PRIMEROS PASOS DE DESARROLLO DE APLICACIONES CON VUE.JS

Jaime Solís Martínez: solisjaime@uniovi.es

Tabla de contenido

Introducción

Vue es un framework de desarrollo progresivo para la construcción de interfaces de usuario cuya librería principal está enfocada a la capa de visualización, siendo fácil de integrar con otro tipo de librerías o proyectos existentes (por ejemplo: Bootstrap, del que ya hablamos en la asignatura). Además, la combinación de Vue con otro tipo de herramientas o librerías de terceros permite la construcción de Single-Page Applications. Vue.js utilizará una combinación de HTML, CSS y JavaScript que permitirá al usuario construir estas aplicaciones utilizando diferentes componentes y características que iremos introduciendo a lo largo de la sesión.

Si se desea conocer un poco más acerca de la comparación de Vue con otros frameworks de desarrollo existentes en la actualidad, a través de la web oficial es posible consultar una comparación detallada.

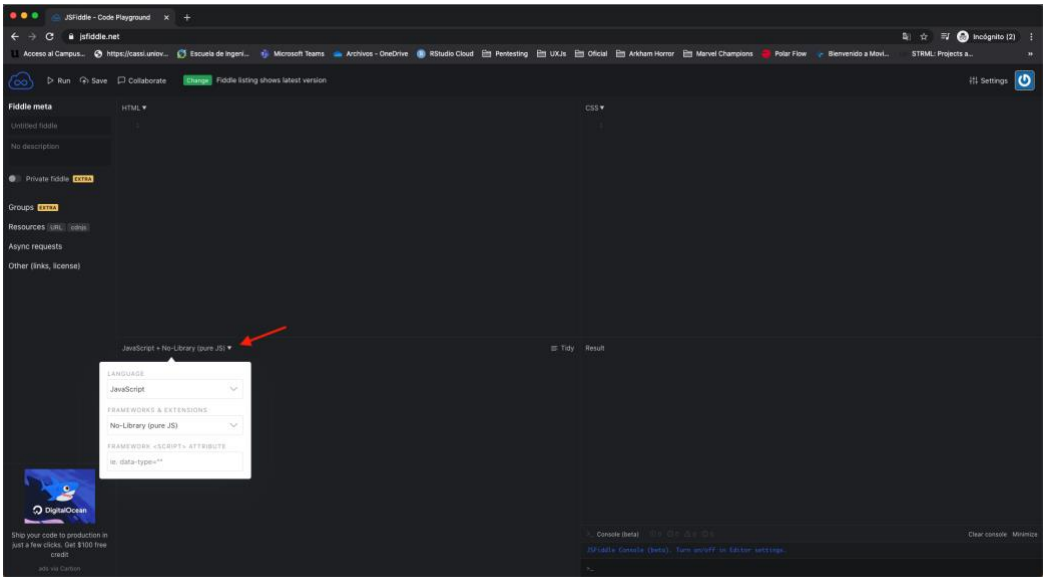
Manejo de Vue.js con JSFiddle

De cara a dar los primeros pasos con este framework utilizaremos la herramienta JSFiddle, que nos ofrece un entorno de pruebas perfecto para la utilización del framework sin necesidad de instalar ningún tipo de componente en el equipo. No obstante, para la realización del ejercicio asociado a este bloque de la asignatura sí será conveniente realizar una instalación en local que permita trabajar más cómodamente y acceder a funcionalidades/componentes más avanzados; es por ello que se recomienda consultar el proceso de instalación recomendado para el Vue CLI aquí.

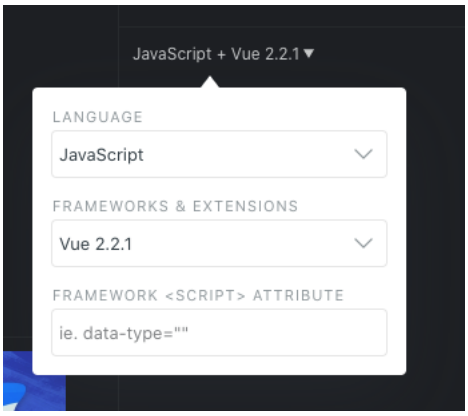
A la hora de utilizar Vue.js dentro de la plataforma JSFiddle, se recomienda configurar el editor de JavaScript de la misma para que utilice las características del framework y las posibles extensiones que tenga asociadas.

Para ello, en un primer lugar entraremos en la web de JSFiddle e iniciaremos sesión con nuestra cuenta de usuario, de tal forma que todo el trabajo que vayamos realizando se pueda almacenar en caso de que se quiera recuperar más adelante.

Una vez en sesión, será necesario ir a la zona de JavaScript y pulsar en la fecha blanca descendente para abrir las opciones de configuración de JavaScript.



Una vez abierto el panel de configuración correspondiente, en el selector de Frameworks y Extensiones será necesario seleccionar el valor Vue 2.2.1, de la manera que se muestra en la siguiente captura. A partir de ese momento el entorno estará preparado para comenzar a funcionar.



