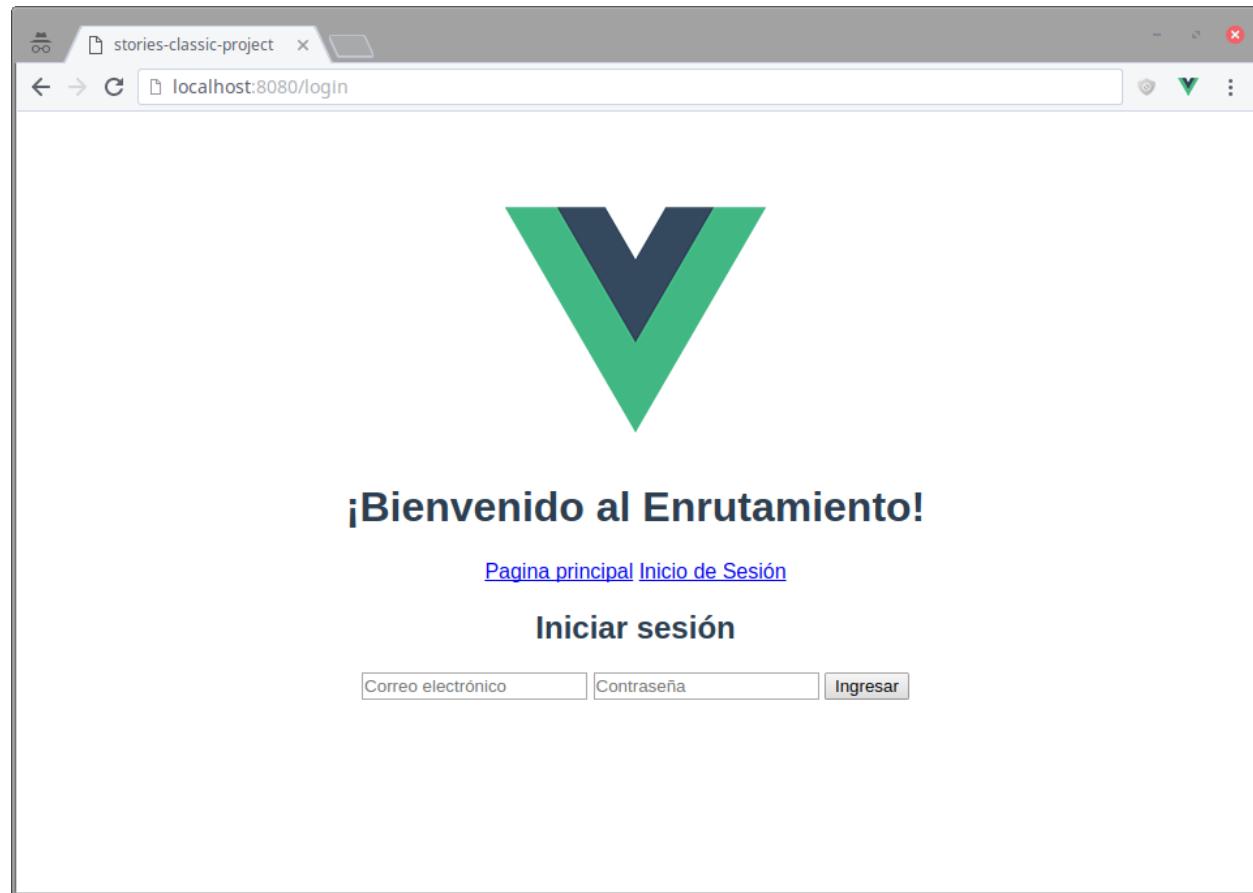
Por ejemplo si establecemos **base** a `/vuejs`, la página de inicio de sesión será `/vuejs/login`.

Puedes establecer cualquier cosa como la **base**, pongamos aquí simplemente `/`.

src/main.js

```
const router = new VueRouter({  
  mode: 'history',  
  base: '/',  
  routes  
})
```

Esto nos quita '#' de las URLs.



URLs Acomodadas



Info

Revisa la lista detallada de opciones disponibles en la [documentación de vue-router](#)¹⁴⁴

Rutas anidadas

Las rutas anidadas, son rutas que residen dentro de otras rutas. Mapear rutas anidadas a componentes es una necesidad común, y también es muy simple con vue-router.

Para demostrarlo, vamos a agregar una página para mostrar las historias con 2 subpáginas.

- Una para mostrar todas las historias (`StoriesAll.vue`)
- Una para mostrar las historias famosas (`StoriesFamous.vue`)

Crearemos los componentes antes mencionados, más uno, que será el contenedor, similar a `App.vue`.

Comencemos registrando las nuevas rutas. Todo lo que tenemos que hacer es usar la propiedad `children` en el array `routes` y agregar un `<router-view>` anidado dentro de nuestra vista contenedora, `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: 'all',
        component: StoriesAll
      },
      {
        path: 'famous',
        component: StoriesFamous
      }
    ]
  }
]

const router = new VueRouter({
  routes
})

const app = document.createElement('div')
app.innerHTML = ''
document.body.appendChild(app)

router.push('/')
```

¹⁴⁴<https://router.vuejs.org/es/api/options.html>

```
        component: Login
    },
{
  path: '/stories',
  component: StoriesPage,
  children: [
    {
      path: '',
      name: 'stories.all',
      component: StoriesAll
    },
    {
      path: 'famous',
      name: 'stories.famous',
      component: StoriesFamous
    }
  ]
}
```

Observa que aquí el `path` para `StoriesAll` está establecido a '''. Esto significa que es la ruta hija por defecto y será renderizada cuando `/stories` sea encontrada. También puedes usar '/' para definir una ruta por defecto.

El contenido de `StoriesFamous` será renderizado cuando `/stories/famous` sea encontrada.

En este punto, no hay necesidad de mostrar que hay dentro de estos componentes. Ambos simplemente muestran un array de historias.

Nuestro componente contenedor, `StoriesPage`, contiene 2 enlaces y la etiqueta `<router-view>`, para renderizar los contenidos de sus componentes hijos.

src/StoriesPage.vue

```
<template>
  <div>
    <h2>Historias</h2>
    <!-- navegación -->
    <router-link :to="{name: 'stories.all'}">Todas</router-link>
    <router-link :to="{name: 'stories.famous'}">Las Famosas</router-link>
    <!-- Salida de ruta -->
    <router-view></router-view>
  </div>
</template>
```

Clases activas Auto-CSS

¿No sería genial si resaltaramos el enlace que dirige a la página activa? Vue Router es suficientemente inteligente para adjuntar una clase css al enlace activo. Esta clase es `vue-router-active`.

Todo lo que tenemos que hacer es agregar una regla para darle estilo en nuestro CSS. Lo agregaré al componente `App.vue`.

`src/App.vue`

