Marcombo

Aprender VueJS

con 100 ejercicios prácticos

Ángel Vázquez Vázquez y Ramón Serrano Valero

CONTENIDOS WEB

Angel Vazquez Vazquez

Aprender VueJS con 100 ejercicios prácticos

«Bookwire»

Vazquez A.

Aprender VueJS con 100 ejercicios prácticos / A. Vazquez — «Bookwire»,

¿Se ha planteado en algún momento desarrollar aplicaciones SPA pero no ha sabido con qué framework iniciarse? Existen muchos, ¿verdad?
Si le contara que la curva de aprendizaje de desarrollar aplicaciones en Vue es mucho menor que el tiempo invertido en aprender React o Angular, ¿se animaría?
Vue, pese a llegar el último al mercado, ha adoptado las buenas prácticas de Angular y React, además de aportar su granito de arena. Con Vue rápidamente podrá desarrollar desde las aplicaciones más sencillas hasta las más complejas. Solo deberá aprender el framework e incorporar otros conocimientos, como TypeScript.
Gracias a los 100 ejercicios prácticos de este libro:
o Aprenderá los principios básicos de Vueo Estudiará el frameworko Conocerá el desarrollo orientado a componenteso Aprenderá la comunicación entre eventoso Utilizará almacenes de estados de los componentes mediante Vuexo Conectará con servidores remotos mediante Axioso Estudiará los distintos modos de compilación de aplicación y componenteso Reutilizará los componentes desarrollados en Vue, en otros frameworkso Realizará pruebas unitarias mediante Jest
Si ya conoce otros frameworks, verá que Vue tiene muchas similitudes con el resto y le costará muy poco aprenderlo. Si no conoce ninguno, solo necesita saber JavaScript y HTML. Fácil, ¿no? ¡Anímese y aprenda Vue de un modo sencillo!

Содержание

Aprender

VueJS

con 100 ejercicios prácticos

Aprender VueJS con 100 ejercicios prácticos

www.marcombo.com
Diseño de la cubierta: Giancarlo Salinas
Director de colecciión: Pablo Martínez Izurzu
Directora de producción: M.ª Rosa Castillo Hidalgo
Maquetación: María Paz Mora Encinas
Correctora: Laura Seoane

ISBN: 978-84-267-2802-9
Producción del ebook: booqlab.com

Presentación

APRENDER VUEJS CON 100 EJERCICIOS PRÁCTICOS

Los 100 capítulos que contiene este libro realizan un recorrido por los principios de desarrollo utilizando el framework de Vue. Es recomendable realizar los ejercicios en orden, sobre todo a partir del ejercicio 40, donde empieza la aplicación SPA mediante el CLI. Puede apoyarse en los ejercicios, ya implementados subidos en la plataforma Marcombo, plenamente funcionales.

Una vez finalizado este libro, el lector habrá adquirido el conocimiento para crear su propia aplicación SPA, habilitándole para afrontar proyectos en esta tecnología. Tendrá capacidad de crear aplicaciones rápidamente, mejorar su pensamiento crítico a la hora de abordar un nuevo desarrollo, así como la habilidad para tomar como base la arquitectura propuesta y poder mejorarla conforme nuevos conocimientos sean incorporados.

A QUIÉN VA DIRIGIDO

A todo aquel que busque iniciarse en el desarrollo web, la programación o adentrarse en el mundo de este nuevo framework Vue.

También está dirigido a aquellos que ya hayan realizado sus primeros pasos con Vue, pues en este libro encontrarán las respuestas a las dudas que les surjan y aquellas que no hayan podido resolver hasta ahora.

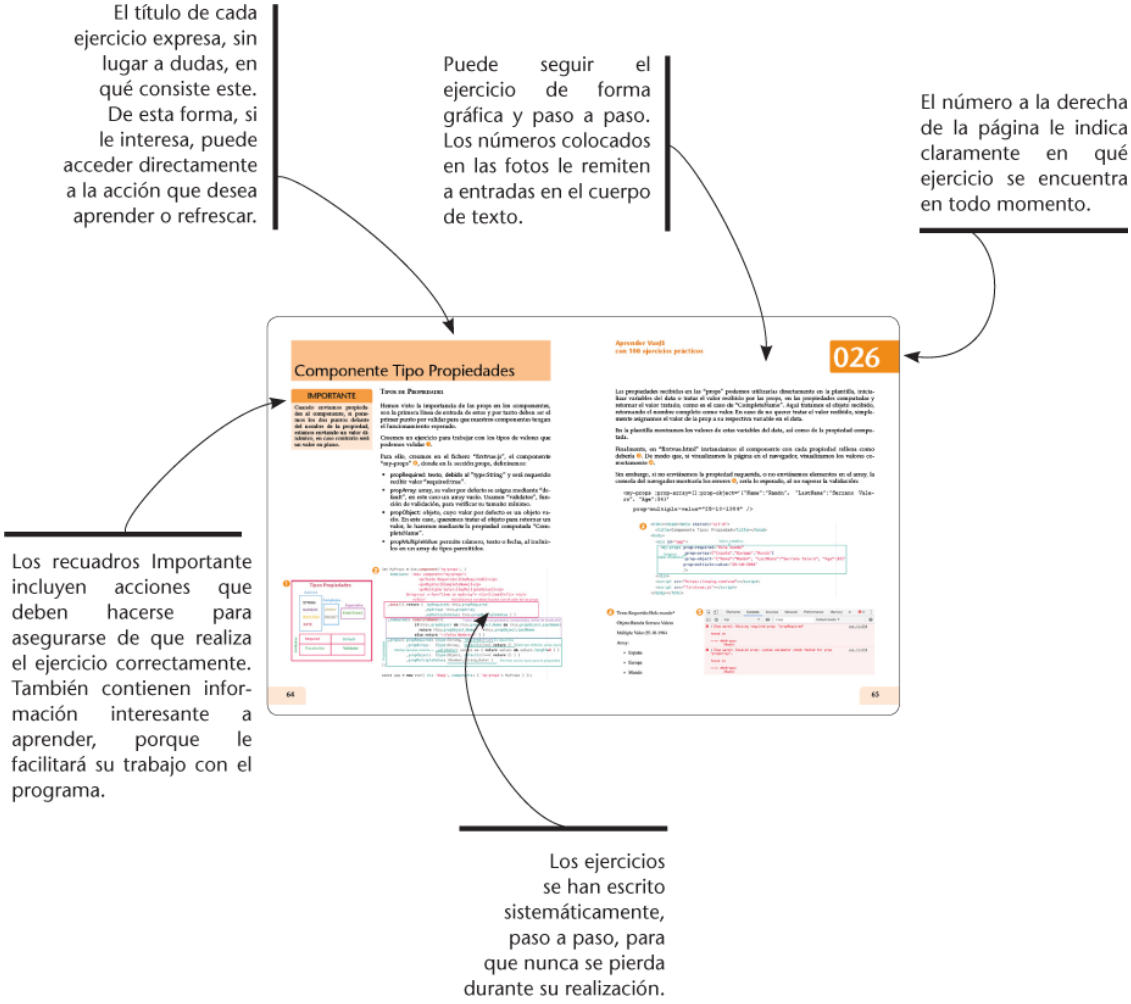
LA FORMA DE APRENDER

Nuestra experiencia en el ámbito de la enseñanza nos ha llevado a diseñar este tipo de manual, en el que cada una de las funciones se ejercita mediante la realización de un ejercicio práctico. Dicho ejercicio se halla explicado paso a paso y pulsación a pulsación, a fin de no dejar ninguna duda en su proceso de ejecución. Además, lo hemos ilustrado con imágenes descriptivas de los pasos más importantes o de los resultados que deberían obtenerse y con recuadros IMPORTANTE que ofrecen información complementaria sobre los temas tratados en los ejercicios.

LOS ARCHIVOS NECESARIOS

En la parte inferior de la primera página del libro encontrará el código de acceso que le permitirá descargar de forma gratuita los contenidos adicionales del libro en *www.marcombo.info*.

Cómo leer los libros "Aprender..."



Ramón: *Ha sido duro compaginar el mundo laboral con el Máster en Big Data y la escritura de este libro, sin embargo, gracias al apoyo de mi mujer Sara de la V., he podido sacar fuerzas para poder finalizar este libro, ¡eres mi fuerza, siempre sacas lo mejor de mi! Agradezco a mis padres por inculcarme el espíritu de esfuerzo. A Xavi y Martín, mis sobrinos, que, algún día, cuando comprendan este libro, aprendan este framework y los nuevos que aparezcan. A mi amigo Ángel V., una mente siempre activa, preparado para cualquier reto, un placer escribir junto a ti. A Gabriel G. por años compartiendo conocimientos que nos permitieron innovar. Así como a mis compañeros Miguel N. y Alonso V. por habernos aventurado juntos en este mundo Vuetizado.*

Ángel: *Este libro es para las personas que me rodean: mis padres que siempre me apoyan, mis amigos y compañeros, mis gatas que tanta compañía me hicieron a la hora de trabajar en este libro, ¡y para tí, Lucía!*

Mención especial para mi compañero Ramón con el que colaboro ya por tercera vez en este tipo de aventuras. ¡Gracias por hacer tan fácil trabajar contigo, nano!

No pudisteis estar presentes Juanto G. y Pablo F. en este libro, pero formáis parte de este equipo, esperamos abordar nuevos retos juntos de nuevo.

Índice

¡Hola, Vue!
LOS COMIENZOS

¡Bienvenidos a Vue, queridos lectores! En este ejercicio veremos cómo empezar a trabajar con Vue con el mínimo esfuerzo, pero en primer lugar daremos algunas pinceladas sobre Vue.

¿Qué es Vue? Vue es un framework progresivo para construir interfaces de usuario [1].

¿Qué significa progresivo? Simplemente que está modularizado de manera que su librería principal está enfocada solo a la parte visual y, usándola en conjunto con otras librerías o proyectos, puede construir fantásticas aplicaciones web SPA. Esto significa que Vue no es un todo o nada sino que puede escoger las partes que más le interesen e incluso usarlas conjuntamente con otros frameworks o librerías web.

Para ilustrar el concepto de progresivo vamos a importar lo mínimo necesario para crear nuestra primera aplicación.

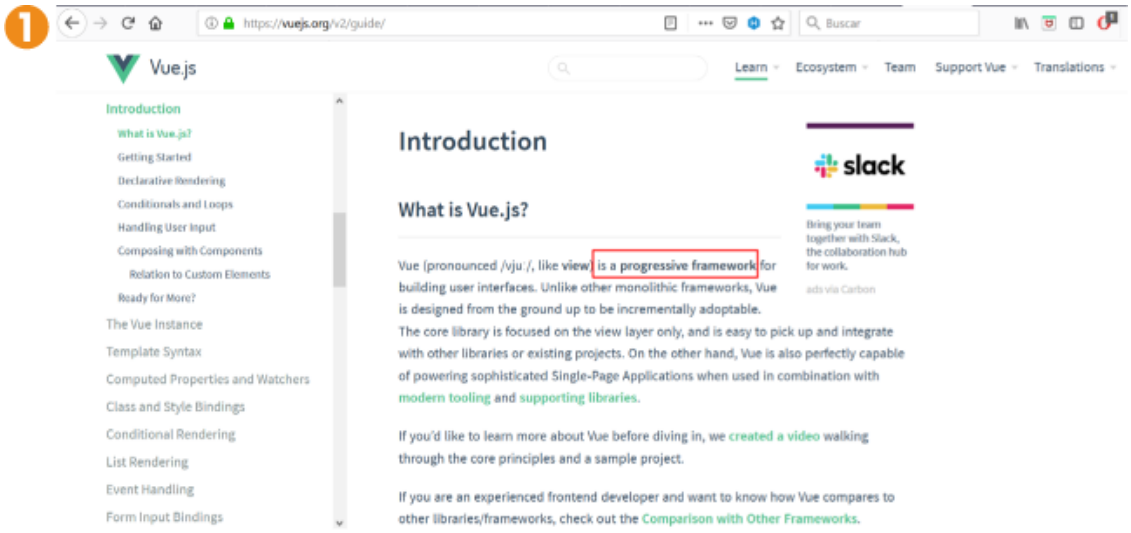
En primer lugar, vamos a utilizar un editor online como JSFiddle (https://jsfiddle.net/) y en la ventana de HTML añadiremos el siguiente contenido [2].

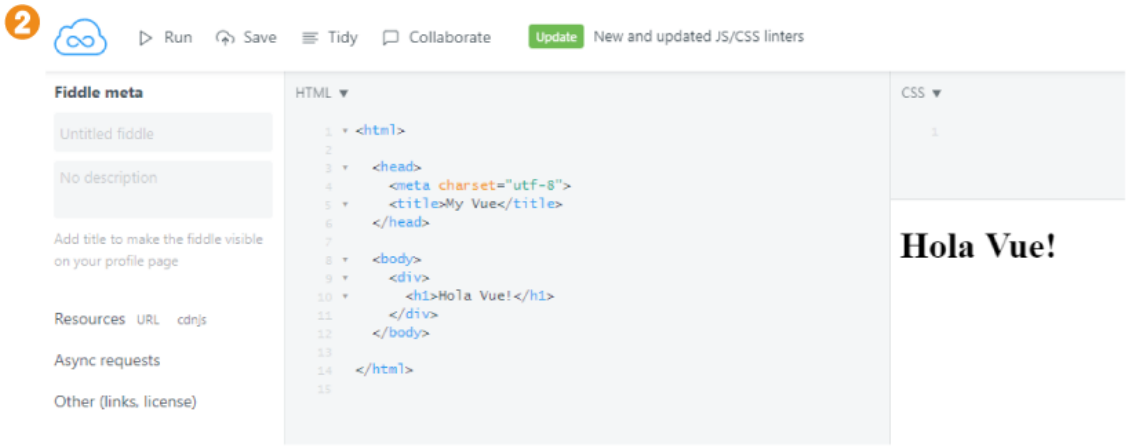
Tenemos nuestro “Hola Vue” que renderiza una página de forma estática. Para trabajar de forma dinámica añadiremos los scripts necesarios para importar la librería de Vue [3].

En la ventana del editor para el código Javascript añadimos la instancia de Vue, la ligamos al elemento con id #app y añadimos una variable en el bloque data denominada message[4].

Volviendo al HTML asociamos a uno de los elementos el id que hemos declarado en la parte de Javascript [5]. ¡Dentro de este elemento añadimos un título (h1) con la variable message y ya tenemos nuestro “¡Hola Vue!”

¡Así de fácil ha sido crear nuestra primera aplicación y de forma online! En siguientes ejercicios configuraremos nuestro entorno en local para trabajar cómodamente y con la ayuda de múltiples herramientas para el desarrollo web.




2
Run
Save
Tidy
Collaborate
Update
New and updated JS/CSS linters
Fiddle meta
Untitled fiddle
No description
Add title to make the fiddle visible on your profile page
Resources URL cdnjs
Async requests
Other (links, license)
HTML
<html>
<head>
<meta charset="utf-8">
<title>My Vue</title>
</head>
<body>

<h1>Hola Vue!</h1>

</body>
</html>
CSS
Hola Vue!


