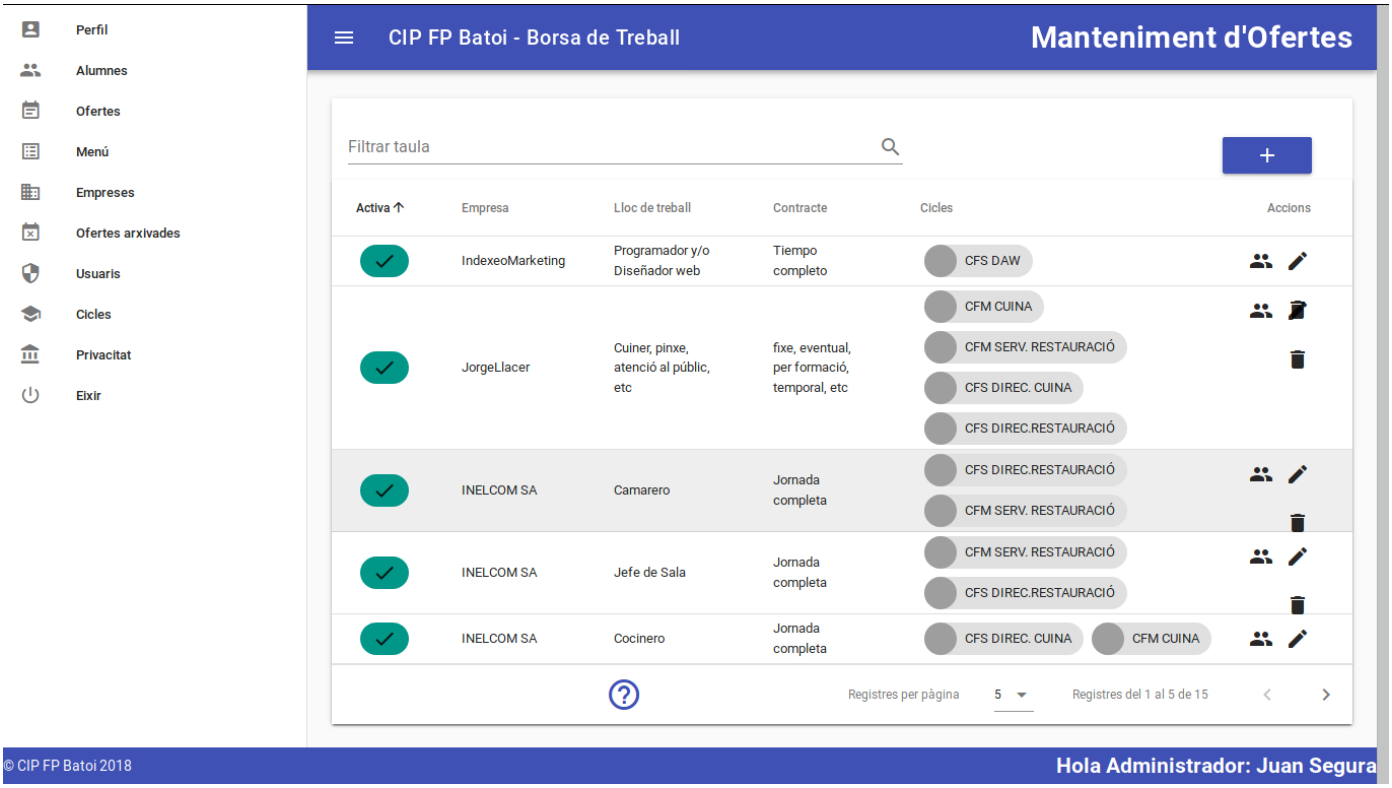
UD 10 - Componentes en Vue

1. Introducción

El sistema de componentes es un concepto importante en Vue y en cualquier framework moderno. En lugar de separar nuestra aplicación en ficheros según el tipo de información que contienen (ficheros html, css o js) es más lógico separarla según su funcionalidad. Una página web muestra una UI donde se pueden distinguir diferentes partes. En el siguiente ejemplo tenemos:



- un menú que es una lista que contiene
    - (repetido) un elemento del menú, cada uno formado por un logo y un texto
- un título
- una tabla con la información a mostrar, formada por
    - un elemento para filtrar la información formado por un input y un botón de buscar
    - un botón para añadir nuevos elementos a la tabla
    - una cabecera con los nombres de cada campo
    - (repetido) una fila para mostrar cada elemento de información, con botones para realizar diferentes acciones
    - un pie de tabla con información sobre los datos mostrados
- un pie de página

Pues estos elementos podrían constituir diferentes componentes: nuestras aplicaciones estarán compuestas de pequeños componentes independientes y reusables en diferentes partes de nuestra aplicación o en otras aplicaciones (podemos usar el elemento de buscar para otras páginas de nuestra web o incluso para otras aplicaciones). También es habitual que un componente contenga otros subcomponentes, estableciéndose relaciones padre-hijo (por ejemplo en componente fila contendrá un subcomponente por cada botón que queramos poner en ella).