Hola Mundo con Vue.js

De forma análoga a lo que sucede con la mayor parte de los lenguajes de programación, la primera incursión que realizaremos con Vue.js será la creación de una aplicación “Hola mundo”. Para ello partiremos de una estructura tradicional de una aplicación de este estilo en HTML plano y la iremos modificando hasta terminar con la aplicación en Vue.js

Para ello, copiaremos el siguiente código de HTML en la parte de JSFiddle destinada para este tipo de código y daremos al botón Run (parte superior izquierda). Al presionar sobre dicho botón, se mostrará la ejecución de dicho ejemplo en una ventana que representa al navegador en la esquina inferior derecha de la pantalla, donde veremos el mensaje “Hola mundo!!”.

```
<div>
  <h1>
    Hola mundo!!
  </h1>
</div>
```

Una vez completo el ejemplo en HTML plano, comenzaremos a declarar la estructura del código que permite pasar este ejemplo a Vue.js. En primer lugar, en la zona de código JavaScript escribiremos el siguiente código:

```
new Vue({
  el: '#root',
  data: {
    message: 'Hola mundo!!'
  }
})
```

Con las líneas de código JavaScript anteriores estamos procediendo a declarar una aplicación de Vue.js que se asocia al elemento con id = root de nuestro código HTML (a través de la propiedad **el**) y que tiene una variable de datos llamada message que tiene el valor ‘Hola mundo!!’ dentro de la propiedad **data**.

En el caso de que procediéramos a pulsar de nuevo sobre el botón Run no habría ningún cambio a nivel de la aplicación actual, ya que el código JavaScript que hemos escrito no está vinculado de ninguna manera con el código HTML de la primera parte del ejemplo. Para vincular correctamente los dos trozos de código, será necesario modificar el código HTML de la siguiente manera:

- Añadir al elemento div el identificador root
- Modificar el valor contenido en las etiquetas h1 a {{ message }}

La sintaxis de {{ variable }} se conoce como mustaches y es compartida por varios frameworks de programación web como Vue y AngularJS; esta sintaxis permite al código HTML acceder a los valores en forma de texto plano de las variables que están declaradas en el código JavaScript y mostrar sus valores entre las marcas estáticas del HTML, convirtiendo a la aplicación resultante en dinámica ante los cambios en los valores de sus variables y objetos. El código HTML resultante sería el siguiente.

```
<div id="root">
  <h1>
    {{ message }}
  </h1>
</div>
```

Ahora, al pulsar sobre el botón Run, la aplicación seguirá mostrando el mismo mensaje pero en este caso estará obteniendo el valor del mensaje de la variable JavaScript y no del código estático del HTML. Este sería el código resultante de nuestra primera aplicación con Vue.js.

Propiedades computadas

Tal y como se puede observar en el ejemplo anterior, que es una de las aplicaciones más simples que se pueda programar, Vue.js es capaz de realizar cómputo a nivel de JavaScript dentro de las etiquetas HTML que componen el árbol DOM y utilizar ese resultado como valor para la etiqueta en cuestión.

A pesar de que Vue sea capaz de realizar dicha tarea, no es conveniente llenar la parte del HTML (conocido como **template**) de expresiones con demasiada lógica de programación, ya que el objetivo principal de las expresiones a nivel de diseño es dejarlas para la realización de operaciones simples. Uno de los principales consejos a nivel del **template** es mantenerlo lo más simple y declarativo posible; además, cuanta más información se meta en el **template** y más compleja sea esta información, más difícil y complejo de mantener se volverá ese código.

Para favorecer la existencia de expresiones simples Vue.js incluye un mecanismo denominado propiedades computadas, que a través de la directiva **computed** se hacen accesibles al código de la plantilla y permiten realizar aquellas operaciones más complejas como si de funciones normales se tratara. En el ejemplo que se incluye a continuación se puede observar la declaración de una propiedad computada que devuelve la versión reversa de una cadena de texto.

```
computed: {
    reversedMessage: function () {
      return this.message.split('').reverse().join('')
    }
  }
```

A la hora de utilizar este tipo de variables, simplemente se añade su nombre al código de la misma manera que se utilizan las variables normales que están declaradas dentro de la propiedad data.

```
<div id="example">

  <p>Mensaje original: "{{ message }}"</p>

  <p>Mensaje invertido computado: "{{ reversedMessage }}"</p>

</div>
```

¿Por qué creéis que se recomienda el uso de las propiedades computadas en vez de simplemente declarar una función normal dentro de methods que realice la misma operación?

Directivas

Las directivas de Vue.js son un conjunto de atributos especiales que se pueden utilizar en el marcado del código HTML de las aplicaciones y que empiezan por v-. A excepción de un caso concreto (que se especificará más adelante), el valor de una directiva tiene que ser una única expresión de JavaScript. A través de este componente se logra aplicar, sobre el árbol DOM de la aplicación, los cambios en los valores de las distintas variables de una aplicación.

Esto se debe a la característica denominada vinculación de datos, que en inglés se conoce como two-way data binding.

Asociando una variable a un control

Una de las características de los frameworks de desarrollo como Vue es la posibilidad de asociar una variable a alguno de los controles del interfaz, de tal manera que los cambios que se produzcan sobre el valor de dicha variable se propaguen a la interfaz automáticamente. La ***directiva*** asociada a este funcionamiento es **v-model**.

