Introducción

¿Qué es Vue.js?

Vue (pronunciado /vjuː/ en inglés, como **view**) es un **framework progresivo** para construir interfaces de usuario. A diferencia de otros *frameworks* monolíticos, Vue está diseñado desde el inicio para ser adoptado incrementalmente. La biblioteca principal se enfoca solo en la capa de la vista, y es muy simple de utilizar e integrar con otros proyectos o bibliotecas existentes. Por otro lado, Vue también es perfectamente capaz de soportar aplicaciones sofisticadas de una sola página (en inglés *single-page-application* o SPA) cuando se utiliza en combinación con **herramientas modernas** y **librerías compatibles**.

Si eres un desarrollador de *frontend* con experiencia y quieres saber como Vue se compara con otras bibliotecas/frameworks, revisa **esta comparación**.

Empezando

> ! La guia oficial asume un conocimiento intermedio de HTML, CSS y JavaScript. Si eres totalmente nuevo en el desarrollo de *frontend*, puede no ser la mejor idea empezar a utilizar un *framework* - ¡aprende los conceptos básicos y luego regresa aquí! La experiencia previa con otros *frameworks* ayuda, pero no es obligatoria.

La manera más sencilla de probar Vue.js es usando el **ejemplo "hola mundo" en JSFiddle**. Siéntete libre de abrilo en otra pestaña y revisarlo a medida que avanzamos con ejemlos básicos. Si no, puedes crear un archivo `.html` e incluir Vue con:

HTML

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

La **página de instalación** provee más opciones para instalar Vue. Nota que **no** recomendamos a los principiantes comenzar con `vue-cli`, especialmente si no estás familiarizado con las herramientas de trabajo basadas en Node.js.

Renderizado declarativo

En el corazón de Vue.js se encuentra un sistema que nos permite renderizar declarativamente datos en el DOM utilizando una sintaxis de plantillas directa:

HTML

```
<div id="app">
  {{ message }}
</div>
```

JS

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

Hello Vue!

¡Ya hemos creado nuestra primera aplicación Vue! Esto parece bastante similar a renderizar una plantilla de texto, pero internamente Vue ha hecho muchas cosas. Los datos y el DOM están enlazados, y todo es **reactivo**. ¿Cómo lo sabemos? Abre la consola de JavaScript en tu navegador (ahora mismo, en esta página) y cambia el valor de `app.message`. Deberías ver el ejemplo renderizado actualizarse acorde a lo que has ingresado.

Además de interpolación de texto, también podemos enlazar atributos de un elemento, por ejemplo:

HTM

```
<div id="app-2">
  <span v-bind:title="message">
    ¡Deja tu mouse sobre este mensaje unos segundos para ver el atributo
  </span>
</div>
```

JS

```
var app2 = new Vue({
  el: '#app-2',
  data: {
    message: 'You loaded this page on ' + new Date()
  }
})
```

> ¡Deja tu mouse sobre este mensaje unos segundos para ver el atributo `title` enlazado dinámicamente!

Aquí nos encontramos con algo nuevo. El atributo `v-bind` que estás viendo es conocido como una **directiva**. Las directivas llevan el prefijo `v-` para indicar que son atributos especiales provistos por Vue y, como debes haber adivinado, aplican un comportamiento reactivo especial al DOM renderizado. En este caso, básicamente está diciendo “mantén el atributo `title` de este elemento enlazado con la propiedad `message` en la instancia de Vue”.

Si abres nuevamente tu consola JavaScript y escribes `app2.message = 'some new message'`, verás que el HTML enlazado (en este caso, el atributo `title`) ha sido actualizado.

Condicionales y bucles

Es bastante sencillo alternar la presencia de un elemento:

HTML

```
<div id="app-3">
  <p v-if="seen">Now you see me</p>
</div>
```

JS

```
var app3 = new Vue({
  el: '#app-3',
  data: {
    seen: true
  }
})
```

Now you see me

Adelante, escribe `app3.seen = false` en la consola. Deberías ver desaparecer el mensaje.

Este ejemplo demuestra que no solo podemos enlazar datos con texto y atributos, sino también con la **estructura** del DOM. Además, Vue provee un sistema de transiciones muy poderoso que puede aplicar automáticamente **efectos de transición** cuando los elementos son agregados/actualizados/removidos por Vue.

Hay unas cuantas otras directivas, cada una con una funcionalidad especial. Por ejemplo, la directiva `v-for` puede ser utilizada para mostrar una lista de elementos usando los datos de un array:

HTML

```
<div id="app-4">
  <ol>
    <li v-for="todo in todos">
      {{ todo.text }}
    </li>
  </ol>
</div>
```

JS

```
var app4 = new Vue({
  el: '#app-4',
  data: {
    todos: [
      { text: 'Learn JavaScript' },
      { text: 'Learn Vue' },
      { text: 'Build something awesome' }
    ]
```

```
  }
})
```

1. Learn JavaScript
2. Learn Vue
3. Build something awesome

En la consola, escribe `app4.todos.push({ text: 'New item' })`. Deberías ver un nuevo elemento agregado a la lista.