Para saber qué debe ser un componente y que no, podemos considerar un componente como un elemento que tiene entidad propia, tanto a nivel funcional como visual, es decir, que puede ponerse en el lugar que queramos 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
- son objetos JS

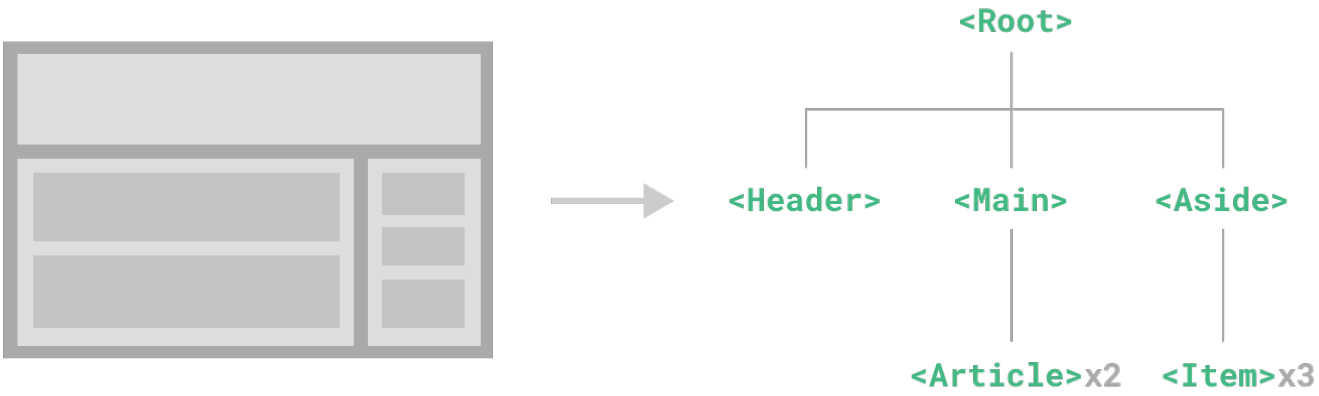
El componente es un objeto con una parte de **HTML** donde definimos su estructura, una parte **JS** que le da su funcionalidad y una parte (opcional) **CSS** para establecer su apariencia.

Separar nuestra aplicación en componentes nos va a ofrecer muchas ventajas:

- encapsulamos el código de la aplicación en elementos más sencillos
- facilita la reutilización de código
- evita tener código repetido

El primer paso a la hora de hacer una aplicación debe ser analizar qué componentes tendrá

En definitiva nuestra aplicación será como un árbol de componentes con la instancia principal de Vue como raíz.



2. Usar un componente

Para usarlo basta con crearlo con `app.component`, darle un nombre y definir el objeto con sus propiedades *data*, *methods*, ... Además tendrá una propiedad *template* con el código HTML que se insertará donde pongamos el componente. Lo hacemos en nuestro fichero JS.

Por ejemplo, vamos a crear un componente para mostrar cada elemento de la lista de tareas a hacer:

```
const app = Vue.createApp({
  ...
})

app.component('todo-item', {
  template: `
    <li>
```

```
      <input type="checkbox" v-model="elem.done">
      <del v-if="elem.done">
        { { elem.title }}
      </del>
      <span v-else>
        { { elem.title }}
      </span>
    </li>`,
  data: ()=>({
    elem: { title: 'Cosa a hacer', done: true }
  })
})
...
app.mount('#app')
```

El nombre de un componente puede estar en *PascalCase* (MyComponentName) o en *kebab-case* (my-component-name). Lo recomendado es que en Javascript lo pongamos en PascalCase y en el HTML en kebab-case (Vue hace la traducción automáticamente). Se recomienda que el nombre de un componente tenga al menos 2 palabras para evitar que pueda llamarse como alguna futura etiqueta HTML.

Ahora ya podemos usar el componente en nuestro HTML:

```
<ul>
  <todo-item></todo-item>
</ul>
```

Resultado:

- [x] ~~Cosa a hacer~~

Podemos utilizar la etiqueta tal cual (`<todo-item>`) o usar una etiqueta estándar y poner la nuestra como valor de su atributo *is*:

```
<ul>
  <li is="todo-item"></li>
</ul>
```

De esta forma evitamos errores de validación de HTML ya que algunos elementos sólo pueden tener determinados elementos hijos (por ejemplo los hijos de un <ul> deben ser <li> o los de un <tr> deben ser <td>).

2.1 Parámetros: *props*

Podemos pasar parámetros a un componente añadiendo atributos a su etiqueta:

```
<ul>
  <todo-item :todo="{ title: 'Nueva cosa', done: false }"></todo-item>
</ul>
```

NOTA: recuerda que si no ponemos el *v-bind* estaríamos pasando texto y no una variable.

El parámetro lo recibimos en el componente en *props*:

```
app.component('todo-item', {
  props: {
    todo: String
  },
  template: `
    <li>
      <input type="checkbox" v-model="todo.done">
      ...`
})
```

Se pueden declarar las *props* recibidas como un array de cadenas (`props: ['todo']`), aunque si los declaramos como un objeto podemos hacer ciertas comprobaciones (en este caso que se recibe un *String*).

NOTA: si un parámetro tiene más de 1 palabra en el HTML lo pondremos en forma kebab-case (ej.: `<todo-item :todo-elem=...>`) pero en el Javascript irá en camelCase (`app.component('todo-item',{ props: ['todoElem'],...})`). Vue hace la traducción automáticamente.

Resultado:

- ☐ Nueva cosa a hacer

En nuestro caso queremos un componente *todo-item* para cada elemento del array *todos*:

```
<ul>
  <todo-item v-for="item in todos" :key="item.id" :todo="item"></todo-item>
</ul>
```

Resultado:

- [x] ~~Learn JavaScript~~
- [ ] Learn Vue

...