Para asociar la variable message a un control de nuestro HTML, crearemos en primer lugar un input de HTML que nos permita modificar el valor de nuestra variable. En este caso, la sintaxis del html quedará de la siguiente manera:

```
<div id="root">
  <h1>
    {{ message }}
  </h1>
  <input v-model="message"/>
</div>
```

Tras el cambio anterior sobre el HTML, la ventana que muestra el estado de la aplicación mostrará el mensaje “Hola mundo!!” y debajo un input de texto con el mismo texto, ya que este se corresponde con el valor que toma actualmente la variable message.

Pero, ¿qué pasará en el caso de que el usuario edite el texto del input? Importante: esperar al debate en clase para realizar la prueba.

¿Qué efectos tiene esto sobre el funcionamiento de nuestra aplicación? Pensad en casos concretos de uso.

Condicionales

Vue.js dispone de un conjunto de directivas condicionales que permite al usuario controlar la visualización y/o el renderizado de los elementos en el árbol DOM del HTML a partir de los valores que tengan las variables de la propia aplicación. En este caso, las directivas implicadas serían **v-if** y **v-show**.

La primera de esas directivas, **v-if**, evalúa la expresión que se le pasa como valor y en el caso de que dicha expresión evalúe a verdadero muestra el elemento en el árbol DOM del HTML. De la

misma forma que sucede en los entornos y lenguajes de programación tradicionales, esta etiqueta está acompañada de etiquetas complementarias: **v-else-if** y **v-else**.

Vamos a partir ahora de un ejemplo completamente nuevo, con el código HTML y el código JavaScript que se pueden ver a continuación.

```
<div id="root">
  <div v-if="count > 0">
    Valor positivo
  </div>
  <div v-else-if="count < 0">
    Valor negativo
  </div>
  <div v-else>
    Cero
  </div>
</div>
```

```
new Vue({
  el: '#root',
  data: {
    count : 0
  }
})
```

En los fragmentos de código anteriores se puede observar que se ha modificado la variable existente en el código JavaScript para que sea una variable de tipo numérico y que el código HTML se ha modificado para que, en función del signo del número almacenado en dicha variable, se muestre un mensaje u otro. Si ejecutamos el código y vamos modificando el valor de la variable count, se observa como el resultado que se muestra en la pantalla es diferente.

¿ Cómo se verán afectados por el valor de la expresión a nivel del árbol DOM los divs de código que están marcados con esta directiva? Importante: Esperar al debate en clase para mirar el código con el inspector.

Esta es la característica que diferencia a las dos directivas condicionales que introdujimos en este bloque de directivas condicionales, ya que si todas las directivas del ejemplo anterior se sustituyen por la directiva v-show, el árbol DOM cambiará.

Al utilizar el código incluido a continuación, ¿qué pasará con el código HTML de la aplicación?
Importante: Esperar al debate en clase para mirar el código con el inspector.

```
<div id="root">
  <div v-show="count > 0">
    Valor positivo
  </div>
  <div v-show="count < 0">
    Valor negativo
  </div>
  <div v-show="count === 0">
    Cero
  </div>
</div>
```

A pesar de que en estos ejemplos estemos usando una variable de carácter numérico para hacer las expresiones que muestran/ocultan los diferentes elementos, las expresiones pueden ser también variables de tipo booleano o el resultado de la ejecución de una función que devuelva un valor booleano. Veamos esto con un ejemplo: vamos a hacer que un mensaje se muestre o no en función de la longitud de una variable.

```
new Vue({
  el: '#root',
  data: {
    message : 'Un texto largo para que salga el mensaje'
  }
})
```

```
<div id="root">
  <div v-if="message.length > 10">
    Texto largo
  </div>
  <div v-else>
    Texto corto
```

```
  </div>

  <br/>

  <input v-model="message"/>

</div>
```

Si ahora procedéis a modificar el valor de la variable message a partir de la manipulación del mensaje que se muestra en el interior del input, veréis como aparece y desaparece cada uno de los mensajes de los divs.

Directivas con argumentos

Algunas de las directivas incluidas en Vue.js tienen una sintaxis que permite asociarles un argumento, característica denotada por la utilización del carácter ":".

La directiva v-bind

Una de estas directivas es la directiva **v-bind**, que permite actualizar los atributos HTML asociados a las etiquetas.

En el ejemplo que se incluye a continuación veremos como es posible modificar el atributo disabled asociado a un input en función del valor que se encuentra almacenado en una variable. En este caso, intentaremos replicar un ejemplo real: deshabilitar un botón de submit de un formulario de suscripción a una newsletter en el caso de que el valor introducido en el campo no tenga la estructura de una dirección de email.

```
new Vue({

  el: '#root',

  data: {

    email : ''

  }

})
```

```
<div id="root">

  <input v-model="email"/>

  <button onclick="alert('tiene pinta de email!!')" v-
bind:disabled="email.length < 5 || email.indexOf('@') === -
1">Suscribirse!!</button>

</div>
```

El código HTML anterior muestra un input de texto donde poder escribir el valor de la variable email y un botón de suscripción que se habilita bajo ciertas circunstancias; en este caso, el botón se activará cuando la longitud del email sea mayor de 5 caracteres y, además, dicha cadena

contenga el carácter '@'. Observar como va cambiando el valor del atributo disabled del botón en función de esas condiciones de la variable.

Aplicando clases de CSS en función de condicionantes

Una de las cosas que nos permite la directiva v-bind es jugar con la asignación de las clases de css que se aplican a cada uno de los elementos del DOM a partir de la comprobación de una serie de condicionantes.

