Vue

Introdución

Un **framework** é basicamente unha biblioteca de terceiros que proporciona funcionalidades, métodos e ferramentas listas para ser utilizadas. Ademais, proporciona unha serie de regras e convencións que se deben seguir para construír aplicacións altamente dinámicas e interactivas de forma fácil.

Nos últimos anos popularizouse a creación de aplicacións **SPA** (*Single Page Application*) que consisten basicamente nunha páxina HTML xunto con un script, que é o encargado de ir actualizando a información, sen necesidade de recargar a páxina por completo. Exemplos deste tipo de páxinas son a versión web de WhatsApp, Twitter ou Google Drive. Observar que neste tipo de aplicacións a persoa usuaria pode interactuar co navegador e a información actualízase, mais a páxina non se recarga por completo.

No desenvolvemento de aplicacións SPA utilízanse diferentes frameworks de cliente, entre outros Angular, React eVue:

- Angular é un framework de JavaScript de código aberto baseado en **TypeScript** e mantido por Google.
- React é realmente unha biblioteca de JavaScript, de código aberto, mantida por Facebook. Úsase para construír interfaces de usuario baseadas en compoñentes.
- Vue é un framework de JavaScript de código aberto baseado en compoñentes. Foi creado por Evan You despois de traballar para Google usando AngujarJS.

O obxectivo destas ferramentas é facilitar o desenvolvemento da interface gráfica de aplicacións tipo SPA.

Imaxina unha aplicación típica de creación dunha lista de tarefas: a aplicación permitirá listar as tarefas e tamén crealas e eliminalas. Simplemente a creación dos novos elementos do DOM para engadir unha nova tarefa require unha grande cantidade de código. Pode verse un exemplo deste tipo de aplicación en CodePen.

Os frameworks de JavaScript foron creados para facer que este tipo de traballo sexa moito máis fácil. Non proporcionan novas funcionalidades a JavaScript, simplemente proporcionan á persoa desenvolvedora novas ferramentas para crear aplicacións de forma máis rápida e fácil.

Os framework de JavaScript ofrecen a posibilidade de escribir o código da interface web de forma declarativa, é dicir, permiten á persoa desenvolvedora escribir código que describa o aspecto que terá a interface. Será o framework o encargado de traducir o código a JavaScript de forma automática e transparente.

Tomando como exemplo a aplicación da lista de tarefas, o seguinte anaco de código ilustra a forma na que podería usarse Vue para describir a lista de tarefas. Observar que este código utiliza instrucións que non son JavaScript (v-for, v-bind:key), que describen como será o código JavaScript final. Este código é máis simple que o código que sería necesario para crear a estrutura DOM usando JavaScript puro:

```
<ul>
  <li v-for="task in tasks" v-bind:key="task.id">
    <span>{{task.name}}</span>
    <button type="button">Delete</button>
  </li>
</ul>
```

Hai que dicir que tamén sería posible o uso de Template literal strings para a construción do DOM, o que proporcionaría un código similar ao anterior. Isto sería unha boa idea para aplicacións simples e pequenas, non así para aplicacións grandes, pois o código sería difícil de xestionar e pouco eficiente.

Como escoller un framework

Cada framework proporciona diferentes solucións para o desenvolvemento web e ten as súas vantaxes e inconvenientes, polo que a decisión de cal usar debe ser tomada polo equipo en función do proxecto a desenvolver.

Algunhas consideracións que se deben ter en conta para tomar unha mellor decisión á hora de escoller un framework é valorar os seguintes puntos:

- soporte polos navegadores
- documentación e soporte do framework
- linguaxes específicas de dominio (DSL - *domain specific languages*) que utiliza.

  Domain-specific languages (DSLs) son linguaxes relevantes en áreas específicas de desenvolvemento de software. No caso dos frameworks, os DSL son variacións de JavaScript ou HTML que facilitan o desenvolvemento de aplicacións con un framework particular. Aínda que ningún framework require o uso obrigatorio dunha linguaxe DSL específica, usala optimiza o proceso de desenvolvemento da aplicación.

A continuación móstrase a información relativa aos diferentes frameworks:

Framework	**Soporte navegador**	**DSL preferida**	**DSLs soportadas**
Angular	Modern	TypeScript	HTML-based; TypeScript
React	Modern	JSX	JSX; TypeScript
Vue	Modern (IE9+ in Vue 2)	HTML-based	HTML-based, JSX, Pug

JSX (JavaScript XML) é unha extensión de sintaxe de JavaScript que permite escribir código HTML nun ficheiro JavaScript. As páxinas web están construídas con HTML, CSS e JavaScript, normalmente en ficheiros separados. Hoxe en día, debido á interactividade da web, é frecuente que JavaScript determine o contido dunha páxina (HTML). Por iso React permite crear compoñentes escribindo JavaScript e HTML no mesmo ficheiro. Exemplo:

```
// Sidebar.js
Sidebar() {
    if (isLoggedIn()) {
        <p>Welcome</p>
    } else {
        <Form />
    }
}
```

TypeScript é basicamente JavaScript con tipado de datos. Exemplo:

// JavaScript
function add(a, b) {
  return a + b;
}
	// TypeScript
	function add(a: **number**, b: **number**): **number** {
	  return a + b;
	}

O framework escollido para traballar é Vue, xa que é máis fácil de aprender que React ou Angular. Unha vez coñecido Vue será máis fácil aprender outro, dado que, aínda que teñan diferencias, comparten conceptos similares.

Vue

Vue.js é un framework de JavaScript para construír interfaces de usuario. Funciona sobre HTML, CSS e JavaScript e proporciona un modelo de programación **declarativo** e **baseado en compoñentes** que permite desenvolver interfaces de usuario de forma eficiente, xa sexan simples ou complexas.

A palabra **vue** vén do francés e significa vista (*view*), que é como se denomina a parte visual do MVC (*Model View Controller*), parte na que se centran estes frameworks.

Actualmente a versión de Vue é a 3, polo que este documento fai referencia a esta versión.

As principais características de Vue son:

- **Renderización declarativa**: Vue estende o HTML estándar cunha sintaxe que permite describir de forma declarativa a saída HTML en base ao **estado** JavaScript. Cando o estado cambie, o HTML actualízase automaticamente.
- **Reactividade**: Vue detecta automaticamente os cambios de estado en JavaScript e actualiza o DOM de forma eficiente cando ocorren estes cambios.

Estilos da API

Vue3 proporciona dous estilos diferentes de programar:

- **Options API**: a lóxica dun compoñente defínese usando un obxecto con propiedades chave para o seu funcionamento como **data, methods** e **mounted**. Estas propiedades estarán accesibles nas funcións a través da palabra chave **this,** que apunta á instancia do compoñente.

  A modalidade Options API está baseada no concepto "instancia do compoñente", accesible a través da palabra chave **this,** que resulta máis fácil de usar para persoas que veñen da programación orientada a obxectos. A información está separada en diferentes propiedades do obxecto e permite abstraer os detalles da reactividade de Vue, de modo que a curva de aprendizaxe é moi sinxela e agradable, resultando máis apropiado para persoas que comezan a traballar con frameworks.

- **Composition API**: a lóxica dun compoñente defínese usando funcións importadas da API. É algo máis complexa que a opción anterior, mais mellora a reutilización de código e a súa organización por funcionalidades. Está indicada para aplicacións grandes. Utilízase habitualmente con SFC.

  Soe usarse en proxectos Vue nos que se utiliza algunha ferramenta de ensamblaxe. Neste caso os compoñentes Vue créanse utilizando unha sintaxe denominada **Single-File Component** (SFC), na que os ficheiros teñen extensión ***.vue**. Un arquivo SFC encapsula a lóxica do compoñente (JavaScript), o contido (HTML) e os estilos (CSS) nun único ficheiro

No sitio web de Vue hai un titorial interesante para ir probando de forma dinámica a sintaxe e vendo o resultado de forma automática. Este titorial permite escoller:

- O estilo da API: Options API ou Composition API
- O modo: HTML ou SFC.

Inicialmente, para empezar, escolleremos **Options API** e **HTML**.

Traballando con Vue

Neste apartado explicaranse diferentes formas de traballar con Vue.