IMPORTANTE: al usar *v-for* en un componente debemos indicarle en la propiedad *key* la clave de cada elemento. Si no tuviera ninguna podemos usar como clave su índice en el array:

```
<ul>
  <todo-item v-for="(item, index) in todos" :key="index" :todo="item">
</todo-item>
</ul>
```

2.2 A tener en cuenta

A la hora de definir componentes hay un par de cosa que debemos tener en cuenta

2.2.1_template_ es recomendable que contenga un único elemento

El template de un componente debe tener un único elemento raíz por lo que, si queremos tener más de uno hay que englobarlos en un elemento (normalmente un

):

```
// NO RECOMENDABLE
app.component('my-comp', {
  template: `<input id="query">
             <button id="search">Buscar</button>`,
})

// RECOMENDABLE
app.component('my-comp', {
  template: `<div>
               <input id="query">
               <button id="search">Buscar</button>
             </div>`,
})
```

NOTA: Vue2 solo acepta en el template un único elemento.

2.2.2_data_ debe ser una función

Además de las variables que se le pasan a un componente en *props* este puede tener sus propias variables internas (definidas en *data*) y sus propios métodos, *hooks*, etc.

```
// Componente
app.component('my-comp', {
  data() {
    return {
      message: 'Hello',
      counter: 0
    }
  }
})
```

También podemos ponerlo en notación de *arrow function*:

```
app.component('my-comp', {
  data: ()=>({
      message: 'Hello',
      counter: 0
  })
})
```

2.3 Registrar un componente localmente

Un componente registrado como hemos visto es *global* y puede usarse en cualquier instancia raíz de Vue creada posteriormente (con *new Vue()*) y también dentro de subcomponentes de dicha instancia.

Pero eso no es lo más correcto ya que lo normal, igual que con las variables, es registrarlo localmente donde vaya a usarse, de forma que sólo se pueda usar dentro de la instancia Vue o del subcomponente en que se registra.

En ese caso el componente a registrar se guarda en un objeto

```
const ComponentA={ /* .... */ }
```

y se registra en cada instancia o subcomponente en que quiera usarse:

```
// Para usarlo en la instancia raíz
new Vue({
  el: '#app',
  components: {
    'component-a': ComponentA,
  }
})

// Para usarlo en un subcomponente
var ComponentB={
  ...,
  components: {
    'component-a': ComponentA,
  }
}
```

NOTA: al ser igual el nombre de la propiedad (*component-a*) y su valor (*ComponentA*) podemos usar la notación de ES2015 y no poner el valor:

```
 components: {
   ComponentA,
 }
```

Cuando trabajamos con componentes lo normal es que no estén en el mismo fichero sino que cada componente se guarde en su propio fichero (con extensión *.vue*) y se importe donde vaya a usarse:

```
// fichero ComponentB.vue
import ComponentA from './ComponentA.vue'

export default {
  ...,
  components: {
    ComponentA,
  }
}
```

3. Vue-cli

Aunque puede usarse *Vue* como hemos visto, enlazándolo directamente en el *index.html* lo más habitual es utilizar la herramienta **vue-cli** que nos facilita enormemente la creación de proyectos *Vue*. Esta herramienta:

- Crea automáticamente el *scaffolding* básico de nuestro proyecto basándose en una serie de plantillas predefinidas
- Facilita el trabajo con componentes, permitiendo que cada uno de ellos esté en su propio fichero (**SFC**, *Single File Components*)
- Incluye utilidades y herramientas como Webpack, Babel, Uglify, ... que permiten
  - gestionar las dependencias de nuestro código
  - empaquetar todos los ficheros *.vue* y librerías en un único fichero JS y CSS
  - traspilar el código ES2015/2016, SCSS, etc a ES5 y CSS3 estándar
  - minimizar el código generado
- Incluye herramientas que facilitan el desarrollo

La versión actual es la 4 que ha cambiado de una arquitectura basada en plantillas a una basada en plugins lo que mejora enormemente su rendimiento. Podemos encontrar toda la documentación en Vue CLI.

3.1 Instalación

Para usar **vue-cli** necesitamos tener instalado **npm** (el gestor de paquetes de Node.js). Si no lo tenemos instalaremos **node.js**.

Podemos instalarlo desde los repositorios como cualquier otro programa (`apt install nodejs`), pero no es lo recomendado porque nos instalará una versión poco actualizada por lo que es mejor instalarlo desde NodeSource_ siguiendo las instrucciones que se indican y que básicamente son:

```
curl -sL https://deb.nodesource.com/setup_X.y | sudo -E bash -
sudo apt-get install -y nodejs
```

(cambiaremos *X.y* por la versión que queramos, vue-cli recomienda al menos la 10.x).

También podemos descargarlo desde NodeJS.org, descomprimir el paquete e instalarlo (`dpkg -i _nombrepaquete_`).