Simplificando el código del ejemplo anterior, pongamos que tenemos un campo de texto para un email y queremos que el color del borde del campo se modifique en función del resultado de la validación del email que contiene. A nivel del objeto Vue de nuestra aplicación no cambiaría nada, ya que dispone de la variable email que utilizaremos para realizar la validación; sin embargo, el código HTML del ejemplo sí que variará, siendo necesario añadirle la directiva **v-bind**:class.

¿Cuál sería el contenido que le pondríais a la directiva para que el borde del input sea rojo si no valida una condición o verde en el caso de que la valide? Importante: esperar al debate en clase para continuar con la lectura del guión.

```
<div id="root">
  <input v-model="email" v-bind:class="???"/>
</div>
```

La solución a la pregunta anterior es: ***un operador ternario***. Este tipo de condicionales tienen tres elementos: el primero es la condición que se ha de comprobar, el segundo el valor devuelto en caso de que la condición evalúe a verdadero y el tercero el valor devuelto en caso de que la condición evalúe a falso; el primer y segundo valor estarán separados por el carácter '?' mientras que el segundo y tercer valor estarán separados por el carácter ':'.

Teniendo en cuenta lo que se pretende lograr a nivel de estilos, el primer paso radica en la creación de dos clases de CSS que modifiquen la propiedad border de un elemento para cambiarle su color.

```
.valid {
  border: 2px solid green;
}
.invalid {
  border: 2px solid red;
}
```

Con las clases de CSS ya creadas, el siguiente paso es crear el operador ternario dentro del atributo **v-bind:class**; en este caso, tomaremos un email como inválido mientras no tenga el carácter '**@**'. Para conseguir este efecto, se modificará el atributo **v-bind:class** para que tenga el siguiente valor:

```
<div id="root">

  <input v-model="email" v-bind:class="[email.indexOf('@') === -1 ? 'invalid'
: 'valid']"/>

</div>
```

Como se puede observar a partir del ejemplo anterior, en este caso es necesario envolver el valor propuesto para la directiva **v-bind** con unos corchetes, de cara a asegurar que se toma todo como un mismo valor y que se evalúa la expresión correspondiente de forma correcta.

¿Qué pasa con el color del borde del campo de texto según vais escribiendo?

La directiva v-on

Otra de las directivas importantes de Vue.js que también dispone de una ejecución con argumentos es la directiva **v-on**, que escucha los eventos del DOM y reacciona en función de ellos.

¿Cómo modificaríais el ejemplo de la directiva v-bind para utilizar la directiva v-on sobre el atributo click de un botón?

La pregunta anterior tiene un poco más de complejidad que las anteriores, ya que en este caso no basta con cambiar solamente esa propiedad del botón, si no que será necesario añadir una nueva propiedad al objeto Vue.

Hasta ahora solamente teníamos las propiedades **el**, que hace referencia al elemento raíz asociado a la aplicación Vue, y **data**, que hace referencia al apartado de datos de la aplicación. Para poder acometer los cambios mencionados anteriormente, será necesario añadir una propiedad **methods** al objeto Vue; esta propiedad **methods** es la encargada de albergar todas las funciones de JavaScript que se pueden utilizar en el código HTML para realizar las operaciones correspondientes.

En este caso, declararemos una función subscribe que no recibirá ningún parámetro y cuyo objetivo es mostrar un alert que diga que se procede a enviar el email. Al no modificar el resto del código del botón podemos estar seguros de que esta función solamente se invocará en el caso de que el email cumpla con las restricciones que habíamos añadido en el paso anterior, ya que en caso contrario el botón estaría deshabilitado. La propiedad **methods** a añadir al código del objeto Vue tiene el siguiente código:

```
methods: {

  subscribe: function() {

    alert('envío el email');

  }

}
```

Una vez añadida la función anterior, es necesario modificar el código del botón y sustituir el atributo onclick y su valor por v-on:click=”subscribe”. A partir de ese momento y tras volver a

ejecutar el programa, cuando se haga clic en el botón y este esté habilitado, se llamará a la función subscribe.

Modo abreviado de algunas directivas

Dentro de las directivas mencionadas anteriormente, existen dos que poseen una formula abreviada para su aplicación en el código, siendo esta fórmula abreviada bastante utilizada por los desarrolladores. Los casos concretos son:

- **v-bind** se sustituye por **:** (por ejemplo, v-bind:disabled sería solamente :disabled)
- **v-on** se sustituye por **@** (por ejemplo, v-on:click sería solamente @click)

Manejo de datos

Una vez visto todo lo relacionado con las directivas más importantes y sabiendo el cometido de las distintas propiedades que hemos utilizado hasta el momento, es hora de comprobar cómo se manejan los datos en Vue.

Recorrido de una colección

Partiremos de nuevo de un ejemplo en blanco y crearemos un objeto de Vue con el siguiente código.

```
new Vue({
  el: '#root',
  data: {
    players : [
       { name: 'Messi', club: 'Barcelona' },
      { name: 'Benzema', club: 'Madrid' },
      { name: 'Joao Felix', club: 'Atlético' }
    ],
    newName : '',
    newClub : ''
  }
})
```

Como se puede observar en el código anterior, existen tres variables de datos declaradas en ese momento: un array de nombre players (que en su interior contiene objetos con propiedades name y club) y dos variables sencillas de nombre newName y newClub.