A forma máis fácil para empezar e probar Vue é engadir un import ao paquete Vue dende un CDN.

```
<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
```

A liña anterior carga o script Vue global facendo que a API estea exposta mediante propiedades dun obxecto global **Vue**. Exemplo completo:

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

  <script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>

  <script>
    // desestruturación de obxectos equivalente a
    // const createApp = Vue.createApp;
    const { createApp } = Vue;
    createApp({
      data() {
        return {
          message: "Ola mundo!",
        };
      },
    }).mount("#app");
  </script>
</body>
```

Observar que a función **data** é unha propiedade do obxecto pasado como parámetro a createApp. Dende ES2015 é posible utilizar a sintaxe que aparece na columna da dereita:

```
createApp({
  data: function () {
    return {
      message: "Ola mundo!",
    };
  },
}).mount('#app');
```

```
createApp({
  data() {
    return {
      message: "Ola mundo!",
    };
  },
}).mount('#app');
```

Na documentación de Vue utilízase a sintaxe de módulos **ES Modules** de JavaScript, xa que é soportada na maioría dos navegadores modernos. É posible utilizar Vue dende un CDN usando **ES Modules**. Observar que no exemplo seguinte se importa o módulo de Vue dende o CDN:

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

<script type="module">
  import { createApp } from 'https://unpkg.com/vue@3/dist/vue.esm-browser.js'

  createApp({
    data() {
      return {
        message: "Ola mundo!"
      }
    }
  }).mount('#app')
</script>
```

A diferencia do exemplo anterior, na documentación de Vue utilizan ***import maps***, polo que o código equivalente ao anterior é o seguinte:

```
<script type="importmap">
  {
    "imports": {
      "vue": "https://unpkg.com/vue@3/dist/vue.esm-browser.js"
    }
  }
</script>

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

<script type="module">
  import { createApp } from 'vue'

  createApp({
    data() {
      return {
        message: "Ola mundo!"
      }
    }
  }).mount('#app')
</script>
```

NOTA: import maps é unha característica relativamente recente, polo que hai que asegurarse que está soportada polo navegador.

NOTA: os exemplos anteriores utilizan a versión de desenvolvemento de Vue. Para usar Vue en produción, hai que consultar a guía de produción.

Os exemplos anteriores inclúen todo o código no ficheiro HTML, aínda que podería separarse o código JavaScript e colocalo nun ficheiro script enlazado no HTML.

Extensións

Para traballar con Vue será interesente instalar as seguintes extensións en Visual Studio Code:

- Vue - Official - Visual Studio Marketplace en Visual Studio Code.
- Vue 3 Snippets - Visual Studio Marketplace

A ferramenta Vue devtools serve de axuda no desenvolvemento de aplicacións con Vue. Na ligazón pode encontrarse información de como instalar esta ferramenta como extensión do navegador ou como aplicación independente.

Pode instalarse como unha aplicación independente instalando o paquete como dependencia do proxecto:

```
npm add -D @vue/devtools
```

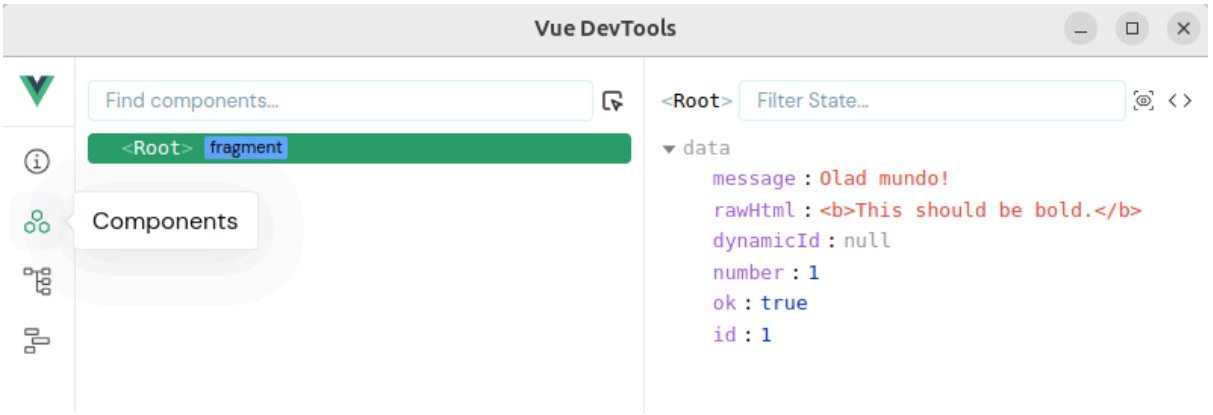
Unha vez instalado o paquete pode lanzarse co seguinte comando:

```
./node_modules/.bin/vue-devtools
```

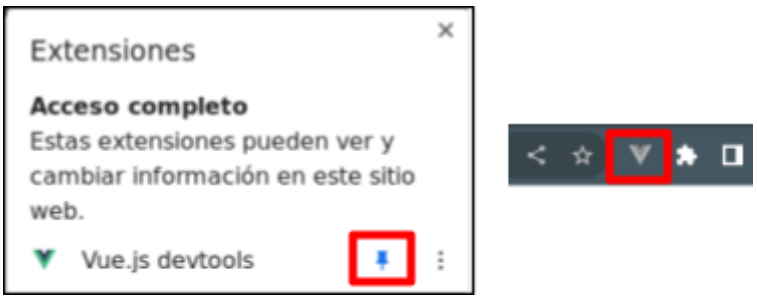
Unha vez se inicia a ferramenta haberá que engadir o seguinte código ao inicio da páxina:

```
<script src="http://localhost:8098"></script>
```

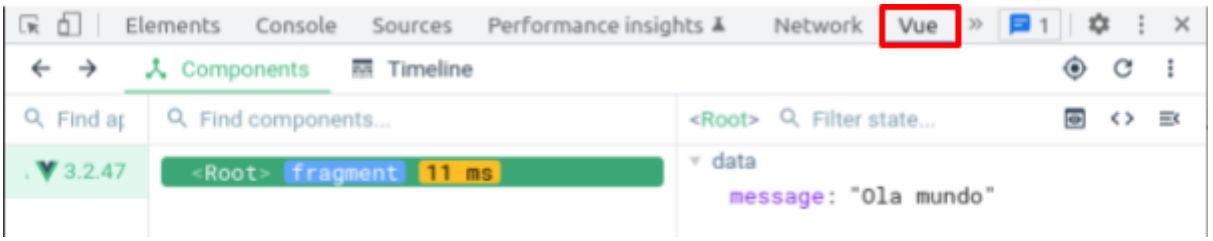
Cando se cargue a páxina no navegador, poderase acceder ás ferramentas de desenvolvemento de Vue dende a aplicación.



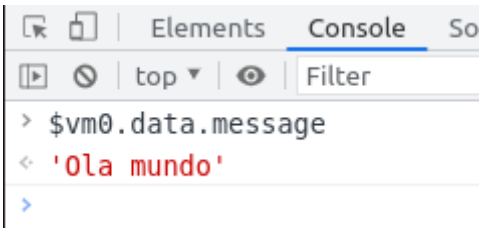
Cando se instala como extensión do navegador é recomendable engadila e fixar a súa icona á barra de ferramentas.



Cando se estea traballando nun proxecto con Vue, pode accederse ás ferramentas de desenvolvemento e analizar o código:



Tamén é posible acceder dende a consola ás instancias de Vue. Utilizando $vm0 faise referencia á instancia da primeira aplicación:



Crear unha aplicación Vue

Toda aplicación Vue comeza creando unha nova instancia da aplicación coa función createApp.

```
const app = createApp({

  /* root component options */

})
```

O obxecto pasado como parámetro a **createApp** é un compoñente. As aplicacións Vue necesitan un **compoñente raíz,** que pode ser un compoñente simple ou conter outros compoñentes como descendentes.