```
...
<style type="text/css">
  .router-link-active {
    color: green;
  }
</style>
```

Así que, ahora, cada vez que visitemos una página, el enlace correspondiente estará en verde.

Si pruebas esto en el navegador, verás que el enlace `Home` siempre es verde. Esto sucede porque la ruta de `Home` es `/`, así que cuando por ejemplo visitas `/login`, `Home` permanece activo. Para deshacernos de este comportamiento podemos agregar la propiedad `exact` a este enlace en específico.

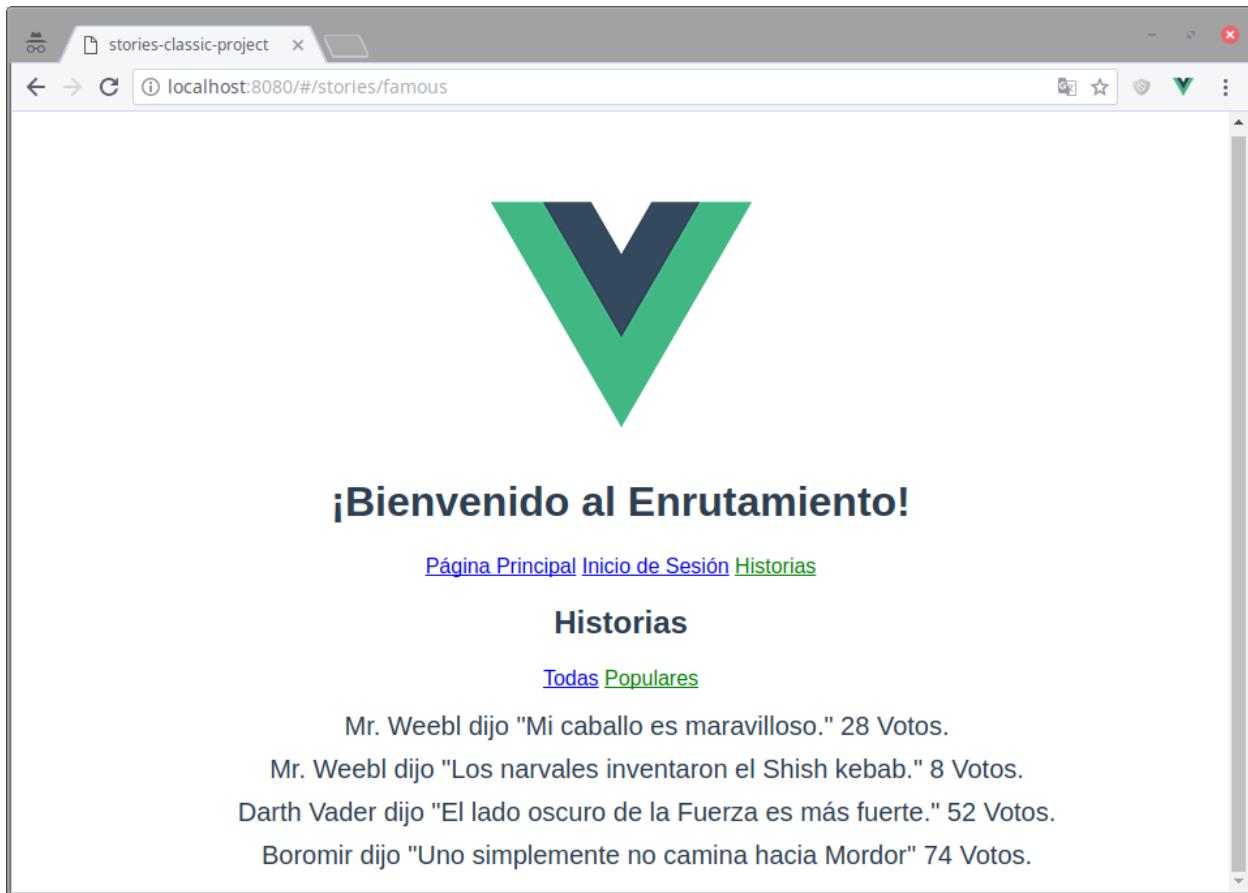
Nuestros enlaces de navegación se verán así:

`src/App.vue`

```
<template>
  <div>
    ...
    <router-link :to="{ name: 'hello'}" exact>Página Principal</router-link>
    <router-link :to="{ name: 'login'}">Iniciar Sesión</router-link>
    <router-link :to="{ name: 'stories.all'}">Historias</router-link>
    <router-view></router-view>
  </div>
</template>
```

También hemos anexado `exact` al primer enlace dentro de `StoriesPage.vue`.

La mejor parte es que el enlace activo también es resaltado en la navegación secundaria.



Clase Activa

Clases Activas Personalizadas

Puedes cambiar el nombre de la clase activa (`router-link-active`) para **un enlace en específico**, usando la propiedad `active-class` o **globalmente** usando la propiedad `linkActiveClass` del constructor del router.

Clase activa personalizada local

```
<router-link :to="{ name: 'hello'}" active-class="my-active-class" exact>
  Página Principal
</router-link>
```

Clase activa personalizada global