Una vez creada la estructura de datos del array players, es necesario declarar el código que permita recorrer cada uno de los objetos que contiene y mostrar la información de los mismos por pantalla. Para ello utilizaremos la directiva **v-for**, que se corresponde con el caso comentado anteriormente de la única directiva cuyo valor no es una única expresión JavaScript. La sintaxis de esta directiva permite asignar cada uno de los elementos del array en una variable que posteriormente puede ser accedida para extraer su información, tal y como muestra el siguiente código.

```
<div id="root">
  <ul>
    <li v-for="player in players">
      {{ player.name }} juega en el {{ player.club }}
    </li>
```

```
  </ul>
</div>
```

En el caso anterior, el código HTML está creando una lista no ordenada de jugadores donde se muestra para cada uno de ellos su nombre y el club al que pertenecen, utilizando una expresión de JavaScript entre los mustaches para concatenar los valores de las variables correspondientes. Es necesario fijarse bien en la sintaxis del bucle **v-for**, ya que en un primer momento se establece el nombre de la variable donde se van recorriendo uno a uno los datos y después se hace referencia a la colección que se recorre.

Adición de nuevos objetos a la colección

Después de haber visto cómo se recorre una colección para mostrar su contenido, es momento de conocer cómo se pueden añadir nuevos elementos a una colección. Para ello será necesario añadir, en primer lugar, los controles HTML que permitan al usuario introducir la información deseada, así como un botón que permita enviar los datos para su almacenaje. En consecuencia, el código HTML del ejemplo queda de la siguiente manera.

```
<div id="root">
  <input v-model="newName" placeholder="Introduce el nombre"/>
  <input v-model="newClub" placeholder="Introduce el club" />
  <button v-on:click="createPlayer"> Nuevo jugador </button>
  <ul>
    <li v-for="player in players">
      {{ player.name }} juega en el {{ player.club }}
    </li>
  </ul>
</div>
```

Para facilitar la comprensión del interfaz gráfico a la hora de saber cual de los controles almacena el nombre y cual el club, se han añadido unos textos de ayuda a través del atributo placeholder de cada uno de estos controles. Además, se ha procedido a asociar cada uno de los controles con la variable de datos que le corresponde a través de la directiva **v-model**, de tal manera que toda la información que se introduzca en los campos de texto se vuelque automáticamente a la variable que corresponda. El botón que va a permitir el almacenaje de la información en la colección de datos existente ha sido creado de tal forma que con cada clic sobre el mismo se va a llamar a la función createPlayer, que cogerá los valores introducidos en los campos y los volcará a la colección como un nuevo objeto.

La siguiente modificación a realizar será la relativa al apartado de funciones de nuestro objeto Vue, ya que será necesario añadir el método createPlayer al conjunto de métodos disponibles en el objeto, de cara a que el botón existente en el HTML pueda ejercer su función. De momento,

lo único que tiene que hacer este método es volcar la información de los campos a la colección de jugadores existentes. Así, se añadirá el siguiente código al objeto Vue.

```
methods: {

  createPlayer : function() {

       return this.players.push({name: this.newName, club: this.newClub});

  }

}
```

Al introducir datos en los campos de texto y pulsar en el botón “Nuevo jugador”, la aplicación coge los datos almacenados en las variables y los añade a la colección de jugadores, por lo que además el usuario verá refrescada la lista de jugadores automáticamente en el interfaz. **Probar a añadir al jugador Koundé que juega en el Sevilla y observar como la lista crece a 4 jugadores.**

Ajustes en la interacción

Si bien el método de añadir elementos a la lista funciona correctamente, existen dos cosas del ejemplo anterior que se pueden mejorar.

En primer lugar, se puede observar como al introducir la información de un jugador y pulsar en el botón “Nuevo jugador” para confirmarla, el objeto se añade correctamente a la lista pero los valores introducidos en los campos permanecen en su lugar una vez que el jugador ya ha sido añadido; sería mucho más cómodo que los valores se borraran una vez que el elemento ha sido añadido. Para lograr esta mejora es necesario modificar la función createPlayer para añadir la línea que elimine los datos de las variables una vez que ya han sido usados; además, se puede ajustar un poco más la función con la eliminación de la palabra clave return, ya que no es necesario que dicha función retorne nada.

```
methods: {

  createPlayer : function() {

    this.players.push({name: this.newName, club: this.newClub});

    this.newName = '';

    this.newClub = '';

  }

}
```

Con las modificaciones anteriores, cada vez que se añade un nuevo jugador a la colección se borran los datos almacenados en las variables utilizadas, de tal manera que la creación de varios elementos uno detrás de otro se simplifica.

En segundo lugar, a pesar de que solamente existe un botón en el interfaz, sería más cómodo que el usuario pudiera presionar enter después de meter la información del último campo para desencadenar la creación del nuevo jugador. Para hacer esto solamente es necesario añadir la

directiva v-on:keyup.enter=”createPlayer” al segundo de los campos de texto del interfaz, de tal manera que el código HTML quede así.

```
<div id="root">

  <input v-model="newName" placeholder="Introduce el nombre"/>

  <input v-model="newClub" placeholder="Introduce el club"
@keyup.enter="createPlayer"/>

  <button v-on:click="createPlayer"> Nuevo jugador </button>

  <ul>

    <li v-for="player in players">

      {{ player.name }} juega en el {{ player.club }}

    </li>

  </ul>

</div>
```

Observar como en este caso se ha optado por utilizada la sintaxis abreviada @ para hacer referencia a la directiva v-on. **Probar a añadir un jugador de elección personal pulsando enter al tener el cursor sobre el segundo campo después de haber introducido el club.**

Filtros sobre datos

En algunos casos es necesario actualizar o transformar la información que se muestra en la vista al usuario sin modificar realmente los datos de los que se está tomando el valor correspondiente. Esta funcionalidad se logra en Vue.js utilizando los filtros que se pueden aplicar sobre los datos.