O código HTML debe proporcionar un elemento con un **id** para poder seleccionalo dende Vue e montar aí o compoñente:

```
<div id="app"></div>
```

A aplicación non mostrará nada ata que se invoque o método **.mount()**. Ao método mount hai que pasarlle un “contedor” como argumento, que pode ser un elemento do DOM ou un selector CSS. No caso de usar un selector CSS, montarase no primeiro elemento do DOM que coincida co selector. O contido do compoñente Vue será renderizado no contedor pasado como parámetro:

```
<!-- HTML -->
<div id="app"></div>
```

```
// js
createApp({
   …
  },
}).mount("#app");
```

O método **mount**() devolve como resultado a instancia do compoñente raíz. Debe invocarse despois de ter rexistrados os recursos e configurada a aplicación.

A configuración da conexión entre a aplicación e o HTML realízase a través das propiedades e métodos do obxecto pasado como parámetro ao método **createApp**. Este obxecto ten, entre outros, un método denominado **data()** que devolve un obxecto tal que as súas propiedades constitúen o **estado reactivo** do compoñente. Estas propiedades son accesibles dende HTML, no contedor onde se monta a aplicación, utilizando unha sintaxe especial denominada interpolación {{ }}. Ademais, tamén son accesibles dende outras partes do compoñente, por exemplo, dende os seus métodos utilizando a palabra reservada **this**.

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

```
// js
createApp({
  data() {
    return {
      message: 'Ola mundo!',
    };
  },
}).mount('#app');
```

Exercicio:

1. Proba a crear diferentes propiedades no obxecto devolto por data() e mostralas dende o HTML. Proba a mostrar diferentes tipos de datos: unha cadea, un número, un booleano, un array, un obxecto, etc.

Sintaxe de Templates

Vue usa unha sintaxe de templates baseada en HTML que permite, de forma declarativa, enlazar o DOM co compoñente.

Internamente, Vue compila os templates a código JavaScript altamente optimizado. Combinado co seu sistema de reactividade, Vue é capaz de saber o número mínimo de compoñentes que é necesario re-renderizar e aplicar as mínimas modificacións no DOM para visualizar os cambios no estado da aplicación.

Interpolación de texto

A forma máis básica para enlazar información do compoñente é a interpolación de texto usando a sintaxe **{{ }}**.

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

```
createApp({
  data() {
    return {
      message: 'Ola mundo!',
    };
  },
}).mount('#app');
```

A etiqueta **{{ message }}** será substituída polo valor da propiedade **message** da instancia do compoñente. Ademais, será actualizado cada vez que a propiedade **message** cambie (pode probarse a cambiar o valor da variable dende a consola de desenvolvemento despois de instalar a extensión para Vue).

Exercicio:

1. Exercicio titorial - Declarative Rendering.

HTML puro

A interpolación de texto interpreta a información como texto simple por motivos de seguridade. Se se quere inserir contido HTML haberá que usar a directiva v-html.

As directivas son atributos HTML que empezan por **v-,** para indicar que son atributos especiais proporcionados por Vue, e aplican un comportamento reactivo especial aos compoñentes DOM.

A directiva **v-html** permite sincronizar o **innerHTML** do elemento do DOM coa propiedade do compoñente interpretada como HTML plano:

```
<div id="app">
    <p>Using text interpolation: {{ rawHtml }}</p>
    <p>Using v-html directive: <span v-html="rawHtml"></span></p>
</div>
```

```
createApp({
  data() {
    return {
      rawHtml: '<b>This should be bold.</b>',
    };
  },
}).mount('#app');
```

NOTA: renderizar HTML directamente na páxina web pode ser perigoso, polo que se recomenda usar esta directiva só con contido no que se poida confiar e nunca contido proporcionado polas persoas usuarias.

Enlazar atributos

A interpolación de texto non se pode usar dentro de atributos HTML. En lugar disto, hai que usar a directiva v-bind.

A directiva **v-bind** permite sincronizar un atributo HTML con unha propiedade dun compoñente. No seguinte exemplo sincronízase o atributo id coa propiedade **dynamicId** do compoñente.

```
<div id="app">
  <div v-bind:id="dynamicId"></div>
</div>
```

```
createApp({
  data() {
    return {
      dynamicId: 'atributoDinamico',
    };
  },
}).mount('#app');
```

Se o valor enlazado é **null** ou **undefined**, o atributo será eliminado do DOM.

Dado que **v-bind** é moi usada, ten unha forma abreviada:

```
<div :id="dynamicId"></div>
```

Dende a versión 3.4 de Vue, se o atributo ten o mesmo nome que o valor JavaScript co que se enlaza, a sintaxe pode ser aínda máis abreviada:

```
<!-- same as :id="id" -->
<div :id></div>

<!-- this also works -->
<div v-bind:id></div>
```

```
import { createApp } from 'vue';
createApp({
  data() {
    return {
      id: 1,
    };
  },
}).mount('#app');
```

```
<!-- renderización -->
<div id="1"></div>
```

Especial mención precisan os atributos booleanos, que son aqueles que indican un valor true/false mediante a súa presencia/ausencia, como por exemplo **checked** e **disabled**. Nestes casos **v-bind** funciona un pouco diferente. No seguinte exemplo, o atributo **disabled** será incluído se **isButtonDisabled** ten un valor **true.** Tamén se incluirá se o valor é a cadea baleira, para manter a consistencia con **<button disabled="">**. Calquera valor falso fará que o atributo se omita.

```
<button :disabled="isButtonDisabled">Button</button>
```

Tamén é posible declarar un obxecto JavaScript con varias propiedades que fagan referencia a atributos e enlazarse utilizando un único **v-bind**:

```
<div id="app"><div v-bind="objectOfAttrs"></div></div>
```

```
import { createApp } from 'vue';
createApp({
  data() {
    return {
      objectOfAttrs: {
        id: 'container',
        class: 'wrapper',
      },
    };
  },
}).mount('#app');
```

```
<!-- renderización -->
<div id="container" class="wrapper"></div>
```

Exercicios:

1. Exercicio titorial - Attribute Bindings.

Usar expresións JavaScript

Ata agora usáronse propiedades simples nas templates. Sen embargo, Vue permite usar calquera expresión válida en JavaScript nas seguintes posicións dunha template:

- Dentro dunha interpolación {{ }}.
- No valor dun atributo de calquera directiva.

Exemplos:

```
{{ number + 1 }}
{{ ok ? 'YES' : 'NO' }}
{{ message.split('').reverse().join('') }}
<div :id="`list-${id}`"></div>
```

Hai que ter en conta que só se pode usar unha expresión de cada vez. Unha expresión é un anaco de código que pode ser avaliado a un valor. Para saber se algo se pode usar como unha expresión simplemente se comprobará se pode ser usado despois de **return**:

```
<!-- this is a statement, not an expression: -->
{{ var a = 1 }}

<!-- flow control won't work either, use ternary expressions -->
{{ if (ok) { return message } }}
```

Directivas

As directivas son atributos especiais co prefixo **v-**. Vue proporciona varias directivas, como por exemplo v-html, v-bind, etc.

Os valores dos atributos das directivas son habitualmente expresións simples de JavaScript. A función da directiva é actualizar o DOM cando o valor da expresión cambia. No seguinte exemplo, a directiva v-if insertará ou eliminará o parágrafo en función do valor da expresión **seen**:

```
<p v-if="seen">Now you see me</p>
```

Algunhas directivas reciben un **argumento** que se especifica utilizando ":" despois do nome da directiva. Por exemplo, a directiva **v-bind** utilízase para actualizar un atributo HTML de forma reactiva. No seguinte exemplo **href** é o argumento da directiva **v-bind**, que especifica que se sincronice o atributo **href** co valor da expresión JavaScript **url**:

```
<a v-bind:href="url"> ... </a>

<!-- shorthand -->
<a :href="url"> ... </a>
```

Utilizando corchetes é posible usar expresións JavaScript nun argumento dunha directiva, o que fará que sexan avaliados dinamicamente. No seguinte exemplo, **attributeName** será avaliado dinamicamente como unha expresión JavaScript e o seu valor final será usado como argumento. Por exemplo, nun compoñente onde **attributeName** teña o valor **"href"** o código xerado será equivalente a **v-bind:href="url"**:

```
<a v-bind:[attributeName]="url"> ... </a>

<!-- shorthand -->
<a :[attributeName]="url"> ... </a>
```

Reactividade

Cando se usa Options API, o estado reactivo dun compoñente declárase utilizando a función **data(),** que devolve un obxecto JavaScript.