3
Untitled fiddle
No description
Add title to make the fiddle visible on your profile page
Resources URL cdnjs
Async requests
Other (links, license)
<html>
<head>
<meta charset="utf-8">
<title>My Vue</title>
</head>
<body>

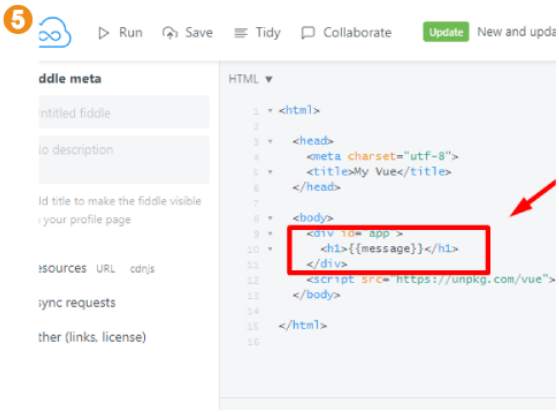
<h1>{{message}}</h1>

<script src="https://unpkg.com/vue"></script>
</body>
</html>
JavaScript + No-Library (pure JS)
const app = new Vue({
el: '#app',
data: {
message: 'Hello Vue!'
}
});
Hello Vue!


4
<meta charset="utf-8">
<title>My Vue</title>
</head>
<body>

<h1>{{message}}</h1>

<script src="https://unpkg.com/vue"></scr
</body>
</html>
JavaScript + No-Library (pure JS)
const app = new Vue({
el: '#app',
data: {
message: 'Hello Vue!'
}
});


5
Run
Save
Tidy
Collaborate
Update
HTML
<html>
<head>
<meta charset="utf-8">
<title>My Vue</title>
</head>
<body>

<h1>{{message}}</h1>

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

Configurar el entorno
PRIMEROS PASOS CON ATOM

Comenzaremos este viaje de aprendizaje configurando un entorno que nos facilite el desarrollo de nuestras aplicaciones. Para este libro como editor hemos elegido Atom pero podría utilizar cualquier otro con el que se sienta cómodo.

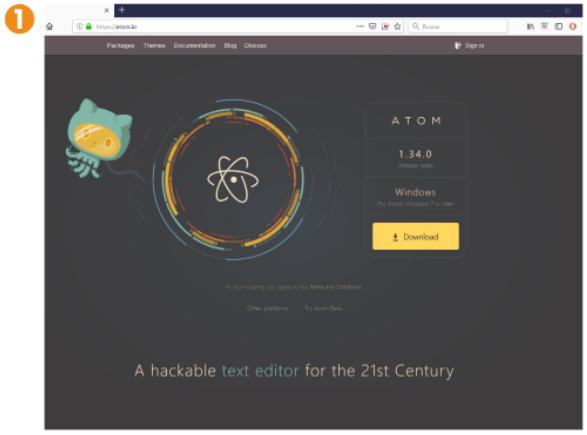
En primer lugar, vamos a la página de Atom (https://atom.io/)[1] y una vez descargado instalamos [2] y lo iniciamos [3] y a partir de este momento ya tenemos un editor para nuestro código.

Esto no se acaba aquí, querremos tener también a nuestro alcance las herramientas que nos hagan desarrollar nuestras aplicaciones más cómoda y rápidamente, con lo que vamos a añadir algunos plugins.

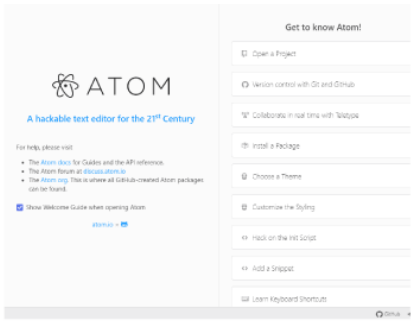
Para ello iremos a la vista de ajustes en la que podremos buscar y seleccionar plugins para su instalación [4]. Entre la ingente cantidad de herramientas que podremos añadir a nuestro editor nosotros hemos elegido algunas que pueden ser útiles. Sin embargo, utilice tantas como quiera y necesite para sentirse cómodo ya que esta es una elección muy personal.

Entre las que hemos seleccionado están language-vue [5] que resalta la sintaxis de vue, vue-format [6], atom-beautify [7] que dan formato a nuestro código y atom-html-preview [8] que integra dentro de nuestro editor una ventana para renderizar el contenido automático de código en vivo.

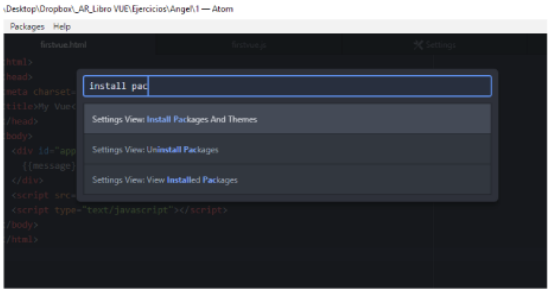
De esta forma, si retomamos los archivos de nuestra primera aplicación, podremos trabajar cómodamente en nuestro editor local con todos los plugins para la ayuda al desarrollo y el renderizado HTML integrado [9].



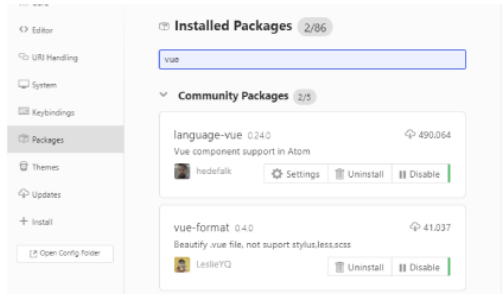
3



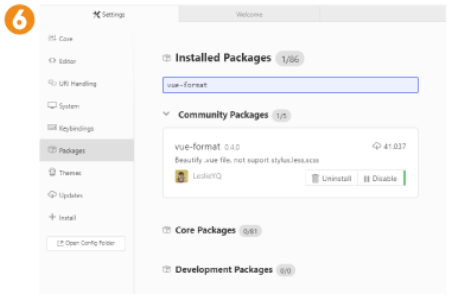
4



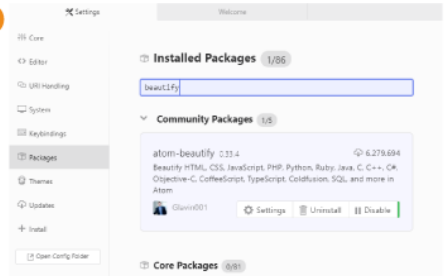
5



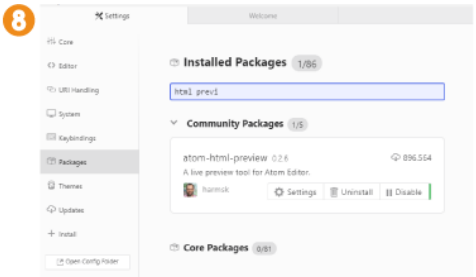
6



7



8



9

```
<html>

<head>
  <meta charset="utf-8">
  <title>My Vue</title>
</head>

<body>
  <div id="app">
    <h1>{{message}}</h1>
  </div>
  <script src="https://unpkg.com/vue"></script>
  <script src="firstvue.js"></script>
</body>

</html>
```

Hello Vue!

Instancia Vue
MANEJO DE VARIABLES

Vamos a construir un pequeño blog de viajes y, para ello, como hemos visto en anteriores ejercicios, creamos la instancia Vue que manejará el contexto del elemento con id #app asociado [1].

Dentro de este elemento crearemos un título general y una serie de artículos con un pequeño titular y contenido [2]. De esta forma definimos una plantilla para nuestro blog y para la que nos faltaría añadir contenido. Este contenido variará, con lo que como primera aproximación crearemos unas cuantas variables dentro de la instancia Vue que representarán los textos que hayamos escrito sobre nuestros últimos viajes.

Tenemos nuestro pequeño blog de viajes listo para añadir contenido, este contenido podría cargarse en las variables desde alguna fuente externa tanto para imágenes como para texto, pero como primer paso simplemente representamos el valor contenido dentro de las variables de la instancia Vue [3].

Esto Vue lo realiza así de fácil, podemos considerar el data como el lugar donde definimos las variables del modelo con las que trabaja nuestra instancia de Vue. Hasta ahora la definición de estos valores se realiza de forma estática, ya veremos más adelante cómo podemos ir modificando estos valores dentro de la instancia, donde su valor se reflejaría automáticamente en la plantilla, al haber sufrido un cambio, esto es propio de la reactividad de Vue. ¡Genial!

1

firstvue.html

```
<body>
  <h1>My Vue Blog</h1>
  <div class="flex-container">
    <div id="app" class="flex-item flex-item-content">
      <h2>Content</h2>

      <h3>{{title1}}</h3>
      {{post1}}

      <h3>{{title2}}</h3>
      {{post2}}

      <h3>{{title3}}</h3>
      {{post3}}
    </div>
  </div>
  <script src="https://unpkg.com/vue"></script>
  <script src="firstvue.js"></script>
</body>
```

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    title1:'Trip 1',
    title2:'Trip 2',
    title3:'Trip 3',
    post1: 'Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiu
```

firstvue.html Preview

My Vue Blog

Content

Trip 1

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Trip 2

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Trip 3

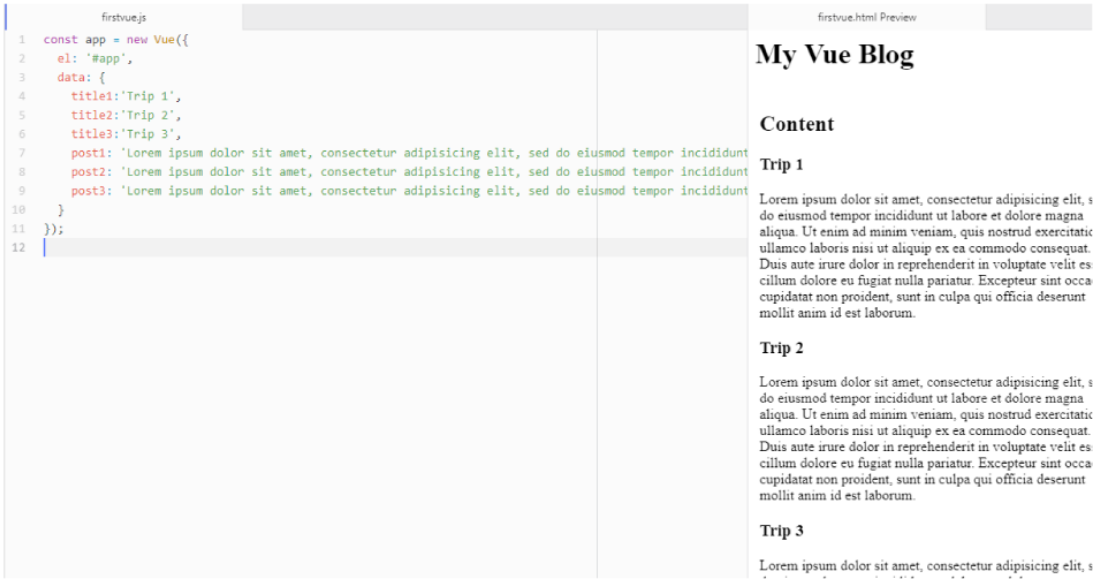
2


firstvue.html
<body>
<h1>My Vue Blog</h1>

<h2>Content</h2>
<h3>{{title1}}</h3>
{{post1}}
<h3>{{title2}}</h3>
{{post2}}
<h3>{{title3}}</h3>
{{post3}}

<script src="https://unpkg.com/vue"></script>
<script src="firstvue.js"></script>
</body>
firstvue.js
const app = new Vue({
el: '#app',
data: {
title1:'Trip 1',
title2:'Trip 2',
title3:'Trip 3',
post1: 'Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiu
post2: 'Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiu
firstvue.html Preview
My Vue Blog
Content
Trip 1
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Trip 2
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Trip 3
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed

3


firstvue.js
const app = new Vue({
el: '#app',
data: {
title1:'Trip 1',
title2:'Trip 2',
title3:'Trip 3',
post1: 'Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt
post2: 'Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt
post3: 'Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt
}
});
firstvue.html Preview
My Vue Blog
Content
Trip 1
Trip 2
Trip 3

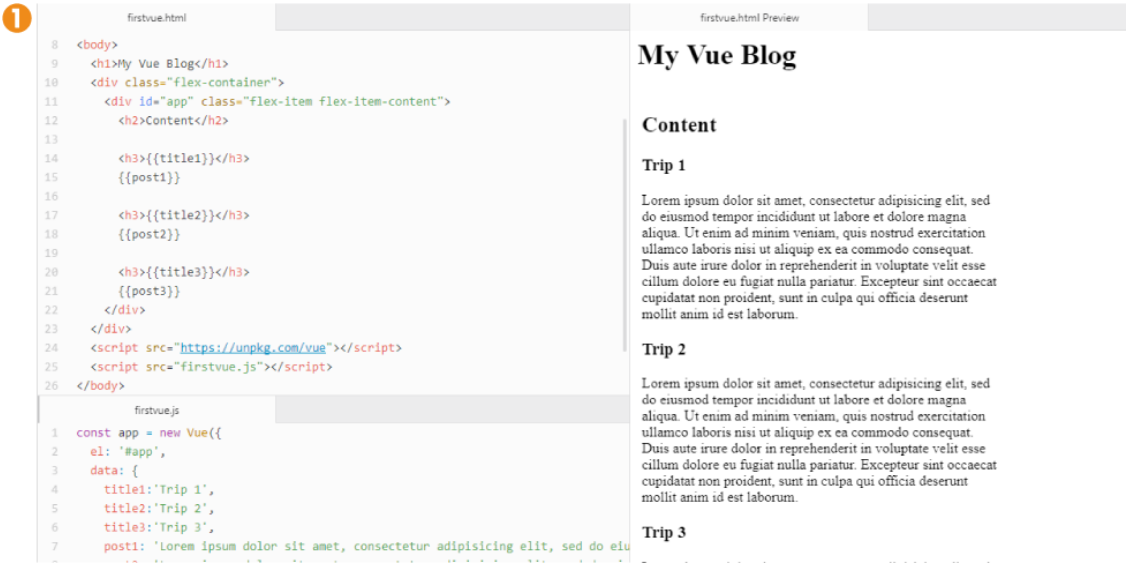
Instancias Vue
GESTIONANDO MÁS DE UNA INSTANCIA

En nuestro blog de viajes además de una colección de artículos queremos añadir un pequeño índice con información sobre los títulos y en qué fecha se publicaron. Con este fin podemos añadir una nueva instancia de Vue que controle este componente índice.

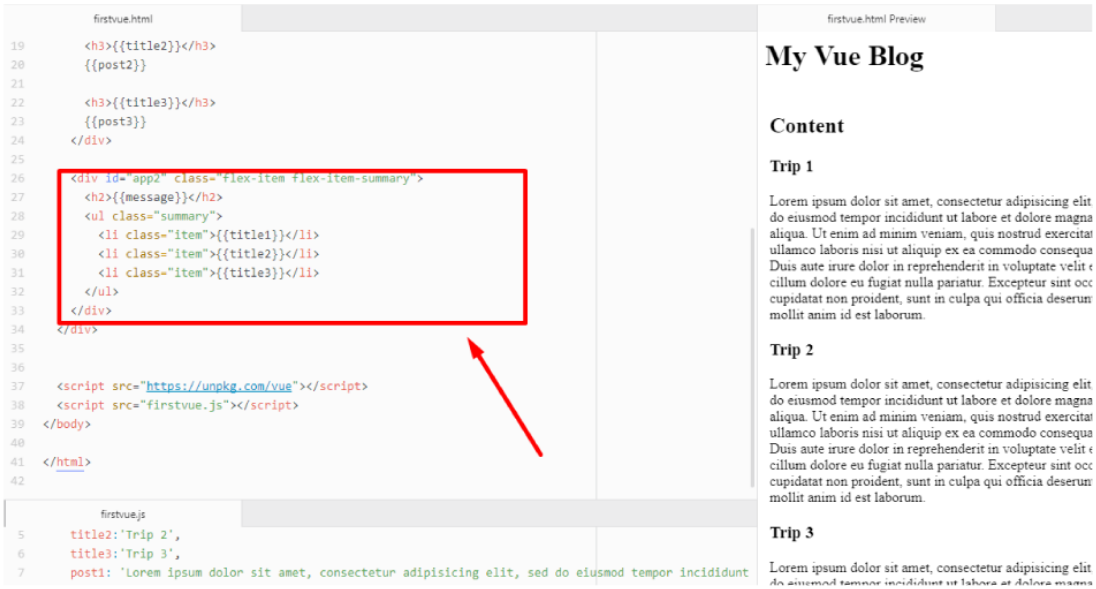
Partimos de nuestro ejemplo anterior [1] creando una nueva columna en la parte derecha con nuestro nuevo componente. Este nuevo elemento contendrá una lista no ordenada de títulos con fecha de publicación, y para ello crearemos el HTML adecuado y lo identificamos con el id app2.

La segunda parte del ejercicio será crear una nueva instancia Vue en el código Javascript con la que manejar nuestro nuevo componente. Hasta este momento tenemos declarada una lista con elementos en HTML con una variable por ítem de la lista y ahora declararemos y daremos valores a todas y cada una de ellas [2[[[3[].

Hemos creado nuestro blog de artículos de viaje y lo hemos completado con la creación de un pequeño índice resumen con títulos y fechas. Comprobamos así que pueden convivir varias instancias Vue, una para cada componente, sin ningún problema.



2


firstvue.html
<h3>{{title2}}</h3>
{{post2}}
<h3>{{title3}}</h3>
{{post3}}

<h2>{{message}}</h2>
<ul class="summary">
<li class="item">{{title1}}
<li class="item">{{title2}}
<li class="item">{{title3}}
<script src="https://unpkg.com/vue"></script>
<script src="firstvue.js"></script>
firstvue.js
title2:'Trip 2',
title3:'Trip 3',
post1: 'Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt
firstvue.html Preview
My Vue Blog
Content
Trip 1
Lorem ipsum dolor sit amet, consectetur adipisicing elit
Trip 2
Lorem ipsum dolor sit amet, consectetur adipisicing elit
Trip 3
Lorem ipsum dolor sit amet, consectetur adipisicing elit

3


firstvue.html
firstvue.js
const app = new Vue({
el: '#app',
data: {
title1:'Trip 1',
title2:'Trip 2',
title3:'Trip 3',
post1: 'Lorem ipsum dolor sit amet, consectetur adipisicing
post2: 'Lorem ipsum dolor sit amet, consectetur adipisicing
post3: 'Lorem ipsum dolor sit amet, consectetur adipisicing
}
});
const app2 = new Vue({
el: '#app2',
data: {
message: 'Summary',
title1: '16/06/2019 Trip 1',
title2: '05/12/2018 Trip 2',
title3: '04/10/2018 Trip 3'
}
});
firstvue.html Preview
My Vue Blog
Content
Trip 1
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Trip 2
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Trip 3
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed
Summary
16/06/2019 Trip 1
05/12/2018 Trip 2
04/10/2018 Trip 3

Ciclo de vida
FASES DEL CICLO DE VIDA

Se denomina ciclo de vida de una instancia Vue a una serie de estados por los que va pasando el componente. Estos estados son básicamente cuatro: created, mounted, updated y destroyed.

Vue nos permitirá definir acciones anteriores y posteriores a la transición desde o hacia cada estado interno del componente. Los métodos en cuestión para implementar estas acciones son:

• beforeCreate: evento lanzado antes de tener el componente cargado, implica no poder acceder al componente a nivel de Dom.

• created: evento donde se verifica si el componente tiene plantilla, entonces se compila y se observan las propiedades computadas, data, métodos y eventos. Pero no podemos acceder al $el.

• beforeMount: evento que ocurre antes de representar el componente.

• mounted: evento que implica que el componente está cargado por completo, se añade al DOM, quedando el componente accesible a través de $el.

• beforeUpdate: evento que se ejecuta cuando el valor del data del componente cambia.

• updated: evento invocado tras finalizar la modificación de valor del data.

• beforeDestroy: evento que elimina eventos que estuvieran activos en el componente, antes de eliminar la instancia.

• destroyed: evento lanzado tras desacoplar el componente.

En estos métodos podremos reservar o liberar recursos, realizar cálculos, hacer llamadas http, etc.

Para ilustrar mejor cómo funciona el ciclo de vida implementaremos cada uno de estos métodos escribiendo en una consola del navegador un mensaje por cada vez que entre en los métodos mencionados [1].

Una vez que carga la página vemos que la instancia es creada, cargada en el DOM y está disponible cuando pasa al estado mount [2].

Para comprobar qué pasa cuando actualizamos el componente podemos editar el texto ”Hola Vue!” [3] y añadir alguna modificación para ver que pasa por los métodos beforeUpdate y updated [4].

En el caso de que eliminemos el componente podremos forzar su destrucción a través del botón Destroy comprobar que la instancia pasa por los métodos beforeDestroy y destroyed [5].

1

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    message: 'Hola Vue!',
  },
  beforeCreate: function() {
    console.log("beforeCreate");
  },
  created: function() {
    console.log("created");
  },
  beforeMount: function() {
    console.log("beforeMount");
  },
  mounted: function() {
    console.log("mounted");
  },
  beforeUpdate: function() {
    console.log("beforeUpdate");
  },
  updated: function() {
    console.log("updated");
  },
  beforeDestroy: function() {
    console.log("beforeDestroy");
  },
  destroyed: function() {
    console.log("destroyed");
  },
  methods: {
    handleClick: function(){
      app.$destroy();
    }
  }
});

