GUIA VUE JS

¿Qué es Vue.js?

Vue (pronunciado /vjuː/, como view) es un framework progresivo para construir interfaces de usuario. A diferencia de otros frameworks monolíticos, Vue está diseñado desde cero para ser utilizado incrementalmente. La librería central está enfocada solo en la capa de visualización, y es fácil de utilizar e integrar con otras librerías o proyectos existentes. Por otro lado, Vue también es perfectamente capaz de impulsar sofisticadas Single-Page Applications cuando se utiliza en combinación con herramientas modernas y librerías de apoyo.

Herramientas y configuración

Instalación NPM

Lo primero que vamos a hacer es descargar Node.js desde su página oficial Descarga Node.js(https://nodejs.org/es/) su versión recomendada.

Una vez descargado el archivo procedemos con la instalación, que no es más que seguir el wizard y dar Next.

Cabe mencionar que conjuntamente con Node.js se va a instalar el gestor de paquetes NPM.

PROBAR QUE NODE.JS SE HA INSTALADO CORRECTAMENTE

Primero vamos a verificar que Node.js se ha instalado correctamente para esto abrimos una consola cmd y ejecutamos el siguiente comando:

node -v

```
C:\Users\andre>node -v
v12.14.1
```

Este comando retornará la versión actual de Node.js.

Lo siguiente que vamos a hacer es verificar la versión del gestor de paquetes NPM instalada, para esto usando la consola abierta ejecutamos el comando:

npm -v

```
C:\Users\andre>npm -v
6.13.4
```

¿Cómo empezamos?

Para empezar, lo único que tendremos que hacer es incluir la dependencia de VueJS a nuestro proyecto. Dependiendo de nuestras necesidades, podremos hacer esto de varias maneras. Lo primero que hacemos es ejecutar los siguientes comandos en el terminal:

$ mkdir example-vue

$ cd example-vue

$ npm init

```
C:\Users\andre>mkdir example-vue

C:\Users\andre>cd example-vue

C:\Users\andre\example-vue>npm init
```

Esto nos genera una nueva carpeta para el proyecto y nos inicia un paquete de NodeJS. Una vez que tengamos esto, añadiremos VueJS como dependencia de la siguiente manera:

$ npm install vue --save

```
C:\Users\andre\example-vue>npm install vue --save
```

De esta forma, ya tendremos todo lo necesario para trabajar en nuestro primer ejemplo. Lo que esta dependencia nos descarga son diferentes VueJS dependiendo del entorno que necesitemos. Lo que hacemos ahora es añadir un fichero **index.html** en la raíz e incluimos tanto la librería de Vue, como nuestro fichero JS, donde desarrollaremos este primer ejemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>VueJS Example</title>
</head>
<body>
    <div id="app"></div>

    <script src="node_modules/vue/dist/vue.js"></script>
    <script src="app.js"></script>
</body>
</html>
```

Lo siguiente que hacemos es crear una instancia de nuestra aplicación VueJS en nuestro fichero app.js

Creando una instancia de Vue

Cada aplicación de Vue se comienza creando una nueva Instancia de Vue con la función