Vue invoca automaticamente esta función ao crear unha nova instancia do compoñente e garda o obxecto devolto no seu sistema de reactividade.

As propiedades de alto nivel do obxecto devolto por **data()** están accesibles nos métodos do compoñente a través da palabra chave **this**. Aínda que é posible engadir novas propiedades a **this,** sen incluílas en **data**, as propiedades así engadidas non provocarán actualizacións reactivas. Se durante a creación do compoñente non se coñece o valor dunha propiedade recoméndase inicializala a **null** ou **undefined** para asegurarse que a propiedade formará parte do sistema de reactividade.

Debe evitarse poñer nomes a estas propiedades que empecen por $ ou _, xa que estes prefixos son usados internamente por Vue e poden dar lugar a conflitos.

Declaración de métodos

Para engadir métodos a unha instancia dun compoñente úsase a opción **methods**, que é un obxecto que conterá os métodos desexados.

```
createApp({
  data() {
    return {
      count: 0
    }
  },
  methods: {
    increment() {
      this.count++
    }
  }
}).mount('#app');
```

Vue enlaza automaticamente o valor **this** nos métodos para que faga referencia á instancia do compoñente.

NOTA: debe **evitarse o uso de funcións frecha na definición de métodos**, xa que neste caso a palabra chave **this** non se referirá á instancia do compoñente.

Ao igual que as propiedades da instancia do compoñente, os métodos son accesibles dende o template HTML, donde son usados maioritariamente como manexadores de eventos:

```
<div id="app">
  <button @click="increment">{{ count }}</button>
</div>
```

```
createApp({
  data() {
    return {
      count: 0,
    };
  },
  methods: {
    increment() {
      this.count++;
    },
  },
}).mount('#app');
```

No exemplo anterior, o método **increment** será invocado cando se pulse o botón.

Reactividade profunda

En Vue, o estado é moi reactivo de forma predeterminada. Isto significa que se detectarán cambios incluso cando se modifiquen obxectos aniñados ou arrays:

```
<div id="app">
  <p>{{obj.nested.count}}</p>
  <p>{{obj.arr}}</p>
  <button @click="cambiar">Cambiar valores</button>
</div>
```

```
createApp({
  data() {
    return {
      obj: {
        nested: { count: 0 },
        arr: ["foo", "bar"],
      },
    };
  },
  methods: {
    cambiar() {
      this.obj.nested.count++;
      this.obj.arr.push(this.obj.nested.count);
    },
  },
}).mount("#app");
```

Exercicio:

1. Crea unha páxina web onde se mostre:
    a. o teu nome
    b. o ano actual
    c. usando unha expresión, suma 5 ao ano actual.
    d. unha imaxe (pode utilizarse o atributo src ou mostrala como HTML puro)
    e. unha caixa de texto que conteña como texto o teu nome.
    f. un contador inicializado a 0
    g. Dous botóns: un para incrementar nunha unidade o contador e outro para decrementalo.

    A información debe estar almacenada nun compoñente Vue e dende HTML debe accederse a ela. Con respecto á imaxe, a ruta está almacenada en Vue.

Xestión de eventos

Pode usarse a directiva **v-on:**, abreviada mediante o símbolo **@,** para escoitar eventos do DOM e executar código JavaScript cando suceden. A continuación de **v-on:** irá o evento a escoitar, que pode ser calquera evento do DOM. Pode consultarse o Listado completo de eventos ou o listado específico de eventos de Element.

A sintaxe é **v-on:<evento>="handler",** ou de forma abreviada **@<evento>="handler".**

O manexador de eventos pode ser:

- **un manexador en liña**: código en liña JavaScript a executar cando suceda o evento. Normalmente úsanse para casos simples:

```
<div id="app">
  <button @click="count++">Add 1</button>
  <p>Count is: {{ count }}</p>
</div>
```

```
createApp({
  data() {
    return {
      count: 0,
    };
  },
}).mount('#app');
```

  Observar no caso anterior, que o valor do contador se actualiza automaticamente na vista.

  NOTA: aínda que é posible modificar o contador dende HTML, non é recomendable. En HTML só debería estar o código relativo á vista. A lóxica da aplicación debería estar en JavaScript.

- **un método**: o nome dun método definido no compoñente.

```
<div id="app">
  <button @click="greet">Greet</button>
  <button @click="greet2">Greet2</button>
</div>
```

```
createApp({
  data() {
    return {
      message: 'Ola mundo',
    };
  },
  methods: {
    greet(event) {
      console.log(`${this.message}!!!`);
      // `event` is the native DOM event
      if (event) {
        console.log(event.target.tagName);
      }
    },
    greet2() {
      console.log('Ola mundo');
    },
  },
}).mount('#app');
```

Observar que o método recibe automaticamente como parámetro o **evento do DOM,** polo que é posible acceder ao elemento que lanzou o evento a través de **event.target.**

Observar tamén que no caso do segundo botón pode usarse como manexador de eventos greet2 ou greet2(), ambos funcionan. Vue comproba o valor da directiva **v-on** e fará a traducción correcta, ou ben traducirá por un manexador de eventos ou por un manexador en liña. É dicir, pódese usar calquera dos dous, ao contrario que en JavaScript que había que especificar a definición da función e non invocala.

Tamén sería posible invocar un método e pasarlle parámetros personalizados ao especificar o manexador de eventos:

```
<div id="app">
  <p>{{ message}}</p>
  <button @click="say('hello')">
    Say hello
  </button>
  <button @click="say('bye')">
    Say bye
  </button>
</div>
```

```
createApp({
  data() {
    return {
      message: "Ola",
    };
  },
  methods: {
    say(message) {
      this.message = message;
    },
  },
}).mount("#app");
```

Se no caso anterior fose necesario acceder ao evento dende o manexador de eventos, podería pasarse unha variable especial **$event**:

```
<div id="app">
   <p>{{ message}}</p>
   <button @click="say('hello', $event)">Say hello</button>
   <button @click="say('bye', $event)">Say bye</button>
</div>
```

```
createApp({
  data() {
    return {
      message: "Ola",
    };
  },
  methods: {
    say(message, evento) {
      this.message = message;
      console.log(evento.target);
    },
  },
}).mount("#app");
```

Exercicio:

1. Crea unha páxina web que conteña unha caixa de texto e un parágrafo sincronizados. É dicir, o parágrafo debe mostrar o mesmo contido que a caixa de texto e debe actualizarse a medida que se engaden novas letras. Podes usar o evento input.

Modificadores de eventos

É frecuente que dentro dun manexador de eventos se teñan que invocar os métodos **event.preventDefault()** ou **event.stopPropagation()**. Aínda que se poden escribir estas instrucións JavaScript, Vue proporciona modificadores de eventos para a directiva **v-on,** que se engaden como sufixos do evento.

Algúns destes modificadores son:

- **.stop:** detén a propagación do evento.

```
<!-- the click event's propagation will be stopped -->
<a @click.stop="doThis"></a>
```

- **.prevent:** non se realizará a acción por defecto (recarga da páxina).

```
<!-- the submit event will no longer reload the page -->
<form @submit.prevent="onSubmit"></form>
```

```
<!-- just the modifier -->
<form @submit.prevent></form>
```

- **.self**: só se executará o manexador de eventos se event.target é o propio elemento, é dicir, se non se produciu o evento nalgún elemento descendente.

```
<!-- only trigger handler if event.target is the element itself -->
<!-- i.e. not from a child element -->
<div @click.self="doThat">...</div>
```

Exemplo:

```
<div id="app">
  <button @click="increment">{{count}}</button>
  <form v-on:submit.prevent="submitForm">
    <input type="text">
    <button>Enviar</button>
  </form>
</div>
```

```
createApp({
  data() {
    return { count: 0 };
  },
  methods: {
    submitForm(e) {
      //e.preventDefault()
      console.log('Enviado');
    },
    increment() {
      this.count++;
    },
  },
}).mount('#app');
```

Os modificadores poden encadearse:

```
<a @click.stop.prevent="doThat"></a>
```

NOTA: a orde en que se encadean os modificadores importa, pois o código é xerado na mesma orde. Así, **@click.prevent.self** evitará a acción por defecto no elemento e os seus descendentes, mentres que **@click.self.prevent** só evitará a acción por defecto no propio elemento.

