



Desarrollo web en entorno cliente





Tema 9: Componentes y Vue CLI

1. Componentes

1.1. ¿Qué es un componente?

Un componente es un elemento de un sistema que ofrece un servicio predefinido, y es capaz de comunicarse con otros componentes. Un componente correcto tendrá un mayor nivel de abstracción que un objeto. Una de las principales características de un componente es la reusabilidad.

Los web components, son elementos HTML reutilizables y compatibles con todos los marcos de trabajo. Se trata de bloques de código que encapsulan tanto los elementos html como CSS y código javascript, permitiendo separar la información por funcionalidad en lugar de según su contenido (ficheros html, css o js). Se trata de instancias reutilizables, pudiendo así estructurar la lógica de un proyecto en diferentes secciones o partes.

La mayor parte de los navegadores son compatibles con los componentes web.

Para saber qué debe ser un componente y que no, se puede tener en cuenta si éste puede ponerse en cualquier lugar de la aplicación y se verá y funcionará correctamente. Además, es algo que es muy posible que pueda aparecer en más de un lugar de la aplicación. En definitiva, un componente:

- Es una parte de la UI.
- Debe poder reutilizarse y combinarse con otros componentes para formar componentes mayores.
- Es un objeto JS.

El componente web tendrá una parte de HTML donde definimos su estructura y una parte JS que le da su funcionalidad. Puede además tener o no CSS para establecer su apariencia.

1.2. Props

Para entender que son las props, vamos a ver el ejemplo del proyecto “Introducción a componentes” del Aula Virtual.

En el ejemplo se puede ver que hay sendas variables llamadas “mensaje” en app y en el componente creado. Éstas, son variables propias de cada componente, y cada una de ellas podrá ser referenciada desde el mismo, pero no desde el otro.

Para poder tener el valor de una variable del objeto Vue en un componente hijo (o en cualquier componente, una variable de su componente padre) se utiliza un recurso llamado **props**, permitiendo pasar parámetros al componente.

Estos parámetros se han de añadir como atributos a la etiqueta del componente. Cuando se trata de pasar el valor de una variable, se ha de utilizar la directiva v-bind.

```
<mi-componente attr1="25" :attr2="var1" />
```



El parámetro pasado se recibe en el componente en el recurso **props** ya mencionado, y éste normalmente se incluye como primera propiedad del objeto del componente:

```
app.component("mi-componente ", {  
  props:[' attr1', ' attr2'],  
  template: ...  
});
```

** Si se quisiera dar un nombre compuesto por más de una palabra al atributo, en el documento html tendrá formato kebab-case, pero en el componente ha de referenciarse en formato CamelCase.

```
<mi-componente attr-num="25" :attr2="var1" />  
.....  
app.component("mi-componente ", {  
  props:[' attrNum', ' attr2'],  
  template: ...  
});
```



2. Single File Components

En el ejemplo visto, el componente se declara de forma global. Esta opción deja de ser buena cuando los proyectos son grandes y los componentes se utilizan para algo más que para mejorar ciertas vistas. Declarar los componentes así en un fichero js genera varios problemas:

- Los componentes al ser globales han de tener un nombre único.
- El código html de la plantilla está en medio de código js.
- El código css si queda fuera, por lo que no se encapsula del todo.

Una opción más adecuada es hacer uso de componentes de un solo archivo (SFC). Esta solución pasa por guardar cada componente completo en un único fichero (posible gracias a herramientas como vue-loader de Webpack). Tendrán extensión **.vue**, **su nombre comenzará por mayúscula** y las secciones que incluirán son:

- **<template>**: donde se incluirá el código html del componente.
- **<script>**: que contendrá el código javascript del componente.
- **<style>**: donde se pone el CSS del componente.

Siguiendo esto, html, js y css siguen separados, pero no en diferentes ficheros si no que en diferentes secciones del mismo.