```


2
firstvue.html
<title>My Vue</title>
<style src="firstvue.css"></style>
<link rel="stylesheet" href="firstvue.css">
</head>
<body>

<h1>{{ message }}</h1>
<input type="text" v-model="message"/>
<button type="button" v-on:click="handleClick">Destroy</button>

firstvue.js
const app = new Vue({
el: '#app',
firstvue.html Preview
Hola Vue!
Hola Vue!
Destroy
1. beforeCreate
2. created
3. beforeMount
4. mounted
5. Download the Vue Devtools extension for a better development experience:
https://github.com/vuejs/vue-devtools
6. You are running Vue in development mode.
Make sure to turn on production mode when deploying for production.
See more tips at https://vuejs.org/guide/deployment.html


3
firstvue.html
<head>
<meta charset="utf-8">
<title>My Vue</title>
<style src="firstvue.css"></style>
<link rel="stylesheet" href="firstvue.css">
</head>
<body>

<h1>{{ message }}</h1>
<input type="text" v-model="message"/>
<button type="button" v-on:click="handleClick">Destroy</button>

<ul id="myULContainer" class="console">
<script src="https://unpkg.com/vue"></script>
<script type="application/javascript" src="https://cdn.rawgit.com/Alorel/co
firstvue.html Preview
Hola Vue!
Hola Vue!
Destroy
1. beforeCreate
2. created
3. beforeMount
4. mounted
5. Download the Vue Devtools extension for a bett
https://github.com/vuejs/vue-devtools
6. You are running Vue in development mode.
Make sure to turn on production mode when dep
See more tips at https://vuejs.org/guide/deploym


4
firstvue.html Preview
Hola Vue!!!
Hola Vue!!!
Destroy
1. beforeCreate
2. created
3. beforeMount
4. mounted
5. Download the Vue Devtools extension for a better development experience:
https://github.com/vuejs/vue-devtools
6. You are running Vue in development mode.
Make sure to turn on production mode when deploying for production.
See more tips at https://vuejs.org/guide/deployment.html
7. beforeUpdate
8. updated
9. beforeUpdate
10. updated
5
Hola Vue!!!
Destroy
1. beforeCreate
2. created
3. beforeMount
4. mounted
5. Download the Vue Devtools extension for a better development experience:
https://github.com/vuejs/vue-devtools
6. You are running Vue in development mode.
Make sure to turn on production mode when deploying for production.
See more tips at https://vuejs.org/guide/deployment.html
7. beforeUpdate
8. updated
9. beforeUpdate
10. updated
11. beforeDestroy
12. destroyed

Data binding
VALORES Y VARIABLES

Volvemos de nuevo con nuestro blog de viajes, en esta ocasión además de listar los artículos escritos y el índice querremos añadir un pequeño apartado para editar nuevos artículos.

En primer término, crearemos un input para escribir un título para nuestro artículo y una caja de texto más amplia para poder escribir todo nuestro artículo en la misma [1].

Además de crear el marcado en lenguaje HTML añadiremos a cada input un nuevo atributo con el siguiente formato v-model=”nombreVariable” para enlazar la variable con el contenido que escribamos. Del mismo modo tendremos que declarar esas variables con los nombres que habremos elegido en el código Javascript de la instancia Vue [2].

Posteriormente añadiremos el título y el contenido en la página para que se renderice a medida que se escribe y podamos ir viendo cómo queda [3]. Esta es la “magia” del enlazado de variables con los inputs ya que somos capaces de darle un valor a una variable dentro de la instancia e ir mostrando lo que guarda esa variable dentro del contexto de nuestra instancia. [4]

Simplemente con estos pasos tendremos una forma fácil de añadir un nuevo artículo con su título y contenido y poder ver el renderizado en la página inmediatamente a medida que vamos generando contenido[5].

1
```
<body>
  <h1>My Vue Blog</h1>
  <div id="app"><h2>Nueva entrada</h2>
    <h3>Título</h3><input v-model="title" placeholder="Write a title">
    <h3>Entrada</h3>
    <textarea cols="50" v-model="post" placeholder="Write a post"></textarea>
    <h2>{{ title }}</h2>
    <p>{{post}}</p>
    <div class="flex-container">
      <div class="flex-item flex-item-content"><h2>Content</h2>
        <h3>{{title1}}</h3> {{post1}}
        <h3>{{title2}}</h3> {{post2}}
        <h3>{{title3}}</h3> {{post3}}
      </div>
      <div class="flex-item flex-item-summary"><h2>{{message}}</h2>
        <ul class="summary"><li class="item">{{title1}}</li>
                            <li class="item">{{title2}}</li>
                            <li class="item">{{title3}}</li></ul></div>
    </div>
  </div>
  <script src="https://unpkg.com/vue"></script>
  <script src="firstvue.js"></script>
</body>
```

2
```
const app = new Vue({ el: '#app',data: { title: '', post: '',
    title1: 'Trip 1', title2: 'Trip 2', title3: 'Trip 3',
    post1: 'Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempo
    post2: 'Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempo
    post3: 'Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempo
    message: 'Summary',
    title1: '16/06/2019 Trip 1', title2: '05/12/2018 Trip 2',title3: '04/10/2018 Trip 3' }
```

3
```
<h2>{{ title }}</h2>
<p>{{post}}</p>
```

4


```
<body>
  <h1>My Vue Blog</h1>
  <div id="app">
    <h2>Nueva entrada</h2>
    <h3>Título</h3>
    <input v-model="title" placeholder="Write a title">
    <h3>Entrada</h3>
    <textarea cols="50" v-model="post" placeholder="Write a post">
    </textarea>
    <h2>{{ title }}</h2>
    <p>{{post}}</p>
    <div class="flex-container">
```

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    title: '',
    post: '',
    title1: 'Trip 1',
    title2: 'Trip 2',
    title3: 'Trip 3',
    post1: 'Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eius
```

5


```
<body>
  <h1>My Vue Blog</h1>
  <div id="app">
    <h2>Nueva entrada</h2>
    <h3>Título</h3>
    <input v-model="title" placeholder="Write a title">
    <h3>Entrada</h3>
    <textarea cols="50" v-model="post" placeholder="Write a post">
    </textarea>
    <h2>{{ title }}</h2>
    <p>{{post}}</p>
    <div class="flex-container">
```

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    title: '',
    post: '',
    title1: 'Trip 1',
    title2: 'Trip 2',
```

Watch
PROPIEDADES OBSERVADAS

Hasta ahora hemos visto cómo hacer el enlazado entre las variables y su contenido y la renderización en la web de los cambios. Hemos hablado de las variables de la instancia de Vue: cómo declararlas, manejarlas y mostrar su contenido.

Para observar una propiedad lo único que tendremos que hacer es declarar dentro en la instancia una variable dentro del bloque watch. El bloque watch se define como un objeto, y cada variable dentro del mismo es una función con el mismo nombre, con lo que cada vez que la variable cambie se ejecutará una función con dos argumentos: un valor antes del cambio y el nuevo valor.

Para ilustrar este concepto en este ejercicio veremos cómo observar variables de la instancia y mostrar su resultado. Compondremos una nueva página con un input y veremos cómo cambia la propiedad observada visualizándolo en una consola dentro de la propia página.

En el apartado del HTML tenemos nuestro título de página y un input para meter texto libre que enlazamos con el atributo v-model [1], y en el apartado del código Javascript creamos un bloque watch en el que observamos la variable message además de escribir la función que se ejecutará cuando cambie su valor [2].

Vemos que cada vez que hacemos cambios en el texto [3] se ejecuta la función con el valor antes del cambio y el nuevo valor [4].

Watch permite detectar cambios en los valores que se les proporciona a componentes hijos a través de las “props”, detectando cambios en el valor proporcionado en sus props, para finalmente refrescar el valor del componente, aunque este uso lo veremos con más detalle en otros ejercicios.

1

```
</head>

<body>
  <div id="app">
    <h1>{{ message }}</h1>
    <input type="text" v-model="message" />
  </div>


  <ul id="myULContainer" class="console"></ul>

```

2

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    message: 'Hola Vue!',
  },
  watch: {
    message: function(newValue, oldValue) {
      console.log('oldValue=> ',oldValue, ' newValue=>', newValue);
    }
  }
});
```

3

```
<head>
  <meta charset="utf-8">
  <title>My Vue</title>
  <style src="firstvue.css"></style>
  <link rel="stylesheet" href="firstvue.css">
</head>

<body>
  <div id="app">
    <h1>{{ message }}</h1>
    <input type="text" v-model="message" />
  </div>


  <ul id="myULContainer" class="console"></ul>

  <script src="https://unpkg.com/vue"></script>
  <script type="application/javascript" src="https://cdn.rawgit.com/Alorel/consol
```

firstvue.js

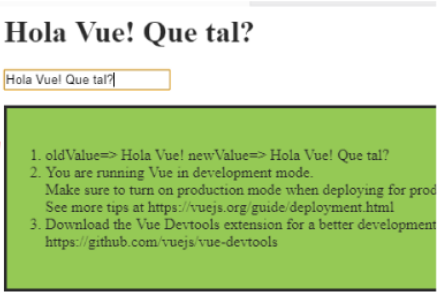
```
const app = new Vue({
  el: '#app',
  data: {
    message: 'Hola Vue!',
  },
  watch: {
    message: function(newValue, oldValue) {
      console.log('oldValue=> ',oldValue, ' newValue=>', newValue);
    }
```

4

```
<head>
  <meta charset="utf-8">
  <title>My Vue</title>
  <style src="firstvue.css"></style>
  <link rel="stylesheet" href="firstvue.css">
</head>

<body>
  <div id="app">
    <h1>{{ message }}</h1>
    <input type="text" v-model="message" />
  </div>


  <ul id="myULContainer" class="console"></ul>

  <script src="https://unpkg.com/vue"></script>
  <script type="application/javascript" src="https://cdn.rawgit.com/Alorel/consol
```

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    message: 'Hola Vue!',
  },
  watch: {
    message: function(newValue, oldValue) {
      console.log('oldValue=> ',oldValue, ' newValue=>', newValue);
    }
  }
});
```



Computed properties
EVITANDO EL RECÓMPUTO

Hasta ahora hemos manejado propiedades y métodos de la instancia. Sabemos cómo declarar propiedades, cómo modificarlas y con ello vemos cómo se renderizan automáticamente de nuevo en la página.