```
const router = new VueRouter({  
  mode: 'history',  
  base: '/',  
  linkActiveClass: 'my-active-class',  
  routes  
})
```

Objeto de Ruta

Toda la información de una ruta analizada será accesible en el **Objeto de Contexto de la Ruta** expuesto , también llamado **Objeto de Ruta**.

El objeto de ruta será inyectado en cada componente en una aplicación habilitada para enrutador y será accesible como `this.$route`. Será actualizado cada vez que una transición de ruta es realizada.

A continuación una lista con las propiedades del objeto `$route`.

Propiedad	Descripción
<code>path</code>	Una cadena de texto que equivale a la dirección de la ruta actual, siempre se resuelve como una ruta absoluta. p.ej. <code>/foo/bar</code> .
<code>params</code>	Un objeto que contiene pares llave/valor de segmentos dinámicos y segmentos estrella. Más detalles abajo.
<code>query</code>	Un objeto que contiene pares llave/valor de la cadena de consulta. Por ejemplo, para una ruta <code>/foo?user=1</code> , obtenemos <code>\$route.query.user == 1</code> .
<code>hash</code>	El hash de la ruta actual (sin #), si esta tiene uno. Si no hay un hash presente el valor será una
<code>fullPath</code>	Toda la URL resuelta incluyendo la consulta y hash.

Propiedad	Descripción		
<i>matched</i>	Un Array que contiene registros de ruta para todos los segmentos de ruta anidados de la ruta actual. Los	registros de las rutas son las copias de los objetos en el arreglo de configuración de las rutas (y en arreglos	hijos).
<i>name</i>	El nombre de la ruta actual, si tiene uno.		

Segmentos Dinámicos

Vue Router proporciona la habilidad de formar rutas usando segmentos dinámicos. Los segmentos dinámicos son segmentos antepuestos por dos puntos. Son llamados “dinámicos” porque su valor es variable.



Info

Los segmentos de URL son partes de la URL o ruta delimitados por barras diagonales. Si tuvieras la ruta `/user/:id/posts`, user, :id y posts serían cada uno un segmento.

En esta ruta, `:id` es el segmento dinámico y coincidirá con cualquier valor proporcionado, por ejemplo `/user/11/posts`, `/user/37/posts`, etc.

Cuando una dirección que contiene un segmento dinámico coincide con una ruta, el segmento dinámico estará disponible dentro de `$route.params`.

En nuestro ejemplo, podemos usar un segmento dinámico para acceder a una determinada historia por su `id`, para poder crear una vista donde podamos editarla.

src/main.js

```

1 const routes = [
2   // other routes
3   {
4     path: ':id/edit',
5     name: 'stories.edit',
6     component: StoriesEdit
7   }
8   ...
9 ]

```

Ahora, necesitamos una forma de enlazar `StoriesAll.vue` con `StoriesEdit.vue`. Vamos a manipular el archivo.



Nota

He creado el archivo StoriesEdit.vue en segundo plano. Lo verás pronto, luego de completar su enrutado.

src/component/StoriesAll.vue

```
1 <template>
2   <div class="">
3     <h3>Todas las Historias ({{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 }}"</h4>
8           <span class="badge">{{ story.upvotes }}</span>
9         </div>
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>
```



Nota

Nuestros archivos de ejemplo contienen casi lo mismo que antes, con alteraciones menores, mayormente de estilo, que no afectan su funcionalidad. Un cambio notable es la adición de un `id` a cada historia dentro de `store.js`.

Hemos agregado un botón para enlazar la ruta `stories.edit`. Bien, esto no es suficiente, porque también necesitamos pasar el `id` de la historia a la ruta.

Para hacerlo, vamos a editar la propiedad `to` y hacer que el `:id` se convierta en el `id` correspondiente de cada historia.

src/component/StoriesAll.vue

```
1 <template>
2   <div>
3     <h3>Todas las Historias ({{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>
```

Aquí queremos que `<router-link>` renderice una etiqueta `<button>` en lugar de `<a>`. Podemos usar la propiedad `tag` para especificar que etiqueta renderizar. La etiqueta renderizada escuchará eventos clic para la navegación.

Dentro de `$route.params`, el `id` de la historia escogida está disponible cuando alcanzamos a nuestro destino. Con esto a mano, podemos elegir la historia que el usuario quiere editar y traerla a él.

Es hora de mostrar el archivo `StoriesEdit.vue`, donde la edición tomará lugar.

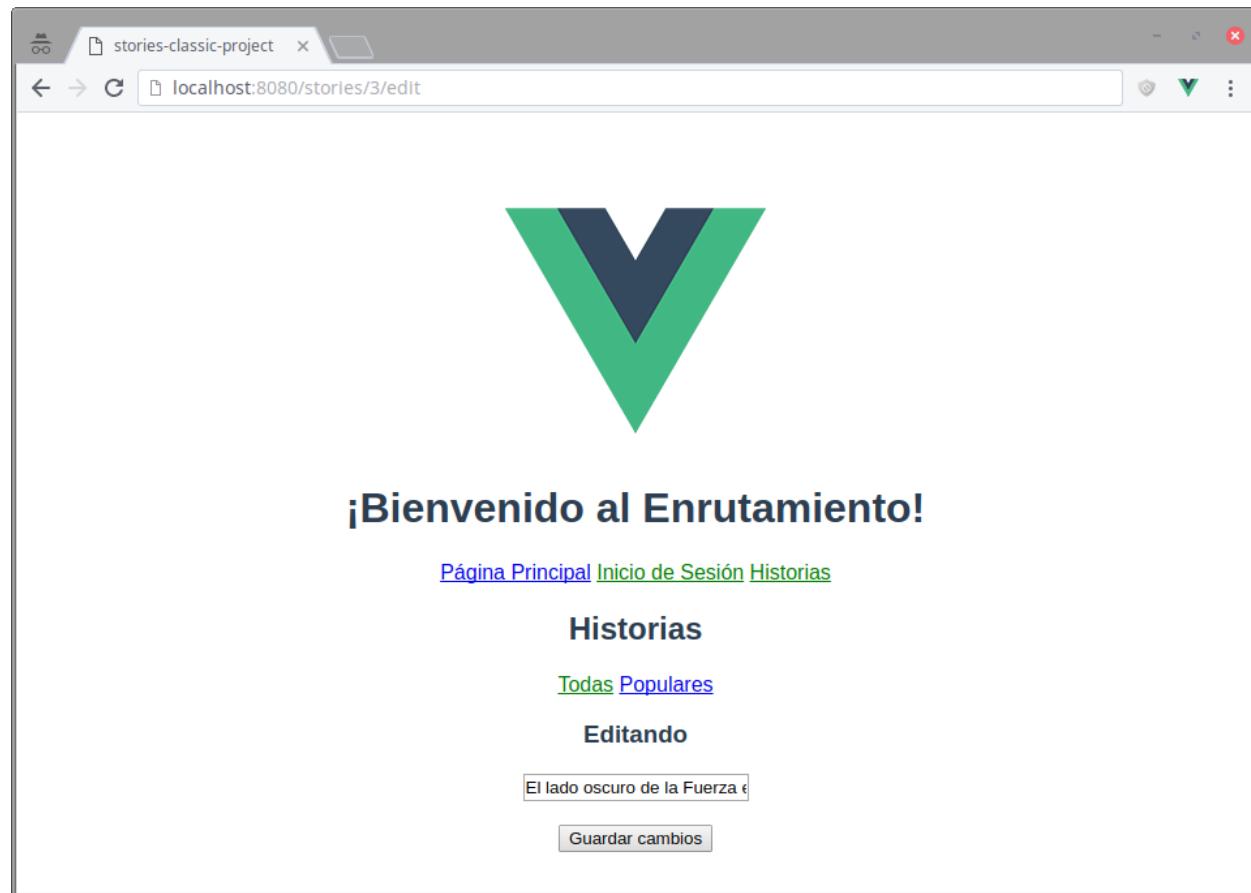
src/components/StoriesEdit.vue