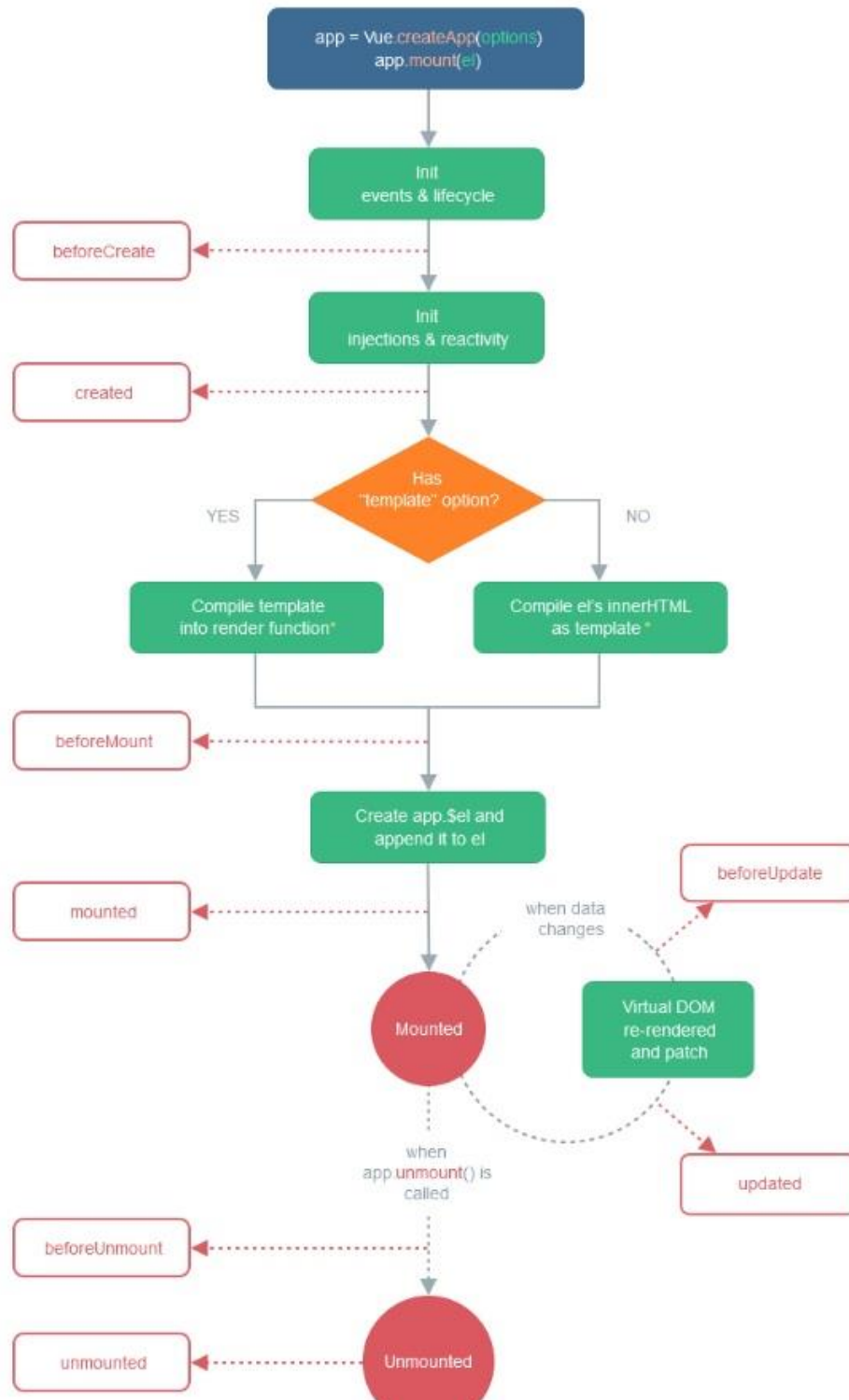
****Aquí entra en juego la extensión para VSC Vetur. Es de ayuda también la extensión Vue VSCode Snippets.**

2.1. Ciclo de vida de un componente

Todo componente pasa por diferentes estados, y, dependiendo del framework usado, existe la posibilidad de incluir funciones en los diferentes estados por los que va pasando.