En este ejercicio veremos cómo usar las computed properties que nos dan un punto extra de eficiencia ya que solo se evaluarán en el momento que afecte a alguna parte de su cálculo.

Imaginemos que tenemos una pequeña calculadora de sumas y restas en la que tenemos cuatro inputs que se corresponden con los diferentes operandos involucrados [1].

Contamos con cuatro propiedades en el apartado data y dos métodos diferentes encargados de hacer el cómputo correspondiente y mostrarnos el resultado [2].

Ahora mismo si editamos alguno de los operandos vemos que se efectúa el cálculo y se muestra el resultado después del signo igual. Hasta aquí es lo que se espera de nuestra calculadora personal, ¿no? Sin embargo, si vamos al detalle de cómo se efectúan los cálculos revisando la consola, podemos comprobar que cada vez que modificamos un solo operando se hacen los cálculos de todas las operaciones que tengamos declaradas.

Para nuestro caso quizá no es algo muy crítico desde el punto de vista de la eficiencia, pero nos vale para ver que, si se hiciera algún cálculo costoso, este repetiría fuese cual fuese el cambio en la instancia [3].

Para evitar este recálculo haremos uso de las computed properties simplemente sustituyendo el bloque de methods por un bloque computed y cambiar las llamadas para el cálculo de los resultados como si fueran propiedades de la instancia [4].

Con esta modificación podemos comprobar que si modificamos alguno de los operandos solo se volverán a calcular aquellas operaciones en las que ese operando esté involucrado, mientras que el resto no se recalcularán [5].

Para comprobar cómo se hace el recálculo utilizaremos el operando 2 en ambas propiedades computadas, con lo que podremos jugar y ver qué es lo que se recalcula en función del operando que modifiquemos [6].

1

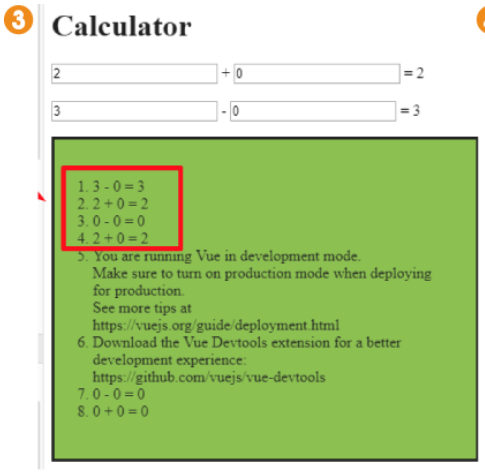
```
const app = new Vue({ el: '#app',
  data: { title: 'Calculator', operator1: '0'
          ,operator2: '0', operator3: '0', operator4: '0' },
  methods: {
    resultSum: function() {},
    resultSub: function() {} } });
```

firstvue.html

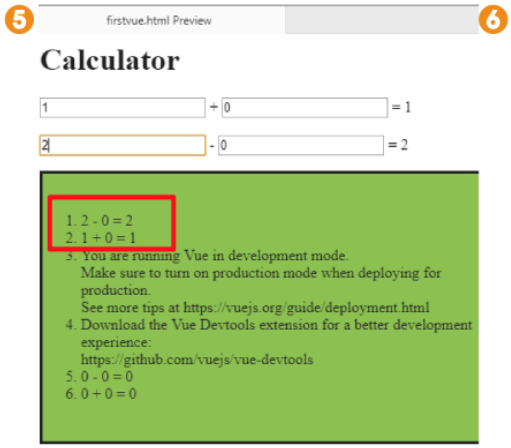
```
<html><head><meta charset="utf-8">
<body>
  <div id="app"><h1>{{ title }}</h1>
    <input type="text" v-model="operator1" /> +
    <input type="text" v-model="operator2" /> = {{ resultSum }}
    <input type="text" v-model="operator3" /> -
    <input type="text" v-model="operator4" /> = {{ resultSub }}
  </div>
```

2

```
const app = new Vue({ el: '#app',
  data: { title: 'Calculator', operator1: '0'
          ,operator2: '0', operator3: '0', operator4: '0' },
  methods: {
    resultSum: function() {
      let op1 = parseInt(this.operator1===''?'0':this.operator1);
      let op2 = parseInt(this.operator2===''?'0':this.operator2);
      console.log( op1 + ' + ' + op2 + ' = ' + (op1+op2));
      return op1 + op2;
    },
    resultSub: function() {
      let op3 = parseInt(this.operator3===''?'0':this.operator3);
      let op4 = parseInt(this.operator4===''?'0':this.operator4);
      console.log( op3 + ' - ' + op4 + ' = ' + (op3-op4));
      return op3 - op4;
  } });
```


3
Calculator
2 + 0 = 2
3 - 0 = 3
1. 3 - 0 = 3
2. 2 + 0 = 2
3. 0 - 0 = 0
4. 2 + 0 = 2
5. You are running Vue in development mode.
Make sure to turn on production mode when deploying for production.
See more tips at https://vuejs.org/guide/deployment.html
6. Download the Vue Devtools extension for a better development experience:
https://github.com/vuejs/vue-devtools
7. 0 - 0 = 0
8. 0 + 0 = 0


4
firstvue.js
const app = new Vue({ el: '#app',
data: { title: 'Calculator', operator1: '0'
,operator2: '0', operator3: '0', operator4: '0' },
computed: {
resultSum: function() {
let op1 = parseInt(this.operator1===''?'0':this.operator1);
let op2 = parseInt(this.operator2===''?'0':this.operator2);
console.log(op1 + ' + ' + op2 + ' = ' + (op1+op2));
return op1 + op2;
},
resultSub: function() {
let op3 = parseInt(this.operator3===''?'0':this.operator3);
let op4 = parseInt(this.operator4===''?'0':this.operator4);
console.log(op3 + ' - ' + op4 + ' = ' + (op3-op4));
return op3 - op4;
}
}
});


5
firstvue.html Preview
Calculator
1 + 0 = 1
2 - 0 = 2
1. 2 - 0 = 2
2. 1 + 0 = 1
3. You are running Vue in development mode.
Make sure to turn on production mode when deploying for production.
See more tips at https://vuejs.org/guide/deployment.html
4. Download the Vue Devtools extension for a better development experience:
https://github.com/vuejs/vue-devtools
5. 0 - 0 = 0
6. 0 + 0 = 0


6
computed: {
resultSum: function() {
let op1 = parseInt(this.operator1===''?'0':this.operator1);
let op2 = parseInt(this.operator2===''?'0':this.operator2);
console.log(op1 + ' + ' + op2 + ' = ' + (op1+op2));
return op1 + op2;
},
resultSub: function() {
let op3 = parseInt(this.operator3===''?'0':this.operator3);
let op4 = parseInt(this.operator4===''?'0':this.operator4);
console.log(op3 + ' - ' + op4 + ' = ' + (op3-op4));
return op3 - op4;
}
}
});

Métodos
OPERANDO CON LAS PROPIEDADES

Dentro de una instancia Vue además de las propiedades se exponen métodos que operan sobre los datos o ejecutan acciones. Los métodos se ejecutan cada vez que las invocamos, cosa que no ocurre con las propiedades computadas, ejecutadas cada vez que alguna de las variables utilizadas en la obtención de un resultado en las propiedades computadas cambia su valor; en dicho caso se ejecutaría la propiedad, para recalcular.

Estos métodos se declaran dentro de la instancia dentro del bloque methods, y se pueden asociar a los elementos HTML de la instancia.

En este ejercicio vamos a relacionar un componente de tipo botón con una acción que se corresponderá con un método declarado en la instancia Vue. Para indicar esta relación usamos la directiva v-on que se coloca dentro del HTML del botón y el evento (en este caso el evento de click).

<button type=”button” v-on:click=”showMessage”>Mostrar Título</button>

Una vez dado este paso creamos un bloque methods en la instancia y dentro de este creamos la función showMessage que concatenará la propiedad title a un texto.

methods: {
showMessage: function() {
console.log(‘Showing message: ‘+this.title + ‘ Aprendamos!’);
return this.title + ‘ Aprendamos!’;
}
}

Posteriormente declaramos dentro de una etiqueta h1 el método para renderizar el resultado de la función [1].

<h1>{{ showMessage() }}</h1>

Para volver a renderizar el resultado de la función accionamos el botón, si nos fijamos se calcula cada vez que salta el evento asociado al botón [2].


2
firstvue.html
<link rel="stylesheet" href="firstvue.css">
</head>
<body>

<h1>{{ showMessage() }}</h1>
<button type="button" v-on:click="showMessage">Mostrar Título</button>

<ul id="myULContainer" class="console">
firstvue.js
const app = new Vue({
el: '#app',
data: {
title: 'Hola Vue!'
},
methods: {
showMessage: function() {
console.log('Showing message: '+this.title + ' Aprendamos!');
return this.title + ' Aprendamos!';
}
}
});
firstvue.html Preview
Hola Vue! Aprendamos!
Mostrar Título
1. Showing message: Hola Vue! Aprendamos!
2. Download the Vue Devtools extension for a better developm
https://github.com/vuejs/vue-devtools
3. You are running Vue in development mode.
Make sure to turn on production mode when deploying for p
See more tips at https://vuejs.org/guide/deployment.html
4. Showing message: Hola Vue! Aprendamos!
5. Showing message: Hola Vue! Aprendamos!
6. Showing message: Hola Vue! Aprendamos!
7. Showing message: Hola Vue! Aprendamos!

Componentes de la instancia
INSTANCIA VUE

Hasta ahora hemos estado jugando con la instancia de Vue, su ciclo de vida, el renderizado y el doble enlazado de propiedades, los métodos, etc., pero no nos hemos parado con los bloques básicos que tiene.

Una instancia se compone de:

• $el: es el objeto componente HTML al que estará asociado mediante el id correspondiente
• $data: es el objeto que contiene las propiedades de la instancia
• $refs: es el objeto donde se registran los elementos marcados con el atributo ref.

Para ilustrarlo en este ejercicio hemos creado una instancia de Vue con cada uno de los bloques mencionados a la que hemos añadido unas sentencias que imprimen por consola cada uno de estos elementos para que podamos ver su contenido cuando se carga la página.

El contenido de $el es efectivamente el objeto que contiene la definición del HTML de nuestra instancia. Si inspeccionamos el objeto accediendo a la propiedad innerHTML nos devuelve en formato texto el contenido en lenguaje HTML [1].

El contenido de $data nos devuelve el objecto con las variables como la de nuestro ejercicio la denominada *message*.

Podemos también invocar los métodos de nuestra instancia tal y como vemos en el ejercicio y nos ejecutará la función.

Dentro de $refs tenemos un objeto con los elementos marcados con el atributo ref en el HTML de nuestra instancia, en este caso son dos elementos h2 (mysubtitle y mysubtitle2).

Hemos añadido un botón que tiene asociado el método clickedButton y si lo accionamos se nos mostrarán dos mensajes en los elementos h2 referenciados. En este método accedemos a las propiedades de los componentes para añadirles un mensaje y un estilo que se hará visible en nuestra página una vez se ejecute [2].

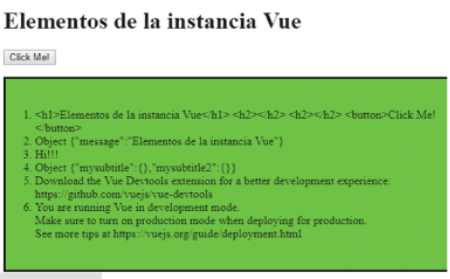
1

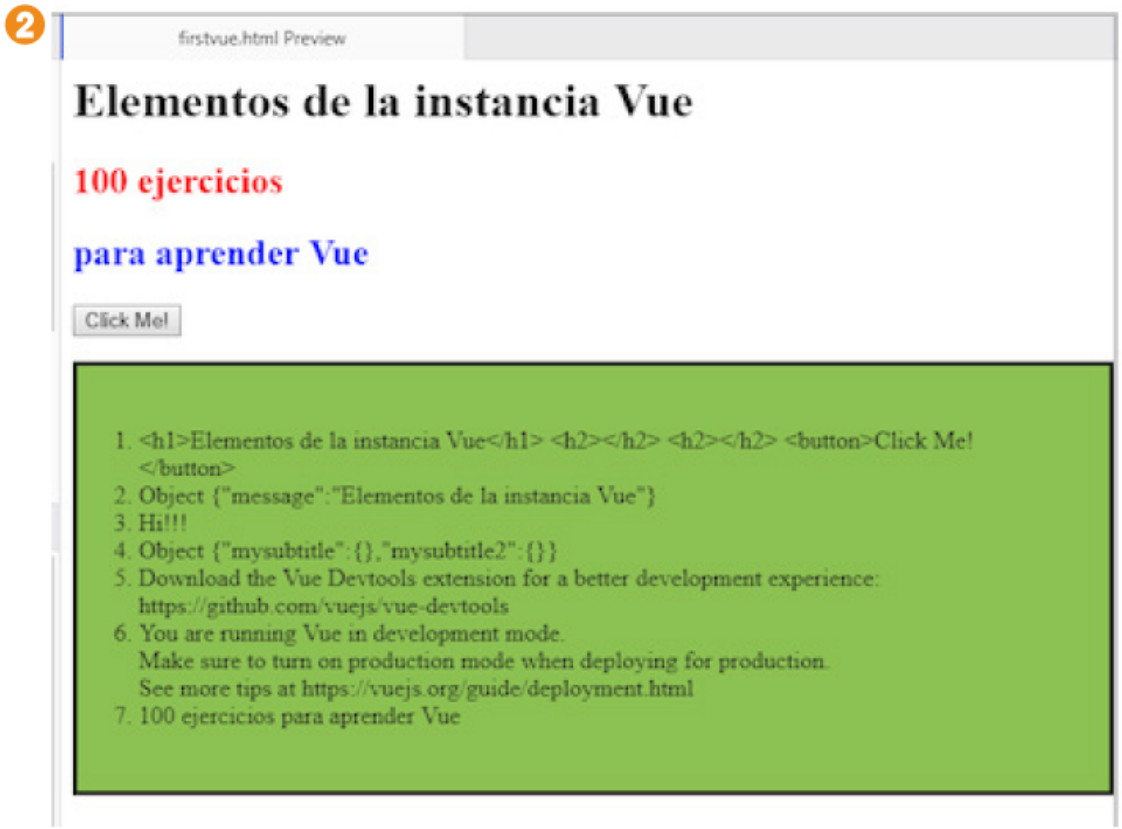
firstvue.html

```
<html><head><meta charset="utf-8">
  <title>My Vue</title><style src="firstvue.css"></style>
  <link rel="stylesheet" href="firstvue.css"></head>
<body> <div id="app">
      <h1>{{ message }}</h1> <h2 ref="mysubtitle"></h2>
      <h2 ref="mysubtitle2"></h2>
      <button @click="clickedButton">Click Me!</button>
  </div>
  <ul id="myULContainer" class="console"></ul>
```

firstvue.js

```
const app = new Vue({ el: '#app',
  data: { message: 'Elementos de la instancia Vue'},
  methods: { sayHi: function() { return 'Hi!!!'; },
             clickedButton: function() {
                this.$refs.mysubtitle.innerText = "100 ejercicios";
                this.$refs.mysubtitle.style.color = "red";
                this.$refs.mysubtitle2.innerText = "para aprender Vue";
                this.$refs.mysubtitle2.style.color = "blue";
                console.log([app.$refs.mysubtitle.innerText,
                      app.$refs.mysubtitle2.innerText].join(" ")); } }
});
console.log(app.$el.innerHTML); console.log(app.$data);
console.log(app.sayHi());       console.log(app.$refs);
```




2
firstvue.html Preview
Elementos de la instancia Vue
100 ejercicios
para aprender Vue
Click Me!
1. <h1>Elementos de la instancia Vue</h1> <h2></h2> <h2></h2> <button>Click Me!
</button>
2. Object {"message":"Elementos de la instancia Vue"}
3. Hi!!!
4. Object {"mysubtitle":{},"mysubtitle2":{}}
5. Download the Vue Devtools extension for a better development experience:
https://github.com/vuejs/vue-devtools
6. You are running Vue in development mode.
Make sure to turn on production mode when deploying for production.
See more tips at https://vuejs.org/guide/deployment.html
7. 100 ejercicios para aprender Vue

Eventos
ESCUCHANDO EVENTOS

Podemos empezar a escuchar eventos añadiendo al input la directiva v-on seguido de que evento queremos escuchar para reaccionar invocando a un método concreto.