Manejando entradas de usuario

Para permitir a los usuarios interactuar con tu aplicación, podemos usar la directiva `v-on` para añadir *listeners* de eventos que invocan métodos en nuestras instancias de Vue:

HTML

```
<div id="app-5">
  <p>{{ message }}</p>
  <button v-on:click="reverseMessage">Reverse Message</button>
</div>
```

JS

```
var app5 = new Vue({
  el: '#app-5',
  data: {
    message: 'Hello Vue.js!'
  },
  methods: {
    reverseMessage: function () {
      this.message = this.message.split('').reverse().join('')
    }
  }
})
```

Hello Vue.js!

Reverse Message

Nota que en el método simplemente actualizamos el estado de nuestra aplicación sin modificar del DOM - todas las manipulaciones del mismo son manejadas por Vue, y el código que escribes se enfoca en la lógica subyacente.

Vue también provee la directiva `v-model` que hace muy sencillo el enlace de dos vías entre un `input` de un formulario y el estado de la aplicación:

HTML

```
<div id="app-6">
  <p>{{ message }}</p>
  <input v-model="message">
</div>
```

JS

```
var app6 = new Vue({
  el: '#app-6',
  data: {
    message: 'Hello Vue!'
  }
})
```

Hello Vue!

Hello Vue!

Componentes

El sistema de componentes es otro concepto importante en Vue, porque es una abstracción que nos permite construir aplicaciones de gran escala compuestas por componentes pequeños, autocontenidos y, normalmente, reutilizables. Si lo pensamos, casi cualquier tipo de interfaz gráfica de una aplicación puede ser representada de manera abstracta como un árbol de componentes:

Component

En Vue, un componente es esencialmente una instancia de Vue con opciones predefinidas. Registrar un componente en Vue es directo y sencillo:

JS

```
// Define un nuevo componente llamado todo-item
Vue.component('todo-item', {
  template: '<li>This is a todo</li>'
})
```

Ahora puedes utilizarlo en la plantilla de otro componente:

HTML

```
<ol>
  <!-- Crea una instancia del componente todo-item -->
  <todo-item></todo-item>
</ol>
```

Pero esto renderizaría el mismo texto para cada *todo*, lo cual no es muy interesante. Deberíamos ser capaces de pasar datos desde el padre a los componentes hijo. Vamos a modificar la definición del componente para aceptar **propiedades**:

JS

```
Vue.component('todo-item', {
  // El componente todo-item ahora acepta
  // "prop", el cual es similar a un atributo personalizado
  // La rpopiedad se llama _todo_.
  props: ['todo'],
  template: '<li>{{ todo.text }}</li>'
})
```

Ahora podemos pasar *todo* a cada componente repetido utilizando `v-bind` :

HTML

```
<div id="app-7">
  <ol>
    <!-- Ahora le pasamos a cada todo-item with el objeto todo    -->
    <!-- que representa, para que su contenido pueda ser dinámico -->
    <todo-item v-for="item in groceryList" v-bind:todo="item"></todo-item
  </ol>
</div>
```

JS

```
Vue.component('todo-item', {
  props: ['todo'],
  template: '<li>{{ todo.text }}</li>'
})

var app7 = new Vue({
  el: '#app-7',
  data: {
    groceryList: [
      { text: 'Vegetables' },
      { text: 'Cheese' },
      { text: 'Whatever else humans are supposed to eat' }
    ]
  }
})
```

1. Vegetables
2. Cheese
3. Whatever else humans are supposed to eat

Este es simplemente un ejemplo imaginario, pero hemos logrado separar nuestra aplicación en porciones más pequeñas, y el hijo está razonablemente desacoplado del padre a través de la interfaz de propiedades. Podemos mejorar aún más nuestro componente `<todo-item>` con una plantilla más compleja o diferente lógica sin afectar a la aplicación padre.

En aplicaciones grandes, es necesario dividir la aplicación entera en componentes para un desarrollo manejable. Hablaremos mucho más acerca de los componentes **más adelante en la guia**, pero aquí tienes un ejemplo (imaginario) de como luciría una plantilla de aplicación utilizando componentes:

HTML

```
<div id="app">
  <app-nav></app-nav>
  <app-view>
    <app-sidebar></app-sidebar>
    <app-content></app-content>
  </app-view>
</div>
```

Relación con los elementos personalizados

Puedes haber notado que los componentes de Vue son muy similares a los **Elementos Personalizados** (*Custom Elements*), los cuales son parte de la **especificación de Componentes Web** (*Web Components*). Esto es porque la sintaxis de componente de Vue está modelada basándose en ideas de la especificación. Por ejemplo, los componentes de Vue implementan la **API de Slot** y el atributo especial `is` . Sin embargo, hay unas cuantas diferencias clave:

1. La especificación de Componentes Web todavía está en un estado de borrador, y no está implementada nativamente en todos los navegadores. En comparación, los componentes de Vue no requieren ningún *polyfill* y funcionan consistentemente en todos los navegadores soportados (IE9 y superiores). Cuando se necesite, los componentes de Vue pueden ser envueltos dentro de un elemento personalizado nativo.

2. Los componentes de Vue proveen características importantes que no están disponibles en los elementos personalizados, siendo las más notables el flujo de datos entre componentes, la comunicación con eventos personalizados, y la integración con herramientas de desarrollo.

¿Listo para más?

Hemos introducido brevemente las características más básicas del corazón de Vue.js - el resto de esta guia cubrirá estas y otras características avanzadas con mucho más detalle, ¡asegúrate de leerla entera!