En el caso de VueJS existen 4 estados posibles y permite incluir acciones(hooks) antes y después de que un componente se encuentre en un estado determinado:

- **beforeCreate**: Su ejecución es justo después de la inicialización de la instancia y antes de que el componente este creado.
 - **created**: Se ejecuta cuando la instancia, eventos, computed properties, el data y los métodos están creados. Normalmente se utiliza para inicializar propiedades del objeto data, llamadas a servicios, etc... En este punto el componente no está en el DOM todavía.
 - **beforeMount**: Su ejecución es justo antes de que se añada al DOM.
 - **mounted**: Su ejecución es después de añadirlo al DOM, y se puede utilizar para inicializar librerías que dependan del DOM.
 - **beforeUpdate**: Su ejecución es cuando el objeto data cambia, pero el DOM aún no ha realizado los cambios.
 - **updated**: Su ejecución es después de que el objeto data cambie y el DOM muestre estos cambios.
 - **beforeDestroy**: Su ejecución es justo antes de eliminar la instancia.
 - **destroyed**: Su ejecución es cuando la instancia, eventos, directivas e hijos del componente se han eliminado.
-



Cuando se crea un componente, incluso antes de su creación, es posible ejecutar código (*beforeCreated*), así como una vez creado (*created*). El componente puede ser creado, pero aún no montado, y también se lanzan eventos antes y después (*beforeMount* y *mounted*).

Durante la actividad del componente, también se lanzan eventos antes de actualizarse y justo tras su actualización (*beforeUpdate* y *updated*). Y, por último, cuando se termina la vida del componente, se lanzan eventos antes del desmontaje y una vez concluido: *beforeUnmount* y *unmounted*.

2.2. <template>

En esta sección se añadirá todo el código html que, en el ejemplo de introducción a componentes, se incluía en la propiedad template del componente.

Si el código HTML a incluir es muy extenso, se puede guardar en un fichero .html externo y referenciarlo desde la plantilla.

```
<template>
  <div>
    <h1 :class="{ anchoFuente: isBold }">Sitio Web de Sonia
    Gutiérrez</h1>
    <h4>{{ mensaje }}</h4>
    <MiComponente />
  </div>
</template>
<!-- Se podría incluir también el html en un fichero aparte y
referenciarlo de la siguiente manera:
<template src="../../src/componente.html"> </template> -->
```

2.3. <script>

El componente en sí se declarará en esta sección. Será un objeto que se exportará con sus propiedades.

Si se van a utilizar subcomponentes, han de importarse antes de definir el objeto y registrarlos dentro del mismo:

```
import MiComponente from "../components/MiComponente.vue";
```

Algunas de las propiedades mas usadas del objeto son:

- **name:** nombre del componente. Obligatorio para componentes recursivos.
- **components:** sección donde se van de registrar los componentes anteriormente importados. En el momento de usarse en la sección template, se referenciarán a través del nombre aquí declarado.
- **props, data, methods, computed.**
- **created(), mounted(),** etc: funciones hook que se ejecutarán al crearse el componente, al montarse, etc.

```
<script>
import MiComponente from "../components/MiComponente.vue";
export default {
  name: "micomponente",
  components: { MiComponente },
  data() {
    return {
      mensaje: "Hola Mundo",
      negrita: true };
  }
};
</script>
```



2.4. <style>

La última sección incluirá los estilos a aplicar en el componente. Es posible importar ficheros de estilos, pero tendrán que estar almacenados en una carpeta llamada assets.

Algunos atributos:

- **scoped:** Si aparece, indica que los estilos definidos no se pueden utilizar en otros componentes de la aplicación no descendientes de este. Para diferenciar estilos “internos” de otro, en el SFC puede haber dos etiquetas style.
- **lang:** para indicar el formato de estilos que se va a utilizar. Puede ser CSS, SASS o PostCSS.

```
<style lang="css">
  .anchoFuente {
    font-weight: bold;
  }
</style>
```

2.5. Custom blocks

Existe también la posibilidad de añadir bloques definidos por el desarrollador (por ejemplo, para incluir documentación, test unitarios, etc.).

2.6. Estructura de un proyecto con SFC

Como ya se ha especificado, se creará un SFC por cada componente. Pero, además, se creará un componente principal para la aplicación llamado App.vue que incluirá al resto de componentes.

Se ha visto que es posible crear un proyecto con VUE, haciendo uso de SFCs, utilizando herramientas como webpack, etc. Existe también la posibilidad de que esa estructura inicial la cree automáticamente la herramienta que se va a ver en el apartado siguiente.



3. Vue CLI

Vue CLI(Command Line Interface) es una herramienta que facilita la programación Vue:

- Crea automáticamente el scaffolding(andamiaje) básico del proyecto basándose en plantillas predefinidas.
- Incluye herramientas como Webpack, Babel, Uglify, ... que permiten gestionar las dependencias del código.
- Habilita empaquetar todos los ficheros .vue y librerías en un único fichero JS y CSS.
- Permite traspilar código ES2015/2016, SCSS, etc a ES5 y CSS3 estándar.
- minimiza el código generado.
- Incluye herramientas que facilitan el desarrollo.

3.1. Instalación de Vue CLI

EJERCICIO 1

- Para hacer uso de Vue CLI, se va a descargar e instalar primero nodeJS(versión LTS) de la página oficial(encontrarás un enlace en el Aula Virtual), ya que CLI utiliza npm(node package manager), que es la gestión de paquetes de node, para crear los proyectos.
- Una vez instalado node y reiniciado el equipo, comprobamos que efectivamente está instalado:

```
Símbolo del sistema
Microsoft Windows [Versión 10.0.17763.805]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Portatil>node -v
v14.15.4

C:\Users\Portatil>npm -v
6.14.10

C:\Users\Portatil>
```

- Y a continuación, ahora sí, se va a instalar CLI a través de la terminal. Si no se tienen permisos de administrador es conveniente ejecutar la terminal como administrador:

```
npm install -g @vue/cli
```

Con -g se instala CLI de forma global. Y comprobamos la instalación:

```
Símbolo del sistema

C:\Users\Portatil>vue --version
@vue/cli 4.5.10

C:\Users\Portatil>
```


3.2. Primer proyecto con CLI

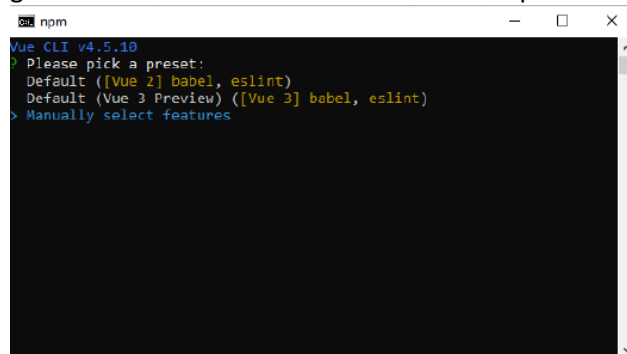
Crear el proyecto con CLI

Con el comando **create** de vue se puede crear un proyecto, teniendo en cuenta que el nombre del mismo no puede contener letras mayúsculas. Desde la terminal, posicionándose en la carpeta donde se quiera crear el proyecto, se ejecutará:

```
vue create nombre_proyecto
```

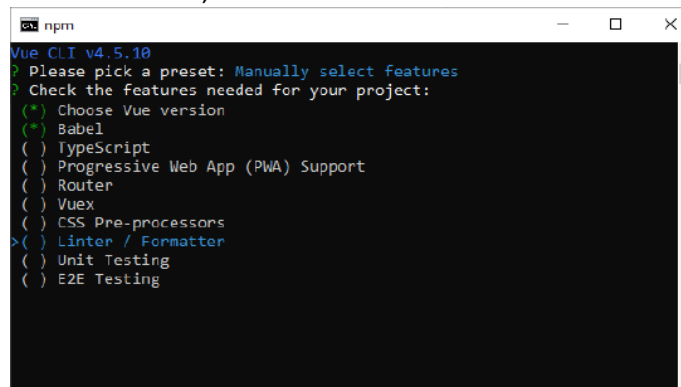
Durante la creación, habrá que seleccionar una serie de variables:

- Primero se podrá crear el proyecto con unas características preestablecidas o configurarlas manualmente. Seleccionamos la opción manual:



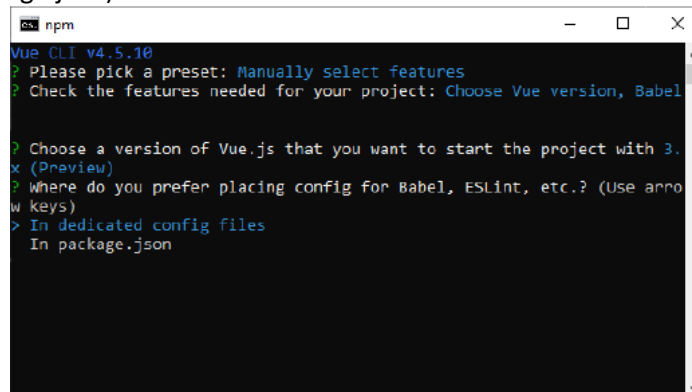
```
npm
Vue CLI v4.5.10
? Please pick a preset:
  Default ([Vue 2] babel, eslint)
  Default (Vue 3 Preview) ([Vue 3] babel, eslint)
> Manually select features
```

- Cuando se muestren las opciones, dejaremos seleccionado Versión y Babel, el resto fuera (con la barra de espacio se seleccionan o se quita la selección). Una vez realizada la selección, enter:



```
npm
Vue CLI v4.5.10
? Please pick a preset: Manually select features
? Check the features needed for your project:
  (*) Choose Vue version
  (*) Babel
  ( ) TypeScript
  ( ) Progressive Web App (PWA) Support
  ( ) Router
  ( ) Vuex
  ( ) CSS Pre-processors
> ( ) Linter / Formatter
  ( ) Unit Testing
  ( ) E2E Testing
```

- Seleccionamos versión 3 de Vue. Enter. Indicamos que la configuración de Babel y demás se almacenará en archivos de configuración dedicados(y no en package.json). Enter:

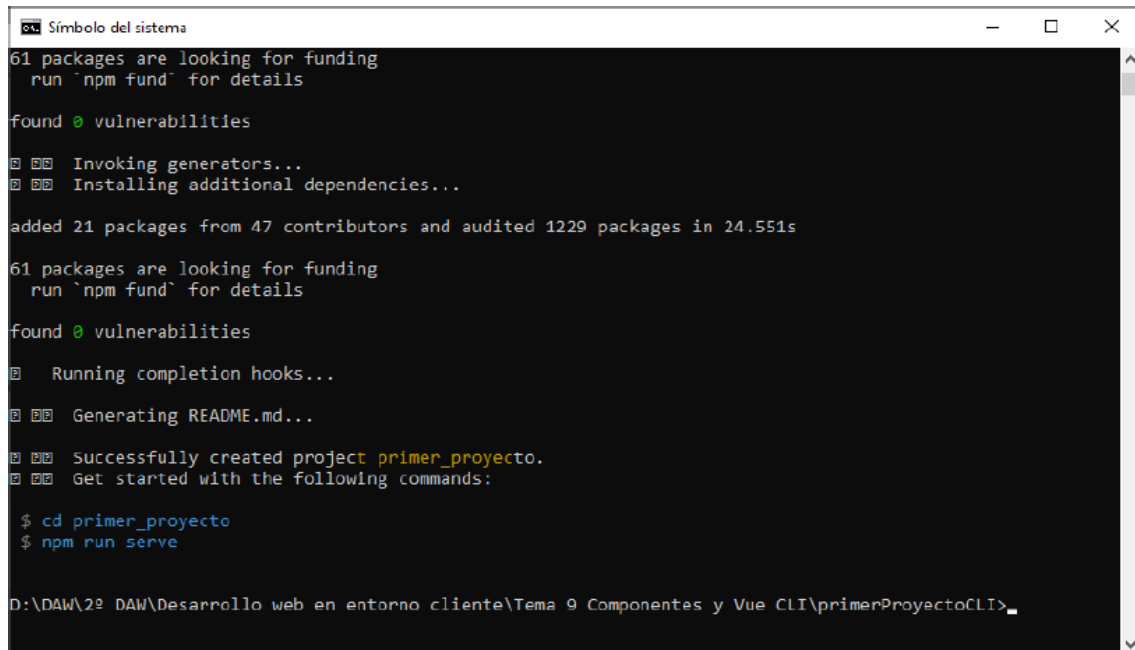


```
npm
Vue CLI v4.5.10
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel

> Choose a version of Vue.js that you want to start the project with 3.
x (Preview)
? Where do you prefer placing config for Babel, ESLint, etc.? (Use arrow
w keys)
> In dedicated config files
  In package.json
```



- Después será necesario indicar si se quiere guardar el proyecto como plantilla para futuros usos, o no. Enter. Y en este punto comenzará la generación del proyecto en sí:



```
Símbolo del sistema
61 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

Invoking generators...
Installing additional dependencies...

added 21 packages from 47 contributors and audited 1229 packages in 24.551s

61 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

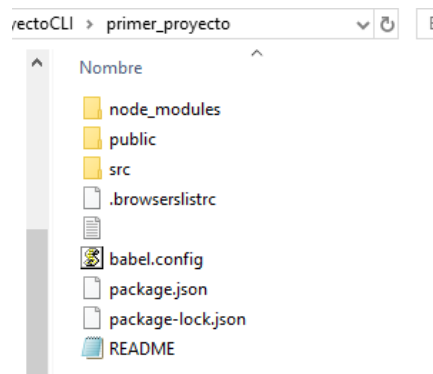
Running completion hooks...
Generating README.md...

Successfully created project primer_proyecto.
Get started with the following commands:

$ cd primer_proyecto
$ npm run serve

D:\DAW\2º DAW\Desarrollo web en entorno cliente\Tema 9 Componentes y Vue CLI\primerProyectoCLI>
```

En este momento, en la carpeta seleccionada, se habrá creado una nueva carpeta con el nombre que se le ha dado al proyecto, y con distintos ficheros y directorios:



Al estar utilizando CLI, que a su vez hace uso de NodeJS, y como NodeJS se ejecuta del lado del servidor, es posible arrancar un servidor para probar el proyecto. npm ofrece, similar al Live Server que se ha utilizado hasta el momento, la opción con el comando:

`npm run serve`

Se abrirá una nueva terminal, con el servidor ya corriendo. Mostrará la url para poder abrir la aplicación en el navegador. Esa nueva terminal tendrá el siguiente aspecto:



```
1: node
[DONE] Compiled successfully in 9662ms
18:22:24

App running at:
- Local: http://localhost:8080/
- Network: http://192.168.1.131:8080/

Note that the development build is not optimized.
To create a production build, run npm run build.
```

El servidor se parará cuando se cierra VSC. También se puede parar con Ctrl+C.

EJERCICIO 2

- Crea una carpeta en la que, de ahora en adelante, crearás todos tus proyectos con CLI.
- Crea un primer proyecto, dentro de la carpeta anterior, siguiendo los pasos del punto 3.2.

Entendiendo el contenido generado con CLI

De todo el contenido generado al crear el proyecto, las carpetas y archivos a destacar son:

- **public:** En esta carpeta se encuentra el documento index.html. Este documento, en principio, no es necesario modificarlo más que para añadir, por ejemplo, un cdn para bootstrap. En él ya se incluye el elemento html en el que se va a ejecutar Vue.
- **src:** Carpeta con la aplicación en sí:
 - **main.js:** Fichero principal JavaScript donde se crea el componente principal y lo monta en el elemento div con id app(paso ya conocido).
 - **App.vue:** Componente principal del sitio web generado con Vue.
 - **assets:** Directorio para almacenar los ficheros estáticos del sitio web.
- El resto fuera de src y public, son documentos de configuración. Entre ellos:
 - **.gitignore:** para ignorar, en principio, todos los componentes node de la carpeta node_modules(paquetes necesarios para que Vue funcione), mas todos aquellos que se quiera.
 - **package.json:** configuración necesaria para poder ejecutar el servidor. Por ejemplo, el apartado scripts incluye los comandos que se ejecutan cuando en la terminal se ejecuta npm run serve.

EJERCICIO 3

- Descarga el proyecto “Primer proyecto con Vue CLI” que encontrarás en el Aula Virtual:
 - Incluye los distintos ficheros, en el proyecto que has creado en el ejercicio 2.
 - Revisa los ficheros index.html, App.vue, PiePagina.Vue y main.js y plantea las dudas que te surjan.
 - Ejecútalo y comprueba que la página se ve correctamente.
- Crea un nuevo proyecto con Vue CLI. En este nuevo proyecto desarrolla la funcionalidad que se detalla a continuación. Partiendo del ejercicio de propósitos de año nuevo de la unidad anterior (coge la solución que tengas del ejercicio 4):
 - Separa las distintas funcionalidades en componentes. Los componentes a crear son:
 - Uno para mostrar la lista de propósitos.



- Otro para añadir un nuevo propósito.
- El borrado de la lista completa quedará fuera de cualquier componente.
- Además, se desencadenará la acción de añadir nuevo propósito no solo cuando se pulse el botón, sino que también cuando, teniendo el foco el input, se pulse intro. (Para hacerlo se utilizarán manejadores de eventos)

Añadiendo plugins a un proyecto

Para instalar un nuevo plugin al proyecto se puede ejecutar, dentro de la carpeta del proyecto, el siguiente comando:

```
vue add nombrePlugin
```

Este comando desencadenará las siguientes acciones:

- instala el plugin dentro de node-modules.
- modifica el fichero package.json.
- crea un fichero JS dentro de la carpeta plugins que importa y registra la librería.
- importa dicho fichero al main.js.

Por ejemplo, si se quiere importar el plugin de bootstrap para Vue(habría que comprobar primero si ya está para Vue3), se ejecutará:

```
vue add bootstrap-vue
```

Si se va a instalar un paquete que no funcione como plugin, habrá que hacerlo con npm:

```
npm install nombre-paquete
```

Al hacerlo a través de esta opción, sólo se instalará el paquete en node-modules, pero para que lo añada a las dependencias del package.json habrá que utilizar la opción --save o -S (si se trata de una dependencia de producción) o bien --dev o -D (si es de desarrollo):

```
npm install -S nombrePaquete
```

La importación y el registro en main.js habrá que hacerlo a mano.

Props con tipo declarado

Otra opción para declarar props es el tipado de las mismas. Es posible indicar el tipo de dato que tiene que recibir la props:

```
props: {  
  libros: Array,  
  telefono: Number,  
}
```

3.3. Reutilización de componentes

EJERCICIO 4

Partiendo del primer apartado del ejercicio 3 (terminado):

- Modifica el código html del componente PiePagina para que, en lugar de un elemento footer, sea un elemento div.





- Desde el componente padre, incluye un nuevo elemento PiePagina al principio de la página (haciendo las veces de cabecera). Modifica los estilos de tal forma que sea vea el componente (da igual que el último no esté abajo del todo) y para que el color de fondo sea solo para esos dos elementos PiePagina.
- Modifica el valor de la props “dir” de tal forma que el valor no sea el mismo para ambos elementos PiePagina.

4. Comunicación entre componentes

Los datos propios de un componente se consideran **datos a nivel de componente**. Cuando varios componentes acceden a los mismos datos, estos datos son **datos a nivel de aplicación**. Para tratar los datos a nivel de aplicación se ha visto ya la opción de las props, pero éstas no pueden ser modificadas por el componente hijo.

******En algún ejemplo de los vistos hasta el momento, se modificaba un array del padre añadiendo desde el hijo un nuevo elemento. Los objetos o arrays se pasan por referencia por lo que, si se modifican en el hijo, sí se actualizarán en el padre, pero no es una buena práctica.

A continuación, se van a ver otras formas de comunicación entre componentes.

4.1. Custom Events

Si un componente hijo quiere pasarle un dato al padre tiene que emitir un evento, previamente configurado en el padre, que el padre capturará y tratará convenientemente.

El componente que vaya a emitir el evento usará la sintaxis siguiente:

```
this.$emit(nombreEvento, parametro);
```

Y el componente padre lo capturará:

```
<Componente @nombreevento="funcionManejadora"></Componente>
```

Y, en la sección methods del padre estará incluida la función manejadora.

******Si al evento se le da un nombre en formato kebab-case, no se hará la conversión de forma automática como si pasa en las props, por lo que habrá que nombrarlo siempre con el mismo formato.

Al igual que se hace con las props, los eventos que emite un componente pueden (y es lo recomendable) definirse en la opción emits:

```
export default {  
  ...  
  emits: ["nombreEvento1", "nombreEvento", ...],  
  ...  
};
```

EJERCICIO 5

- En el Aula Virtual encontrarás un ejemplo de eventos en el proyecto “Comunicación entre Componentes”. Descárgalo, incluye las carpetas en un proyecto Vue CLI para probarlo y plantea las dudas que te surjan sobre la implementación.
 - Siguiendo con el ejercicio de propósitos de año nuevo:
-



- modifica los componentes para que todos los cambios que se realizan en la lista de propósitos (eliminarla, añadir uno, borrar uno) se haga a través de eventos.
- Añade un nuevo apartado a tu página, paralelo a la lista de propósitos (será un nuevo componente).
 - En dicho apartado se mostrará una tabla con todos los propósitos que se hayan cumplido.
 - Esa tabla se actualizará de forma automática cada vez que se cumpla un nuevo propósito o se elimine uno cumplido.
- Modifica los estilos de los componentes para que cada uno de ellos tenga una fuente de texto diferente, salvo para los apartados que ya tienen fuentes concretas asignadas.
- Añade un título h2 a cada apartado (si no lo incluye ya). Cada uno de los h2 tendrá un color de fondo diferente.
- El botón de Eliminar Lista estará desactivado mientras que la lista de propósitos esté vacía.

4.2. Otros tipos de comunicación

Para implementar comunicación entre componentes con una relación diferente a la de padre-hijo, se pueden encontrar diferentes opciones, por ejemplo:

- el uso de variables globales, no recomendado salvo para casos muy concretos.
- En Vue 3 se puede implementar una solución haciendo uso de la librería Mitt.

La recomendación de Vue es no salirle de la comunicación directa padre-hijo o hijo-padre y, para cuando sea necesario otro patrón, hacer uso de Vuex.