Tamén existen **modificadores para os botóns** do rato: **.left, .right, .middle**. Estes modificadores restrinxen a execución do manexador do evento ao uso dun botón específico do rato:

```
<div id="app">
  <p>{{count}}</p>
  <p>
    <button @click="increment">Incrementar</button>
    <button @click.middle="decrement">Decrementar</button>
  </p>
</div>
```

```
createApp({
  data() {
    return {
      count: 0,
    };
  },
  methods: {
    increment() {
      this.count++;
    },
    decrement() {
      this.count--;
    },
  },
}).mount('#app');
```

Cando se están escoitando eventos de teclado, habitualmente compróbase que se pulsase unha tecla específica. Vue tamén proporciona modificadores de teclas para a directiva v-on ou @.

```
<!-- only call `submit` when the `key` is `Enter` -->
<input @keyup.enter="submit" />
```

Poderá usarse calquera nome de chave válida de KeyboardEvent.key traducida usando a nomenclatura kebab-case (une as palabras con guións):

```
<!-- só se executará se $event.key é igual a ‘PageDown’ -->
<input @keyup.page-down="onPageDown" />
```

Vue proporciona alias para as teclas máis usadas: .enter, .tab, .delete (captures both "Delete" and "Backspace" keys), .esc, .space, .up, .down, .left, .right

Exemplo:

```
<div id="app">
  <input type="text" @keyup.enter="showText">
  <p>{{texto}}</p>
</div>
```

```
createApp({
  data() {
    return {
      texto: '',
    };
  },
  methods: {
    showText(e) {
      this.texto = e.target.value;
    },
  },
}).mount('#app');
```

Exercicio:

1. Crea unha páxina web cos seguintes elementos:
    a. Un botón que mostre unha mensaxe por consola ao ser pulsado co botón do medio.
    b. Unha caixa de texto onde ao pulsar calquera tecla se mostre o texto actual da caixa nun parágrafo da mesma páxina HTML.
    Observar que se se usa o evento keydown prodúcese un atraso nun carácter, pois o evento lánzase ao pulsar a tecla e o valor da caixa de texto actualízase cando se solta a tecla (o evento keydown xa rematou).
    c. Unha caixa de texto tal que ao pulsar a tecla ENTER se mostre o seu contido nun novo parágrafo da páxina HTML. Ademais, fai que o contido da caixa de texto se borre despois de pulsar ENTER.

Directiva v-once

Permite renderizar o elemento/compoñente só unha vez e obviar futuras actualizacións. En futuras re-renderizacións, o elemento/compoñente e todos os seus descendentes serán tratados como contido estático e non se modificarán.

Exemplo:

```
<div id="app">
  <p v-once>Contador inicial: {{count}}</p>
  <p>{{count}}</p>
  <p>
    <button @click="increment">Incrementar</button>
    <button @click="decrement">Decrementar</button>
  </p>
</div>
```

```
createApp({
  data() {
    return {
      count: 10,
    };
  },
  methods: {
    increment() {
      this.count++;
    },
    decrement() {
      this.count--;
    },
  },
}).mount('#app');
```

Formularios

Cando se usan formularios en HTML é necesario sincronizar o estado dun elemento do formulario co estado correspondente en JavaScript, de tal forma que se un cambia, o outro debe actualizarse. Isto pode facerse utilizando atributos enlazados e manexadores de eventos, aínda que pode resultar un pouco complicado:

```
<input
  :value="text"
  @input="event => text = event.target.value">
```

A directiva **v-model** permite obter o mesmo comportamento simplificando o código:

```
<input v-model="text">
```

A directiva **v-model** tamén pode usarse con diferentes elementos do formulario, como áreas de texto e elementos <select>. Vue traduce automaticamente o código desta directiva adaptándoo ás propiedades do DOM e aos eventos apropiados ao compoñente onde se usa:

- <input> de tipo texto e <textarea> usan a propiedade **value** e o evento **input**.
- <input type="checkbox"> e <input type="radio"> usan a propiedade **checked** e o evento **change**.
- <select> usa a propiedade **value** e o evento **change**.

NOTA: v-model ignorará os valores dos atributos **value, checked** ou **selected** dos elementos dun formulario engadidos en HTML. É dicir, terá prioridade a o estado definido en JavaScript como propiedade do obxecto devolto por **data().**

Exemplo caixa de texto:

```
<p>Message is: {{ message }}</p>
<input v-model="message" placeholder="edit me" />
```

Exemplo área de texto:

```
<span>Multiline message is:</span>
<p style="white-space: pre-line;">{{ message }}</p>
<textarea v-model="message" placeholder="add multiple lines"></textarea>
```

Exemplo con un único checkbox:

```
<input type="checkbox" id="checkbox" v-model="checked" />
<label for="checkbox">{{ checked }}</label>
```

Tamén poden enlazarse múltiples checkboxes utilizando un array. Ver exemplo.

Exemplo radio button:

```
<div>Picked: {{ picked }}</div>

<input type="radio" id="one" value="One" v-model="picked" />
<label for="one">One</label>

<input type="radio" id="two" value="Two" v-model="picked" />
<label for="two">Two</label>
```

Exemplo select:

```
<div>Selected: {{ selected }}</div>

<select v-model="selected">
  <option disabled value="">Please select one</option>
  <option>A</option>
  <option>B</option>
  <option>C</option>
</select>
```

NOTA: se o valor inicial da expresión v-model non coincide coas opcións do select, o elemento mostrarase nun estado “unselected”. Recoméndase proporcionar unha opción deshabilitada cun valor baleiro, como no exemplo anterior.

A directiva **v-model** ofrece tres modificadores:

- **v-model.lazy**: por defecto v-model sincroniza os datos por cada evento input. Pode engadirse o modificador lazy para que a sincronización se faga despois de cada evento change.

  ```
  <input v-model.lazy="msg" />
  ```

- **v-model.number**: permite converter automaticamente a entrada a un número. Este modificador aplícase automaticamente se o campo de entrada é **type=”number”.**

  ```
  <input v-model.number="age" />
  ```

  Se o valor non pode ser parseado con parseFloat(), usarase por defecto o valor orixinal.

- **v-model.trim**: elimina os espazos en branco do campo do formulario:

  ```
  <input v-model.trim="msg" />
  ```

Exercicio:

1. Crea un formulario con, polo menos, unha caixa de texto, un área de texto, múltiples checkboxes, radiobuttons e select. Fai que o valor seleccionado en cada un dos campos se mostre en HTML.

   Engade un botón para resetear o formulario, de forma que os valores escritos/seleccionados se borren.

Propiedades calculadas

Aínda que se poden usar expresións JavaScript nunha interpolación {{ expresión }} non se recomenda facelo, xa que están pensadas para cálculos simples. Utilizar expresións complexas nunha interpolación fai que a aplicación sexa difícil de manter.

Cando a expresión a calcular non é simple e se ademais inclúe información reactiva, a recomendación é usar propiedades calculadas.

As **propiedades calculadas** permiten mostrar valores calculados a partir das propiedades dun compoñente Vue.

Características das propiedades calculadas:

- son como propiedades normais, excepto que dependen doutra propiedade.
- son escritas como métodos, aínda que non aceptan argumentos.
- poden usarse dende HTML como se fosen unha propiedade normal (non se engade () como nas invocacións de métodos).
- son actualizadas automaticamente cando a súa dependencia cambia.

O seguinte exemplo crea unha propiedade calculada chamada **publishedBooksMessage**:

```
<div id="app">
    <span>Libros publicados: {{ author.books.length > 0 ? 'Yes' : 'No' }}</span>
    <p>Libros publicados: {{publishedBooksMessage}}</p>
</div>
```

```
createApp({
  data() {
    return {
      author: {
        name: 'John Doe',
        books: [
          'Vue 2 - Advanced Guide',
          'Vue 3 - Basic Guide',
          'Vue 4 - The Mystery',
        ],
      },
    };
  },
  computed: {
    publishedBooksMessage() {
      return this.author.books.length > 0 ? 'Yes' : 'No';
    },
  },
}).mount('#app');
```

En lugar de declarar unha propiedade calculada, podería obterse o mesmo resultado invocando un método nunha expresión:

```
<div id="app">
  <p>Libros publicados: {{calculateBooksMessage() }}</p>
</div>
```

```
createApp({
  ….
  methods: {
    calculateBooksMessage() {
      return this.author.books.length > 0 ? 'Yes' : 'No';
    },
  },
}).mount('#app');
```

O resultado sería o mesmo, sen embargo, a diferencia está en que as propiedades calculadas están **cacheadas** e só se recalcularán cando cambien as súas dependencias reactivas. Vue sabe que **publishedBooksMessage** depende de **this.author.books**, polo que a propiedade calculada só se recalcula cando this.author.books cambia. É dicir, se author.books non cambia, múltiples accesos a **publishedBooksMessage** devolverán o resultado previo sen necesidade de executar a función outra vez. Polo contrario, a invocación ao método sempre executaría a función.

O seguinte exemplo mostra unha propiedade calculada que nunca se actualiza porque non ten ningunha dependencia reactiva:

```
computed: {
  now() {
    return Date.now()
  }
}
```

A continuación descríbense varios casos de uso nos que as propiedades calculadas son útiles:

- Facer cálculos, como no exemplo anterior de **publishedBooksMessage.**
- Personalizar a etiqueta asociada a un checkbox. Imaxinar un formulario onde se quere marcar se un elemento é importante ou non, de tal forma que a etiqueta asociada ao checkbox se actualice en función de se está ou non seleccionado (ver exemplo). O exemplo anterior utiliza os valores true/false asociados ao checkbox cando a maioría de persoas entendería mellor o formulario se se utilizase o texto “si/non”. Pode reescribirse o exemplo anterior utilizando propiedades calculadas, demostrando a súa utilidade.
- Filtrar datos dunha propiedade de tipo array. Exemplo.

As propiedades calculadas só deben usarse para realizar un cálculo sen efectos secundarios. É dicir, **non deben mutar outro estado, nin facer peticións asíncronas, nin modificar o DOM.**

Exercicios:

1. Exercicio titorial.

Watchers

As propiedades calculadas permiten calcular valores derivados sen que se produzan efectos secundarios. Sen embargo, hai casos nos que se necesita provocar “efectos” como por exemplo cambiar o estado, facer unha petición asíncrona ou modificar o DOM. Neste caso recomendase usar un watcher.

Un watcher permite observar unha propiedade declarada en **data** ou **computed** e executar unha función cando cambie. É dicir, os watchers permiten observar e reaccionar ante cambios na instancia do compoñente Vue.

Con un watcher non só é posible observar o valor dunha variable e realizar certas accións, tamén se ten acceso ao valor antigo e ao novo da variable.

Un **watcher**:

- é un método que permite observar unha propiedade do mesmo nome.
- execútase cada vez que o valor da propiedade cambia.
- utilízase para executar unha acción que provoque “efectos secundarios” cando a propiedade cambia.

Exemplo:

```
<div id="app">
  <p>
    Ask a yes/no question:
    <input v-model="question" />
  </p>
  <p>{{ answer }}</p>
</div>
```

```
createApp({
  data() {
    return {
      question: '',
      answer: 'Questions usually contain a question mark. ;-)',
    };
  },
  watch: {
    // whenever question changes, this function will run
    question(newQuestion, oldQuestion) {
      if (newQuestion.includes('?')) {
        this.getAnswer();
      }
    },
  },
```

```
  methods: {
    async getAnswer() {
      this.answer = 'Thinking...';
      try {
        const res = await fetch('https://yesno.wtf/api');
        this.answer = (await res.json()).answer;
      } catch (error) {
        this.answer = 'Error! Could not reach the API. ' + error;
      }
    },
  },
}).mount('#app');
```

Observar que a función watch recibe como parámetros o valor actual e antigo da propiedade modificada.

Exercicios:

1. Exercicio titorial.
2. Crea unha páxina web con **dúas** caixas de texto para introducir números. A páxina web debe facer a suma dos dous números introducidos e mostrar o resultado. Cada vez que se modifique unha das caixas de texto, debe actualizarse e mostrarse o resultado da suma dos dous números. Ademais, unha vez obtido o resultado da suma, debe mostrarse tamén na páxina web información indicando se o resultado é un número par ou impar.
3. Crea unha páxina web con dúas caixas de texto que permitan converter valores de kilómetros a metros automaticamente. Cada vez que se escriba en calquera das dúas caixas de texto, mostraranse automaticamente a unidade equivalente na outra caixa.

Clases e estilos

Nunha aplicación web é frecuente ter que manipular a lista de clases dun elemento HTML e os estilos en liña aplicados. Dado que **class** e **style** son atributos HTML, pode usarse a directiva **v-bind** para asignarlles un valor dinamicamente, igual que a calquera dos outros atributos. Sen embargo, xerar estes atributos utilizando a concatenación de strings pode ser tedioso e propenso a erros. Por esta razón, Vue facilita e simplifica o uso de **v-bind** cos atributos **class** e **style**, permitindo utilizar cadeas, obxectos ou arrays.

A directiva **:class** (abreviatura de v-bind:class) permite enlazar dinamicamente o atributo class con JavaScript.

O valor da directiva :class pode ser unha **variable**:

```
<style>
  .red {
    color: red;
  }
</style>
```

```
<div id="app">
    <div :class="className">The class is set with Vue</div>
</div>
```

```
data() {
  return {
    className: 'red',
  };
},
```

RESULTADO
** The class is set with Vue **

O valor da directiva :class tamén pode ser un **obxecto** con propiedades que fagan referencia a clases CSS. Vue só engadirá as clases das propiedades que sexan avaliadas a **true**:

```
<style>
  .red {
    color: red;
  }
</style>
```

```
<div id="app">
  <!-- A presencia da clase "red" vén determinada polo valor de "isRed" -->
  <p :class="{red: isRed}">
    Texto
  </p>
</div>
```

```
data() {
  return {
    isRed: true,
  };
},
```

RESULTADO
** Texto **

O obxecto pasado á directiva **:class** pode ter máis dunha propiedade. Ademais, a directiva **:class** pode coexistir co atributo **class**. Vue fusiona as clases do atributo e da directiva. Exemplo:

```
<div id="app">

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

</div>
```

```
data() {
  return {
    isActive: true,
    hasError: false,
  };
},
```

RESULTADO

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

No exemplo anterior, cando **isActive** ou **hasError** cambien, a lista de clases actualizarase en consecuencia.

Non é necesario que o obxecto estea escrito en liña. O seguinte exemplo producirá o mesmo resultado que o anterior e resulta máis fácil de ler:

```
  <div id="app">

    <div class="static" :class="classObject">
    </div>

  </div>
```

```
    data() {
      return {
        classObject: {
          active: true,
          'text-danger': false,
        },
      };
    },
```

RESULTADO

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

Tamén é posible enlazar unha **propiedade calculada** que devolva un obxecto. Este é un patrón moi útil en Vue:

```
<div :class="classObject"></div>
```

```
data() {
  return {
    isActive: true,
    error: null,
  };
},
computed: {
  classObject() {
    return {
      active: this.isActive && !this.error,
      'text-danger': this.error && this.error.type === 'fatal',
    };
  },
},
```

RESULTADO
** **

Á directiva **:class** tamén se lle pode asignar un array, o que permitirá engadir múltiples clases a un elemento:

```
<div :class="[activeClass, errorClass]"></div>
```

```
data() {
  return {
    activeClass: 'active',
    errorClass: 'text-danger',
  };
},
```

RESULTADO
** **

Tamén é posible usar a sintaxe de obxectos dentro dun array:

```
<div :class="[errorClass, {active: isActive}]">Proba</div>
```

```
data() {
  return {
    errorClass: 'text-danger',
    isActive: true,
  };
},
```

RESULTADO
** **

A directiva **:style** permite establecer estilos en liña. Como valor da directiva :**style** debe proporcionarse un obxecto JavaScript coas propiedades CSS e os valores a asignar ao elemento:

```
<div :style="{ color: activeColor, fontSize: fontSize + 'px' }"></div>
```

```
data() {
  return {
    activeColor: 'red',
    fontSize: 30,
  };
},
```

RESULTADO
** **

Observar que as propiedades do obxecto estilo están escritas en notación camelCase, que é a recomendada neste caso. No caso de que se usase a notación kebab-case, deberían poñerse entre comiñas: { 'font-size': fontSize + 'px' }. Isto é así porque non se pode acceder a unha propiedade dun obxecto JavaScript que teñan un guión usando a notación con punto, habería que usar os corchetes (mdn).

Para utilizar a directiva :style, a documentación de Vue recomenda enlazar directamente cun obxecto, pois proporciona claridade no código:

```
<div :style="styleObject"></div>
```

```
data() {
  return {
    styleObject: {
      color: 'red',
      fontSize: '13px',
    },
  };
},
```

RESULTADO
** **

Aínda que se poden usar estilos en liña dende Vue, non é recomendable abusar deles, xa que hai que recordar que son os que teñen máis especificidade, o que implica que terán prioridade sobre o resto de estilos aplicados.

Exercicios:

1. Crea unha páxina web que teña dous parágrafos:

    a. O primeiro parágrafo ten por defecto unha clase CSS (configúraa coas propiedades que queiras). Configura este parágrafo para que ao pulsar sobre el se quite e se engada a clase CSS alternativamente.

    b. O segundo parágrafo ten cor de letra verde, por defecto. Configúrao para que ao pulsar sobre el a cor de letra alterne entre o verde e o vermello.

2. Crea unha páxina web que teña un taboleiro con **dúas** celas. Define unha clase CSS que vai ser usada para aplicar estilos ás celas. Por exemplo, crea unha clase chamada "**activa**" e configúraa cunha cor de fondo e un borde diferentes dos establecidos por defecto. Fai que ao pulsar sobre cada cela se alterne o estilo, é dicir, que se engada a clase CSS **activa** se non estaba asignada e viceversa.

    Fai unha segunda versión do mesmo exercicio, pero esta vez só unha das celas pode ter a clase CSS "activa". Configura a páxina para que ao pulsar sobre a cela que non ten a clase "activa" se lle engada e se elimine da outra cela.

3. Crea unha páxina web coas seguintes condicións:

    a. Define dúas clases CSS (claseA e claseB) cos estilos que queiras para aplicar a un parágrafo.
    b. Engade á túa páxina web unha caixa de texto e un parágrafo. Configura o código para que cando unha persoa usuaria escriba "claseA" na caixa de texto, se aplique esta clase ao parágrafo e cando se escriba "claseB" na caixa de texto se aplique esta clase ao parágrafo.
    c. Engade as seguintes clases CSS á túa páxina web:

```
.hidden {
  display: none;
}

.visible {
  display: block;
}
```

    d. Engade un botón á páxina de tal forma que, usando as clases anteriores, alterne a visibilidade do parágrafo usado anteriormente. É dicir, inicialmente o parágrafo está visible e se se pulsa o botón, ocultarase. Se se volve a pulsar o botón, o parágrafo farase visible.
    e. Engade á páxina web unha nova caixa de texto e un novo parágrafo. Usando estilos en liña, fai que cando unha persoa escriba unha cor de fondo na caixa de texto, o parágrafo cambie a súa cor de fondo pola indicada na caixa de texto.

Renderizado condicional

A directiva **v-if** úsase para mostrar un bloque de código en función dunha expresión booleana. O código só se mostrará se a expresión é true.

```
<h1 v-if="awesome">Vue is awesome!</h1>
```

A directiva **v-else** úsase para especificar o bloque **else** da directiva **v-if**. Para que funcione ten que colocarse xusto debaixo de **v-if**, non pode haber ningún bloque de código polo medio. Exemplo:

```
<h1 v-if="awesome">Vue is awesome!</h1>
<h1 v-else>Oh no 😢</h1>
```

A directiva **v-else-if**, como o seu nome indica, funciona como o bloque “else if” da directiva **v-if** e pode encadearse múltiples veces:

```
<div v-if="type === 'A'">
  A
</div>
<div v-else-if="type === 'B'">
  B
</div>
<div v-else-if="type === 'C'">
  C
</div>
<div v-else>
  Not A/B/C
</div>
```

Igual que **v-else**, o elemento **v-else-if** debe estar inmediatamente despois da directiva **v-if** ou **v-else-if**.

Dado que **v-if** é unha directiva debe utilizarse nun elemento HTML. No caso de querer asociala a un grupo de elementos, poden agruparse nun elemento <template>, que funciona como un “contedor invisible” (o resultado final non inclúe o elemento <template>):

```
<template v-if="ok">
  <h1>Title</h1>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
</template>
```

Outra opción para mostrar contido de forma condicional é usar a directiva **v-show**, que modifica a visibilidade do elemento en base ao valor true/false da expresión**:**

```
<h1 v-show="ok">Hello!</h1>
```

A diferencia de v-if, coa directiva **v-show** o elemento sempre estará no DOM, dado que **v-show** só alterna a propiedade CSS **display** do elemento. Polo tanto, a directiva **v-show** é moito máis simple, xa que o elemento sempre está no DOM. O que se cambia é a súa propiedade display de CSS.

En xeral, a directiva **v-if** ten maior custo computacional cando cambia o valor da condición, xa que implica a modificación do DOM engadindo ou eliminando elementos. Recoméndase usar **v-show** cando un elemento se mostra/oculta frecuentemente. Pola contra, é mellor usar **v-if** se a condición non é frecuente que cambie.

Exercicios:

1. Exercicio titorial.
2. Crea dende JavaScript un array con tarefas, que poden ser inicialmente un array de cadeas de texto. Para practicar o renderizado condicional crea unha páxina web que mostre un parágrafo “Non hai tarefas pendentes” se o array coa lista de tarefas está baleiro ou que mostre “Hai X tarefas pendentes”, sendo X o número de tarefas no array.

Renderizado de listas

A directiva **v-for** pode usarse para iterar sobre elementos dun array. Esta directiva utiliza a sintaxe **item in items,** onde **items** é o array e **item** é un alias para o elemento do array sobre o que se vai iterar. Exemplo:

```
  <ul>
    <li v-for="item in items">{{ item.message }}</li>
  </ul>
```

```
data() {
  return {
    items: [{ message: 'Foo' }, { message: 'Bar' }]
  }
}
```

Tamén é posible acceder ao índice do elemento actual coa sintaxe **(item, index)**. Exemplo:

```
<li v-for="(item, index) in items">
  {{ parentMessage }} - {{ index }} - {{ item.message }}
</li>
```

```
data() {
  return {
    parentMessage: 'Parent',
    items: [{ message: 'Foo' }, { message: 'Bar' }]
  }
}
```

É posible usar a desestruturación coa directiva **v-for**:

```
<ul>
  <li v-for="{ message } in items">{{ message }}</li>
</ul>
```

```
<ul>
  <li v-for="({ message }, index) in items">{{ message }} {{ index }}</li>
</ul>
```

NOTA: pode usarse o delimitador **of** en lugar de **in**, o que proporciona unha sintaxe máis parecida aos iteradores de JavaScript:

```
<div v-for="item of items"></div>
```

Pode usarse **v-for** para **iterar sobre as propiedades dun obxecto**. A orde de iteración baséase no resultado de invocar Object.keys(). Exemplo:

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

```
data() {
  return {
    myObject: {
      title: 'How to do lists in Vue',
      author: 'Jane Doe',
      publishedAt: '2016-04-10'
    }
  }
}
```

Pode utilizarse **v-for** para iterar sobre un rango de valores pasándolle un número. Isto fará que o bucle se repita varias veces, utilizando o rango de **1..n**. Observar que a iteración empeza en 1 e non en 0. Exemplo:

```
<span v-for="n in 10">{{ n }}</span>
```

NOTA: Non é recomendable usar no mesmo elemento as directivas **v-if** e **v-for**, debido a que v-if ten prioridade e será avaliada en primeiro lugar. Revisar as regras de estilo para máis detalles.