```
1 <template>
2   <div class="row">
3     <h3>Edición</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          Guardar Cambios
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       // Usaremos esto más adelante
32     }
33   },
34   mounted () {
35     this.story = store.stories.find(this.isTheOne)
36   }
37 }
38 </script>
```

Estamos usando el id de la historia para elegir la historia deseada del array de stories con el

método `find145` de JavaScript.

La historia escogida está lista para edición. Observa que el `id` de la historia es mostrado en la URL.



Editando la historia seleccionada

Esto funciona bien si visitas la página `StoriesEdit` usando el botón pero si intentas escribir la URL directamente en la barra de direcciones del navegador p.ej. `/stories/2/edit`, obtendrás un error y el componente no será renderizado.

La razón detrás de esto es que usamos `igualdad estricta146` (`==`) para encontrar la historia activa correcta. Al visitar la página directamente, el `id` es pasado como una cadena de texto (no número). Por tanto, `isTheOne` siempre devuelve false.

¹⁴⁵https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find

¹⁴⁶https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Identity

src/components/StoriesEdit.vue

```
isTheOne (story) {
  // devuelve false cuando this.id no es un número
  return story.id === this.id
}
```

La solución fácil es convertir el `id` a un número, usando el objeto contenedor de JavaScript `Number`.

src/components/StoriesEdit.vue

```
export default {
  ...
  mounted () {
    this.id = Number(this.$route.params.id)
  }
}
```

Si revisas esto en el navegador, encontrarás que la historia aparece como si hicieramos clic en el botón de editar.

Aunque, hay una solución mucho mejor que nos ayuda a desacoplar el componente `StoriesEdit` de `$route`. Podemos convertir el parametro `id` a número dentro de la configuración de la ruta y usarlo como una propiedad en el componente `StoriesEdit`.

src/main.js

```
const routes = [
  ...
  {
    path: ':id/edit',
    props: (route) => ({ id: Number(route.params.id) }),
    name: 'stories.edit',
    component: StoriesEdit
  },
  ...
]
```

Ahora, cuando `stories/:id/edit` es visitada, el enrutador pasará el valor numérico de `:id` como una propiedad al componente `StoriesEdit`. Lo que falta es definir la propiedad y eliminar `this.$route.params.id`. Por tanto, nuestro componente se verá así

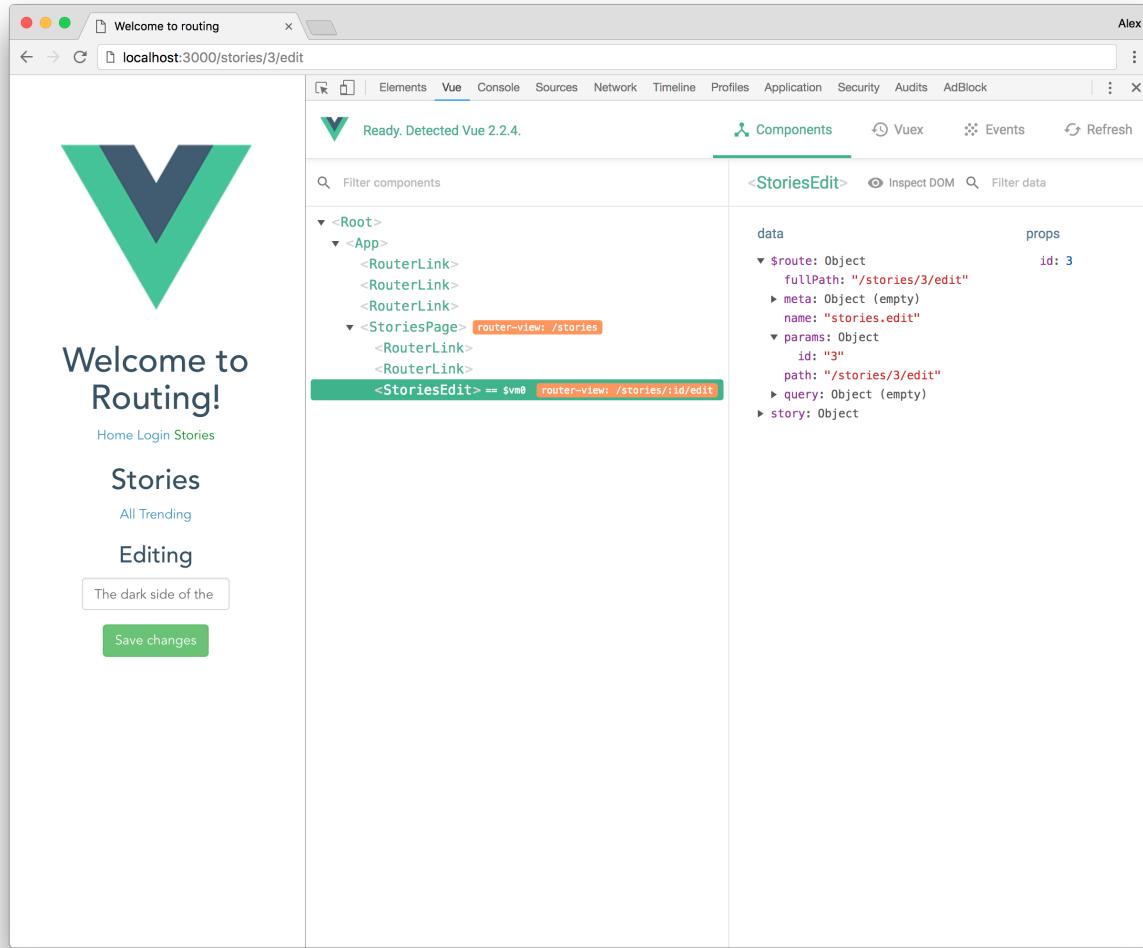
src/components/StoriesEdit.vue