```
var vm = new Vue({
  // opciones
})
```

```
new Vue({
el: '#app',
template: `<div v-bind:style="estilos" v-on:click="log()"`,
data: {
    titulo: 'Rectángulo',
    largo: 100,
    alto: 40
},
computed: {
    area: function() {
        return this.largo * this.alto;
    },
    estilos: function() {
        return {
            width: this.largo + 'px',
            height: this.alto + 'px'
        }
    }
},
methods: {
    log: function() {
        console.log('Soy un rectángulo con un area de ' + this
            .area + ' pixels');
    }
},
created: function() {
    console.log('se crea la instancia: todavía no se ha
        reemplazado "#app" por el template de estainstancia '); },
    mounted: function() {
        console.log('ya se ha renderizado el template en el DOM');
    }
});
```

El

La propiedad **EL** determina el tag html sobre los que operará nuestra instancia Vue. Es un selector CSS como los que se utilizan en las hojas de estilo o jQuery.Es importante notar que cada instancia de Vue apunta a un solo nodo html y sus hijos. Podemos utilizar como selector una clase o una etiqueta, pero la instancia solamente operará sobre el primer nodo que encuentre.

Template

Como template, se indica una cadena con html que reemplazará el html que hemos fijado como el. Nota que hay dos cosas distintas denominadas 'template': todo contenido entre {{}} y la propiedad template de la instancia. Son dos cosas distintas, aunque generalmente van juntas.

No es necesario usar la propiedad template para trabajar con Vue. Podríamos haber puesto ese mismo código directamente en el html dentro del div#app y hubiera funcionado de manera similar. A veces es deseable mantener el html en ficheros .html y no llevarlo a javaScript. Pero la opción template también tiene sus ventajas:

Si el html y el javaScript van a cambiar a la vez a menudo, los mantenemos en el mismo fichero.

Si el html que estamos escribiendo no es válido antes de que Vue lo parsee y lo modifique, evitamos que el navegador lo elimine antes de que Vue pueda operar sobre él. Podemos usar JavaScript para generar el template, lo que en según que casos puede ser cómodo y evitar duplicidades en el html.

Data

La propiedad data es un objeto con propiedades que podemos utilizar en nuestra instancia. En este caso, tanto la directiva v-bind como el método area() refieren estas propiedades. También se utiliza entre corchetes en el template. Básicamente, es el estado de nuestro objeto, el model del patrón view-model. Cuando se crea la instancia, el motor de Vue lee todas las propiedades en data y les añade getters y setters, utilizando la 'magia' de Object.defineProperty para que sean reactivas y quedar a la escucha de sus cambios. La reactividad en Vue tiene ciertas limitaciones: si añadimos propiedades a un objeto dentro de data directamente con JavaScript, Vue no podrá escuchar estos cambios. Así que, si uno de los campos de data es una lista de objetos y renderizamos html en función de estas listas, añadir un objeto a la lista o cambiar a mano el índice puede provocar que los cambios no se propaguen a la vista. De todos modos, en esos casos, el método $set nos tiene cubiertos.

Computed

Computed es un objeto que contiene una serie de métodos que devuelven valores calculados a partir de las propiedades del objeto. En nuestro ejemplo, valores derivados como el área, o el perímetro del rectángulo son los típicos casos en los que usaríamos computed.

Además, tenemos un campo estilos que indica algunos estilos que se aplicarán al componente. Esto se consigue mediante el atributo v-bind que hay en el template. Más abajo veremos el tema de las directivas.

Methods

Methods es un objeto con funciones que podemos invocar, y cuyo contexto está enlazado con el propio objeto. Cuando queramos modificar las propiedades de nuestro objeto, o enviar eventos a otros componentes o instancias de la página, lo haremos desde alguno de estos métodos.

Nota que, moviendo área a methods y añadiendo () a su llamada en el método log, podríamos usar un método en vez de una computed property para indicar el área del rectángulo. La cosa quedaría como:

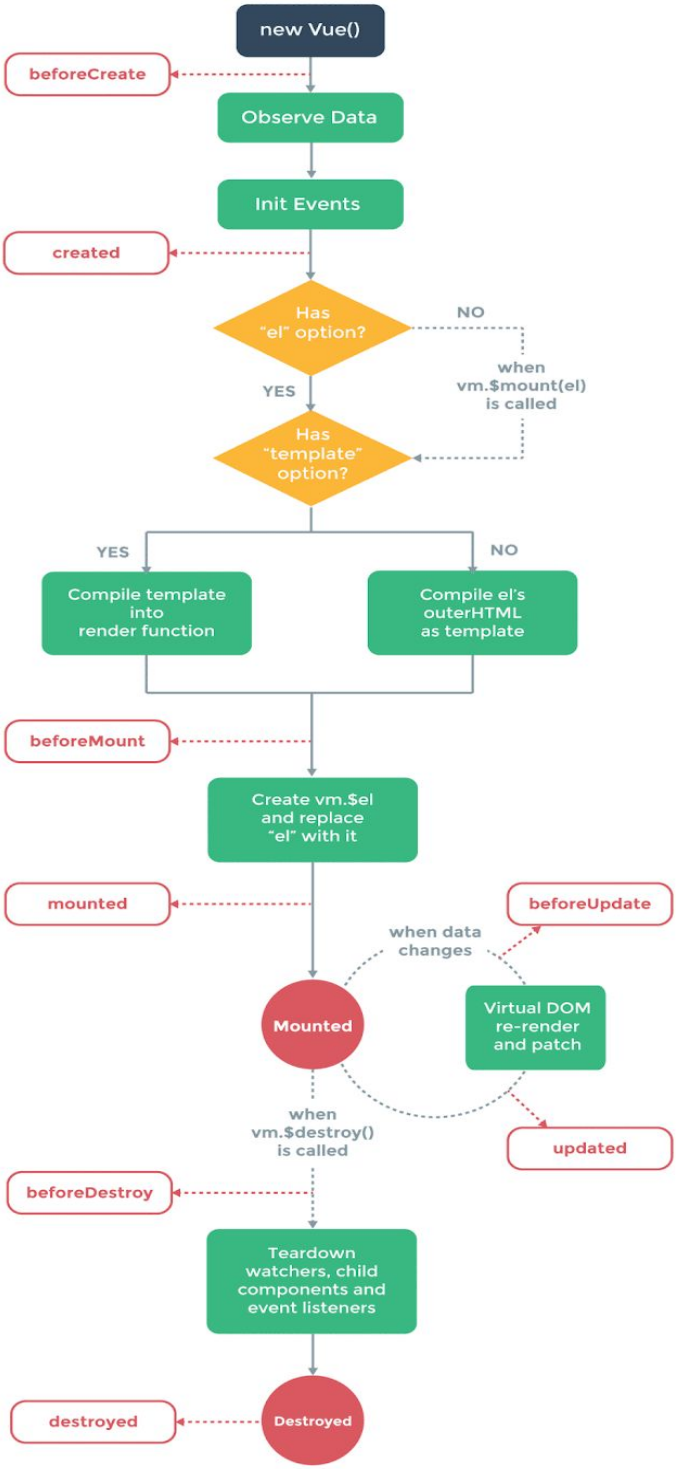
```
methods : {
    log : function(){
        console.log('Soy un rectángulo con un área de ' + this.area() + ' pixels' );
    },
    area : function(){
       return this.largo * this.alto;
    }