- Para filtrar elementos nunha lista (p.ex.: v-for="user in users" v-if="user.isActive") recoméndase substituír **users** por unha propiedade calculada que devolva o array filtrado (p.ex.: **activeUsers**).
- Para evitar mostrar unha lista que debe estar oculta (p.ex.: v-for="user in users" v-if="showUsers") recoméndase mover o v-if ao contedor ascendente (ul ou ol).

Manter o estado con unha chave

Cando se engaden ou eliminan elementos nun array visualizado coa directiva **v-for**, Vue actualiza o DOM de forma óptima en canto a rendemento, reutilizando elementos do DOM en lugar de crear novos elementos e eliminar antigos.

Dado que os elementos do DOM non teñen un identificador único, esta reutilización pode provocar erros, facendo que se misturen propiedades de diferentes elementos do array. Por exemplo, cando hai dous elementos nunha lista e se borra o primeiro, Vue non volve a crear toda a lista, senón que en realidade move o contido do segundo ao primeiro elemento da lista.

O seguinte exemplo demostra un bug que existe coa directiva v-for. Observar o que pasa cando se elimina o primeiro elemento da lista e hai algo escrito na súa caixa de texto.

```
<div id="app">
  <input type="text" v-model="novaTarefa" />
  <button @click="engadirTarefa">Engadir tarefa</button>
  <p v-if="tarefas.length === 0">Aínda non hai tarefas</p>
  <ul v-else>
    <li v-for="(tarefa, index) in tarefas">
      <p>
        {{tarefa}} - {{index}}
        <button @click="borrarTarefa(index)">X</button>
      </p>
      <input type="text" @click.stop />
    </li>
  </ul>
</div>
```

```
    data() {
      return {
        novaTarefa: '',
        tarefas: [],
      };
    },
    methods: {
      engadirTarefa() {
        this.tarefas.push(this.novaTarefa);
        this.novaTarefa = '';
      },
      borrarTarefa(index) {
        this.tarefas.splice(index, 1);
      },
    },
```

No exemplo anterior demóstrase que Vue, ao eliminar o primeiro elemento da lista, move só o contido dinámico do segundo elemento ao primeiro, en lugar de volver a crear toda a árbore DOM.

Para resolver este problema recoméndase proporcionar un atributo **key** con v-for, sempre que sexa posible, de forma que se identifique de forma única os elementos da lista. No exemplo anterior podería usarse o nome da tarefa como key (**:key="tarefa"**), supoñendo

que non hai nomes repetidos. O ideal sería que o atributo key estivese enlazado con un identificador único do elemento da lista.

```
<div v-for="item in items" :key="item.id">
  <!-- content -->
</div>
```

Neste outro exemplo pode comprobarse que ao pulsar o botón “Remove item”, que elimina o segundo elemento, fai que o elemento “Fish” perda a súa marca de “Favorito”. Versión arreglada utilizando o atributo **key**.

NOTA: non é posible utilizar o índice do array (index) como key xa que o índice indica a posición no array e non identifica o elemento. Ademais, o índice pode cambiar cando se engaden ou eliminan elementos do array.

Exercicios:

1. Exercicio titorial.
2. Dado un array de números almacenado en JavaScript, utiliza a directiva v-for para mostrar só os números pares.
3. Crea unha páxina web que mostre unha lista de elementos (escolle ti o significado), coas seguintes funcionalidades:
    a. Deben mostrarse os elementos en forma de lista.
    b. Deben poder engadirse novos elementos á lista.
    c. Deben poder eliminarse elementos da lista.
    d. Engade tamén un botón para mostrar/ocultar a lista. Fai que o texto deste botón sexa “Mostrar lista” ou “Ocultar lista” dependendo de se a lista é visible ou non.
4. Batalla. Descarga o código base e realiza as modificacións propostas usando Vue. Recoméndase que a lóxica da aplicación vaia no ficheiro JavaScript e non no HTML:
    a. Tanto a vida do monstro como a da persoa que xoga terá valores entre 0 - 100. É dicir, o valor de vida non pode ser un número negativo nin superar o valor 100. Debe comprobarse en todo momento que está no rango permitido. Para visualizalo de forma gráfica establecerase o valor correspondente na propiedade **width** de CSS no .
    b. Cada vez que se pulsa o botón “**Ataque**” a vida do monstro diminúe, pero ao mesmo tempo este contraataca e a vida da persoa que está xogando tamén diminuirá. Para que o xogo non sexa preditivo, a vida do monstro diminuirá nun valor aleatorio entre 5 e 12 e a vida da persoa que xoga diminuirá nun valor aleatorio entre 8 e 15.
    c. Cada vez que se pulsa o botón “**Ataque especial**” a vida do monstro diminúe un número aleatorio entre 10 e 25. Igual que no caso anterior, o monstro contraataca e a vida da persoa que está xogando tamén diminuirá un número aleatorio entre 8 e 15 (igual que antes).

d. Cada vez que se pulsa un dos botóns “**Ataque**”, “**Ataque especial**” ou “**Curación**” realízase unha acción no xogo. Fai que o botón “**Ataque especial**” só estea habilitado unha vez de cada 3 accións realizadas.
e. Ao pulsar o botón “**Curación**” a vida da persoa que xoga verase incrementada nun valor aleatorio entre 8 e 20 (hai que ter en conta de non superar o límite de 100). Ademais, cada vez que se use este botón, a vida da persoa que xoga tamén se verá diminuída nun número aleatorio entre 8 e 15 (como se sufrise un ataque).
f. Comprobación de fin de partida. Cando a vida do monstro ou da persoa xogadora chegue a 0, a partida remata. Comproba cando suceda isto e mostra no navegador información de que a partida rematou e quen ganou: o monstro, a persoa xogadora ou produciuse un empate.
g. Cando a partida remate debe mostrarse un botón “Novo xogo” para poder iniciar unha nova partida. Este botón non estará visible cando unha partida estea activa.
h. Cando se pulsa o botón “**Renderse**” significa que a persoa que está xogando se rende, polo que a partida remata. Non hai que modificar o estado das vidas nin do monstro nin da persoa que xoga.
i. Cando a partida remata, deben ocultarse os botóns “**Ataque**”, “**Ataque especial**”, “**Curación**” e “**Renderse**”.
j. Mostra, a modo de rexistro, as accións realizadas na partida. A última acción realizada debe ser a primeira que apareza. Utiliza as clases CSS definidas para os estilos: **log--player, log--monster, log--heal** (curación) e **log--damage** (ataque). Na imaxe, tanto o ataque como o ataque especial non están diferenciados. Exemplo:

Template Refs

Aínda que Vue abstrae a maioría das operacións directas sobre o DOM, hai algúns casos nos que é necesario ter acceso directo aos elementos HTML. Para conseguilo pode usarse o atributo especial **ref**.

Cando se establece o atributo **ref** nunha etiqueta HTML, dito elemento do DOM é engadido ao obxecto **$refs**, que está accesible dende JavaScript:

```
<div id="app">
  <input type="text" v-model="novaTarefa" ref="inputTarefa" />
  <button @click="engadirTarefa">Engadir tarefa</button>
</div>
```

```
  data() {
    return {
      novaTarefa: "",
    };
  },
  methods: {
    engadirTarefa() {
      console.log(this.$refs.inputTarefa);
    },
  },
```

O atributo **ref** é un atributo especial que permite obter unha referencia directa a un elemento específico do DOM, unha vez que o compoñente está montado.

Dende JavaScript, accedederase ao elemento coa sintaxe **this.$refs.<chave>**

Os elementos HTML creados con v-for aos que se lle engade o atributo **ref,** serán engadidos ao obxecto **$refs** como un array. Exemplo.

Referencias

Para a elaboración deste material utilizáronse, entre outros, os recursos que se enumeran a continuación:

- JavaScript | MDN
- Client-side web APIs - Learn web development | MDN
- JavaScript Tutorial - w3schools
- Desarrollo Web en Entorno Cliente | materials
- Javascript en español - Lenguaje JS
- The Modern JavaScript Tutorial
- Eloquent JavaScript
- Vue.js
- Getting started with Vue - Learn web development | MDN
- Vue Mastery
- Vue.js 3 Fundamentals, a FREE Vue.js Course