La aplicación de un filtro se corresponde con la ejecución de una función que modifica la información que se va a mostrar por pantalla sin alterar el valor real de la misma dentro del objeto que la contiene. Para aplicar un filtro solamente es necesario invocar a la función del filtro al mostrar el valor de la variable que corresponda, concatenando el nombre de la función con el de la variable representada a través del uso del carácter |.

Vamos a añadir un filtro a nuestra colección de jugadores para que los nombres propios de los futbolistas se muestren siempre con la primera letra en mayúsculas y el resto en minúsculas, sin importar si el usuario los ha introducido de esta o de otra forma. Así, en primer lugar añadiremos la llamada al filtro detrás del acceso a la variable name del jugador, dentro del bucle for creado anteriormente.

```
<li v-for="player in players">

  {{ player.name | capitalize }} juega en el {{ player.club }}

</li>
```

Si se procede a ejecutar dicho código se observará que se produce un error en la renderización de los valores del listado, ya que la función de capitalización todavía no está declarada y, por lo tanto, no puede ser ejecutada.

Para añadir esta función al código del objeto Vue de nuestra aplicación será necesario añadir la propiedad **filters** a dicho objeto, ya que esta propiedad es la encargada de albergar todas las funciones de aplicación de filtros que existan. Así, añadiremos el siguiente código a nuestro objeto Vue a continuación de la propiedad **methods**:

```
filters: {

  capitalize : function(playerName) {

    return playerName.charAt(0).toUpperCase() + playerName.slice(1);

  }

}
```

A partir de este momento, el listado de jugadores que se muestra por pantalla siempre tendrá la primera letra del nombre del jugador en mayúscula, a pesar de que el usuario pueda introducirlo de otra manera. Es importante quedarse con el concepto de que los datos almacenados en la colección no se modifican pero sí que se ven de manera diferente a cómo pueden estar almacenados.

Añadir al jugador iago aspas del Celta de Vigo al listado de jugadores, respetando la capitalización de las letras indicada. Comprobar que al mostrar el listado por pantalla la primera letra del nombre se muestra como una mayúscula.

Modificar la función capitalize para que ponga en mayúsculas la primera letra de cada una de las palabras que componga el nombre del jugador. Añadir al jugador enes unal del Getafe a la lista de jugadores y comprobar que la función anterior realiza su tarea de forma correcta.

Orígenes de datos externos

Hasta el momento hemos estado trabajando con un conjunto de datos que se encuentra siempre almacenado, de forma estática, en el objeto Vue (por ejemplo, el array de jugadores de fútbol). Sin embargo, los datos no siempre estarán presentes de esta manera en nuestra aplicación y, por lo tanto, es necesario conocer la manera de cargar datos en memoria desde fuentes externas.

Para realizar ciertas acciones de forma correcta es necesario conocer el ciclo de vida de una aplicación de Vue.js, que se encuentra explicado en el siguiente enlace. A modo de resumen, una instancia de Vue.js pasa por un conjunto de estados a lo largo de su ciclo de vida (desde que se crea hasta que se destruye) y existen una serie de métodos que permiten al programador ejecutar diversas acciones en los momentos inmediatamente anteriores o posteriores al cambio de cada uno de estos estados. En el caso que nos ocupa actualmente, el momento idóneo para la carga de los datos desde la fuente externa será una vez que la instancia ha sido montada, por lo que se utilizará el hook **mounted**. Para utilizar este hook, será necesario añadir al objeto Vue

de la aplicación la entrada **mounted** después del bloque de datos del objeto y meter dentro de dicha función el código que se requiera ejecutar.

La obtención de datos externos al entorno de la aplicación se puede realizar utilizando la función **fetch**, que permite buscar datos en la dirección que se le pasa como parámetro, y encadenar su ejecución con la llamada a la función **then**, para asegurarnos de que la parte del código que va a almacenar los valores leídos en una variable se ejecute después de que la obtención de los datos haya resultado satisfactoria. En este caso vamos a utilizar la API JSONPlaceholder, que devuelve una serie de datos de pruebas en formato JSON y que nos servirá para explicar cómo hacer las peticiones.

```
mounted () {

  fetch('https://jsonplaceholder.typicode.com/posts')

    .then(res => res.json())

    .then(finalResult => {

      this.info = finalResult

    })

}
```

En el código anterior se realiza la búsqueda de todos los posts existentes en el API mencionado anteriormente y el resultado se vuelca finalmente a la variable info del apartado de datos, de tal forma que luego es recorrer la colección para extraer los datos.

¿De qué manera creéis que se puede recorrer la colección que se ha obtenido de este API?