```
<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>
```

Es todo. Ahora el componente StoriesEdit está desacoplado de la ruta y podemos usarlo en cualquier lugar así: `<stories-edit :id="story.id"></stories-edit>`.

Puedes darle un vistazo a los atributos del objeto `$route`, usando Vue Devtools.



Dentro de \$route

Alias de Ruta

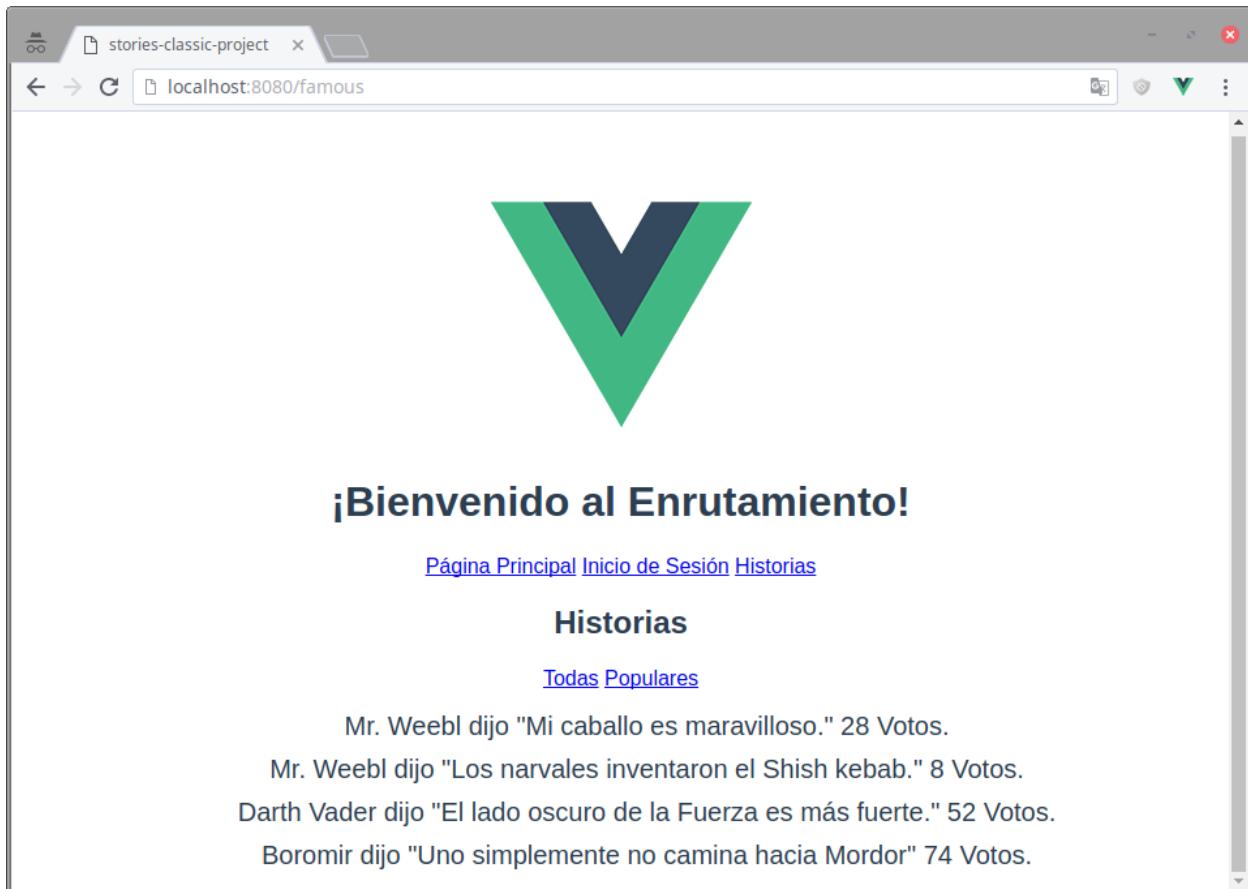
Cuando definimos nuevas rutas, generalmente tratamos de hacerlas claras y representativas. Sin embargo, algunas veces podemos terminar con direcciones largas o complejas, que pueden ser difíciles de manejar más adelante.

Cuando queremos ver las rutas de historias famosas a través del navegador, tenemos que visitar '/stories/famous'. Podemos hacer esto más corto definiendo un alias global para esta ruta, donde una URL más corta nos llevaría al mismo lugar.

src/main.js

```
{  
  path: 'famous',  
  name: 'stories.famous',  
  // Concuerda con '/famous' como si fuera '/stories/famous'  
  alias: '/famous',  
  component: StoriesFamous  
}
```

Usando esta configuración, podemos simplemente usar el alias en lugar del path.



Usando alias de ruta

Navegación Programática

En algún punto, queremos navegar a una ruta no mediante enlaces, sino programáticamente.

Para navegar a una ruta, podemos usar `router.push(path)`. La ruta puede ser tanto una cadena de texto o un objeto.

Si es una cadena de texto, la dirección debe estar en forma plana, lo que significa que no puede contener segmentos dinámicos. Por ejemplo `router.push('/stories/11/edit')`.

Si es un objeto, puedes pasar cualquier argumento necesario.

Navegación programática

```
router.push({ path: '/stories/11/edit' })
router.push({ name: 'stories.edit', params: { id: '11' } })
```

Podemos usar `router.push` para navegar a la página de la lista de historias luego de completar la edición.

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>
```

Hemos actualizado el método `saveChanges`. Cuando se llama, registra un mensaje en la consola y usando `this.$router.push()`, navega de vuelta a `/stories`.

Si quieres dirigir al usuario a la URL que visitó anteriormente, en lugar de una URL específica, puedes usar `router.back()`.

En nuestro caso podemos agregar un botón y llamar a la función `router.back`, en lugar de `router.push`, y esta vez el usuario podrá navegar a la página anterior (cualquiera que sea esta, por ejemplo <https://google.com>).

```
... <button @click="goBack">Regresar</button>  
methods: { ... goBack () { this.$router.back() }, ... }
```

Otra forma de hacer esto es usar el método `router.go(n)`, que toma un entero como parámetro el cual indica cuantas veces ir hacia adelante o hacia atrás en el historial¹⁴⁷.

```
goBack () {  
  this.$router.go(-1)  
}
```



Advertencia

Al usar `$router.back()` o cualquier otro método del router, estamos acoplando el componente al router. Si quieres mantenerlo desacoplado, puedes usar `window.history.back()`.

Transiciones

Introducción

Cada vez que navegamos a otra página de nuestra aplicación nada sofisticado sucede. Podemos cambiar eso utilizando una transición para animar el componente que entra en la página y también el que sale.