Una vez instalado **npm** Vue-cli se instala con

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

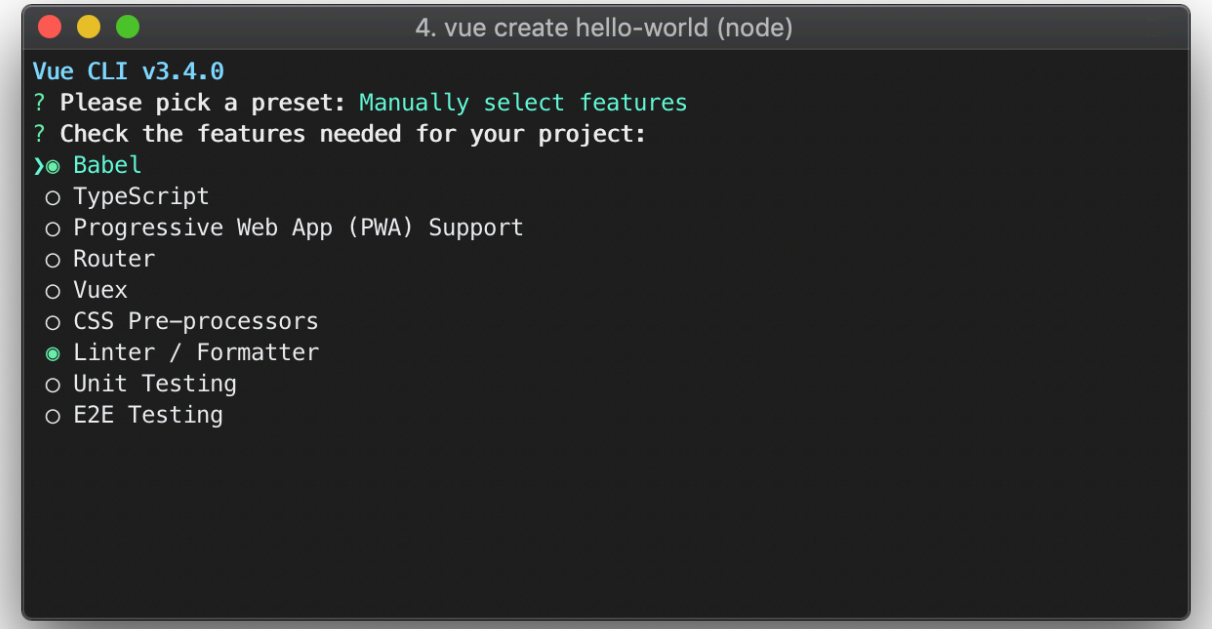
La opción -g es para que lo instale globalmente en el sistema y no tengamos que instalar una copia para cada proyecto.

3.2. Creación de un nuevo proyecto

Para crear un nuevo proyecto haremos:

```
vue create _<directorio_proyecto>_
```

Vue nos ofrece la opción de crear el nuevo proyecto para Vue2 o Vue3 por defecto con los plugins para *Babel* y *esLint* (mäs adelante podremos añadir más si los necesitamos) o bien la opción **manual** donde escogemos que plugins instalar para el proyecto de entre los siguientes:



También podemos crear y gestionar nuestros proyectos desde el entorno gráfico ejecutando el comando:

```
vue ui
```

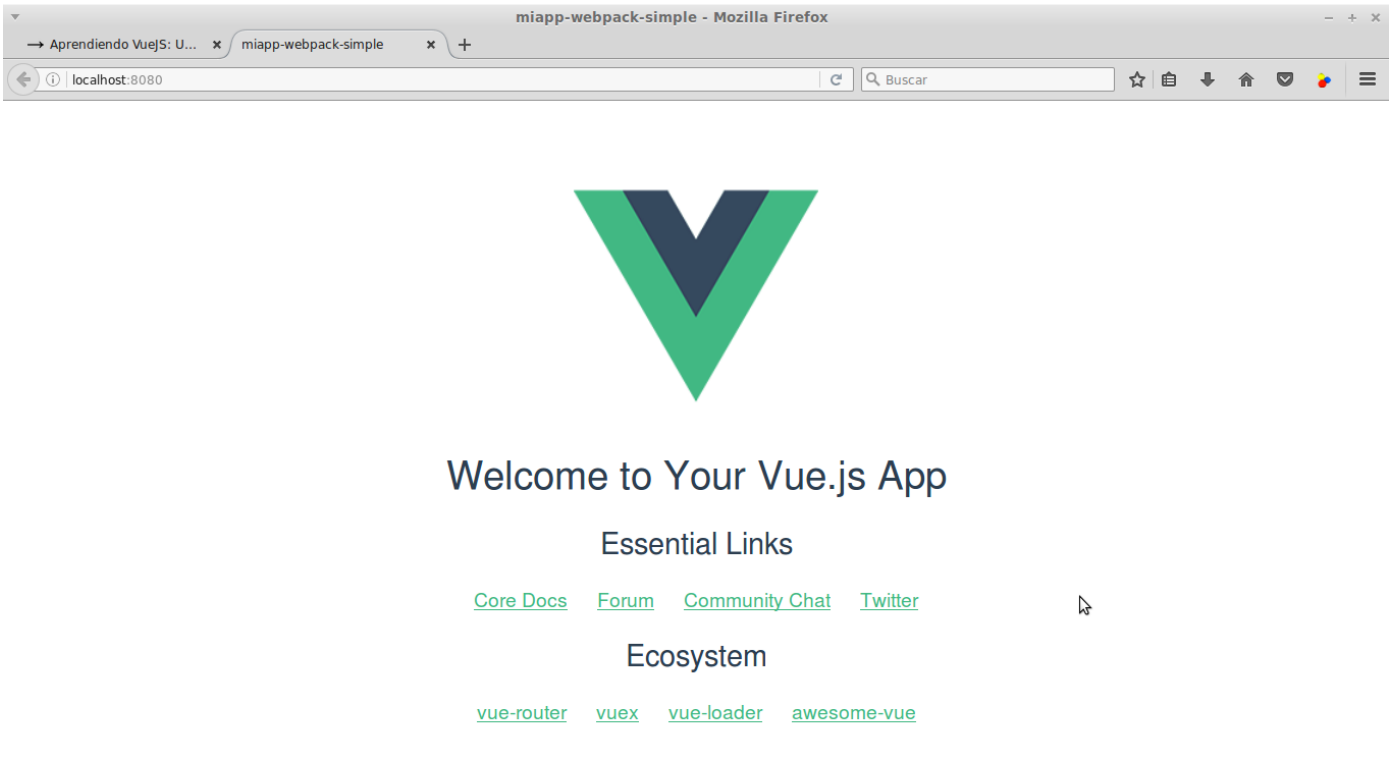
Este comando arranca un servidor web en el puerto 8000 y abre el navegador para gestionar nuestros proyectos.