Componentización

Vue.js permite al usuario definir de forma sencilla sus propios componentes para después reutilizarlos a lo largo de la aplicación tantas veces como se necesite. Este es un enfoque promovido por este tipo de frameworks para evitar uno de los puntos débiles que tiene HTML a la hora de realizar el desarrollo de aplicaciones web: la imposibilidad de incrustar contenido de un fichero en otro y, por lo tanto, la imposibilidad de reutilizar el código.

A la hora de definir y utilizar un componente para Vue es necesario realizar los pasos que se detallan a continuación:

- Declarar y crear el componente utilizando la sintaxis **Vue.component**
- Rellenar el código asociado al componente en sus diferentes variables, incluyendo la necesidad de parámetros para funcionar
- Vincular el componente a la aplicación Vue
- Instanciar el componente en el código

Declaración de un nuevo componente

La declaración del componente es el primer paso que se ha de realizar para crear este nuevo elemento del código y posteriormente poder utilizar en todo el proyecto. La declaración del componente player-list se realiza utilizando la siguiente sintaxis:

```
Vue.component('player-list', {

  data: function () {

    return {

      ...

    }

  },

  template: ...

})
```

En función de las necesidades del componente en cuestión se irán rellenando los diferentes elementos para que cumplan con su cometido. Merece especial atención en esta sintaxis el hecho de que la variable data no sea un conjunto de datos si no que sea una función que devuelve unos datos.

¿Alguien sabría decirme por qué? Está relacionado con el modo de acceso a los datos y su actualización…

Creación del código del componente

Una vez que el componente ya ha sido declarado, llega el momento de dotarlo del código necesario para que cumpla con su cometido. En este caso el componente albergará una lista de jugadores por lo que el código de la plantilla, que se almacena en el elemento **template**, será el mismo código que se estaba usando hasta este momento para pintar el listado de jugadores.

Adicionalmente, el componente deberá recibir como parámetro la colección de los jugadores que tiene que pintar, por lo que será necesario utilizar la variable **props** en la declaración del componente. Así, el código del componente se rellenaría de la siguiente manera:

```
Vue.component('player-list', {

  data: function () {

    return {}

  },

  props: ['players']

  template: '

    <ul>

      <li v-for="player in players">

        {{ player.name }} juega en el {{ player.club }}

      </li>

    </ul>

  '

})
```

Vinculación del componente al código de la aplicación

Después de rellenar el componente con el código y las variables que necesita, llega el momento de vincular el componente a la aplicación de Vue para que pueda ser utilizado en la misma. Para ello es necesario añadir un nuevo elemento al conjunto de características de la aplicación, que en este caso es **component**; dentro de dicho elemento se almacenará un listado con los componentes que utiliza la aplicación. En el caso actual solamente existe un único componente, que será el que conforme el valor de dicha variable.

```
app = new Vue({

  el: '#root',

  component: ['player-list'],

  data: {

    players : [

       { name: 'Messi', club: 'Barcelona' },

      { name: 'Benzema', club: 'Madrid' },

      { name: 'Joao Felix', club: 'Atlético' }

    ],
```

```
    newName : '',
    newClub : '',
  } ...
```

Instancia del nuevo componente

Una vez que el componente ha sido creado, rellenado y declarado a nivel del objeto Vue que gestiona el funcionamiento de la aplicación, ya es posible utilizarlo como parte del código de la aplicación.

En función del nombre que se le haya dado en la declaración y de la cantidad de parámetros que reciba en su invocación, la línea de código HTML utilizada para incluirlo en la aplicación variará. En el caso que nos ocupa, el código HTML que permite utilizar el componente es el siguiente, nótese la sintaxis abreviada a la hora de asociar el valor del parámetro.

```
<div id="root">
  <player-list :players="players"></player-list>
</div>
```

En el caso de haber mantenido el código del filtro de capitalización sobre el nombre del jugador, ¿qué ocurre con el código anterior?

Consejos de realización de la práctica individual

A partir de los diferentes ejemplos y ejercicios contenidos en este guión es posible tener un primer encuentro con Vue.js sin necesidad de instalar ningún tipo de entorno en local que haya que mantener y actualizar.

No obstante, la realización de la práctica individual de entrega requiere de la creación de un entorno de desarrollo en una máquina local, ya que JSFiddle está limitado a los archivos que existen en pantalla y no es posible crear un proyecto completo de Vue para manejar el desarrollo de una aplicación completa por simple que esta sea.

A continuación se detallan algunos de los puntos más importantes a tener en cuenta a la hora de realizar la práctica individual.

Instalación del CLI de Vue.js

El CLI de Vue.js es la mejor manera de trabajar con este framework en local para el desarrollo de aplicaciones de tipo Single-Page. A través de su web es posible acceder a la instalación de este paquete que permite construir y desplegar las aplicaciones de forma sencilla, siendo un sistema perfecto para aquellas personas que tengan cierta experiencia en el manejo de Node.js.