Hasta ahora hemos visto el doble enlazado de las propiedades de nuestra instancia, dentro del bloque data. Podemos declarar una propiedad que se renderiza en nuestra interfaz de usuario y declaramos un input que pueda modificarla mediante la directiva v-model [1].

De esta forma, cada vez que modifiquemos el contenido del input se modificará la propiedad y se renderizará inmediatamente. Sin embargo, ¿cómo haríamos si queremos que se modifique solamente cuando se finalice su edición? ¿Cuándo detectamos que se ha acabado la edición?

En este caso vamos a interpretar que se acaba de editar el valor cuando se pulsa la tecla Enter de nuestro teclado. Para ello tendremos que detectar esta pulsación y esto lo podemos conseguir con la directiva v-on diciendo que ejecutaremos una acción justo cuando se deje de presionar (modificador keyup) la tecla Enter (modificador .enter)

v-on:keyup.enter=”onEnterPressed”

De esta forma vemos que cuando dejamos de escribir y cuando soltamos la tecla Enter se ejecuta la acción que hemos definido, que no será otra que la de renderizar el contenido de la variable en la página [2].

Comprobamos además que solo se ejecuta en este caso como se puede ver en el log de la consola en la que se imprime un mensaje cuando se llama.

Escuchar eventos de teclado suele ser una práctica muy común para interceptar eventos según ocurran. Para invocar determinadas acciones, hemos probado el evento keyup. Además, podemos establecer un modificador concreto a invocar tras la captura del evento “haber pulsado una tecla”, por ejemplo, capturar una tecla en concreto, la tecla escape o incluso definir alias para las F del teclado. Esto último se realizaría mediante la configuración en Vue de los códigos de tecla: “Vue.config.keyCodes.f” [3].

1 firstvue.html

```
<html><head><meta charset="utf-8">
  <title>My Vue</title><style src="firstvue.css"></style>
  <link rel="stylesheet" href="firstvue.css"></head>
<body>
  <div id="app">
    <h1>{{ message }}</h1>
    <input type="text" v-on:keyup.enter="onEnterPressed" v-model="inputText"/>
    <p>{{ enterMessage }}</p>
  </div>
```

firstvue.html Preview

Escuchando evento teclado [Enter]

firstvue.js

```
const app = new Vue({ el: '#app',
  data: { message: 'Escuchando evento teclado [Enter]',
          inputText: '', enterMessage: '' },
  methods: {
    onEnterPressed: function() {
      this.enterMessage = this.inputText;
      console.log(this.enterMessage); } } });
```


2
firstvue.html Preview
Escuchando evento teclado [Enter]
Aprendamos Vue!
Aprendamos Vue!


3
firstvue.html
<html><head><meta charset="utf-8">
<body>

<ul id="myULContainer" class="console">
firstvue.js
const app = new Vue({ el: '#app',
data: { message: 'Escuchando evento teclado [Enter]',
inputText: '', enterMessage: '' },
methods: {
onEnterPressed: function() {
this.enterMessage = this.inputText;
console.log(this.enterMessage); } } });
Escuchando evento teclado [Enter]
Aprendamos Vue!
Aprendamos Vue!
1. Download the Vue Devtools extension for a better development experience:
https://github.com/vuejs/vue-devtools
2. You are running Vue in development mode.
Make sure to turn on production mode when deploying for production.
See more tips at https://vuejs.org/guide/deployment.html
3. Aprendamos Vue!

Renderizar HTML
HTML EN UNA PROPIEDAD

En este ejercicio veremos cómo añadir código HTML para que sea renderizado dentro de nuestra instancia Vue.

En principio podríamos pensar que es algo muy sencillo, ¿no? Creamos una nueva instancia dentro de un elemento, una propiedad myhtml dentro del bloque data a la que asignamos una cadena con el formato de marcado HTML y la mostramos dentro de un bloque {{ myhtml }} [1].

¿Esto funcionaría? Bueno, la respuesta en este caso sería negativa ya que dentro de la instancia no se renderizaría nuestro código HTML si no como un simple texto. Esto es así para evitar una vulnerabilidad crítica en la web (XSS: cross site scripting) que pudiera permitir una modificación maliciosa del código HTML de la variable.

¿Y no existe ninguna forma de añadir código HTML? Sí que la hay, Vue nos da una directiva específica, que igualmente deberemos utilizar siempre con cuidado, denominada v-html.

En el ejercicio declaramos la variable html con el contenido a renderizar que simplemente es un título y un link a otra página y dentro de un elemento div indicamos mediante la directiva v-html que dentro se renderizará el código HTML. Fácil ¿verdad? [2]

1

firstvue.html

```
<title>My Vue</title>
<style src="firstvue.css"></style>
<link rel="stylesheet" href="firstvue.css">
</head>

<body>
  <div id="app">
    <div v-html="html"></div>
  </div>
```

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    html: `
    <div>
       <h1>Ejemplo de renderizado HTML</h1>
       Este en lace nos lleva a <a href="https://www.google.com">Google</a>
    </div>`,
  },
  methods: {
  }
});
```

firstvue.html Preview

Ejemplo de renderizado HTML

Este en lace nos lleva a Google

2

firstvue.html

```
<title>My Vue</title>
<style src="firstvue.css"></style>
<link rel="stylesheet" href="firstvue.css">
</head>

<body>
  <div id="app">
    <div v-html="html"></div>
  </div>
```

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    html: `
    <div>
       <h1>Ejemplo de renderizado HTML</h1>
       Este en lace nos lleva a <a href="https://www.google.com">Google</a>
    </div>`,
  },
  methods: {
  }
});
```

firstvue.html Preview

Buscar con Google Me siento con

Ofrecido por Google en: español català galego

Vincular propiedades
USO DE V-BIND

Con la directiva v-bind podemos vincular un atributo de algún elemento html a una propiedad de nuestra instancia [1].

Así contado parece algo abstracto pero para ilustrarlo en este ejercicio tendremos simplemente dos elementos: un área en la cual se van a mostrar imágenes y un botón que, cuando lo accionemos, modificará esta imagen.

Para la imagen tendremos un elemento HTML como img que se encarga de mostrar una imagen que le indiquemos dentro de la propiedad src.

<img src=”placeholder.jpg” alt=”placeholder”>

Por otro lado en nuestra instancia tenemos una propiedad llamada image que tendrá la ruta a la imagen que queremos mostrar.

Ahora simplemente tendremos que vincular o enlazar la propiedad image con el atributo src del elemento img mediante la directiva v-bind.

<img v-bind:src=”image”></img>

Una alternativa al v-bind sería utilizar los dos puntos, es decir, en lugar de “v-bind:src=…”, podríamos utilizar “:src=…”, muy útil a la hora de trabajar, por ahorrarnos tener que escribir más de la cuenta. Por lo que podríamos trabajar indistintamente con cualquiera de las dos alternativas.

Finalmente, añadimos el botón con una directiva v-on, que escuchará el evento click, que podríamos también haber simplificado con un @click, en lugar de v-on:click , para ejecutar el método responsable de cambiar el valor de la propiedad y cargar una nueva ruta de imagen al azar [2] y 3].

De esta forma tan sencilla hemos ejemplificado la definición que hablábamos al principio. Hemos enlazado un atributo de un elemento HTML con una propiedad de nuestra instancia.

2

firstvue.html

```
</head>

<body>
  <div id="app">
    <h1>v-bind image</h1>
    <div class="button">
      <button type="button" v-on:click="handleClick">Change image!</button>
    </div>
    <div class="image">
      <img v-bind:src="image"></img>
    </div>
  </div>
```

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    image: 'https://via.placeholder.com/320x240.png',
    index: 0
  },
  methods: {
    handleClick: function() {
      this.image = `https://loremflickr.com/320/240?random=${this.index++}`;
    }
  }
});
```

firstvue.html Preview

3

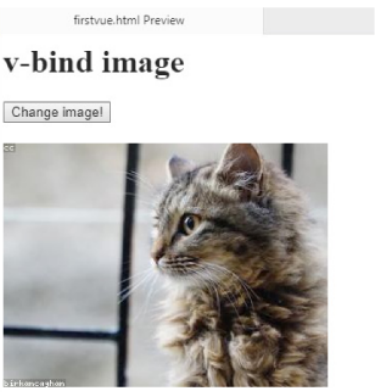
firstvue.html

```
</head>

<body>
  <div id="app">
    <h1>v-bind image</h1>
    <div class="button">
      <button type="button" v-on:click="handleClick">Change image!</button>
    </div>
    <div class="image">
      <img v-bind:src="image"></img>
    </div>
  </div>
```

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    image: 'https://via.placeholder.com/320x240.png',
    index: 0
  },
  methods: {
    handleClick: function() {
      this.image = `https://loremflickr.com/320/240?random=${this.index++}`;
    }
  }
});
```

firstvue.html Preview



Condicionales
USO V-IF Y V-ELSE

Durante el desarrollo de una aplicación es seguro que se nos presentarán ocasiones en las que queramos renderizar un bloque u otro de nuestra interfaz de usuario en función de alguna determinada condición.

En este caso haremos uso de las directivas v-if y v-else y, como ejemplo de las mismas, crearemos un nuevo ejercicio en el que en base a un selector mostraremos uno u otro bloque [1].

En este pequeño ejercicio se pregunta al usuario qué le gusta más: los gatos o los perros; y en función de su elección mostraremos un mensaje y una foto de su animal favorito.

La directiva v-if siempre contiene una condición en su interior, del estilo v-if=”miCondicion”, si quisiéramos añadir una segunda alternativa al bloque anterior, proseguiríamos con un “sino si…”, es decir “v-else-if”, también acompañada por una condición en su interior como la anterior. En caso de añadir la condición de, si no se han cumplido todas las anteriores, entonces habilita este bloque, utilizaríamos la directiva v-else, sin condición.

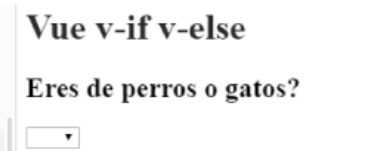
Para llevarlo a cabo necesitaremos una propiedad de la instancia Vue que denominaremos *pet*, que ligaremos con la directiva v-model al selector. Al seleccionar uno de los valores posibles se dispara un cambio en las diferentes propiedades de la instancia. Posteriormente crearemos los dos bloques con el contenido opcional y los marcaremos con las directivas v-if y v-else. Estas directivas evalúan la condición y, en función de lo escogido, en el selector se mostrará el bloque del perro 2 o del gato 3.

1

```
<body>
  <div id="app">
    <h1>Vue v-if v-else</h1>
    <h2>Eres de perros o gatos?</h2>

    <select v-model="pet" v-on:change="choosePet">
      <option></option>
      <option>dog</option>
      <option>cat</option>
    </select>
    <div v-if="this.pet!=''">
      <div v-if="this.pet === 'dog'">
        <h1> Woof!</h1>
        <img v-bind:src="image"></img>
      </div>
      <div v-else="this.pet !== 'dog'">
        <h1> Miau!</h1>
        <img v-bind:src="image"></img>
      </div>
    </div>
  </div>
```

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    pet: '',
    image: ''
  },
  methods: {
    choosePet: function(){
      this.image = `https://loremflickr.com/320/240/${this.pet}`;
    }
  }
```



2

```
<body>
  <div id="app">
    <h1>Vue v-if v-else</h1>
    <h2>Eres de perros o gatos?</h2>

    <select v-model="pet" v-on:change="choosePet">
      <option></option>
      <option>dog</option>
      <option>cat</option>
    </select>
    <div v-if="this.pet!=''">
      <div v-if="this.pet === 'dog'">
        <h1> Woof!</h1>
        <img v-bind:src="image"></img>
      </div>
      <div v-else="this.pet !== 'dog'">
        <h1> Miau!</h1>
        <img v-bind:src="image"></img>
      </div>
    </div>
  </div>
```

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    pet: '',
    image: ''
  },
  methods: {
```

3

```
<body>
  <div id="app">
    <h1>Vue v-if v-else</h1>
    <h2>Eres de perros o gatos?</h2>

    <select v-model="pet" v-on:change="choosePet">
      <option></option>
      <option>dog</option>
      <option>cat</option>
    </select>
    <div v-if="this.pet!=''">
      <div v-if="this.pet === 'dog'">
        <h1> Woof!</h1>
        <img v-bind:src="image"></img>
      </div>
      <div v-else="this.pet !== 'dog'">
        <h1> Miau!</h1>
        <img v-bind:src="image"></img>
      </div>
    </div>
  </div>
```

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    pet: '',
    image: ''
  },
  methods: {
```

Listas
RENDERIZADO DE LISTAS

Otra situación típica en la creación de nuevas interfaces de usuario es la enumeración de elementos dentro de una lista. Para ello en Vue se utiliza la directiva v-for a la que se le pasa una colección de elementos que se representarán dentro del bloque.

Como ejercicio plantearemos una clásica lista de la compra. Simplemente crearemos dentro de la instancia Vue una propiedad que denominaremos *shoppingList* que es un array de cadenas de texto que representan cada uno de los productos de la lista.

Con la propiedad definida solo nos resta indicar mediante la directiva v-for que iteraremos por cada producto de nuestra lista de la compra y renderizaremos cada elemento de la lista <li>.

<li v-for=”product in shoppingList”> {{ product }}

Con esta simple definición ya tenemos nuestra lista de la compra. [1]

Modificando ligeramente el código podemos añadir un nuevo input que añada elementos a mi compra con algunos mecanismos que hemos aprendido en anteriores ejercicios. De esta forma vemos cómo si añadimos otro nuevo elemento a nuestra lista de productos se actualiza nuestra compra [2].

1

firstvue.html

```
<head>
  <meta charset="utf-8">
  <title>My Vue</title>
  <style src="firstvue.css"></style>
  <link rel="stylesheet" href="firstvue.css">
</head>

<body>
  <div id="app">
    <h1>v-for</h1>
    <h2>Lista de la compra</h2>
    <ul>
      <li v-for="product in shoppingList">
        {{ product }}
      </li>
    </ul>
  </div>
```

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    shoppingList: ['manzanas', 'leche', 'hamburguesa', 'patatas fritas'],
  },
  methods: {
  }
});
```

firstvue.html Preview

v-for

Lista de la compra

- manzanas
- leche
- hamburguesa
- patatas fritas

2

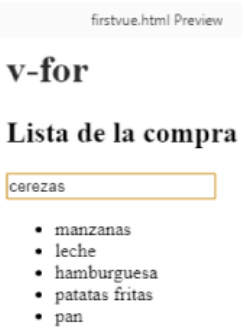
firstvue.html

```
<div id="app">
  <h1>v-for</h1>
  <h2>Lista de la compra</h2>
  <input v-on:keyup.enter="onEnterPressed" v-model="newProduct"/>
  <ul>
    <li v-for="product in shoppingList">
      {{ product }}
    </li>
  </ul>
</div>
```

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    newProduct: '',
    shoppingList: ['manzanas', 'leche', 'hamburguesa', 'patatas fritas'],
  },
  methods: {
    onEnterPressed: function() {
      this.shoppingList.push(this.newProduct);
      this.newProduct = '';
    }
  }
});
```

firstvue.html Preview



Enumeración de propiedades
V-FOR PARA OBJETOS

Hemos visto cómo mostrar una enumeración de productos de nuestra lista de la compra con la directiva v-for y en este ejercicio veremos cómo mostrar un listado de las propiedades de un objeto de nuestra instancia Vue.

Tenemos una propiedad dentro de la instancia Vue que es un objeto que representa la información de uno de los posts de nuestro blog y que queremos renderizar.