3.2.1 Ejemplo proyecto por defecto

Una vez creado entramos a la carpeta y ejecutamos en la terminal

```
npm run serve
```

Este script compila el código, muestra si hay errores, lanza un servidor web en el puerto 8080 y carga el proyecto en el navegador (http://localhost:8080). Si cambiamos cualquier fichero JS de *src* recompila y recarga la página automáticamente. La página generada es:



3.2.2_Build and Deploy_ de nuestra aplicación

Normalmente trabajaremos con algún gestor de versiones como *git*. Para subir nuestro proyecto al repositorio lo creamos (el *GitHub*, *GitLab* o donde queramos) y ejecutamos desde la carpeta del proyecto:

```
git init
git add .
git remote add origin https://github.com/mi-usuario/mi-proyecto
git commit -m "Primer commit"
git push -u origin main
```

Cuando nuestra aplicación esté lista para subir a producción ejecutaremos el script:

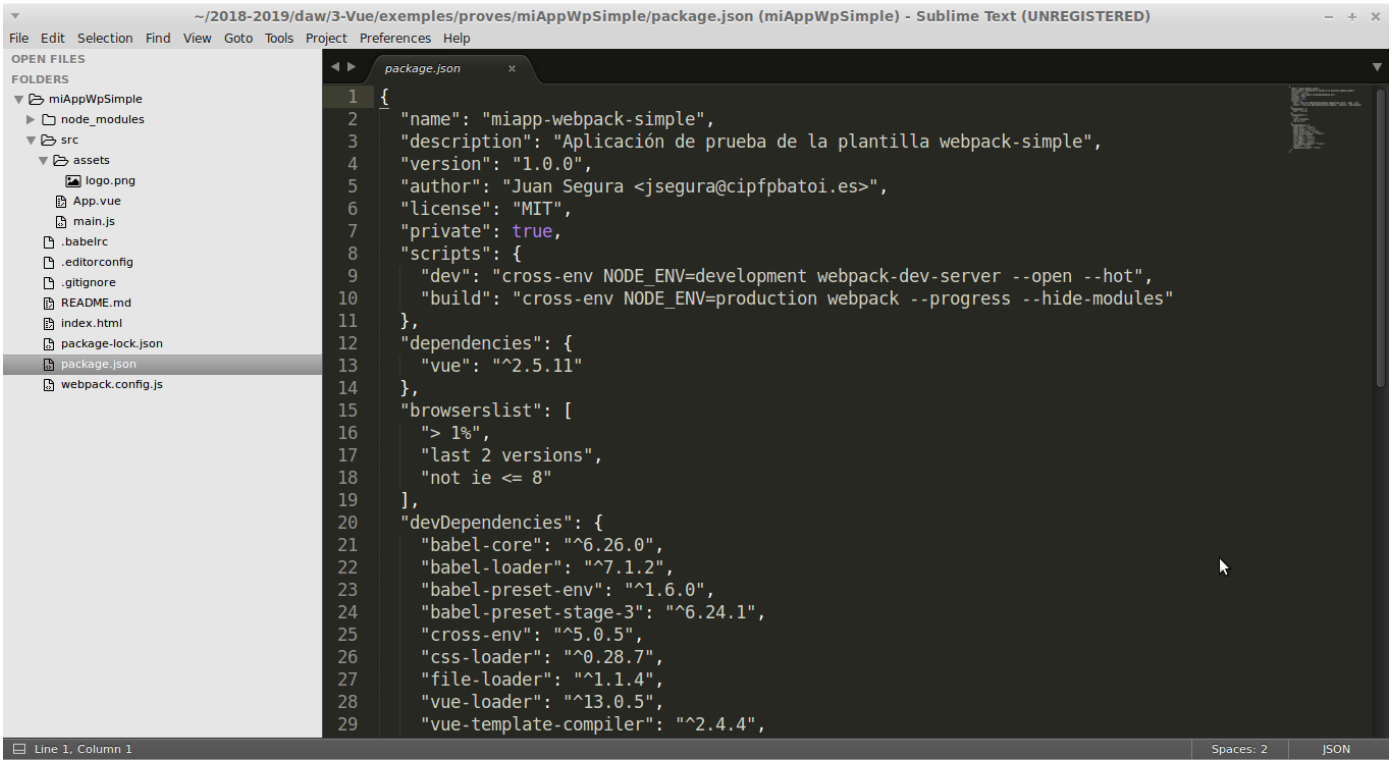
```
npm run build
```

Este comando genera los JS y CSS para subir a producción dentro de la carpeta *dist*. El contenido de esta carpeta es lo que debemos subir a nuestro servidor de producción.

También podemos ejecutar el comando `npm run lint` para ejecutar esta herramienta y comprobar nuestro código.