}
```

Esto es una mala práctica y dará problemas de rendimiento, dado que el resultado de computed queda guardado en memoria y solamente cambia cuando cambia alguna de sus dependencias -en este caso, las propiedades alto y largo, mientras que las funciones en methods se invocan siempre.

Hooks

Created y mounted son funciones que se ejecutan en determinados momentos del ciclo de vida de la instancia. Created se invoca cuando se crea el objeto, mientras que mounted se invoca justo después de que el motor de Vue reemplace el html que hubiera en él.Hay otros hooks que podemos consumir, para realizar acciones cuando una instancia es destruida, es modificada... etc. etc. Generalmente mounted es el más utilizado y de alguna manera es similar a $(function(){}) en jQuery: nos asegura que el código no se va a ejecutar hasta que Vue haya terminado de inicializar el html de la instancia.

La siguiente imagen es una representación esquemática bastante completa del ciclo de vida de una instancia -o un componente- y de sus hooks:


new Vue()
beforeCreate
Observe Data
Init Events
created
Has "el" option?
NO
when vm.$mount(el) is called
YES
Has "template" option?
YES
NO
Compile template into render function
Compile el's outerHTML as template
beforeMount
Create vm.$el and replace "el" with it
mounted
beforeUpdate
when data changes
Mounted
Virtual DOM re-render and patch
when vm.$destroy() is called
updated
beforeDestroy
Teardown watchers, child components and event listeners
destroyed
Destroyed

Directivas

Las directivas son atributos especiales con el prefijo v-. Se espera que los valores de atributo de la directiva sean una única expresión de JavaScript (con la excepción de v-for, que se tratará más adelante). El trabajo de una directiva es aplicar reactivamente efectos secundarios al DOM cuando cambia el valor de su expresión.

Como las siguientes:

Enlace Clases y Estilos

Una necesidad común de data binding es manipular la lista de clases de un elemento y sus estilos en línea. Como ambos son atributos, podemos usar v-bind para manejarlos: solo necesitamos crear una cadena de texto con nuestras expresiones. Sin embargo, concatenar cadenas de texto puede llegar a ser incómodo y propenso a errores. Por esta razón, Vue proporciona mejoras cuando se utiliza **v-bind** con class y style. Además de las cadenas de texto, las expresiones también pueden evaluar objetos o matrices.

Ejemplo:

```
<div class="static"
     v-bind:class="{ active: isActive, 'text-danger': hasError }">
</div>
```

Y el siguiente objeto data:

```
data: { isActive: true, hasError: false }
```

Se renderizará:

```
      div class="static active"></div>
```

Binding en Formularios

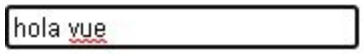
Puede usar la directiva v-model para crear bindings de datos bidireccionales (two-way binding) en elementos input, textarea y select de un formulario. La directiva busca automáticamente la manera correcta de actualizar el elemento según el tipo de entrada. Aunque un poco mágico, v-model es esencialmente syntax sugar para actualización de datos a través de eventos de entradas del usuario, además de mostrar un cuidado especial para algunos casos de borde.

Ejemplo:

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

  <script>
        new Vue({
            el: '#app',
            data: {
                message: 'Hello Vue!'
            }
       })
  </script>
```

Resultado:

hola vue



Renderización Condicional

V-IF

Es una directiva de vue que nos permite mostrar o hacer algo según se cumpla una condición.

Ejemplo

Vamos a utilizar la directiva v-model para crear un enlace por medio de la variable message de un textarea con nuestro código vue.

Agregamos la directiva v-if en la primera etiqueta <h3> con la condición de que si la variable está vacía aparecerá el texto “Escribe tu sugerencia”.

Luego en la segunda etiqueta <h3> la directiva v-if tendrá la condición de que si la variable tiene un mensaje aparecerá el texto “Presiona ENTER para enviar”.

```
<h3 v-if="!message">Escribe tu sugerencia</h3>
<h3 v-if="message">Presiona ENTER para enviar</h3>
<textarea v-model="message" class="form-control"></textarea>
```

V-ELSE

Esta directiva trabaja en función a que si la condición declarada en la directiva v-if no se cumple, esta realizará o mostrará algo diferente.

Ejemplo

En el ejemplo anterior utilizamos dos veces la directiva v-if, ahora con la directiva v-else obtendremos el mismo resultado, pero de esta manera el código y la lógica es más clara y fácil de entender. El v-else se ata a la condición asignada en el v-if; si la variable “message” está vacía muestra el primer <h3>, caso contrario se muestra el segundo.

```
<h3 v-if="!message">Escribe tu sugerencia</h3>
<h3 v-else>Presiona ENTER para enviar</h3>
<textarea v-model="message" class="form-control"></textarea>
```

V-SHOW

El v-show en Vue funciona muy parecido a como lo hace el v-if, pero tiene algunas diferencias. Para empezar, dentro de la condición del v-show puedes colocar todo lo que podías colocar dentro del v-if, por ejemplo:

```
<template>
    <div class="content">
      <p v-show="condition">Este mensaje no se va a mostrar</p>
      <p>Este mensaje si que se va a mostrar</p>
    </div>
  </template>
```

```
<script>
export default {
  data: () => ({
    condition: false
  })
};
</script>
```

El v-show también permitirá o no mostrar un elemento del html pero hay una diferencia fundamental con el v-if.

En el v-if cuando no se muestra un elemento Vue lo elimina directamente del DOM, es decir, elimina toda la etiqueta HTML y la muestra cuando la necesita. El v-show no elimina la etiqueta html, la oculta. Lo que hace el v-show es añadir display:none al elemento para que no se muestre, pero sigue estando en los dentro del DOM.

Otra diferencia fundamental con el v-if es que no puedes hacer v-else con lo cual está más limitado en ese aspecto.

Renderizado de lista

V-FOR

La directiva v-for es muy interesante para crear estructuras repetitivas de código HTML de una forma sencilla y sin que el código resulte excesivamente complejo

Ejemplo:

Podemos usar la directiva v-for para representar una lista de elementos basada en una matriz. La directiva v-for requiere una sintaxis especial en forma de item in items, donde los items son la matriz de datos de origen y el item es un **alias** para el elemento de matriz que se está iterando:

```
<ul id="example-1">
    <li v-for="item in items">
      {{ item.mensaje }}
    </li>
  </ul>

  <script>
      var example1 = new Vue({
    el: '#example-1',
    data: {
      items: [
        { mensaje: 'Foo' },
        { mensaje: 'Bar' }
```

```
      ]
    }
  })
  </script>
```

v-for con un Objeto

También puede usar v-for para iterar a través de las propiedades de un objeto.

```
<ul id="v-for-object" class="demo">
    <li v-for="value in object">
      {{ value }}
    </li>
  </ul>

  <script>
      new Vue({
    el: '#v-for-object',
    data: {
      object: {
        primerNombre: 'John',
        apellido: 'Doe',
        edad: 30
      }
    }
  })
  </script>
```

Manejo de eventos

Escuchar eventos

Podemos usar la directiva v-on para escuchar eventos DOM y ejecutar algunos JavaScript cuando se activan.

Por ejemplo:

```
<div id="example-1">
    <button v-on:click="counter += 1">Add 1</button>
    <p>Se ha hecho clic en el botón de arriba {{ counter }} veces.</p>
  </div>

  <script>
      var example1 = new Vue({
```

```
  el: '#example-1',
  data: {
    counter: 0
  }
})
</script>
```

Resultado:

La variable counter aumentará su valor +1 cada que se haga click en el botón de agregar 1

Componentes

Los componentes son elementos html reusables y configurables, y nos permiten definir etiquetas que podemos usar en nuestro markup dotadas de una determinada api y comportamiento. Aunque su estructura no es idéntica a la de los custom elements, es posible convertir vue components a custom elements con facilidad mediante vue-custom-elements.Aunque la Api más inmediata de Vue es su constructor, utilizando meras instancias de Vue no podemos montar con facilidad instancias dentro de otras ni reutilizar código para casos de uso similares. Para esto, deberemos usar componentes. Un componente es, básicamente, una instancia Vue que será utilizada dentro de otra, y que será referenciada mediante una etiqueta específica. De este modo, podemos extender el html para crear nuevas etiquetas con un comportamiento y unos datos dirigidos por Vue.

Por ejemplo, si definimos un componente como <usuario>, referenciar esa etiqueta dentro de código html parseado por Vue generará html válido -en función de la propia lógica del componente. Los componentes son componibles, de tal modo que se pueden añadir componentes dentro de componentes, y tienen un orden jerárquico: los padres pueden modificar los datos de los hijos, pero no al revés. Los hijos únicamente pueden notificar eventos a los padres, pero no modificar su estado directamente. Esto puede parecer una molestia, pero definitivamente mejora el workflow y evita muchos bugs y condiciones de carrera, y asegura que cuando modifiquemos nuestros modelos haya un punto de entrada centralizado que gestiona estos cambios.

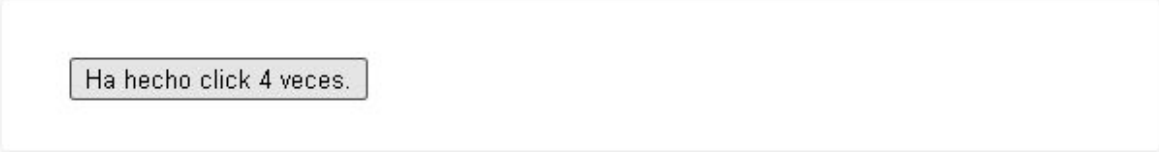
Aquí un ejemplo de un componente Vue:

```
// Definir un nuevo componente llamado button-counter
Vue.component('button-counter', {
  data: function () {
    return {
      count: 0
    }
  },
    template: '<button v-on:click="count++">Me ha pulsado {{ count }} veces.</button>'
})
```

Los componentes son instancias reutilizables de Vue con un nombre: en este caso, <button-counter>. Podemos usar este componente como un elemento personalizado dentro de una instancia de Vue raíz creada con new Vue:

```
<div id="components-demo">
    <button-counter></button-counter>
</div>
<script>
   new Vue({ el: '#components-demo' })

</script>
```

Resultado:



Dado que los componentes son instancias reutilizables de Vue, aceptan las mismas opciones que new Vue, como data, computed, watch, methods, y hooks de ciclo de vida. Las únicas excepciones son algunas opciones específicas de la raíz como el.

Reutilizando Componentes

Los componentes se pueden reutilizar tantas veces como se desee:

```
<div id="components-demo">
    <button-counter></button-counter>
    <button-counter></button-counter>
    <button-counter></button-counter>
</div>
```

Resultado:

Tenga en cuenta que al hacer clic en los botones, cada uno mantiene su propio count por separado. Esto se debe a que cada vez que se utiliza un componente, se crea una nueva instancia del mismo.

data Debe ser una función

Cuando definimos el componente <button-counter>, es posible que haya notado que data no devuelve directamente un objeto, como este:

data: {

count: 0

}

En lugar de eso, la opción data de un componente debe ser una función, de modo que cada instancia pueda mantener una copia independiente del objeto de datos devuelto:

data: function () {

return {

count: 0

}

}

Si Vue no tuviera esta regla, hacer clic en un botón afectaría los datos de todas las demás instancias.

Props

Las props son atributos personalizados que usted puede registrar en un componente. Cuando se pasa un valor a un atributo prop, se convierte en una propiedad en esa instancia de componente. **Ejemplo**:Para pasar un título a nuestro componente de publicación de blog, podemos incluirlo en la lista de props que este componente acepta, usando la opción props:

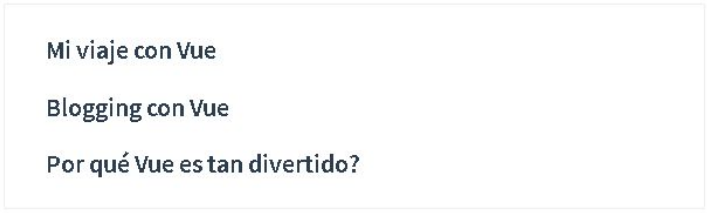
```
Vue.component('blog-post', {
  props: ['title'],
  template: '<h3>{{ title }}</h3>'
})
```

Un componente puede tener tantas props como se desee, y se puede pasar cualquier valor a cualquier prop de forma predeterminada. En el template anterior, verá que podemos acceder a este valor en la instancia del componente, al igual que con data.

Una vez que se registra un prop, puede pasarle datos como un atributo personalizado, de la siguiente manera:

```
<blog-post title="Mi viaje con Vue"></blog-post>
<blog-post title="Blogging con Vue"></blog-post>
<blog-post title="Por qué Vue es tan divertido?"></blog-post>
```

Resultado:



En una aplicación típica, sin embargo, es probable que tenga un array de post en data:

```
new Vue({
  el: '#blog-post-demo',
  data: {
```

```
    posts: [
      { id: 1, title: 'Mi viaje con Vue' },
      { id: 2, title: 'Blogging con Vue' },
      { id: 3, title: 'Por qué Vue es tan divertido?' }
    ]
  }
})
```

Entonces querrá renderizar un componente para cada uno:

```
<blog-post
  v-for="post in posts"
  v-bind:key="post.id"
  v-bind:title="post.title"
></blog-post>
```

Arriba, verá que podemos usar v-bind para pasar propiedades dinámicamente. Esto es especialmente útil cuando no se conoce el contenido exacto que se va a renderizar con anticipación, como cuando se obtienen posts de una API.

vue-cli

Vue tiene excelentes herramientas para trabajar comenzando con Vue CLI. Le permite crear una aplicación rica en funciones casi al instante. De hecho, nos ayuda a configurar nuestro proyecto Vue que proporciona soporte para muchas herramientas populares de JS como Webpack, Babel, etc.

Antes de crear un proyecto, primero deberá instalar Vue CLI:

npm install -g @ vue / cli

```
C:\Users\andre\example-vue>npm install -g @vue/cli
```

y correr el comando **vue create example**

```
C:\Users\andre\example-vue>vue create example
```

Abra su proyecto recién creado con su editor de código favorito (recomiendo VS Code , es la herramienta predeterminada para los desarrolladores de aplicaciones para el usuario a partir de hoy).

Los comandos

Como se vio antes, la creación de un proyecto predeterminado con Vue CLI viene con Babel y ESLint habilitados de manera predeterminada. Una cosa importante a tener en cuenta es que los proyectos de Vue CLI utilizan Webpack : un paquete de módulos.

Básicamente, Webpack analiza todos sus módulos y archivos JS, puede pre procesarlos y agruparlos en un solo archivo minificado. Para preprocesar archivos, Webpack utiliza cargadores que transforman su código fuente. Por ejemplo, Vue tiene su propio cargador debido a los **.vue** archivos. Veremos eso en un momento.

Su proyecto Vue CLI viene con tres package.json scripts:

serve:se usa para iniciar servidores de desarrollo locales. Aproximadamente significa que cuando cambia su código, su aplicación se recarga instantáneamente (y eso es lo que llama una gran experiencia de desarrollador)

build: este comando produce un paquete listo para producción en un dist/directorio, con minificación para JS / CSS / HTML y división automática de partes del proveedor para un mejor almacenamiento en caché. El manifiesto de fragmentos está insertado en el HTML. Estas palabras

complejas significan que su aplicación será más pequeña y más rápida cuando se implemente en producción.

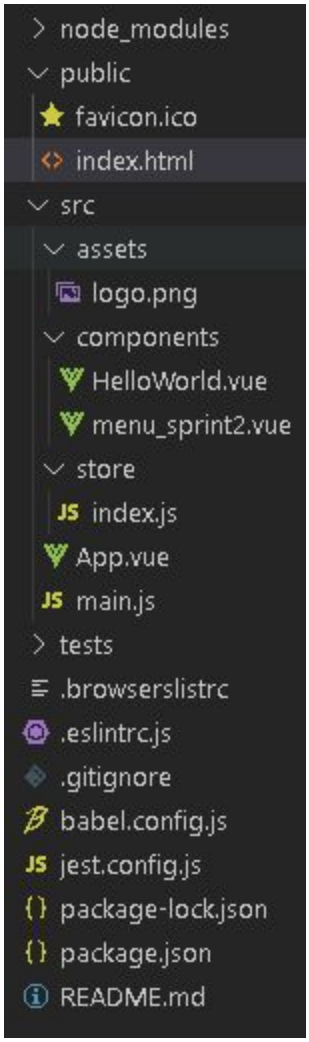
lint: borre sus archivos usando ESLint. Si ESLint no significa nada para usted, es una herramienta para identificar e informar sobre patrones encontrados en código JS, con el objetivo de hacer que el código sea más consistente y evitar errores. Se considera una buena práctica tenerlo en un proyecto y es por eso que Vue lo proporciona de forma predeterminada.

Tarea NPM	Descripción
`npm run serve`	Arranca el proyecto en el navegador en **modo desarrollo** en vivo.
`npm run build`	Genera la versión definitiva de **producción** para subir o desplegar al servidor donde se aloje.
`npm run test:unit`	Lanza los **tests unitarios** con el framework de testing seleccionado en el asistente.
`npm run test:e2e`	Lanza los **tests end-to-end** con el framework de testing seleccionado en el asistente.
`npm run lint`	Ejecuta **ESLint** para analizar el código de nuestro proyecto en busca de problemas.

fuente: https://lenguajejs.com/vuejs/introduccion/@vue-cli/

Los archivos generados

¿Se siente abrumado por la cantidad de archivos? No se preocupe. Le daré una descripción general rápida de los archivos generados por Vue CLI.



public carpeta:

favicon.ico: ¿Ves el pequeño icono de la izquierda en tu pestaña actual? Eso se llama favicon. Es un ícono que representa su aplicación o su sitio web.

index.html: este es su archivo html principal. Contendrá todo su código Vue. Se puede ver en la parte inferior del archivo de esta línea: . Eso es importante ya que es donde Vue montará su aplicación.

src carpeta:

assets carpeta: aquí es donde pondrás tus activos, es decir tus imágenes, tus iconos, etc.

components Carpeta: bueno, esto se explica por sí mismo aquí, contiene sus componentes de Vue.

App.vue: este es el componente principal montado en el DOM.

main.js: este es el punto de entrada de su aplicación . Este archivo es responsable de montar su App Componente en el DOM, específicamente en el #app div que vio index.html.

.browserslistrc: este archivo nos permite definir a qué navegadores queremos apuntar para generar los archivos de salida óptimos . Aquí, el archivo apunta a todas las versiones de los navegadores que tienen más del 1% de cuotas de mercado (> 1%) y las dos últimas versiones de cada navegador (last 2 versions). Puede visualizarlos en browserl.ist .

.eslintrc.js: configura qué guía de estilo de linting / formateo seguiremos. El linting y el formateo son excelentes para usar en cualquier proyecto, ya que permiten tener una guía de estilo común con otros desarrolladores.

.gitignore: su proyecto ya incluye Git si clonó el repositorio. Para asegurarse de no poner archivos confidenciales, carpetas de compilación ni nada por el estilo en GitHub (o GitLab, BitBucket, etc.), solo necesita poner en este archivo lo que no desea enviar a su repositorio.

babel.config.js: Si nunca antes ha oído hablar de Babel , esta puede ser difícil de entender. JavaScript está en constante evolución, cada año ve aparecer nuevas funciones . Pero estas funciones no están disponibles para todos los navegadores o versiones de navegadores antiguos. En pocas palabras, Babel nos permite transpilar el nuevo código JavaScript en el antiguo para que nuestro código funcione en la mayoría de los navegadores. Por lo tanto, podemos utilizar las últimas funciones de JavaScript sin preocupaciones. Tenga en cuenta que aquí, Babel producirá un código JavaScript compatible con los navegadores definidos en browserlistrc.

package.json: este archivo contiene varios metadatos sobre su proyecto: nombre, versión pero principalmente dependencias y scripts. Por ejemplo, cuando ejecutó yarn install, yarn buscó las dependencias usando su package.jsonarchivo, las instaló en la node_modules carpeta y generó yarn.lock.

README.md: eso es lo que contiene las instrucciones para instalar la aplicación.

yarn.lock: Es un archivo generado por yarn que almacena la versión de dependencias de los paquetes que usa.

Componentes de un solo archivo

Mira el App.vue archivo. anteriormente vimos que definimos la instancia de Vue así:

```
new Vue({

  el: "#app",

})
```

Y componentes como este:

```
Vue.component("my-component", {

    // ...

    template: `<div>...</div>`,

  })
```

Eso funcionó bien. Entonces, ¿por qué App.vue está estructurado de manera diferente?

Hay muchas razones:

Estilo: ¿cómo diseñarías tus componentes? ¿Definiendo un css archivo fuera del alcance del componente e importándolo en su html archivo? Eso hace que CSS sea global.

Plantillas: ¿qué pasa con estas cadenas de plantillas que usa en sus componentes? ¿No se ven un poco feos?

Construcción: no podemos usar las últimas funciones de JS porque realmente no podemos agregar pasos de compilación.

Es por eso que Vue usa componentes de un solo archivo . Están estructurados de forma muy clara:

template: Sus elementos HTML, directivas de Vue, etc.

script: Todas las que está vinculado a la instancia Vue: data, methods, etc.

style: Su código CSS.

Por ejemplo, aquí está el App.vue archivo:

App.vue

```
<template>
        <div id="app">
          <img alt="Vue logo" src="./assets/logo.png" />
          <HelloWorld msg="Welcome to Your Vue.js App" />
        </div>
      </template>
      <script>
        import HelloWorld from "./components/HelloWorld.vue"
        export default {
          name: "App",
          components: {
            HelloWorld,
          },
        }
      </script>
      <style>
        #app {
          font-family: Avenir, Helvetica, Arial, sans-serif;
          -webkit-font-smoothing: antialiased;
          -moz-osx-font-smoothing: grayscale;
          text-align: center;
```

```
        color: #2c3e50;
        margin-top: 60px;
      }
    </style>
```

En el <template>, hay una imagen y un componente que importó.

En <script>, define su componente Vue y especifica qué componentes usa en su<template>

En <style>, define el código CSS que utiliza. Precaución, aquí el CSS es global al igual App.vue que el archivo principal. Pero si va a HelloWorld.vue, verá <style scoped>: significa que su CSS definido en HelloWorld.vue estará limitado a él.

Nota : si está codificando con VS Code , le recomiendo que instale la extensión Vetur . Nos hará la vida más fácil al construir un proyecto de Vue (resaltado de sintaxis, fragmentos, etc.).

Los componentes de un solo archivo tienen muchas ventajas sobre lo que solíamos hacer:

CSS: traemos CSS directamente al componente. ¿Y sabes qué es aún más asombroso? Puede restringir el alcance de CSS solo al componente. ¡No más conflictos de reglas CSS!

Modularidad: los componentes tienen su propio archivo y todo está limitado al componente únicamente. Por lo tanto, componer componentes es más fácil de hacer.

Construcción: como podemos incluir pasos de construcción en el proyecto, podemos usar las últimas funciones de JS, linting, etc.

Referencias

Guía de Vue JS [https://es.vuejs.org/v2/guide/installation.html]