```
post: {
title: 'Nuevo artículo',
date: '24/03/2019',
author: 'Ramón',
}
```

Ahora lo que tenemos que hacer es usar la directiva v-for con el siguiente formato, descomponiendo el objeto en sus propiedades en pares clave-valor.

```
<ul>
<li v-for="(value, key) in post" v-blue>
{{ key }}: {{ value }}
</li>
</ul>
```

También podríamos usar el índice en esta descomposición, si lo necesitáramos, de la siguiente manera:

```
<ul>
<li v-for="(value, key, index) in post"
style= "list-style-type:none;" v-blue>
{{ index + 1 }} => {{ key }}: {{ value }}
</li>
</ul>
```

Podemos ver el resultado de la inspección del objeto en la imagen [1] y para seguir jugando con nuestro ejemplo meter un nuevo atributo y ver cómo se renderiza perfectamente cuando guardamos. [2]

1

firstvue.html

```
<div id="app">
  <h1>v-for object</h1>
  <h2></h2>
  <ul>
    <li v-for="(value, key, index)  in post"
        style= "list-style-type:none;">
      {{ index + 1 }} => {{ key }}: {{ value }}
    </li>
  </ul>
</div>
```

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    post: {
      title: 'Nuevo artículo',
      date: '24/03/2019',
      author: 'Ramón',
    },
  }
});
```

firstvue.html Preview

v-for object

1 => title: Nuevo artículo
2 => date: 24/03/2019
3 => author: Ramón

2

firstvue.html

```
<div id="app">
  <h1>v-for object</h1>
  <h2></h2>
  <ul>
    <li v-for="(value, key, index)  in post"
        style= "list-style-type:none;">
      {{ index + 1 }} => {{ key }}: {{ value }}
    </li>
  </ul>
</div>
```

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    post: {
      title: 'Nuevo artículo',
      date: '24/03/2019',
      author: 'Ramón',
      content: 'Este es el contenido'
    },
  }
});
```

firstvue.html Preview

v-for object

1 => title: Nuevo artículo
2 => date: 24/03/2019
3 => author: Ramón
4 => content: Este es el contenido

Filtros
MODIFICADORES DE VALOR

Vamos a imaginarnos que tenemos nuestra interfaz que renderiza la lista de la compra que hemos realizado en anteriores ejercicios. Hemos sacado la aplicación que muestra la lista de la compra pero los usuarios se quejan de que se muestran en letras minúsculas y ellos quieren que se vea bien grande y se muestre en letras mayúsculas aun cuando ellos los introduzcan en minúsculas.[1]

Podremos modificar este comportamiento con una herramienta que pone a nuestra disposición Vue: los filtros.

Definiremos un filtro que pase a mayúsculas cada valor que se le pase en este caso de la siguiente forma:

```
Vue.filter('productName', function(value){
if(!value) return ''
value = value.toString();
return value.toUpperCase();
});
```

Y en la declaración HTML lo haremos de la siguiente forma

```
<ul>
<li v-for="product in shoppingList">
{{ product | productName }}
/li>
</ul>
```

Esta notación es el típico pipe que es muy común y que en este caso se podría leer algo así como "a este valor la aplico esta función o transformación".[2]

1

firstvue.html

```
<div id="app">
  <h1>Vue filters</h1>
  <h2>Lista de la compra</h2>
  <ul>
    <li v-for="product in shoppingList">
      {{ product }}
    </li>
  </ul>
</div>
```

firstvue.js

```
Vue.filter('productName', function(value){
  if(!value) return ''
  value = value.toString();
  return value.toUpperCase();
});

const app = new Vue({
  el: '#app',
  data: {
    shoppingList: ['apple', 'orange', 'banana', 'grapes'],
  },
  methods: {
  }
});
```

firstvue.html Preview

2

firstvue.html

```
<div id="app">
  <h1>Vue filters</h1>
  <h2>Lista de la compra</h2>
  <ul>
    <li v-for="product in shoppingList">
      {{ product | productName }}
    </li>
  </ul>
</div>
```

firstvue.js

```
Vue.filter('productName', function(value){
  if(!value) return ''
  value = value.toString();
  return value.toUpperCase();
});

const app = new Vue({
  el: '#app',
  data: {
    shoppingList: ['apple', 'orange', 'banana', 'grapes'],
  },
  methods: {
  }
});
```

firstvue.html Preview

Renderizar solo una vez
DIRECTIVA V-ONCE

En caso de que necesitemos mostrar un único valor cuando carguemos la página haremos uso de una directiva especial llamada v-once. Esta directiva asegura que un elemento solo tomará un valor durante el renderizado y, en el caso de que la página se vuelva a renderizar, considerará al elemento como un elemento estático que no se tomará en cuenta de nuevo.

Una razón por la que se puede marcar un elemento con la directiva v-once sería optimizar el rendimiento en la actualización de datos. El contenido del elemento que se renderice una sola vez podría ser costoso.

En este ejemplo vemos cómo se renderiza una sola vez aquello que tengamos dentro de la variable incluida en el input [1].

Si posteriormente a la carga modificamos el contenido del input y, por tanto, el contenido de la variable podemos comprobar cómo no se renderiza de nuevo el contenido de nuestro título [2].

1

firstvue.html

```
11   <div id="app">
12     <h1 v-once>{{ message }}</h1>
13
14     <input type="text" v-model="message"/>
15   </div>
```

firstvue.js

```
1  const app = new Vue({
2    el: '#app',
3    data: {
4      message: 'Solo una vez',
5    },
6    methods: {
7    }
8  });
```

firstvue.html Preview

Solo una vez

Solo una vez

2

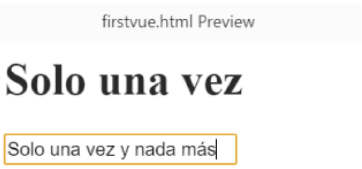
firstvue.html

```
<div id="app">
  <h1 v-once>{{ message }}</h1>

  <input type="text" v-model="message"/>
</div>
```

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    message: 'Solo una vez',
  },
  methods: {
  }
});
```

firstvue.html Preview



firstvue.html

```
<div id="app">
  <h1 v-once>{{ message }}</h1>

  <input type="text" v-model="message"/>
</div>
```

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    message: 'Solo una vez',
  },
  methods: {
  }
});
```

Mouse move
DIRECTIVA V-ON:MOUSEMOVE

En esta ocasión vamos a ver dentro de las directivas para capturar eventos del DOM aquella que nos permite filtrar los generados cuando el ratón se mueve por encima de uno de nuestros componentes.

Estamos hablando de la directiva v-on acompañado del evento mousemove [1]:

A este div le daremos unas dimensiones mediante la clase CSS mousearea [2]:

.mousearea {
width: 300px;
border: 15px solid green;
padding: 50px;
margin: 20px;
}

Enlazaremos el evento con una función de la instancia para ir modificando las coordenadas mientras el ratón está en movimiento. Estas coordenadas las guardaremos y las mostraremos en otro div diferente como resultado.

methods: {
move:function(event){
this.mouseX = event.clientX;
this.mouseY = event.clientY;
}
}

{{mouseX}} - {{mouseY}}

A medida que se captura el movimiento del ratón se ejecuta la función y vamos refrescando las variables de la instancia que se renderizarán en nuestra página [3].

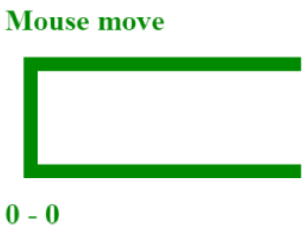
2

firstvue.html

```
h1 {
  color: green;
}
.mousearea {
  width: 300px;
  border: 15px solid green;
  padding: 50px;
  margin: 20px;
}
.result {
  color: green;
  font-size: 2em;
  font-weight: bold;
}
```

firstvue.css

firstvue.html Preview



firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    mouseX:'0',
```

3

firstvue.html

```
</head>

<body>
  <div id="app">
    <h1>Mouse move</h1>
    <div class="mousearea" v-on:mousemove="move">
    </div>
    <div class="result">{{mouseX}} - {{mouseY}}</div>
  </div>
```

firstvue.html Preview

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    mouseX:'0',
    mouseY:'0'
  },
```

Directivas personalizadas
DEFINIENDO UNA DIRECTIVA GLOBAL

Vue permite la creación y registro de nuevas directivas a las que podremos dotar el comportamiento personalizado que queramos.

Las directivas personalizadas en Vue como el resto suelen empezar por v- y el nombre de la directiva. Como primer ejemplo empezaremos definiendo una directiva muy simple que cambie el color del texto renderizado.

De esta forma tendríamos una declaración tal que así:

<h1 v-red>Aprendiendo Vue!</h1>

En la que modificaremos el color del título a rojo tal y como indica la propia directiva. El código necesario en Javascript para definir la directiva sería el siguiente:

Vue.directive(“red”, function(el, binding, vnode) {
el.style.color = “red”;
});

Este código se ejecutará en el bind que es una función que se llama solo una vez cuando se enlaza la primera vez la directiva al elemento. Siendo más estrictos podríamos escribirlo así y con más funciones en función de cuando se quiere ejecutar el código.

Vue.directive(“red”, {
bind: function(el, binding, vnode) {
el.style.color = “red”;
}
});

Poniendo todo junto en el ejercicio como resultado tendríamos nuestros textos cambiados a color rojo ya que modificamos la propiedad color del estilo del elemento [1].

Para seguir jugando con nuestro ejemplo podríamos definir otra directiva con el color azul y aplicarla al otro texto y ver el resultado[2].

2

firstvue.html

```
<div id="app">
  <h1 v-red>Aprendiendo Vue!</h1>
  <p v-blue>
    Usando la directiva personalizada red!
  </p>
</div>
```

firstvue.js

```
Vue.directive("red", function(el, binding, vnode) {
  el.style.color = "red";
});

Vue.directive("blue", function(el, binding, vnode) {
  el.style.color = "blue";
});

const app = new Vue({
  el: '#app'
});
```

firstvue.html Preview

Directivas personalizadas II
DIRECTIVAS CON ARGUMENTOS

Hemos visto cómo crear una directiva personalizada muy sencilla y en este nuevo ejercicio haremos una pequeña variación para que sea algo más genérica. Veíamos que para definir nuevos colores teníamos que crear nuevas directivas pero si tenemos que crear una para cada color… vaya lío, ¿no?

Lo ideal es que pudiéramos indicarlo de alguna forma y solo mantener una directiva y, para conseguirlo, sería genial poder tener una única directiva y pasarle un argumento con el color. ¡Esto es lo que vamos a hacer en este ejercicio!

<h1 v-color:blue>{{message}}</h1>

<p v-color:red> Usando la directiva personalizada v-color:arg! </p>

Declaramos la nueva directiva como v-color pero además le pasamos como argumento el color deseado de forma que colorearemos diferente cada texto según el argumento dado.

Si vamos a la definición podremos hacer algo como esto:

Vue.directive(“color”, function(el, binding, vnode) {

el.style.color = binding.arg;

console.log(binding);

});

El objeto binding guarda muchas propiedades como el nombre de la directiva, el valor, los modificadores, etc., y entre estas propiedades también podemos obtener el argumento [1]. Es este argumento el que recuperaremos y con el que podremos modificar la propiedad color del estilo del elemento [2].

1

firstvue.js

```
1  Vue.directive("color", function(el, binding, vnode) {
2      el.style.color = binding.arg;
3      console.log(binding);
4  });
5  const app = new Vue({
6    el: '#app',
7    data: {
8      message:'Aprendiendo Vue!!!'}
9  });
10
```

2

firstvue.html

```
11   <div id="app">
12     <h1 v-color:blue>{{message}}</h1>
13     <p v-color:red>
14       Usando la directiva personalizada v-color:arg!
15     </p>
16   </div>
```

firstvue.js

```
1  Vue.directive("color", function(el, binding, vnode) {
2      el.style.color = binding.arg;
3      console.log(binding);
4  });
5  const app = new Vue({
6    el: '#app',
7    data: {
8      message:'Aprendiendo Vue!!!'}
9  });
10
```

firstvue.html Preview

Aprendiendo Vue!!!

Usando la directiva personalizada v-color:arg!

Directivas personalizadas III
UTILIZANDO MODIFICADORES

En este nuevo ejercicio utilizaremos más propiedades de las directivas, hasta hora hemos visto como crear una directiva que cambia el color de un texto. Una muy simple con un comportamiento estático modificando el estilo del texto y otra un poco más avanzada en la que introducimos un parámetro.

En esta ocasión introduciremos el uso de los modificadores, que son simplemente nuevos decoradores que se le pueden añadir o no a la directiva.

Volviendo a las modificaciones de estilo de los textos, un buen ejemplo sería decidir el formato en función de ciertos modificadores. Para ello crearemos la directiva v-format que podrá recibir diferentes modificadores que cambien el estilo del texto del elemento.

Los modificadores se declaran a partir de un punto y seguido al nombre de la directiva u otro modificador. Este podría ser un ejemplo de declaración:

```
<h1 v-format.bold.underline.highlight>{{message}}</h1>
```

Para el código Javascript tendríamos algo como lo siguiente:

```
Vue.directive(“format”, function(el, binding, vnode) {
const modifiers = binding.modifiers; if (modifiers.underline) {
el.style.textDecoration = “underline”;
}
if (modifiers.bold) {
el.style.fontWeight = “bold”;
}
if (modifiers.highlight) {
el.style.background = “#eaeaea”;
}
});
```

En el objeto binding tenemos los modificadores [1] que pueden haberse aplicado y que podremos consultar y aplicar en caso de que existan [2].

1

firstvue.js

```
Vue.directive("format", function(el, binding, vnode) {
  const modifiers = binding.modifiers;

if (modifiers.underline) {
    el.style.textDecoration = "underline";
  }

if (modifiers.bold) {
    el.style.fontWeight = "bold";
  }

if (modifiers.highlight) {
    el.style.background = "#eaeaea";
  }
});

const app = new Vue({
  el: '#app',
  data: {
    message: "Aprendiendo Vue!"
  }
});

```

2

firstvue.html

```
<div id="app">
  <h1 v-format.bold.underline.highlight>{{message}}</h
</div>
```

firstvue.js

```
Vue.directive("format", function(el, binding, vnode) {
  const modifiers = binding.modifiers;

if (modifiers.underline) {
    el.style.textDecoration = "underline";
  }

if (modifiers.bold) {
    el.style.fontWeight = "bold";
  }

if (modifiers.highlight) {
    el.style.background = "#eaeaea";
  }
});

const app = new Vue({
  el: '#app',
  data: {
    message: "Aprendiendo Vue!"
  }
});
```

firstvue.html Preview

Aprendiendo Vue!

Directivas personalizadas IV
DIRECTIVAS EN LA INSTANCIA

Hasta ahora hemos visto cómo crear y declarar nuestras directivas personalizadas de forma global pero en este ejercicio podremos ver cómo se definen de forma local dentro de una instancia o componente Vue.

Para ello simplemente tendremos que ir al código javascript de nuestra instancia y añadir un nuevo objeto *directives* con una sintaxis ligeramente diferente a la que hemos visto hasta ahora.

En este ejemplo condensaremos varias de las directivas de los ejercicios anteriores añadiéndolas en este apartado con sus correspondientes funciones.

En la declaración no cambia nada:

```
<h1 v-color:yellow v-format.bold.underline.highlight>{{message}}</h1>
```

Y el código JS quedaría de la siguiente forma:

```
directives: {
'white': {
bind(el, binding, vnode) {
el.style.color = '#fff';
}
},
'color': {
bind(el, binding, vnode) {
el.style.color = binding.arg;
}
},
'format': {
bind(el, binding, vnode) {
const modifiers = binding.modifiers;
if (modifiers.underline) {
el.style.textDecoration = "underline";
}
…
},
}
}
});
```

Tendríamos un resultado similar [1] al de ejercicios anteriores [2].

1

firstvue.html

```
<div id="app">
  <h1 v-color:yellow v-format.bold.underline.highlight>
    {{message}}
  </h1>
</div>
```

firstvue.js

```
const app = new Vue({
  el: '#app',
  data: {
    message: "Aprendiendo Vue!"
  },
  directives: {
    'white': {
      bind(el, binding, vnode) {
          el.style.color = '#fff';
      }
    },
    'color': {
      bind(el, binding, vnode) {
          el.style.color = binding.arg;
      }
    },
    'format': {
      bind(el, binding, vnode) {
        const modifiers = binding.modifiers;
```

firstvue.html Preview

2

firstvue.html

```
directives: {
  'white': {
    bind(el, binding, vnode) {
        el.style.color = '#fff';
    }
  },
  'color': {
    bind(el, binding, vnode) {
        el.style.color = binding.arg;
    }
  },
  'format': {
    bind(el, binding, vnode) {
      const modifiers = binding.modifiers;

      if (modifiers.underline) {
          el.style.textDecoration = "underline";
      }

      if (modifiers.bold) {
          el.style.fontWeight = "bold";
      }

      if (modifiers.highlight) {
          el.style.background = "#ff0000";
      }
    }
```

firstvue.js

firstvue.html Preview

Componente
COMPONENTE

IMPORTANTE

La nomenclatura del componente es importante, debe ser de tipo camel-case “MiComponente” o kebab-case “mi-componente”, en definitiva, una forma de unir palabras.

Los componentes en Vue son unos contenedores donde implementamos lógica para que funcionen de manera aislada y puedan ser reutilizados por la aplicación.

Creemos un componente donde, al escribir una palabra, se muestre su traducción si existe en el diccionario.

Al usar CDN en lugar del CLI, para definir un componente, usamos “Vue.component”, cuyo primer parámetro es el nombre del componente.

Creamos la página html, que incluye la librería de vue, el css y el contenedor de la aplicación, donde incluimos la etiqueta del componente a crear, “<my-translator />” [1].

Además, creamos el fichero “firstvue.css”, con un estilo para agrandar los elementos de tipo texto [2].

Definimos el componente, en el fichero “firstvue.js” [3]. Nombramos al componente como “my-translator”.

let Translator = Vue.component(‘my-translator’, { })

En la propiedad “template”, definimos su plantilla, mediante “template literal”, comillas “`” usadas a principio y fin del html de la plantilla, para visualizarla en varias líneas. Definimos una caja de texto, que representa la palabra “word”, que el usuario escribe para buscar su traducción. Mediante “v-model” vinculamos el input con el modelo.

Incluimos un botón para limpiar el valor del input al pulsarlo, así como las sugerencias que la propiedad computada “AnyMatch”, nos retorna.

En la propiedad “data” incluimos las variables, “placeholderWord”, texto en el placeholder del input, “word” y “dictionary”, que contiene una lista de palabras con sus traducciones entre castellano “ES” e inglés “EN”.

En la propiedad “computed” añadimos la propiedad computada “AnyMatch” que recorre el diccionario y compara cada palabra, de modo que si el valor contiene la palabra “word”, la añade a un array retornado por la función.

En la propiedad “methods” definimos el método “clear”, para vaciar el input al pulsar el botón “limpiar”.

Tras definir el componente, lo añadimos a la instancia de la aplicación, en la propiedad “components”, mediante kebab-case.

const app = new Vue({ el: ‘#app’, components: { ‘my-translator’: Translator } });

Abrimos en el navegador el html, visualizamos el componente, tras introducir letras en la caja de texto, deberían aparecer las sugerencias de traducción [4]. Si pulsamos el botón limpiar, se vacía el contenido del input [5] y las sugerencias.

1
```
<html><head><meta charset="utf-8">
  <title>Componente Traductor</title>
  <link rel="stylesheet" type="text/css" href="firstvue.css"> </head>
<body>                                  Aplicación compuesta por un componente
  <div id="app">                                Componente Traductor
    <my-translator></my-translator>
  </div>
  <script src="https://unpkg.com/vue"></script>
  <script src="firstvue.js"></script>
```

2
```
input[type=text]{ width: 20em; height: 3em; }
```

4 **Traductor**

h Limpiar

Hola->Hello

5 **Traductor Castellano-Inglés**

Introduzca Palabra a traducir Limpiar

3
```
                                                    Nombre del componente
let Translator = Vue.component('my-translator', {
 template: `<div component="my-translator">                                  Plantilla del componente
              <h1>Traductor</h1>
                                                          Data-bind palabra
              <input type="text" :placeholder="placeholderWord" v-model="word" />
              <input type="button" @click="Clear" value="Limpiar" />
              <div v-for="match in AnyMatch">   Evento clic, limpiar palabra
                <span>{{match.ES}}->{{match.EN}}</span>
              </div>
            </div>`
,data(){ return { placeholderWord:'Introduzca Palabra a traducir', word:''   Palabra
                , dictionary:[{'EN':'Hello', 'ES':'Hola'},{'EN':'Bye', 'ES':'Adios'}
    Array de palabras disponibles   ,{'EN':'Good', 'ES':'Bueno'},{'EN':'Bad', 'ES':'Malo'}] } }   Modelo
,computed:{
   AnyMatch(){ let match=false; let words=[];
                                                  Incluir palabra en lista si coincide
     this.dictionary.map((w)=>{if( this.word!='' &&
         w.ES.toLowerCase().includes(this.word.toLowerCase()) ) words.push(w) })
     return words }                                                  Propiedad Computada
}
                  Visualizamos sugerencias de traducciones si contiene la palabra escrita
,methods:{ Clear(){ this.word = '' } }   Método para limpiar el input de palabra
})
                                                  Incluimos los componentes que utiliza la aplicación
                  Instancia de la aplicación
const app = new Vue({ el: '#app', components: { 'my-translator': Translator } });
```

Componente parametrizables REUTILIZABLES

IMPORTANTE

Los componentes en entorno CDN se declaran como Vue.component, sin embargo su definición varía en entornos CLI, que veremos más adelante.

Los componentes pueden reutilizarse múltiples veces y parametrizarse para condicionar su comportamiento. Realicemos unas modificaciones en el ejercicio anterior para crear dos instancias del mismo componente y que su comportamiento sea diferente en base a un parámetro recibido en las props.

Para ello, el html incluye dos componentes del mismo tipo “my-translator” [1], el primero para traducciones de castellano, indicado mediante la propiedad “prop-language” con el valor “ES”, y el segundo para traducciones de inglés, con valor “EN”. La hoja de estilos “firstvue.css” será idéntica a la anterior [2].

La definición del componente [3] es idéntica al anterior, con algunas variaciones. En la sección de la plantilla, comprobamos el valor de la propiedad “propLanguage”, proporcionada en las props al componente, de modo que, dependiendo del idioma, mostraremos un texto u otro como título, mediante un condicional “v-if”. Este mismo condicional entrará en juego a la hora de mostrar las sugerencias recorridas con el “v-for”.

La sección del data no varía. Sin embargo, la sección de las propiedades computadas, en lugar de establecer por defecto el idioma “ES” lo hacemos dinámico mediante la prop.

this.dictionary.map((w)=> {if(this.word != ‘’ &&

w[this.propLanguage].toLowerCase().includes(this.word.toLowerCase())) words.push(w) })

Tanto tiempo hablando de “propLanguage” y no la habíamos definido; podemos ver cómo una propiedad puede influir en el comportamiento de nuestro componente. Definamos esta propiedad, estableciendo su tipo a texto mediante “type” y su valor por defecto “ES”, es decir, en caso de no indicar la propiedad, tomará este valor.

,props:{ propLanguage: {type:String,default:’ES’} }

Si abrimos el html en el navegador aparecen dos componentes, su título condicionado por el valor del idioma proporcionado a cada componente. Si escribimos en el componente de castellano, sugerirá palabras de castellano a inglés [4], si lo hacemos en el componente de inglés será al revés. Finalmente, una vez queramos borrar el contenido de sugerencias y la palabra, pulsaremos el botón “limpiar” del componente que deseemos vaciar [5], observando un comportamiento aislado entre componentes.

1
```
<html><head><meta charset="utf-8">
  <title>Componente Traductor</title>
  <link rel="stylesheet" type="text/css" href="firstvue.css"> </head>
<body>                                  Aplicación compuesta por dos componentes
  <div id="app">                        Componente Traductor. Castellano-Inglés
    <my-translator prop-language="ES"></my-translator>
    <my-translator  prop-language="EN"></my-translator>
  </div>                                Componente Traductor. Inglés-Castellano
  <script src="https://unpkg.com/vue"></script>
  <script src="firstvue.js"></script>
</body></html>
```

4 **Traductor Castellano-Inglés**

ho Limpiar

Hola->Hello

Traductor Inglés-Castellano

b Limpiar

Bye->Adios

Bad->Malo

2
```
input[type=text]{ width: 20em; height: 3em; }
```

5 **Traductor Castellano-Inglés**

Introduzca Palabra a traducir Limpiar

3
```
                                        Nombre del componente
let Translator = Vue.component('my-translator', {
 template: `<div component="my-translator">
                <h1>Traductor <b v-if="propLanguage=='ES'">Castellano-Inglés</b>
                              <b v-else>Inglés-Castellano</b></h1>   Data-bind palabra
                <input type="text" :placeholder="placeholderWord" v-model="word" />
                <input type="button" @click="Clear" value="Limpiar" />
                <div v-for="match in AnyMatch">   Evento clic, limpiar palabra
                  <span v-if="propLanguage=='ES'">{{match.ES}}->{{match.EN}}</span>
                  <span v-else>{{match.EN}}->{{match.ES}}</span>
                </div>   Visualizamos sugerencias de traducciones si contiene la palabra escrita
            </div>`
 ,data(){ return { placeholderWord:'Introduzca Palabra a traducir', word:''   Palabra
                 , dictionary:[{'EN':'Hello', 'ES':'Hola'},{'EN':'Bye', 'ES':'Adios'}
     Array de palabras disponibles    ,{'EN':'Good', 'ES':'Bueno'},{'EN':'Bad', 'ES':'Malo'}] } }
 ,computed:{
    AnyMatch(){ let match=false; let words=[];
      this.dictionary.map((w)=>{if( this.word!='' &&        Incluir palabra en lista si coincide
          w[this.propLanguage].toLowerCase().includes(this.word.toLowerCase()) ) words.push(w) })
      return words }
 }
 ,props:{ propLanguage: {type:String,default:'ES'} }   Propiedades, en este caso Idioma,
                                                       recibe valor por defecto ES y debe ser texto
 ,methods:{ Clear(){ this.word = '' } }   Método para limpiar el input de palabra
})
                                        Incluimos los componentes que utiliza la aplicación
  Instancia de la aplicación
const app = new Vue({ el: '#app', components: { 'my-translator': Translator } });
```

Plantilla del componente

Modelo

Propiedad Computada

Componente Tipo Propiedades
TIPOS DE PROPIEDADES

IMPORTANTE

Cuando enviamos propiedades al componente, si ponemos los dos puntos delante del nombre de la propiedad, estamos enviando un valor dinámico, en caso contrario será un valor en plano.

Hemos visto la importancia de las props en los componentes, son la primera línea de entrada de estos y por tanto deben ser el primer punto por validar para que nuestros componentes tengan el funcionamiento esperado.

Creemos un ejercicio para trabajar con los tipos de valores que podemos validar [1].

Para ello, creamos en el fichero “firstvue.js”, el componente “my-props” [2], donde en la sección props, definiremos:

• propRequired: texto, debido al “type:String” y será requerido recibir valor “required:true”.

• propArray: array, su valor por defecto se asigna mediante “default”, en este caso un array vacío. Usamos “validator”, función de validación, para verificar su tamaño mínimo.

• propObject: objeto, cuyo valor por defecto es un objeto vacío. En este caso, queremos tratar el objeto para retornar un valor, lo haremos mediante la propiedad computada “CompleteName”.

• propMultipleValue: permite número, texto o fecha, al incluirlos en un array de tipos permitidos.

Las propiedades recibidas en las “props” podemos utilizarlas directamente en la plantilla, inicializar variables del data o tratar el valor recibido por las props, en las propiedades computadas y retornar el valor tratado, como es el caso de “CompleteName”. Aquí tratamos el objeto recibido, retornando el nombre completo como valor. En caso de no querer tratar el valor recibido, simplemente asignamos el valor de la prop a su respectiva variable en el data.

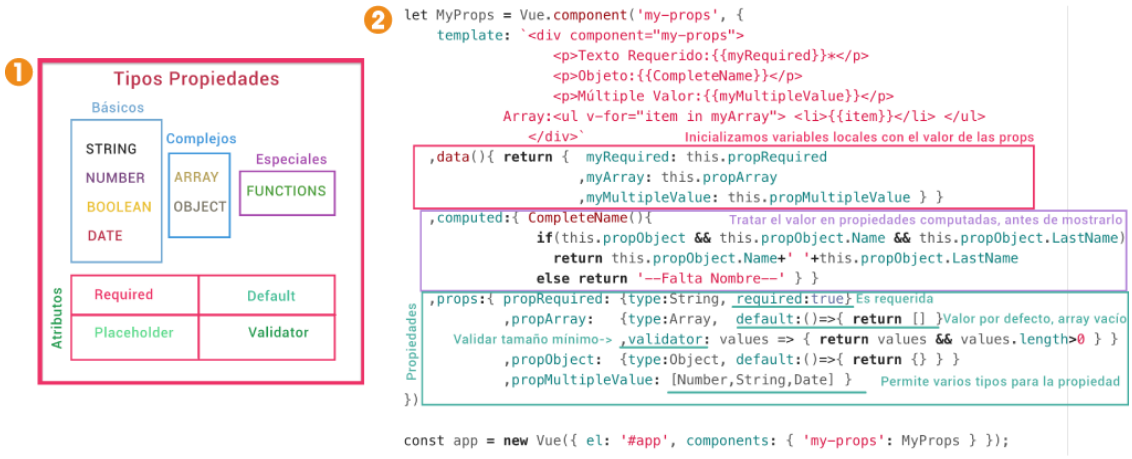
En la plantilla mostramos los valores de estas variables del data, así como de la propiedad computada.

Finalmente, en “firstvue.html” instanciamos el componente con cada propiedad rellena como debería [3]. De modo que, si visualizamos la página en el navegador, visualizamos los valores correctamente [4].

Sin embargo, si no enviásemos la propiedad requerida, o no enviásemos elementos en el array, la consola del navegador mostraría los errores [5], sería lo esperado, al no superar la validación:

<my-props :prop-array=[] :prop-object=’{“Name”:”Ramón”, “LastName”:”Serrano Valero”, “Age”:34}’

prop-multiple-value=”25-10-1984” />



3

```
<html><head><meta charset="utf-8">
  <title>Componente Tipos Propiedad</title></head>
<body>
  <div id="app">
    <my-props prop-required="Hola mundo"
              :prop-array=["España","Europa","Mundo"]
              :prop-object='{"Name":"Ramón", "LastName":"Serrano Valero", "Age":34}'
              prop-multiple-value="25-10-1984"
              />
  </div>
  <script src="https://unpkg.com/vue"></script>
  <script src="firstvue.js"></script>
</body></html>
```

Valor estático

Asignar valor dinámico

4

Texto Requerido:Hola mundo*

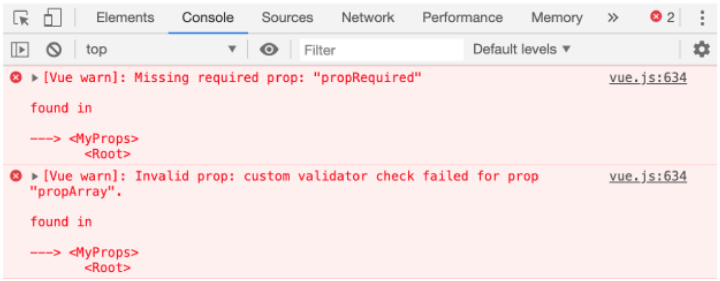
Objeto:Ramón Serrano Valero

Múltiple Valor:25-10-1984

Array:

- España
- Europa
- Mundo

5



Cambiar valor de props
CAMBIO VALOR

IMPORTANTE

Las propiedades recibidas por los componentes se reciben al cargar el componente, cualquier modificación de valor posterior, requiere capturar estos cambios mediante watch.

Hasta ahora, enviábamos un valor a un componente, recibiéndolo por las props, sin embargo, no hemos probado que ocurriría si actualizásemos dicho valor. ¿Refrescaría su valor el componente?

Creemos un ejercicio en el que actualicemos el valor de la prop y esta muestre el nuevo valor recibido.

Definimos en “firstvue.html” [1], dentro de la instancia de la aplicación, una caja de texto cuyo valor realice “data-bind” con la variable “valueData” del data de la aplicación cada vez que modifiquemos el texto de este input. Además, instanciamos el componente “my-props-component” que crearemos, al que proporcionamos por su propiedad “prop-value”, el valor de esta variable del data de la aplicación.

En el fichero “firstvue.js”, creamos el componente “my-props-component” [2], en el que definimos una propiedad de tipo texto “propValue”, en las props. Esta inicializa el valor de la variable local “myValue”, del data del componente.

Finalmente, agregamos el componente a la aplicación, mediante “components” y agregamos la variable “valueData” al data de la aplicación, con el valor por defecto “Valor Inicial”.

Abrimos la página html en el navegador, veremos el valor “Valor Inicial” tanto en el input como en el componente [3], dado que es el valor con el que inicializamos dicha variable de la aplicación y es recibida correctamente por las props del componente. Si ahora intentamos cambiar el valor de dicho input, vemos que el valor se está pasando por las props al componente, pero el componente no representa dichos cambios [4].

Esto se debe a que el valor proporcionado en las props inicializa las variables del data del componente la primera vez. De modo que, al recibir actualizaciones, el valor de dicha propiedad no lo controlamos en el componente.

Para controlar estos cambios de valor debemos hacer uso de las “watch”, y en este caso observar cambios en la propiedad “propValue”, por lo que si añadimos este “watch” al componente [5], asignamos el nuevo valor recibido al data. Abrimos de nuevo la página en el navegador y al cambiar el valor, el componente actualiza la vista [6].

1

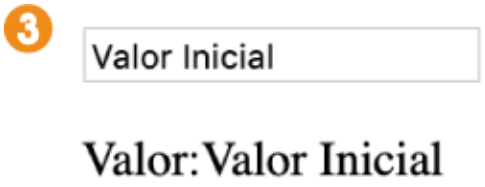
```
<html><head><meta charset="utf-8">
  <title>Actualizar Props</title></head>
<body>
  <div id="app">
    <input type="text" v-model="valueData" />
      Le pasamos lo que escribamos en el input
    <my-props-component :prop-value="valueData"/>
  </div>                Componente

  <script src="https://unpkg.com/vue"></script>
  <script src="firstvue.js"></script>
</body></html>
```

2

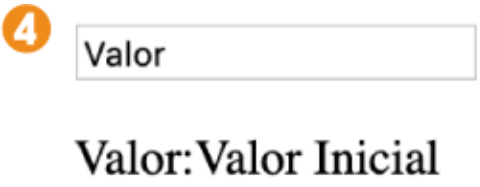
```
let MyPropsComponent = Vue.component('my-props-component', {
    template: '<p component="my-props">Valor:{{myValue}}</p>'
   ,data(){ return {  myValue: this.propValue } }
   ,props:{ propValue: {type:String, required:true} }
})
          Recibe el valor de la prop, por aqui

const app = new Vue({ el: '#app'
                    , data(){ return { valueData: "Valor Inicial" } }
                    , components: { 'my-props-component': MyPropsComponent } });
```

Variable que recibe el valor del input

3



4

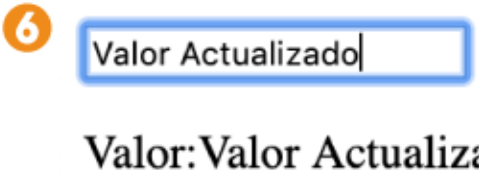


5

```
let MyPropsComponent = Vue.component('my-props-component', {
    template: '<p component="my-props">Valor:{{myValue}}</p>'
   ,watch:{ propValue(){ this.myValue=this.propValue } }
   ,data(){ return {  myValue: this.propValue } }
   ,props:{ propValue: {type:String, required:true} }
})
```

Capturar cambios

6



Reactividad Data
REACTIVIDAD

IMPORTANTE

Cualquier borrado o añadido de un atributo de tipo objeto del data requiere del uso de “Vue.set” y de “Vue.delete” para activar la reactividad al añadir o borrar atributos respectivamente.

La reactividad en Vue es un concepto muy importante: cualquier cambio en una variable del data representada en la vista será detectada por un observador y la vista se refrescará con los cambios [1].

Aunque, como veremos, no siempre ocurre la reactividad. Creemos un ejercicio que nos permita activar reactividad.

Creamos “firstvue.html”, con la etiqueta del componente “my-reactivity” [2], al que le pasamos un objeto persona.

Este valor por defecto es enviado por la propiedad “propPerson” de tipo objeto al componente, por lo que creemos en el fichero “firstvue.js” [3], el componente.

En la plantilla visualizamos las propiedades del objeto persona, únicamente hemos enviado “FirstName” y “LastName”. Declaramos en el data, una propiedad para el input de edad y otra para el del nombre, así como dos botones, uno para borrar la propiedad “FirstName” y otro para añadir la propiedad “Age”. Si nos fijamos en los métodos definidos, cada vez que cambie el valor del input nombre, el método “OnChangeName” se ejecuta, asignando el valor del “name” al atributo “FirstName” del objeto “person”.

Como el atributo “FirstName” existe en el objeto persona, en la pantalla [4] al cambiar el valor al nombre se observa el cambio, de modo que la vista es actualizada mostrando el valor en pantalla [5].

Si cambiamos el valor en el input de edad, la reactividad es instantánea, al cambiar el valor de una variable del data directamente y no al atributo de un objeto con el que la reactividad tiene problemas [6].

Ahora bien, si borramos la propiedad “FirstName” del objeto “person”, la reactividad no sucede debido a que esta no se activa al añadir o borrar atributos a un objeto [7]. Sucediendo lo mismo al añadir el atributo “Age”, tras pulsar el botón “Añade Edad” [8].

Para forzar la reactividad ante añadir nuevos atributos o asignar valores sobre el atributo de un objeto utilizamos:

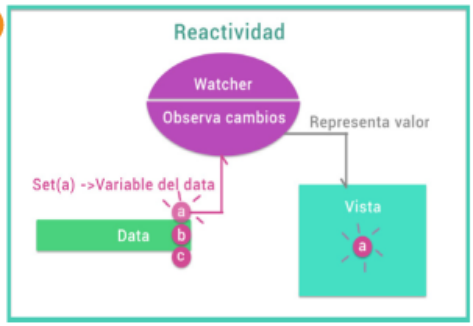
Vue.set(objeto,”NombreAtributo”,valor)

Mientras que para activarla en el borrado de un atributo de un objeto usaremos:

Vue.delete(objeto,”NombreAtributo”)

Si modificamos los métodos anteriores por [9] debería ser reactivo todo cambio sobre el objeto, tanto si borramos atributo ||!, como si añadimos atributo [”@, como si asignamos valor [#].

1



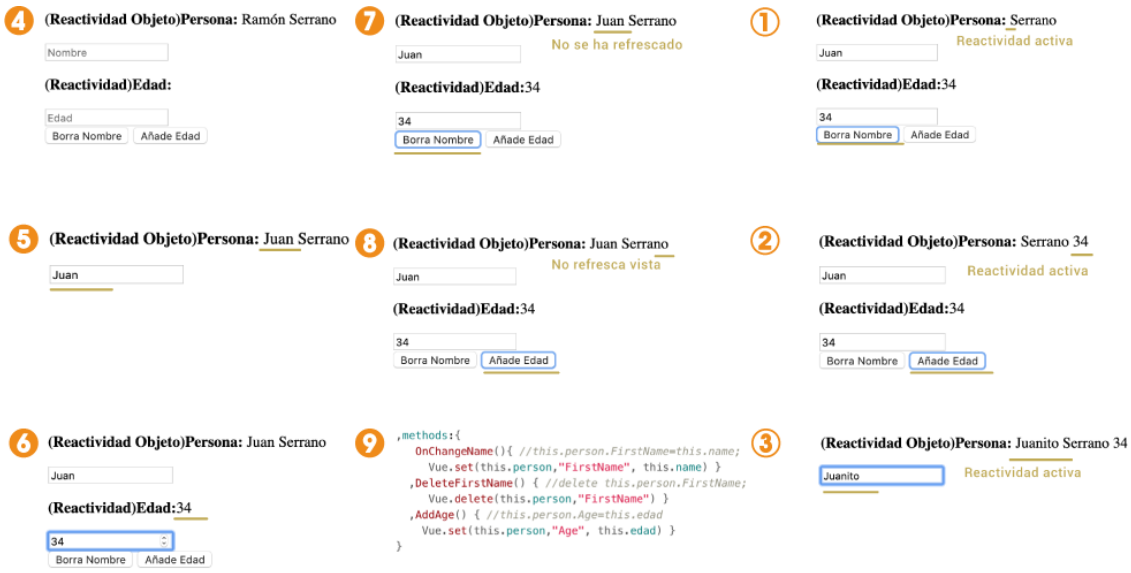
2

```
<html><head><meta charset="utf-8">
  <title>Reactividad</title></head>
<body>
  <div id="app">
    <my-reactivity :prop-person="{ FirstName: 'Ramón', LastName: 'Serrano' }" />
  </div>
```

3

```
let MyReactivityComponent = Vue.component('my-reactivity', {
  template: `<div component="my-reactivity">
              <p v-if="person"><b>(Reactividad Objeto)Persona:</b> Visualizar Datos personales
                <span>{{person.FirstName}} {{person.LastName}} {{person.Age}}</span></p>
Introducir Nombre  <input type="text" placeholder="Nombre"
                           v-model="name" @change="OnChangeName" />

                                                  Visualizar Edad
              <p><b>(Reactividad)Edad:</b>{{edad}}</p>
Introducir Edad  <input type="number" placeholder="Edad" v-model="edad"/>
                <br />
              <input type="button" @click="DeleteFirstName" value="Borra Nombre" />
              <input type="button" @click="AddAge" value="Añade Edad" />
            </div>`

   ,data(){ return {  person: this.propPerson, name:null, edad:null } }
   ,props:{ propPerson: {type:Object} }
   ,methods:{
      OnChangeName(){ this.person.FirstName=this.name; }
     ,DeleteFirstName() { delete this.person.FirstName; }
     ,AddAge() { this.person.Age=this.edad }
   }
})

const app = new Vue({ el: '#app', components: { 'my-reactivity': MyReactivityComponent } });
```

No causan reactividad
Borrar / Añadir propiedad


4
(Reactividad Objeto)Persona: Ramón Serrano
Nombre
(Reactividad)Edad:
Edad
Borra Nombre
Añade Edad
7
(Reactividad Objeto)Persona: Juan Serrano
No se ha refrescado
Juan
(Reactividad)Edad:34
34
Borra Nombre
Añade Edad
1
(Reactividad Objeto)Persona: Serrano
Reactividad activa
Juan
(Reactividad)Edad:34
34
Borra Nombre
Añade Edad
5
(Reactividad Objeto)Persona: Juan Serrano
Juan
8
(Reactividad Objeto)Persona: Juan Serrano
No refresca vista
Juan
(Reactividad)Edad:34
34
Borra Nombre
Añade Edad
2
(Reactividad Objeto)Persona: Serrano 34
Reactividad activa
Juan
(Reactividad)Edad:34
34
Borra Nombre
Añade Edad
6
(Reactividad Objeto)Persona: Juan Serrano
Juan
(Reactividad)Edad:34
34
Borra Nombre
Añade Edad
9
,methods:{
OnChangeName(){ //this.person.FirstName=this.name;
Vue.set(this.person,"FirstName", this.name) }
,DeleteFirstName() { //delete this.person.FirstName;
Vue.delete(this.person,"FirstName") }
,AddAge() { //this.person.Age=this.edad
Vue.set(this.person,"Age", this.edad) }
}
3
(Reactividad Objeto)Persona: Juanito Serrano 34
Reactividad activa
Juanito

Конец ознакомительного фрагмента.