3.2.3_Scaffolding_ creado

Se ha creado la carpeta con el nombre del proyecto y dentro el scaffolding para nuestro proyecto:



Los principales ficheros y directorios creados son:

- `package.json`: configuración del proyecto (nombre, autor, ...) y dependencias
- `babel.config.js`: configuración de Babel
- `public/index.html`: html con un div donde se cargará la app
- `node_modules`: librerías de las dependencias

- `src`: todo nuestro código
  - `assets/`: nuestros CSS, imágenes, etc
  - `main.js`: JS principal que carga componentes y crea la instancia de Vue que carga el componente principal llamado *App.vue* y lo renderiza en *#app*
  - `App.vue`: es el componente principal y constituye nuestra página de inicio del proyecto. Aquí cargaremos la cabecera, el menú,... y los diferentes componentes
  - `components/`: carpeta que contendrá los ficheros .vue de los diferentes componentes
    - `HelloWorld.vue`: componente de ejemplo llamado por App.vue

3.2.3.1 package.json

Aquí se configura nuestra aplicación:

- **name, version, author, license**, ...: configuración general de la aplicación
- **scripts**: ejecutan entornos de configuración para webpack. Por defecto tenemos 3:
  - **serve**: lanza el servidor web de webpack y configura webpack y vue para el entorno de desarrollo
  - **build**: crea los ficheros JS y CSS dentro de **/dist** con todo el código de la aplicación
  - **lint**: lanza el linter
- **dependences**: se incluyen las librerías y plugins que utiliza nuestra aplicación en producción. Todas las dependencias se instalan dentro de **/node-modules**. Posteriormente veremos como añadir nuevas dependencias
- **devDependencies**: igual pero son paquetes que sólo se usan en desarrollo (babel, webpack, etc). También se instalan dentro de node-modules pero no estarán cuando se genere el código para producción. Para instalar una nueva dependencia de desarrollo ejecutaremos `npm install nombre-del-paquete -D` (la opción -D la añade a package.json pero como dependencia de desarrollo).

3.2.3.2 Estructura de nuestra aplicación

Fichero index.html:
Simplemente tiene el *app* que es el que contendrá la aplicación.

Fichero main.js:

```
1 import { createApp } from 'vue'
2 import App from './App.vue'
3
4 createApp(App).mount('#app')
5
```

Es el fichero JS principal. Importa la utilidad *createApp* de la librería *Vue* y el componente *App.vue*. Crea la instancia de Vue con el componente definido en *App.vue* y lo renderiza en el elemento *#app*.

Fichero App.vue:
Es el componente principal de la aplicación, el que contiene el *layout* de la página. Se trata de un *SFC (Single File Component)* y lo que contiene dentro de la etiqueta *<template>* es lo que se renderizará en el div *app* que hay en *index.html*. Si contiene algún otro componente se indica aquí dónde renderizarlo (en este caso).

En el siguiente apartado explicaremos qué es un *SFC* y qué partes lo forman. De momento veamos qué contiene cada sección:

template

```
<template>
  <div id="app">
    <img alt="Vue logo" src="./assets/logo.png">
    <HelloWorld msg="Welcome to Your Vue.js App"/>
  </div>
</template>
```

Muestra la imagen del logo (las imágenes y otros ficheros como ficheros .css se guardan dentro de **/src/assets/**) y el subcomponente *HelloWorld*.

script

```
<script>
import HelloWorld from './components/HelloWorld.vue'

export default {
  name: 'app',
  components: {
    HelloWorld
  }
}
</script>
```

Importa y registra el componente *HelloWorld* que se muestra en el template.

style
Aquí se definen los estilos de este componente. Como la etiqueta NO tiene el atributo *scoped* (`<style scoped>`) significa que los estilos aquí definidos se aplicarán a TODOS los componentes.

Fichero components/HelloWorld.vue:
Es el componente que muestra el texto que aparece bajo la imagen. Recibe como parámetro el título a mostrar. Veamos qué contiene cada sección:

template

```
<template>
  <div class="hello">
    <h1>{{ msg }}</h1>
    <p>
      For a guide and recipes on how to configure / customize this project,<br>
      check out the
      <a href="https://cli.vuejs.org" target="_blank" rel="noopener">vue-cli documentation</a>.
    </p>
    <h3>Installed CLI Plugins</h3>
    <ul>
       ...
  </div>
</template>
```

Muestra el *msg* recibido como parámetro y varios apartados con listas.

script

```
<script>
export default {
  name: 'HelloWorld',
  props: {
    msg: String
  }
}
</script>
```

Recibe el parámetro *msg* que es de tipo String.

style
Aquí la etiqueta SÍ tiene el atributo *scoped* (`<style scoped>`) por lo que los estilos aquí definidos se aplicarán sólo a este componente.

3.3 SFC (*Single File Component*)

Declarar los componentes con `app.component()` en el fichero JS de la instancia como hicimos en el tema anterior genera varios problemas:

- Los componentes así declarados son globales a la aplicación por lo que sus nombres deben ser únicos

- El HTML del template está en ese fichero en medio del JS lo que lo hace menos legible y el editor no lo resalta adecuadamente (ya que espera encontrar código JS no HTML)
- El HTML y el JS del componente están juntos pero no su CSS
- No podemos usar fácilmente herramientas para convertir SCSS a CSS, ES2015 a ES5, etc
- Nuestro fichero crece rápidamente y nos encontramos con código *spaguetti*

Por tanto eso puede ser adecuado para proyectos muy pequeños pero no lo es cuando estos empiezan a crecer.

La solución es guardar cada componente en un único fichero (SFC), que tendrá extensión **.vue**. Estos ficheros contienen 3 secciones diferentes:

- <template>: contiene todo el HTML del componente
- <script>: con el JS del mismo
- <style>: donde pondremos el CSS del componente

Aunque esto va contra la norma de tener el HTML, JS y CSS en ficheros separados en realidad están separados en diferentes secciones y tenemos la ventaja de tener en un único fichero todo lo que necesita el componente.