Vue proporciona una variedad de maneras para aplicar efectos de transición cuando elementos son insertados, actualizados o eliminados del DOM. Esto incluye herramientas para:

- automáticamente aplicar clases para transiciones y animaciones CSS
- integrar CSS de terceros o librerías de animaciones JavaScript, como [Animate.css¹⁴⁸](#), [Velocity.js¹⁴⁹](#), etc
- usar JavaScript para directamente manipular el DOM durante hooks de transiciones

En este punto, sólo cubriremos entrada y salida usando clases de CSS. Si estás interesado en aprender más sobre transiciones revisa la [guía¹⁵⁰](#).

¹⁴⁷<http://router.vuejs.org/en/essentials/history-mode.html>

¹⁴⁸<https://danedenv.github.io/animate.css/>

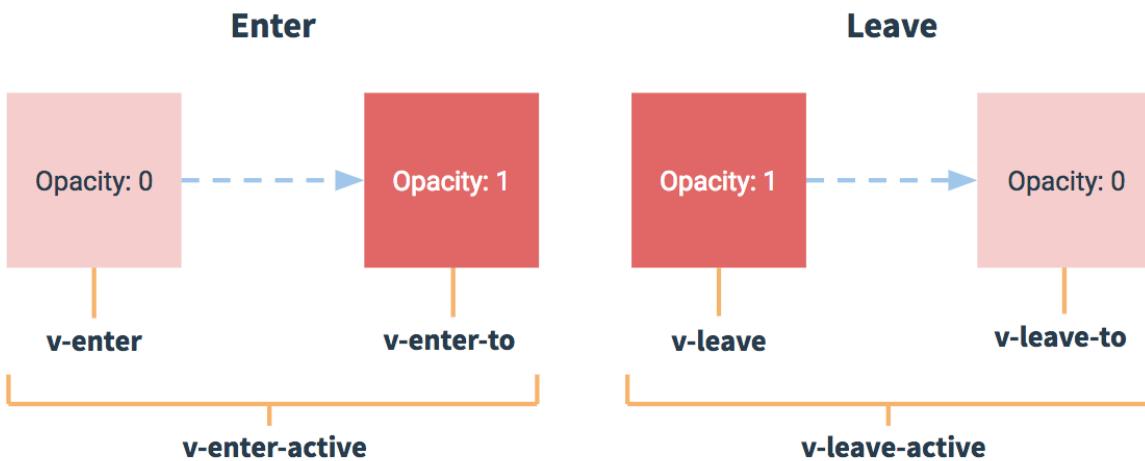
¹⁴⁹<http://velocityjs.org/>

¹⁵⁰<https://vuejs.org/v2/guide/transitions.html>

Para usar una transición, tenemos que encerrar el elemento correspondiente dentro del componente de transición. En nuestro caso, el componente `router-view`.

Vue agregará la clase CSS `v-enter` al elemento antes de que es insertado y `v-enter-active` durante la fase de entrada. `v-enter-to` es la última clase por ser agregada antes de que la transición esté completa. Este es realmente el estado final para entrar.

Respectivamente, cuando el elemento está siendo eliminado del DOM, `v-leave`, `v-leave-active`, y `v-leave-to` serán aplicadas.



Clases de Transición

Si un nombre para la transición es definido, todas las clases antes mencionadas tendrán el nombre en lugar de 'v'. Por ejemplo `fade-enter`, `fade-leave-to`, etc.

Uso

Usemos una transición, llamada `fade`, para nuestra salida de ruta.

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>
```

Echa un vistazo a las clases CSS. La transición comenzará con opacidad en 0 que aumentará gradualmente por 1 segundo. .fade-enter-to no es necesaria, definiéndola así creará un incremento, de 0.8 a 1, justo antes de que la transición termine.

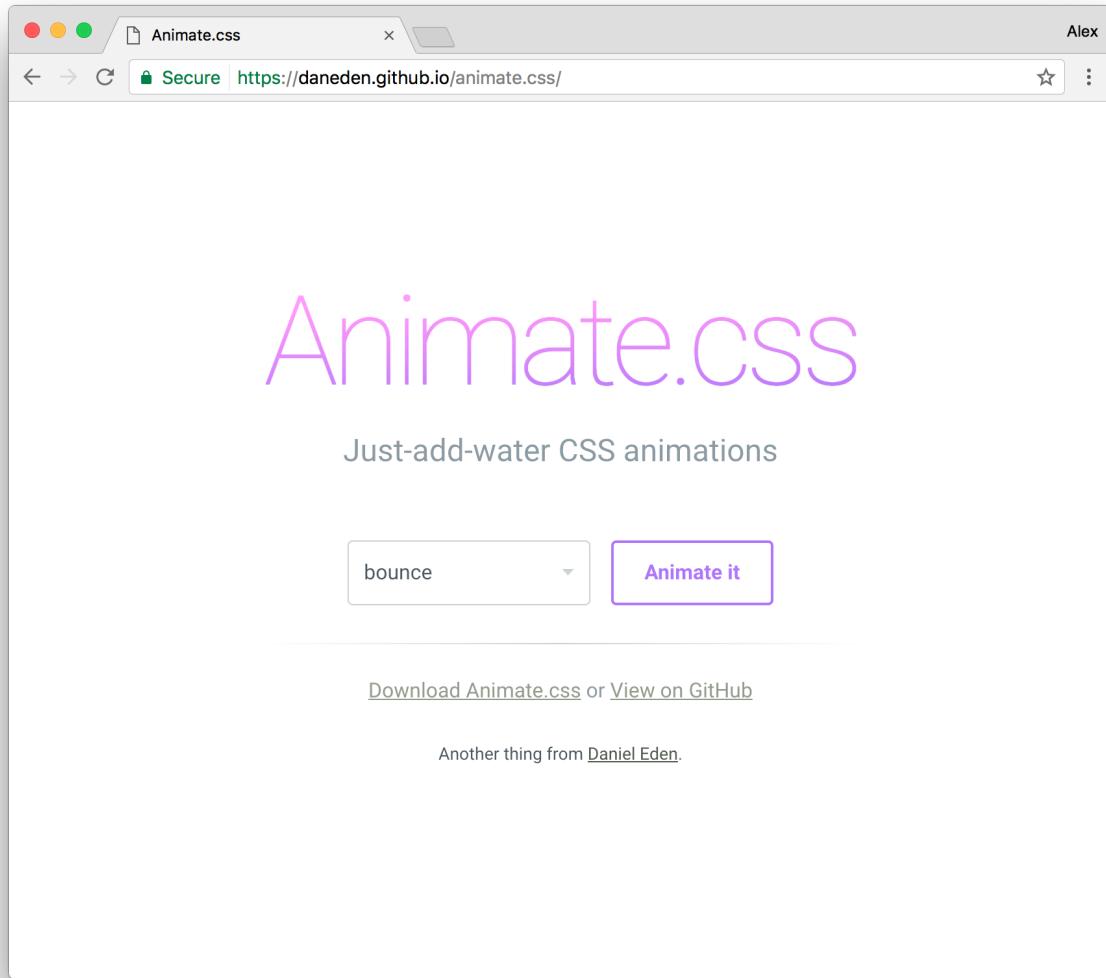
Para crear la animación inversa cuando un componente sale, tenemos que modificar nuestro CSS así:

```
<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>
```

Animaciones CSS de terceros

Crear animaciones desde cero, y diseñar en general, es una tarea difícil para mí. Siempre prefiero confiar en librerías de terceros. Afortunadamente para mí (y tal vez tú), usando transiciones de Vue es muy fácil integrar librerías de animación CSS y JS de terceros.

En este ejemplo usaré [Animate.css¹⁵¹](#).



Animate.css

De acuerdo a la documentación, para usar Animate.css tienes que:

1. Incluir la hoja de estilos en el <head> de tu documento.

¹⁵¹<https://daneden.github.io/animate.css/>

2. Agregar la clase `animated` al elemento que quieras animar. También puede que quieras incluir la clase `infinite` para un bucle infinito.
3. Finalmente debes agregar una de las clases disponibles, como `bounce`, `rollIn`, `fadeIn`, etc. Puedes encontrar una lista con todas las clases disponibles [aquí¹⁵²](#).

Comencemos por importarlo desde CDNJS, en la cabecera de nuestro archivo `index.html`.

En lugar de depender de la propiedad `name` del componente `transition`, esta vez vamos a usar una clase personalizada para `v-enter-active`.

`src/App.vue`

```
<template>
  <div>
    ...
    <transition enter-active-class="animated rollIn">
      <router-view></router-view>
    </transition>
  </div>
</template>
```

También podemos aplicar una animación cuando el componente abandona el DOM agregando un `leave-active-class` personalizado, así: `leave-active-class="animated rollOut"`.

Guardias de Navegación

Vue Router proporciona un mecanismo conveniente para filtrar transiciones. Para filtrar una transición puedes usar `router.beforeEach()` que es activada antes de cada transición, y `router.afterEach()`, que es activada después, sin embargo `afterEach()` no puede afectar la navegación.

`router.beforeEach()` puede ser útil en un escenario de autorización. Por ejemplo si un usuario no tiene permiso para acceder a una página de tu aplicación, debería ser dirigido a la página de inicio de sesión.

Veamos como podemos lograr eso en un breve ejemplo.

¹⁵²<https://github.com/daneden/animate.css>

src/main.js

```
1 // Crear un objeto para representar un usuario
2 let User = {
3   isAdmin: false
4 }
5
6 router.beforeEach((to, from, next) => {
7   if (to.path !== '/login' && !User.isAdmin) {
8     // Si no va a iniciar sesión y tampoco es un administrador, redireccionar a inicio\
9     de sesión
10    next('/login')
11  } else {
12    // Pero si está autorizado, proceda
13    next()
14  }
15 })
```

Aquí aplicamos una regla para que el router no deje a los usuarios acceder a ninguna página excepto login. Asegurate de siempre llamar a la función `next()`, de lo contrario el hook nunca será resuelto.



Video

Vue Router es cubierto a partir de la lección 33 del curso de Vue 2 en Styde¹⁵³.



Código de Ejemplo

Puedes encontrar el código de ejemplo para este capítulo en [GitHub](#)¹⁵⁴.

¹⁵³<https://styde.net/installacion-y-uso-de-vue-router-en-vue-js-2/>

¹⁵⁴<https://github.com/hootlex/the-majesty-of-vuejs-2/tree/master/codes/chapter19>

Ejercicios

A lo largo de este capítulo analizamos muchas cosas y ha pasado un tiempo desde que te hemos asignado ejercicios!

La aplicación que tienes que construir es un mini Pokédex.

La página de inicio mostrará una lista de categorías de Pokémon, como *Fuego*, *Agua*, etc. A partir de ahí, el usuario será capaz de navegar una categoría, ver su Pokémon y agregar nuevos.

Tus rutas pueden ser algo como esto:

Ruta	Descripción
/	Lista las categorías.
/category/:name	Muestra los Pokémons de la categoría.
/category/:name/pokemons/new	Agrega un nuevo Pokémon a la categoría.

Cada transición debe ser registrada en la consola. Por ejemplo, cuando el usuario decide navegar la categoría *Fire*, un mensaje tiene que ser registrado, informando al usuario que va a visitar /category/Fire.

Hemos creado el objeto Pokédex para ayudar a comenzar. Puedes encontrarlo [aquí¹⁵⁵](#).



Info

La ruta /category/:name/pokemons/new es una subruta de /category/:name.

Cuando el usuario visita /category/:name/pokemons/new debería ver un formulario para agregar un nuevo Pokémon junto con el listado de los Pokemones de la categoría.



Pista 1

Para acceder un Pokémon de una categoría en específica, considera usar el [método find¹⁵⁶](#) de JavaScript.

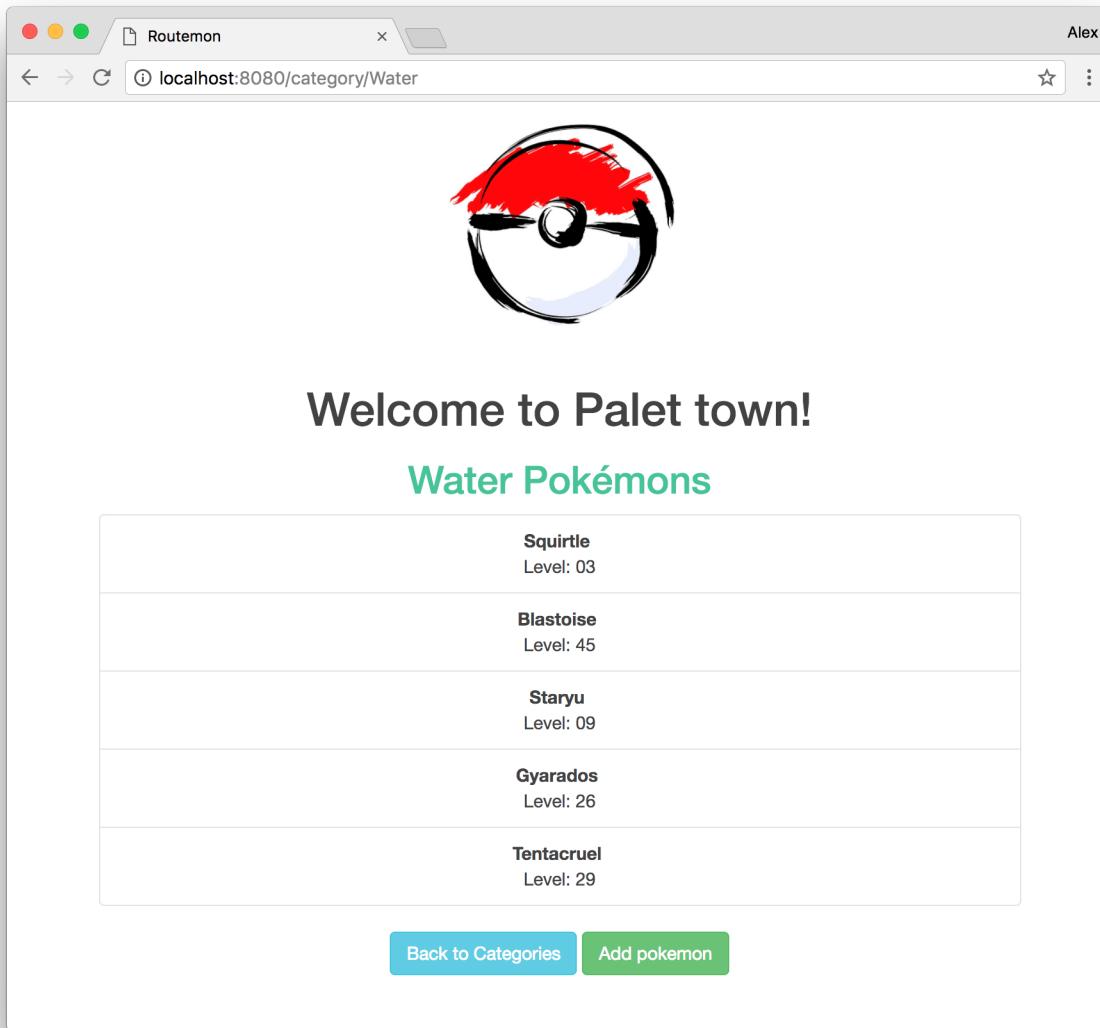


Pista 2

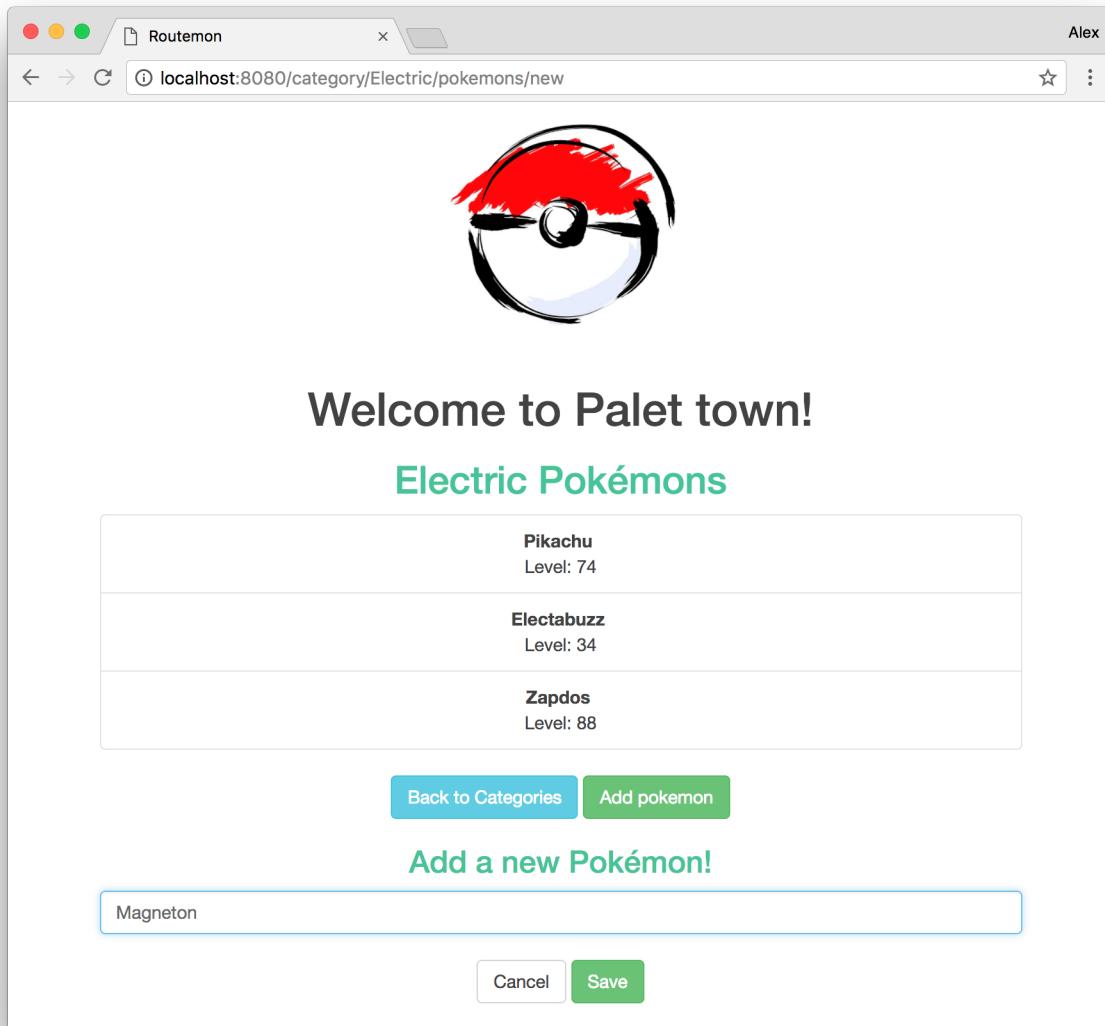
Para registrar un mensaje en la consola antes de cada transición usa `router.beforeEach()`.

¹⁵⁵<https://github.com/hootlex/the-majesty-of-vuejs-2/blob/master/homework/Chapter19/pokedex.js>

¹⁵⁶https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find



Pantalla de Ejemplo



Pantalla de Ejemplo

Puedes encontrar una posible solución a este ejercicio [aquí¹⁵⁷](#)

¹⁵⁷<https://github.com/hoottlex/the-majesty-of-vuejs-2/tree/master/homework/Chapter19>