Es necesario recalcar que el CLI implica la utilización de la línea de comandos para la realización de una serie de tareas como la creación del proyecto o el lanzamiento del servidor de desarrollo para pruebas. A continuación se incluye una lista de los comandos que utilizaréis con mayor frecuencia.

```
npm install -g @vue/cli //instalación del CLI.

vue --version //comprobación de que la instalación se realizó correctamente.

vue create project-name //creación del proyecto project-name

npm run serve //arranque del proyecto para su acceso y desarrollo
```

Entorno de desarrollo integrado

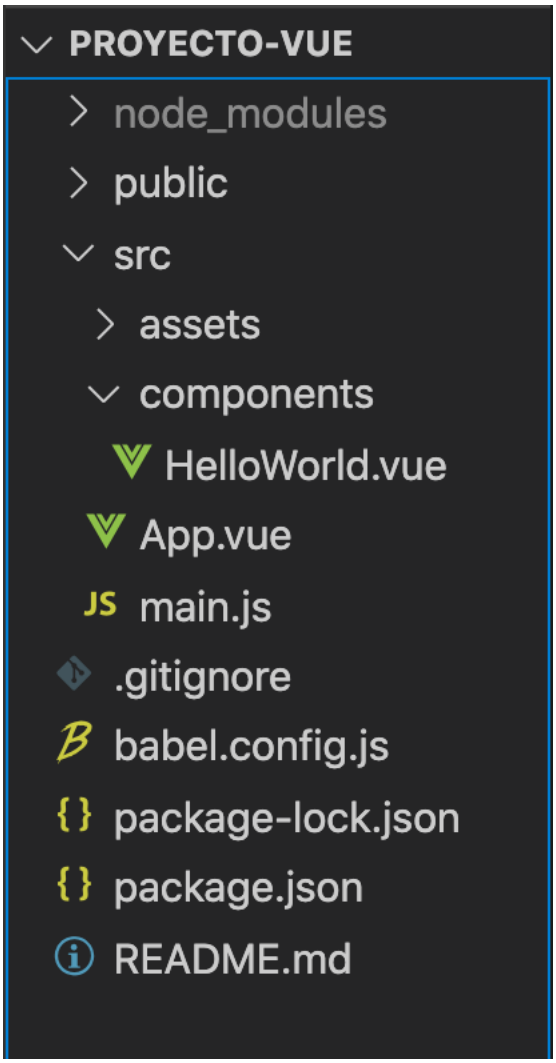
En la actualidad existen infinidad de entornos de desarrollo que permiten trabajar con diversos tipos de lenguajes y frameworks, ofreciendo características que facilitan el proceso de desarrollo a los programadores.

Uno de los entornos de desarrollo más utilizados en la actualidad es VS Code, que se puede descargar desde el siguiente enlace. Utilizando esta herramienta seréis capaces de trabajar con Vue.js sin problemas y podréis gestionar todas las necesidades de la práctica de una manera sencilla y cómoda.

Estructura del proyecto creado

Al utilizar el comando de creación del proyecto de la lista anterior de comandos, el CLI realiza las acciones pertinentes para crear un proyecto en el directorio que le hayáis indicado a la herramienta.

Una vez que esta realiza todas las acciones necesarias, podréis abrir el directorio de vuestro proyecto con el entorno de desarrollo que hayáis escogido. Al abrir el proyecto veréis la siguiente estructura de directorios y ficheros.



Los ficheros que tienen la extensión .vue representan aquellos componentes y elementos de la aplicación que son un compendio de código JavaScript, HTML y CSS; es decir, cada uno de estos ficheros representa el entorno de trabajo que hemos estado utilizando a través de la herramienta JSFiddle.

Del listado de ficheros y directorios anterior, tenéis que tener en cuenta lo siguiente:

- El directorio components irá acogiendo los ficheros .vue de todos los componentes que vayáis declarando para vuestra aplicación.
- El fichero App.vue es el fichero principal de vuestra aplicación de Vue.js y en él iréis añadiendo las referencias y los elementos de código que utilicen los componentes que vayáis declarando.
    - Será necesario que este fichero contenga los siguientes elementos por cada componente que declaréis: una línea de importación (palabra clave import), una entrada en la directiva components y el código HTML que referencie a vuestro componente en la template.

    - En la parte de la template de este fichero está la declaración del div con id app que es el elemento principal de la aplicación (lo que en nuestros ejemplos anteriores era el elemento root).
- El fichero main.js es el punto de entrada de la ejecución de la aplicación y en él se irán declarando los diferentes elementos externos que queráis utilizar (por ejemplo, librerías).

Utilización de la librería Bootstrap

De cara a que sea posible centrarse en la realización de la funcionalidad solicitada, se recomienda utilizar la versión específica para Vue de la librería Boostrap para que podáis poner en práctica parte de los conceptos tratados en la sesión anterior.

Para conocer como se realiza la instalación de la librería de cara a poder ser utilizada en el código, podéis consultar el siguiente enlace.

Cómo organizar la práctica

La práctica no tiene una única solución, ya que es posible realizar la vista de listado y de detalles de los elementos de SWAPI de varias maneras diferentes. A modo de sugerencia, podéis considerar las siguientes ideas:

- Tener una página con un listado de elementos de SWAPI y que al pulsar en algún control de los elementos del listado se abra algún tipo de ventana modal con la información.

- Tener una página con un listado de elementos de SWAPI y que al pulsar en algún control de los elementos del listado se visualice una nueva vista que muestre los detalles del elemento seleccionado. Esta vista deberá tener algún tipo de control que permita volver a la vista anterior para recuperar el listado de elementos.

- Tener una página con un listado de elementos de SWAPI y que al pasar el ratón por encima de los elementos de la tabla se visualice algún tipo de infoView con los detalles de dicho elemento.