La mayoría de editores soportan estos ficheros instalándoles algún plugin, (como *Volar* para Visual Studio Code) por lo que el resaltado de las diferentes partes es correcto. Además **vue-cli** integra *Webpack* de forma que podemos usar ES2015 y los preprocesadores más comunes (SASS, Pug/Jade, Stylus, ...) y ya se se traducirá automáticamente el código a ES5, HTML5 y CSS3.

3.3.1 Secciones de un Single File Component

Veamos en detalle cada una de las secciones del SFC.

<template>

Aquí incluiremos el HTML que sustituirá a la etiqueta del componente. Recuerda que en las versiones anteriores a Vue3 dentro sólo puede haber un único elemento HTML (si queremos poner más de uno los incluiremos en otro que los englobe).

Si el código HTML a incluir en el template es muy largo podemos ponerlo en un fichero externo y vincularlo en el template, así nuestro SFC queda más pequeño y legible:

```
<template src="./myComp.html">
</template>
```

Respecto al lenguaje, podemos usar HTML (la opción por defecto) o PUG que es una forma sencilla de escribir HTML. Lo indicamos como atributo de <template>:

```
<template lang="pug">
...
```

<script>

Aquí definimos y exportamos el componente, que será un objeto con diferentes propiedades. Si utiliza subcomponentes hay que importarlos antes de definir el objeto y registrarlos dentro de este.

Entre las propiedades que puede tener el objeto están:

- **name**: el nombre del componente. Es recomendable ponerlo, aunque sólo es obligatorio en caso de componentes recursivos
- **components**: aquí registramos componentes hijos que queramos usar en el *template* de este componente (debemos haber importado previamente los ficheros *.vue* que los contienen). En el *template* usaremos como etiqueta el nombre con que lo registramos aquí
- **props**: donde registramos los parámetros que nos pasa el componente padre como atributos de la etiqueta que renderiza este componente
- **data**: función que devuelve un objeto con todas las variables locales del componente
- **methods**: objeto con los métodos del componente
- **computed**: aquí pondremos las variables calculadas del componente. Lo veremos en detalle en las siguientes unidades.
- **created()**, **mounted()**, ...: funciones *hook* que se ejecutan al crearse el componente, al montarse, ... Aquí pondremos el código que queremos que se ejecute al cargar un componente, como pedir a la BBDD los datos que necesita. Veremos los diferentes *hooks* en las siguientes unidades.
- **watch**: si queremos observar manualmente cambios en alguna variable y ejecutar código como respuesta a ellos (recuerda que Vue ya se encarga de actualizar la vista al cambiar las variables y viceversa). Lo veremos en detalle en las siguientes unidades.
- ...

<style>

Aquí pondremos estilos CSS que se aplicarán al componente. Podemos usar CSS, SASS o PostCSS. Si queremos importar ficheros de estilo con `@import` deberíamos guardarlos dentro de la carpeta *assets* de nuestra aplicación.

Si la etiqueta incluye el atributo ***scoped*** estos estilos se aplicarán únicamente a este componente (y sus descendientes) y no a todos los componentes de nuestra aplicación. Si tenemos estilos que queremos que se apliquen a toda la aplicación y otros que son sólo para el componente y sus descendientes pondremos 2 etiquetas <style>, una sin el atributo *scoped* y otra con él.

La forma más común de asignar estilos a elementos es usando clases. Para conseguir que su estilo cambie fácilmente podemos asignar al elemento clases dinámicas que hagan referencia a variables del componente. Ej.:

```vue
<template>
  <p :class="[decoration, {weight: isBold}]">Hi!</p>
</template>

<script>
export default {
  data() {
    return {
      decoration: 'underline',
      isBold: true
    }
  }
}
</script>

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

El párrafo tendrá la clase indicada en la variable `decoration` (en este caso *underline*) y además como el valor de `isBold` es verdadero tendrá la clase *weight*. Hacer que cambien las clases del elemento es tan sencillo como cambiar el valor de las variables.

Podemos ver las diferentes maneras de asignar clases a los elementos HTML en la documentación de Vue.

Igual que vimos en la etiqueta <template>, si el código de los estilos es demasiado largo podemos ponerlo en un fichero externo que vinculamos a la etiqueta con el atributo *src*.

Custom blocks

Además de estos 3 bloques un SFC puede tener otros bloques definidos por el programador para, por ejemplo, incluir la documentación del componente o sus test unitarios:

```vue
<custom1 src="./unit-test.js">
    Aquí podríamos incluir la documentación del proyecto
</custom1>
```

3.4 Añadir nuevos paquetes y plugins

Si queremos usar un nuevo paquete en nuestra aplicación lo instalaremos con *npm*:

```
npm install nombre-paquete
```

Este comando sólo instala el paquete en *node-modules*. Para que lo añada a las dependencias del *package.json* le pondremos la opción `--save` o `-S` (si se trata de una dependencia de producción) o bien `--dev` o `-D` (si es una dependencia de desarrollo). Ej.:

```
npm install -S axios
```

Para usarlo en nuestros componentes debemos importarlo y registrarlo tal y como se indique en su documentación. Lo normal es hacerlo en el ***main.js*** (o en algún fichero JS que importemos en *main.js* como en el caso de los plugins) si queremos poderlo usar en todos los componentes.

3.4.1 Bootstrap

Podemos utilizar *Bootstrap 5* directamente en Vue ya que esta versión no necesita de la librería *jQuery*.

Para usarlo simplemente lo instalaremos como una dependencia de producción y después lo añadimos al fichero `src/main.js`:

```
import "bootstrap/dist/css/bootstrap.min.css"
import "bootstrap"
```

Para usar los iconos de *Bootstrap 5* debemos importar el css y ya podemos incluir los iconos en etiquetas *<i>* como se explica en la documentación de Bootstrap. Por ejemplo, incluimos en el *<style>* del componente **App.vue**:

```
@import url("https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.7.2/font/bootstrap-icons.css");
```

y donde queramos incluir el icono de la papelera, por ejemplo, incluimos:

```
<i class="bi bi-trash"></i>
```

Respecto a los componentes de *Bootstrap*, para que funcionen sólo tenemos que usar los atributos `data-bs-`, por ejemplo para hacer un botón colapsable haremos:

```
<button
  class="btn btn-primary"
  data-bs-target="#collapseTarget"
  data-bs-toggle="collapse">
  Bootstrap collapse
</button>
<div class="collapse py-2" id="collapseTarget">
  This is the toggle-able content!
</div>
```

En lugar de usar atributos *data-bs-* podemos *envolver* los componentes bootstrap en componentes Vue como se explica en muchas páginas, como Using Bootstrap 5 with Vue 3.

3.5 Crear un nuevo componente

Creamos un nuevo fichero en **/src/components** (o en alguna subcarpeta dentro) con extensión *.vue*. Donde queramos usar ese componente debemos importarlo y registrarlo como hemos visto con *HelloWorld* (y como se explica en el artículo de los *Single File Components*).

```
import CompName from './CompName.vue'

export default {
  ...
  components: {
    'comp-name': CompName
  }
  ...
}
```

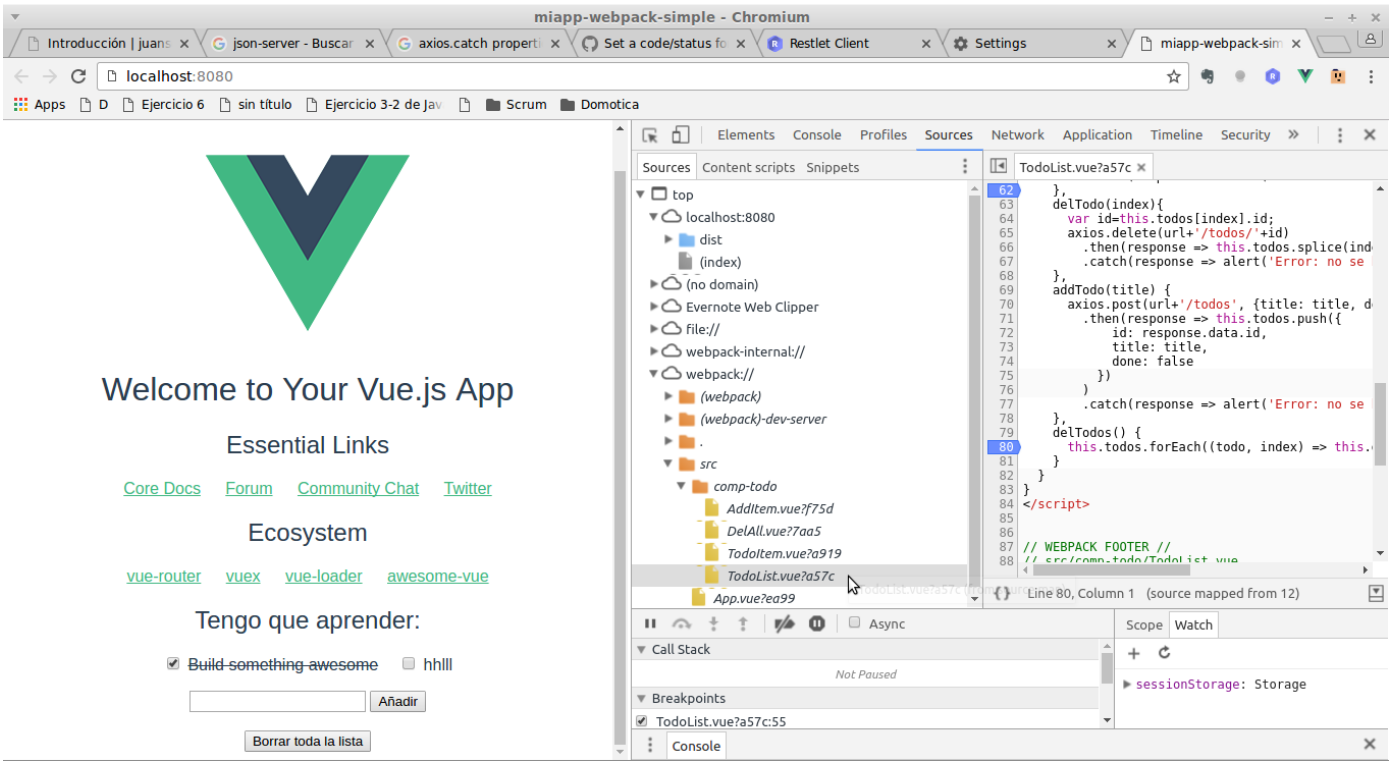
Y ya podemos incluir el componente en el HTML:

```
<comp-name ...> ... </comp-name>
```

3.6 Depurar el código en la consola

Podemos seguir depurando nuestro código, poniendo puntos de interrupción y usando todas las herramientas que nos proporciona la consola mientras estamos en modo de depuración (si hemos abierto la aplicación con `npm run serve`).

Para localizar nuestros fichero varemos que en nuestras fuentes de software aparece **webpack** y dentro nuestras carpetas con el código (**src**, ...):



Recordad que si hemos instalado las **Vue DevTools** tenemos una nueva pestaña en la consola desde la que podemos ver todos nuestros componentes con sus propiedades y datos: