

Primeros pasos

Vue.js



Esta librería surge como creación de Evan You (trabajaba en Google Lab). Evan You se encargaba de desarrollar prototipos con Angular. Angular era un fw muy estricto y pesado para hacer prototipos en los que no era necesario toda la potencia de Angular.

Surgió en 2014 como experimento.

Contenidos

Vue es un framework progresivo	3
Componentes	4
Entorno de desarrollo y CLI	5
Primer componente	8
Single File Component	12
Data y props	13
Computed properties y watchers	18
Ciclo de vida	22
Comunicación entre componentes	24
Bus de eventos	29
Componentes asíncronos	30
Sintaxis en los templates	32
HTML y PUG	33
CSS, SASS o POSTCSS	35

Bindings	38
Directivas	42
Filtros	47
Transiciones	48
Servicios externos. Fetch.	49
Axios	54

Vue es un framework progresivo

Sirve para construir interfaces de usuario, la parte frontend.

Características:

- Accesible: mucha documentación.
- Versátil y escalable: núcleo pequeño y escala mediante pluggins.
- Mantenidos por la organización (core).

Vue Core. Framework progresivo	Vue-router Plugin para la gestión de rutas de nuestra SPA basada en Vue	Vuex Implementación del patrón de arquitectura Flux para Vue
Vue CLI Simple interfaz de línea de comandos para el scaffolding de proyectos basados en Vue	Vue Server Renderer Plugin para hacer que nuestra SPA corra tanto servidor como en el cliente	Vue Devtools Extensión de Chrome para debuggear aplicaciones basadas en Vue

- Mantenido por otros desarrolladores.

Vue-resource Cliente HTTP para conectar con servicios externos en nuestras aplicaciones basadas en Vue	Vue-i18n Plugin para la internacionalización o multi idioma de nuestra app basada en Vue	Nuxt Framework para crear aplicaciones universales con Vue (Webpack+Babel+Vue+Vue-router+Vuex)
Vuetify Framework con componentes basados en Material Design para Vue	Framework 7 Framework para realizar aplicaciones para Android o iOS basadas en Vue	Weex Framework para hacer aplicaciones nativas basadas en Vue. Comenzado por Alibaba, y adoptado por la Apache Software Foundation

- Reactivo.
- Optimizado: core que ocupa 74KB.
- Gran comunidad.
- Licencia MIT (OpenSource)
- Proyectos realizados con **Vue**.

Gitlab Control de versiones, administración de Issues, Code Review, Integración continua, https://gitlab.com/	Stylesage Big-data para moda https://stylesage.co/	Alibaba El mercado online mayorista más grande del mundo https://spanish.alibaba.com/
Sainsbury's Jobs Ofertas de trabajo para la cadena de supermercados Sainsbury's https://sainsburys.jobs/	Team Work Gestor de proyectos online para aumentar la productividad, comunicación y demás. https://www.teamwork.com	Codeship Integración continua, despliegues, ... de forma segura https://codeship.com/

Componentes

Con la nueva versión estándar HTML5 se introdujo el concepto de componente. Los fw como Angular o Vue tienen su propia definición de Web Component y no siguen al pie de la letra la definición de la W3C.

Qué es un componente

Un elemento con una serie de **propiedades, estados y funciones**.



We're not designing pages, we're designing systems of components

Stephen Hay

Un componente se podría equiparar a un *átomo*. Si unimos los componentes, podemos obtener *moléculas* (composición de átomos, de componentes). Si seguimos componiendo podríamos obtener *organismos*, y si seguimos componiendo organismos podríamos obtener *templates*.

Podemos concluir, que una web es un **árbol de componentes**.

Características

- Pequeños
- Contenidos
- Encapsulados: código fuente, estilos y HTML.
- Independientes
- Reutilizables
- Fácil testeo

Estructura

Un componente está formado por una serie de:

- Propiedades
- Datos
- HTML (plantilla)
- Métodos

Entorno de desarrollo y CLI

No necesitamos nada. Sólo necesitamos la propia librería (fichero JS) que lo podemos obtener de un CDN por ejemplo:

<https://unpkg.com/vue@2.6.12/dist/vue.js>

Pero tenemos **CLI (Interfaz de línea de comandos)**. Nos da una serie de características y de utilidades. **VUE-CLI. Se define como un CLI para scaffolding de proyectos VUE.**

El CLI está basado en templates con distintas configuraciones. Por defecto hay 6 pero nosotros podemos hacer la que queramos.

Tenemos opciones simples y más avanzadas. El que nosotros utilizaremos es el que está basado en **PWA (Progressive Web Apps)**. Una **PWA** combina lo mejor de la Web y lo mejor de las apps. Están disponibles para los usuarios a partir de la primera visita en una pestaña del navegador y no requieren instalación. Se cargan rápidamente, incluso en redes débiles o puede utilizarse de forma offline, envía notificaciones push, tiene un icono en la pantalla principal y se carga como experiencia de pantalla completa y de primer nivel.

Una PWA se puede visualizar en cualquier dispositivo, tiene un sistema de cacheo que permite que vaya muy fluida, ...

Simple Es el template más simple en un fichero HTML	Browserify-simple Basado en Browserify, ofrece una configuración básica de este con vuetyfy	Browserify Ofrece una configuración completa basada en Browserify, añadiendo configuraciones y tareas para el hot-reload, linting y test.
Webpack-simple Basado en Webpack, ofrece una configuración básica de este junto a vue-loader	Webpack Ofrece una configuración completa basada en Webpack + vue-loader, añadiendo configuraciones y tareas para el hot-reload, linting, test, y extracción de CSS	PWA Creado por Addy Osmani, configura nuestra APP para que sea una PWA basada en Webpack + vue-loader 

También podemos crear nuestros propios templates!

Instalación

Para instalar **vue-cli**, debemos tener instalado antes **NodeJS**, a ser posible en una versión $\geq 6.x$.

```
inma@MacBook-Pro-de-Inmaculada ~ % sudo npm install -g @vue/cli
```

Vamos a utilizar el template de **PWA** como hemos dicho antes. Este template se basa en **vue-loader**. Es un plugging mantenido por el core de vue y trabaja con Webpack (añade una serie de funcionalidades). Nos permite utilizar **Single File components**, todo está contenido en un fichero.

Vamos a crear nuestro primer proyecto de ejemplo, para ello debemos instalar:

```
inma@MacBook-Pro-de-Inmaculada dwec_2021_vue % sudo npm i -g @vue/cli-init
```

Después ejecutamos:

```
inma@MacBook-Pro-de-Inmaculada dwec_2021_vue % vue init pwa codit
? Project name codit
? Project short name: fewer than 12 characters to not be truncated on homescreens (default: same as name) codit
? Project description First Vue.js project
? Author Inma Gijón <igijoninf@gmail.com>
? Vue build standalone
? Install vue-router? Yes
? Use ESLint to lint your code? Yes
? Pick an ESLint preset Standard
? Setup unit tests with Karma + Mocha? Yes
? Setup e2e tests with Nightwatch? Yes

vue-cli · Generated "codit".

To get started:

  cd codit
  npm install
  npm run dev

Documentation can be found at https://vuejs-templates.github.io/webpack

inma@MacBook-Pro-de-Inmaculada dwec_2021_vue % []
```

Pwa es el nombre del template que usaremos.

Codit es el nombre del proyecto.

Esta es la estructura que nos ha creado en el ejemplo:

- **build**: todo el código de configuración del entorno de desarrollo, webpack ...
- **config**: variables de entorno.
- **src**: nuestra aplicación, componentes, css, assets, rutas, ...
- **static**: lo que esté en este directorio, nuestro template no lo procesa, sólo procesa lo que está en assets.
- **test**: test unitarios.

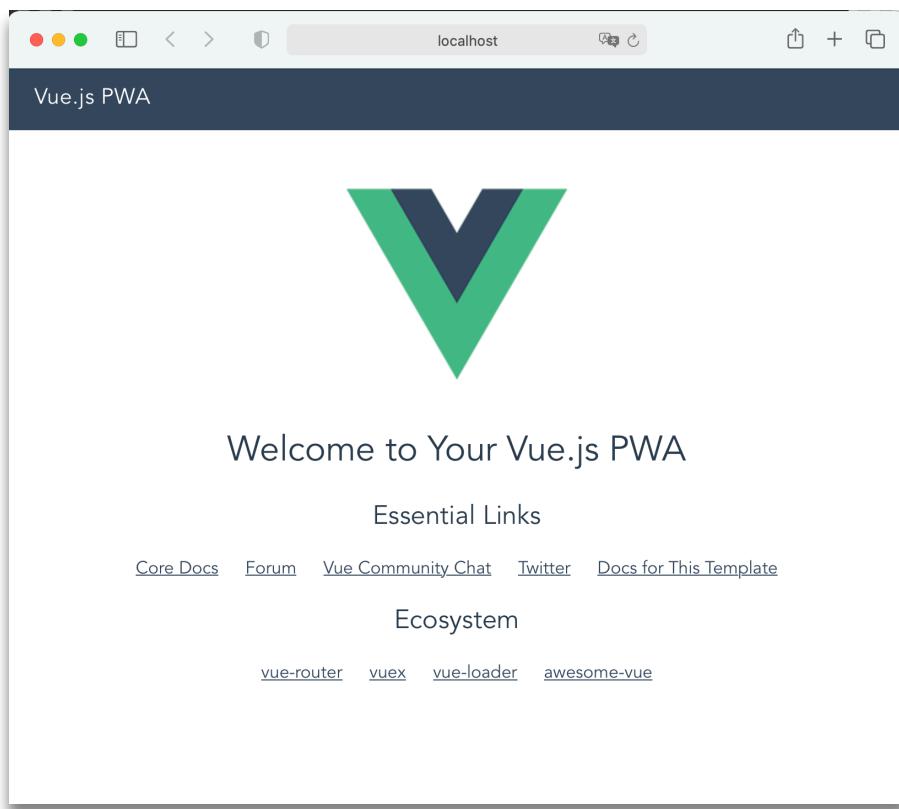
DWEC_2021_VUE	
✓	codit
>	build
>	config
✓	src
✓	assets
🖼	logo.png
✓	components
▼	Hello.vue
✓	router
JS	index.js
▼	App.vue
JS	main.js
>	static
>	test
β	.babelrc
⚙	.editorconfig
⚙	.eslintignore
⚙	.eslintrc.js
❖	.gitignore
JS	.postcssrc.js
▷	index.html
{}	package.json
ⓘ	README.md

Lanzar el proyecto inicial

```
inma@MacBook-Pro-de-Inmaculada codit % sudo npm install
```

```
inma@MacBook-Pro-de-Inmaculada codit % sudo npm start
```

Esto abre en <http://localhost:8080> la aplicación:



Primer componente

¿Qué es un componente?

Los componentes web son un concepto relativamente nuevo, aunque realmente llevas utilizándolos mucho tiempo. Un componente web se trata de un elemento de la página web, con **vista**, **estilos** y **lógica**, que puedes reutilizar a lo largo de tu página web.

Realmente todas las etiquetas html son una especie de componentes web porque se componen también de esas partes.

Por ejemplo al crear un elemento html <input>, realmente estás creando una especie de componente web porque se compone del html que se pinta, los estilos que se pintan por defecto y a lógica de poder escribir.

Pues ahora imagínate que puedes crear tu propio componente con un html mucho más complejo, con más estilos CSS que los de por defecto y con una lógica mucho más compleja. Imagina poder crear un componente web llamado calendario. Simplemente poniendo en la página que lo necesites su etiqueta html, por ejemplo <calendar/> podrías crear un calendario totalmente funcional incluso con la lógica de pasar los meses y los años.

Los componentes no son únicos de Vue, es más, desde hace poco los puedes crear de forma nativa en javascript, pero Vue lo que hace es facilitarnos la tarea de crear los componentes web escribiendo menos código y pudiendo hacer cosas mucho más complejas.

Creación del componente

Lo primero que necesitamos es instancia Vue, para poder arrancarlo.

Todas las aplicaciones que estén desarrolladas con Vue comienzan creando una instancia de Vue.

En **main.js** podemos ver la primera instancia creada de **Vue**:

```
js main.js  x
ejemplo > src > js main.js > ...
1 // The Vue build version to load with the `import` command
2 // (runtime-only or standalone) has been set in webpack.base.conf with an alias.
3 import Vue from 'vue'
4 import App from './App'
5 import router from './router'
6
7 Vue.config.productionTip = false
8
9 /* eslint-disable no-new */
10 new Vue({
11   el: '#app',
12   router,
13   template: '<App/>',
14   components: { App }
15 })
16
```

```

▼ App.vue ×
codit > src > ▼ App.vue > {} "App.vue" > ⚒ script > [e] default
1   <template>
2     <div id="app">
3       <header>
4         <span>Vue.js PWA</span>
5       </header>
6       <main>
7         
8         <router-view></router-view>
9       </main>
10    </div>
11  </template>
12
13  <script>
14  export default [
15    {
16      name: 'app'
17    }
18  ]</script>
19
20  <style>
21  body {
22    margin: 0;
23  }
24
25  #app {
26    font-family: 'Avenir', Helvetica, Arial, sans-serif;
27    -webkit-font-smoothing: antialiased;
28    -moz-osx-font-smoothing: grayscale;
29    color: #2c3e50;
30  }
31
32  main {
33    text-align: center;
34    margin-top: 40px;
35  }
36
37  header {
38    margin: 0;
39    height: 56px;
40    padding: 0 16px 0 24px;
41    background-color: #35495E;
42    color: #ffffff;
43  }
44
45  header span {
46    display: block;
47    position: relative;
48    font-size: 20px;
49    line-height: 1;
50    letter-spacing: .02em;
51    font-weight: 400;
52    box-sizing: border-box;
53    padding-top: 16px;
54  }</style>
55

```

id *app* en el cual todo lo demás:

```

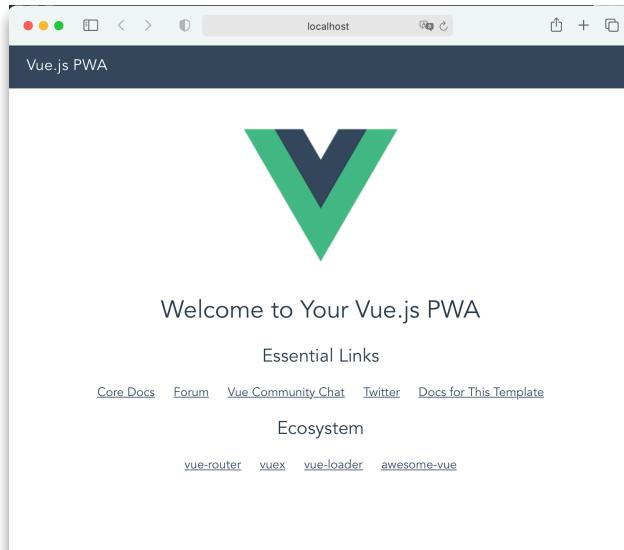
▼ App.vue           JS index.js ×
codit > src > router > JS index.js > [e] default
1   import Vue from 'vue'
2   import Router from 'vue-router'
3   import Hello from '@/components/Hello'
4
5   Vue.use(Router)
6
7   export default new Router({
8     routes: [
9       {
10         path: '/',
11         name: 'Hello',
12         component: Hello
13       }
14     ]
15   })
16

```

El elemento principal es `#app`, este identificador coincide con la etiqueta de nuestro html. Es el punto en el que comienza a inyectar todos los componentes.

Por defecto, Vue tiene una serie de configuraciones, como puede ser **productionTip = false**, quitar el Tip de producción, para que en la consola no nos salga el mensaje cuando detecta que no estamos en producción.

En `index.html` podemos ver el elemento con se va a inyectar



codit > src > components > ▼ Hello.vue > {} "Hello.vue" > ⏪ template

```
1  <template>
2    <div class="hello">
3      <h1>{{ msg }}</h1>
4      <h2>Essential Links</h2>
5      <ul>
6        <li><a href="https://vuejs.org" target="_blank" rel="noopener">Core Docs</a></li>
7        <li><a href="https://forum.vuejs.org" target="_blank" rel="noopener">Forum</a></li>
8        <li><a href="http://chat.vuejs.org/" target="_blank" rel="noopener">Vue Community Chat</a></li>
9        <li><a href="https://twitter.com/vuejs" target="_blank" rel="noopener">Twitter</a></li>
10       <li><a href="http://vuejs-templates.github.io/webpack/" target="_blank" rel="noopener">Docs for This Template</a></li>
11     </ul>
12     <h2>Ecosystem</h2>
13     <ul>
14       <li><a href="http://router.vuejs.org/" target="_blank" rel="noopener">vue-router</a></li>
15       <li><a href="http://vuex.vuejs.org/" target="_blank" rel="noopener">vuex</a></li>
16       <li><a href="http://vue-loader.vuejs.org/" target="_blank" rel="noopener">vue-loader</a></li>
17       <li><a href="https://github.com/vuejs/awesome-vue" target="_blank" rel="noopener">awesome-vue</a></li>
18     </ul>
19   </div>
20 </template>
21
22 <script>
23 export default {
24   name: 'hello',
25   data () {
26     return {
27       msg: 'Welcome to Your Vue.js PWA'
28     }
29   }
30 }
31 </script>
32
33 <!-- Add "scoped" attribute to limit CSS to this component only -->
34 <style>
35 h1, h2 {
36   font-weight: normal;
37 }
38
39 ul {
40   list-style-type: none;
41   padding: 0;
42 }
43
44 li {
45   display: inline-block;
46   margin: 0 10px;
47 }
48
49 a {
50   color: #35495E;
51 }
52 </style>
53
```

```
index.html <--> index.html <--> html <--> body

18  <meta name="apple-mobile-web-app-title" content="codit">
19  <link rel="apple-touch-icon" href="<%= htmlWebpackPlugin.files.publicPath %>static/img/icons/apple-touch-icon-152x152.png">
20  <link rel="mask-icon" href="<%= htmlWebpackPlugin.files.publicPath %>static/img/icons/safari-pinned-tab.svg" color="#4DBA87">
21  <!-- Add to home screen for Windows -->
22  <meta name="msapplication-TileImage" content="<%= htmlWebpackPlugin.files.publicPath %>static/img/icons/msapplication-icon-144x144.png">
23  <meta name="msapplication-TileColor" content="#000000">
24  <!-- for (var chunk of webpack.chunks) {
25    for (var file of chunk.files) {
26      if (file.match(/(.js|css)$/)) %>
27        <link rel="<%= chunk.initial?'preload':'prefetch'" href="<%= htmlWebpackPlugin.files.publicPath + file %>" as="<%>= file.match(/(.css)?(script|style)/)?>= style' %>"} %>
28  </head>
29  <body>
30    <noscript>
31      This is your fallback content in case JavaScript fails to load.
32    </noscript>
33    <div id="app"></div>
34    <!-- Todo: only include in production -->
35    <%= htmlWebpackPlugin.options.serviceWorkerLoader %>
36    <!-- built files will be auto injected -->
37  </body>
38 </html>
39
```

Si nos fijamos, lo que está ocurriendo es que **main.js** tiene la instancia de **Vue** y en **index.html** está cargando en el *div* con id *app*, el componente *App*. El

componente **App**, está implementado en el fichero **App.vue**. Después veremos en detalle las características de estos ficheros, pero básicamente está inyectando en ese punto del **index.html**, el componente **App**.

```
JS main.js  X
codit > src > JS main.js > ...
1 // The Vue build version to load with the `import` command
2 // (runtime-only or standalone) has been set in webpack.base.conf with an alias.
3 import Vue from 'vue'
4 import App from './App'
5 import router from './router'
6
7 Vue.config.productionTip = false
8
9 /* eslint-disable no-new */
10 new Vue({
11   el: '#app',
12   router,
13   template: '<App/>',
14   components: { App }
15 })
16
```

El componente **App**, en su template carga un header y un main. Dentro del main, por defecto está cargando un **router-view** (ya veremos lo que hace en detalle más adelante). Si nos fijamos, en el fichero **router/index.js** está cargando el componente **Hello** en la ruta **/**. Por lo tanto, en esta aplicación **ejemplo creada por defecto se muestra (App.vue) el header, el main y dentro de éste un logo y router-view que a su vez por defecto carga dentro el componente Hello**.

Single File Component

JSX para Vue.

Los SFC son ficheros .vue con todo lo relacionado para la implementación de un componente. Están proporcionados por el plugin **vue-loader**.

Estos ficheros, como podemos ver en **App.vue**, tienen tres grandes bloques:

- **Template:** HTML
- **Script:** ES6
- **Style:** CSS
- **Custom blocks:** por ejemplo para documentar el componente.

Yo podría tener el código en ficheros externos como por ejemplo:

```
<template lang='html' lang='./template.html'>
</template>

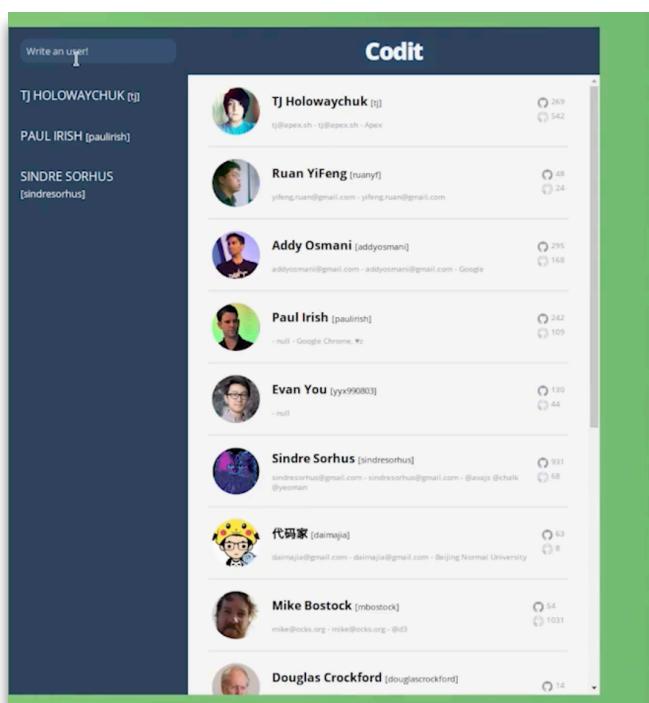
<script src='./script.js'>
</script>

<style lang='css' src='./styles.css'>
</style>

<custom1 src='./unit-test.js'>
</custom1>
```

Data y props

La aplicación ejemplo que vamos a desarrollar va a ser una aplicación que consume el API de GitHub.



Vamos a tener los siguientes componentes: **un componente app, un componente de búsqueda, un componente para el logotipo, un componente para los bookmarks y un componente para los desarrolladores.**

Vamos a ir modificando cosas del proyecto generado para crear la aplicación.

En **main.js**, se define la instancia de Vue, con el elemento principal

en el que se inyectará todo (**codit**) y se inyectará exactamente el componente **CoApp**.

```
JS main.js ×
codit > src > JS main.js > ...
1 // The Vue build version to load with the `import` command
2 // (runtime-only or standalone) has been set in webpack.base.conf with an alias.
3 import Vue from 'vue'
4 import CoApp from '@/App'
5
6 Vue.config.productionTip = false
7
8 /* eslint-disable no-new */
9 new Vue({
10   el: '#codit',
11   template: '<co-app/>',
12   components: { CoApp }
13 })
14
```

```
▼ App.vue ●
codit > src > ▼ App.vue > {} "App.vue"
1 <template lang='html'>
2   <div class='codit'>
3     <div class='container'>
4       <nav class='menu'>
5         <co-search></co-search>
6         <co-bookmarks></co-bookmarks>
7       </nav>
8       <main class='content'>
9         <header class='content__header'>
10          <co-logo></co-logo>
11        </header>
12        <co-developers></co-developers>
13      </main>
14    </div>
15  </div>
16 </template>
17
18 <script>
19 import CoLogo from '@/components/CoLogo'
20 import CoSearch from '@/components/CoSearch'
21 import CoBookmarks from '@/components/CoBookmarks'
22 import CoDevelopers from '@/components/CoDevelopers'
23
24 export default {
25   name: 'CoApp',
26   components: {
27     CoLogo,
28     CoSearch,
29     CoBookmarks,
30     CoDevelopers
31   }
32 }
33 </script>
34
35 <style lang='css'>
36 </style>
37
```

Nuestro componente **CoApp** es el que define nuestros layouts y lo tenemos implementado en el fichero **App.vue**.

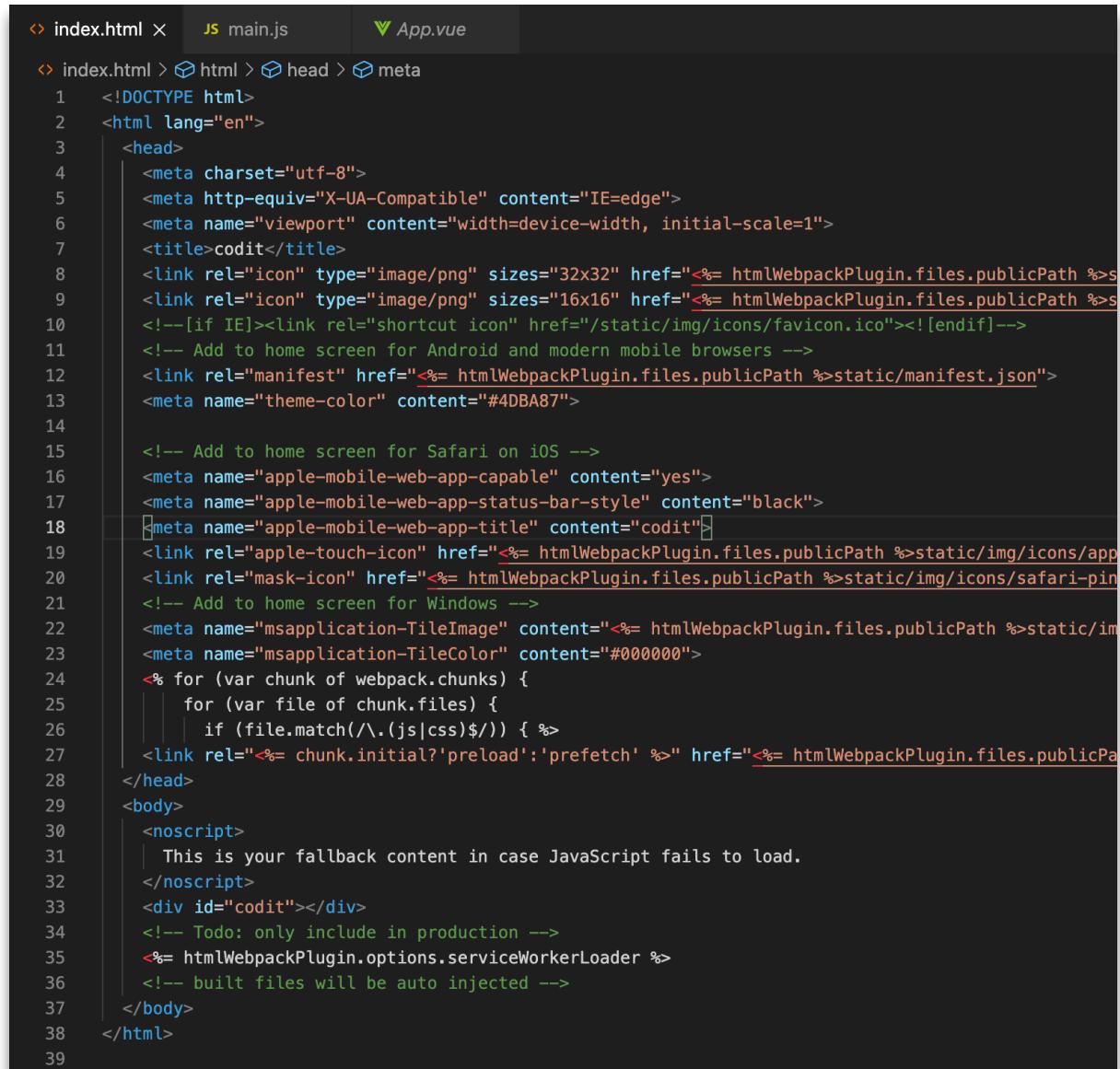
En este componente como podemos ver, están las tres etiquetas:

-Template: aquí tenemos implementado el layout que va a tener por un lado un menú, con un buscador (co-search) y otro componente bookmarks. Después tendremos un main que va a contener un header que tendrá un componente co-logo y un listado de desarrolladores que

se implementará en el componente co-developers (estos componentes los crearemos a continuación).

- **Script:** en la implementación del script, por ahora tendremos el nombre del componente (**CoApp**), lo cual nos permitirá utilizarlo con las etiquetas **co-app** y los componentes que utiliza (los que hemos utilizado en el template) que son: CoLogo, CoSearch, CoBookmarks y CoDeveloper, para ello tendremos que importarlos (aún no los hemos creado pero estarán en la carpeta **components**) y los crearemos a continuación.
- **Style:** aún no hemos introducido estilos.

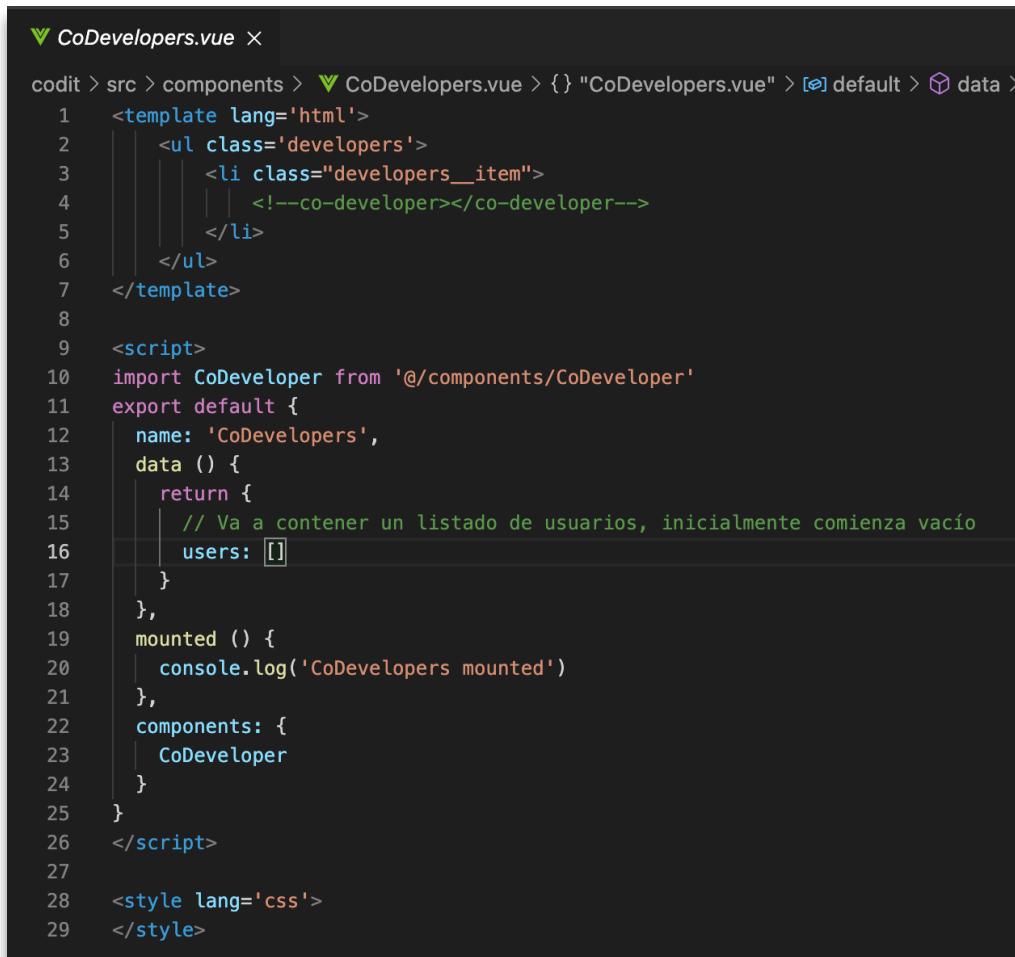
En **index.html** tenemos que llamar al elemento **codit** que hemos indicado en nuestro fichero **main.js** para que se inyecte todo lo demás:



```
index.html × JS main.js App.vue
index.html > html > head > meta
1  <!DOCTYPE html>
2  <html lang="en">
3  | <head>
4  | | <meta charset="utf-8">
5  | | <meta http-equiv="X-UA-Compatible" content="IE=edge">
6  | | <meta name="viewport" content="width=device-width, initial-scale=1">
7  | | <title>codit</title>
8  | | <link rel="icon" type="image/png" sizes="32x32" href="<%= htmlWebpackPlugin.files.publicPath %>s
9  | | <link rel="icon" type="image/png" sizes="16x16" href="<%= htmlWebpackPlugin.files.publicPath %>s
10 | | <!--[if IE]><link rel="shortcut icon" href="/static/img/icons/favicon.ico"><![endif]-->
11 | | <!-- Add to home screen for Android and modern mobile browsers -->
12 | | <link rel="manifest" href="<%= htmlWebpackPlugin.files.publicPath %>static/manifest.json">
13 | | <meta name="theme-color" content="#4DBA87">
14 |
15 | | <!-- Add to home screen for Safari on iOS -->
16 | | <meta name="apple-mobile-web-app-capable" content="yes">
17 | | <meta name="apple-mobile-web-app-status-bar-style" content="black">
18 | | <meta name="apple-mobile-web-app-title" content="codit">
19 | | <link rel="apple-touch-icon" href="<%= htmlWebpackPlugin.files.publicPath %>static/img/icons/app
20 | | <link rel="mask-icon" href="<%= htmlWebpackPlugin.files.publicPath %>static/img/icons/safari-pin
21 | | <!-- Add to home screen for Windows -->
22 | | <meta name="msapplication-TileImage" content="<%= htmlWebpackPlugin.files.publicPath %>static/im
23 | | <meta name="msapplication-TileColor" content="#000000">
24 | | <% for (var chunk of webpack.chunks) {
25 | | | for (var file of chunk.files) {
26 | | | | if (file.match(/\.(js|css)$/)) { %>
27 | | | <link rel="<% chunk.initial?'preload':'prefetch' %>" href="<%= htmlWebpackPlugin.files.publicPa
28 | | }>
29 | | <body>
30 | | | <noscript>
31 | | | | This is your fallback content in case JavaScript fails to load.
32 | | | </noscript>
33 | | | <div id="codit"></div>
34 | | | <!-- Todo: only include in production -->
35 | | | <%= htmlWebpackPlugin.options.serviceWorkerLoader %>
36 | | | <!-- built files will be auto injected -->
37 | | </body>
38 | </html>
39 |
```

Vamos a comenzar a desarrollar los componentes, para ello creamos ficheros con extensión .vue dentro de la carpeta **componente**.

Vamos a ver el componente **CoDevelopers**:



```
▼ CoDevelopers.vue ×
codit > src > components > ▼ CoDevelopers.vue > {} "CoDevelopers.vue" > [o] default > ⚡ data >
1  <template lang='html'>
2    <ul class='developers'>
3      <li class="developers__item">
4        <!--co-developer--></co-developer-->
5      </li>
6    </ul>
7  </template>
8
9  <script>
10 import CoDeveloper from '@/components/CoDeveloper'
11 export default {
12   name: 'CoDevelopers',
13   data () {
14     return {
15       // Va a contener un listado de usuarios, inicialmente comienza vacío
16       users: []
17     }
18   },
19   mounted () {
20     console.log('CoDevelopers mounted')
21   },
22   components: {
23     CoDeveloper
24   }
25 }
26 </script>
27
28 <style lang='css'>
29 </style>
```

Este componente mostrará el listado de desarrolladores, por tanto llamará en cada fila al componente **co-developer**. Por ahora lo tenemos comentado hasta que implementemos cómo llenar esa información.

Vamos a utilizar data que son datos iniciales. Siempre es una función (cuando va en cualquier componente) que va a tener como objeto unos datos iniciales que son los que devuelve la función. En este caso es una función, porque cuando creamos cualquier componente necesita devolver esos datos iniciales.

(Si añadimos **data** a la instancia **Vue** inicial, no es una función porque de esto, sólo se crea una instancia.)

En este caso, el componente **CoDevelopers** va a ser el listado de desarrolladores, por tanto inicialmente **data** devuelve un array de usuarios

vacio. Ya veremos más adelante como rellenarlo. En el template tenemos el listado que se llenará con cada desarrollador.

CoDeveloper va a ser la fila de cada desarrollador.

Para cada desarrollador, vamos a mostrar: nombre, login, número de estrellas GitHub y gists, correo electrónico, ...

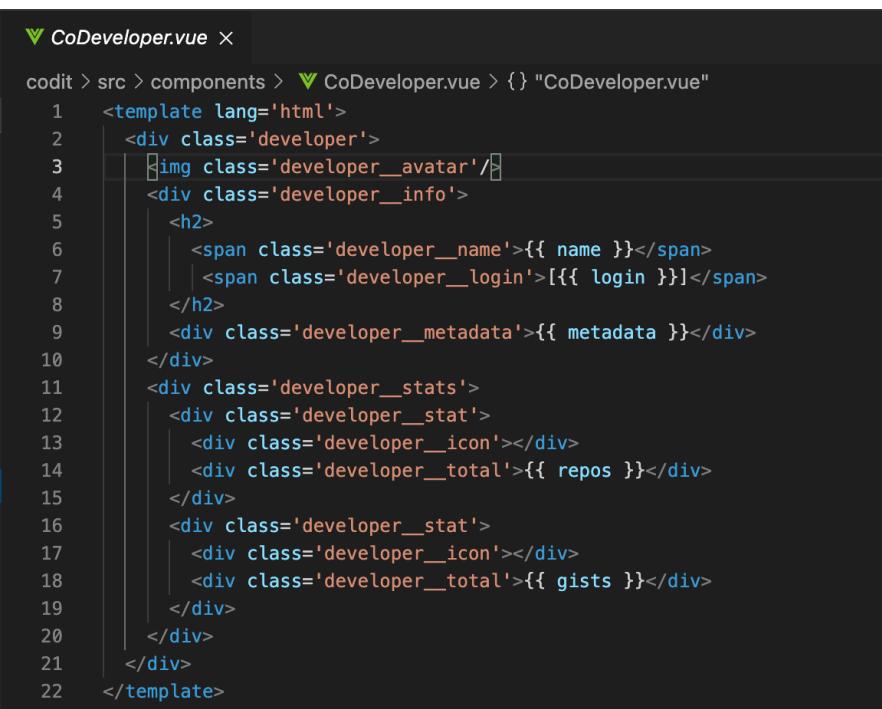
Cómo estos componentes los vamos a reutilizar, es responsabilidad del padre enviar los datos al componente hijo.

Vue es **one-way-down** data flow, lo que significa que todos los datos se envían de arriba a abajo. Un padre puede enviar datos a su hijo a través de las **props**. Las propiedades las recibe el hijo cuando se carga el componente.

En nuestro caso, **CoDevelopers** mandará información a **CoDevelop**.

Recapitulando, data son datos iniciales del componente y props son propiedades que el componente hijo puede recibir del padre y no al revés por la política one-way-down.

Creamos el componente **CoDevelop** con una serie de propiedades que le llegarán desde el padre:



The screenshot shows a code editor with the file 'CoDeveloper.vue' open. The code is a Vue.js template for a developer card. It starts with a template block containing a div with class 'developer'. Inside this div, there is an img element with class 'developer_avatar'. Below it is a div with class 'developer_info' containing an h2 and two spans: one for 'name' and one for 'login'. Another div with class 'developer_metadata' contains 'metadata'. Following this is a div with class 'developer_stats' containing two divs with class 'developer_stat'. Each stat div contains an icon (represented by a placeholder image) and a total value ('repos' or 'gists'). The template concludes with a closing div and template tag.

```
<template lang='html'>
  <div class='developer'>
    <img class='developer_avatar' />
    <div class='developer_info'>
      <h2>
        <span class='developer_name'>{{ name }}</span>
        <span class='developer_login'>[{{ login }}]</span>
      </h2>
      <div class='developer_metadata'>{{ metadata }}</div>
    </div>
    <div class='developer_stats'>
      <div class='developer_stat'>
        <div class='developer_icon'></div>
        <div class='developer_total'>{{ repos }}</div>
      </div>
      <div class='developer_stat'>
        <div class='developer_icon'></div>
        <div class='developer_total'>{{ gists }}</div>
      </div>
    </div>
  </div>
</template>
```

```

▼ CoDeveloper.vue ●
codit > src > components > ▼ CoDeveloper.vue > {} "CoDeveloper.vue" > default > ⚡ props
24  <script>
25  export default {
26    name: 'CoDeveloper',
27    // Las propiedades en este caso las vamos a definir como un objeto con una serie de propiedades,
28    // como por ejemplo avatar
29    props: [
30      avatar: {
31        // Vamos a determinar que el avatar que le llega del padre va a ser una url de la imagen
32        // por tanto va a ser de tipo string, vamos a decidir que sea obligatorio y vamos a validarla
33        // en este caso indicando que el valor que le llega va a comenzar como http y de este modo
34        // sabemos que es una ruta
35        type: String,
36        required: true,
37        validator (value) {
38          return value.startsWith('http')
39        }
40      },
41      name: {
42        type: String,
43        required: true
44      },
45      login: {
46        type: String,
47        required: true
48      },
49      email: {
50        type: String,
51        required: true
52      },
53      company: {
54        type: String,
55        required: false // podríamos no indicarlos
56      },
57      repos: {
58        type: Number
59      },
60      gists: {
61        type: Number
62      }
63    ]
64  }
65  </script>
66
67  <style lang='css'>
68  </style>

```

Computed properties y watchers

Hay casos en los que queremos una propiedad devuelva un procesamiento, en nuestro caso vamos a querer que la información del desarrollador se concatene toda en una cadena, para eso hacemos lo siguiente, utilizando la palabra reservada **computed** al mismo nivel que **props** en el componente **CoDeveloper**.

De este modo, cuando desde la vista, accedamos a la propiedad **metadata** podremos obtener la concatenación de dichas propiedades que ha obtenido del padre.

```
computed: {
  metadata() {
    //Vamos a concatenar tres propiedades
    //A los datos del componente podemos acceder mediante this
    let meta = "";
    if (this.email) {
      meta = `${meta} ${this.email}`;
    }
    if (this.location) {
      meta = `${meta} ${this.location}`;
    }
    if (this.company) {
      meta = `${meta} ${this.company}`;
    }
    return meta;
  },
  othermeta: {
    set() {
      //Va a ejecutar un código cuando en cualquier parte del componente se setee el valor de
      //othermeta
    },
    get() {
      //Va a ejecutar el código cuando se intente obtener un dato
      let meta = "";
      if (this.email) {
        meta = `${meta} ${this.email}`;
      }
      if (this.location) {
        meta = `${meta} ${this.location}`;
      }
      if (this.company) [
        meta = `${meta} ${this.company}`;
      ]
      return meta;
    },
  },
},
};

</script>

<style lang='css'>
</style>
```

```
▼ CoDeveloper.vue ●

codit > src > components > ▼ CoDeveloper.vue > {} "CoDeveloper.vue" > [o] default > ⚡ computed
  40   type: String,
  41   required: true
  42 },
  43   email: {
  44     type: String,
  45     required: true
  46   },
  47   company: {
  48     type: String,
  49     required: false // podríamos no indicarlos
  50   },
  51   repos: {
  52     type: Number
  53   },
  54   gists: {
  55     type: Number
  56   }
  57 },
  58 computed: [
  59   metadata () {
  60     // Vamos a concatenar tres propiedades
  61     // A los datos del componente podemos acceder mediante this
  62     let meta = ''
  63     if (this.email) {
  64       meta = `${meta} ${this.email}`
  65     }
  66     if (this.location) {
  67       meta = `${meta} ${this.location}`
  68     }
  69     if (this.company) {
  70       meta = `${meta} ${this.company}`
  71     }
  72     return meta
  73   }
  74 ],
  75 
```

También podemos decidir cuál va a ser su comportamiento al obtener el dato y al setear el dato. Para ello, podemos crear un objeto con sus funciones **get** y **set**. Lo podemos ver en el ejemplo con **othermeta** que en lugar de ser una función es un objeto.

En nuestro caso nos vamos a quedar con la opción de arriba (**metadata**) porque no vamos a diferenciar entre lo que se hace cuando se setea la variable o cuando se obtiene su valor.

Watchers

¿Qué ocurre si queremos reaccionar a cambios en una variable? En este caso no usaremos ni data, ni props ni computed.

Para ello, Vue nos aconseja **Watchers**. Es un observador que generamos sobre un data, props o computed. De ahí que decimos que Vue es reactiva, reacciona a cambios.

Para ello, definimos un objeto de tipo **watch** para que observe una variable a ver cuándo cambia, por ejemplo la variable **metadata**. Para ello la definimos como clave y le indicamos que el valor nuevo que le asigne se imprima por consola. Con esto estamos consiguiendo que cada vez que **metadata** cambie, reaccionemos al cambio.

Por ejemplo, si tenemos una petición a un servicio externo, reaccionaríamos a ese cambio.

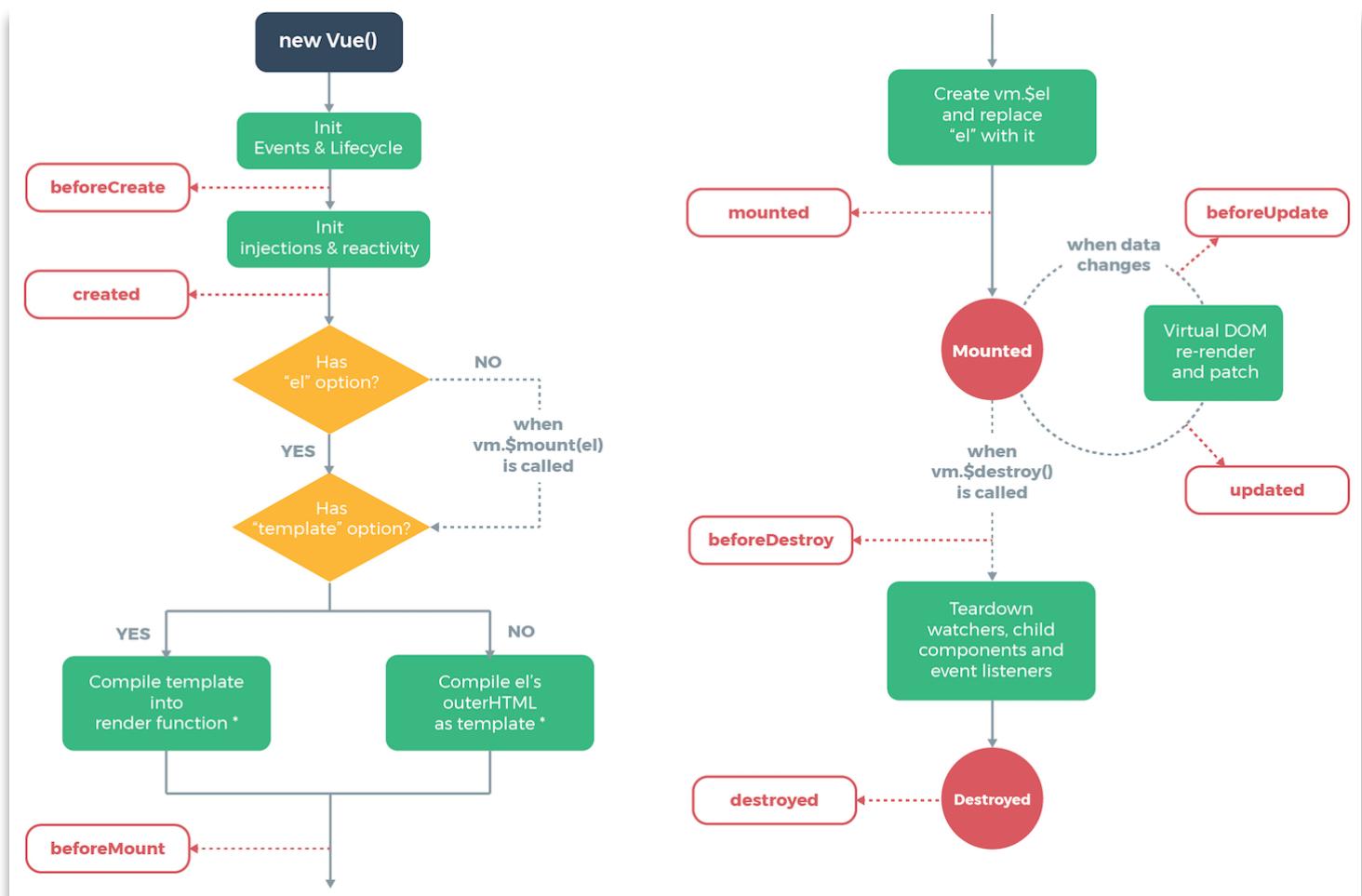
```
▼ CoDeveloper.vue ×
codit > src > components > ▼ CoDeveloper.vue > {} "CoDeveloper.vue" > [?] default > ↴
62   },
63 },
64 computed: {
65   metadata () {
66     // Vamos a concatenar tres propiedades
67     // A los datos del componente podemos acceder mediante this
68     let meta = ''
69     if (this.email) {
70       meta = `${meta} ${this.email}`
71     }
72     if (this.location) {
73       meta = `${meta} ${this.location}`
74     }
75     if (this.company) {
76       meta = `${meta} ${this.company}`
77     }
78     return meta
79   }
80 },
81 watch: {
82   metadata (newValue) {
83     console.log('Metadata:', newValue)
84   }
85 }
86 }
87 </script>
88
89 <style lang='css'>
90 </style>
```

Los **watchers** son costosos y no debemos pasarnos en su uso.

Ciclo de vida

Un componente se crea, pero tiene otros estados y podemos unirnos a un estado determinado y decidir qué hacer en función de los eventos que se disparan.

Lo primero que se ejecuta es la propia instancia de Vue.



Posteriormente se inicializan los eventos y el propio ciclo de vida y hay un hook **beforeCreate** que se dispara. Es el primero de todos. En este momento no tenemos acceso a los datos del componente, porque se ha creado la instancia de Vue pero no nuestro componente con sus datos y propiedades.

Después se inicializan las reactividades y se inyectan dependencias ... y se dispara el hook **create**. Ya se ha inicializado el componente. Sería un buen momento para hacer una petición a un servicio externo por ejemplo.

Una vez inicializado nuestro componente e incrustado en el HTML, se dispara **beforeMount**. Ya tenemos el template de nuestro componente compilado en la función render.

mounted se dispara cuando el componente está montado y se genera la propiedad **\$el**, es decir, nuestro componente ha sido renderizado en el DOM. Otro buen momento para hacer una petición a un servicio externo.

beforeUpdate: se lanza cuando el componente detecta un cambio en sus datos, pero aún nuestro componente no ha sido renderizado de nuevo.

updated: se ha realizado un cambio en algún dato de nuestro componente y se ha realizado el renderizado de nuevo en el virtual DOM.

beforeDestroy: hook que se lanza cuando Vue detecta que el componente debe ser eliminado. Se ejecuta antes de eliminar todas las referencias internas del componente.

destroyed: el componente va a ser eliminado y se han eliminado todas las referencias a los watchers y listeners.

Ejemplo

Nosotros no vamos a utilizar mucho los hooks, pero lo haremos sobre todo cuando hagamos peticiones a servicios externos. En nuestro caso nos interesará el hook **mounted** para enterarnos de cuando se modifican los data del componente por el servicio externo.

```
▼ CoDevelopers.vue ×
codit > src > components > ▼ CoDevelopers.vue > {} "CoDevelopers.vue" > [?] default > ⚙ data >
1  <template lang='html'>
2    <ul class='developers'>
3      <li class="developers__item">
4        |   <!--co-developer></co-developer-->
5      </li>
6    </ul>
7  </template>
8
9  <script>
10 import CoDeveloper from '@/components/CoDeveloper'
11 export default {
12   name: 'CoDevelopers',
13   data () {
14     return {
15       // Va a contener un listado de usuarios, inicialmente comienza vacío
16       users: []
17     }
18   },
19   mounted () {
20     console.log('CoDevelopers mounted')
21   },
22   components: {
23     CoDeveloper
24   }
25 }
26 </script>
27
28 <style lang='css'>
29 </style>
```

En un futuro, el componente **CoDevelopers** será el que haga peticiones a un servicio externo por lo tanto mostraremos la implementación de ejemplo ahí preparándola para un futuro.

Comunicación entre componentes

Vue se define como librería **One-way down Data Flow**. Comunicación de padres a hijos.

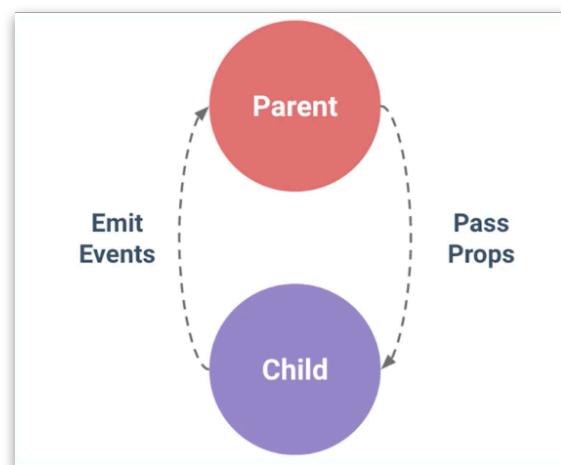
Para comunicar de hijos a padres:

```

▼ App.vue ×
codit > src > ▼ App.vue > {} "App.vue"
12   </header>
13   <co-developers></co-developers>
14   </main>
15   </div>
16 </div>
17 </template>
18
19 <script>
20 import CoLogo from '@/components/CoLogo'
21 import CoSearch from '@/components/CoSearch'
22 import CoBookmarks from '@/components/CoBookmarks'
23 import CoDevelopers from '@/components/CoDevelopers'
24
25 export default {
26   name: 'CoApp',
27   methods: {
28     onSearch (searchCriteria) {
29       console.log('Search', searchCriteria)
30     }
31   },
32   components: {
33     CoLogo,
34     CoSearch,
35     CoBookmarks,
36     CoDevelopers
37   }
38 }
39 </script>
40
41 <style lang='css'>
42 </style>
43

```

-Para



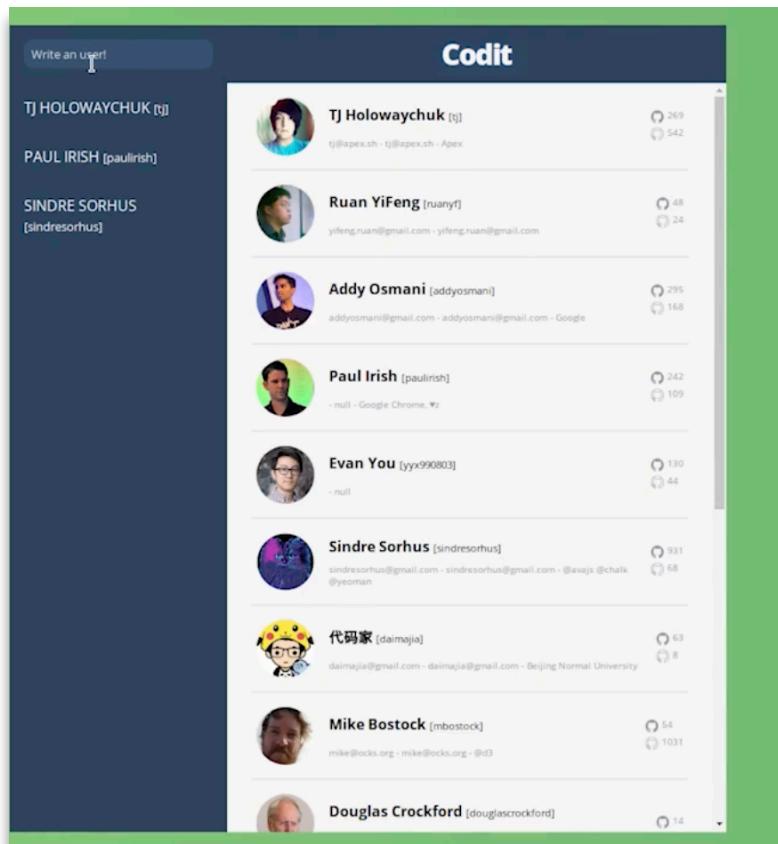
comunicar un padre con un hijo pasaremos los valores a través de las propiedades.

-Para comunicar un hijo con un padre lo haremos emitiendo eventos.

Ejemplo componente de búsqueda

Vamos a probar esta comunicación en nuestra aplicación a través del componente de búsqueda.

En un futuro este componente me permitirá introducir el nombre de un usuario y que GitHub nos dé la información de dicho usuario.



Creamos un componente **CoSearch.vue**:

```
▼ CoSearch.vue ×
codit > src > components > ▼ CoSearch.vue > {} "CoSearch.vue"
1  <template lang='html'>
2    <input
3      name='search'
4      type='text'
5      placeholder='Write an user!'
6      class='search'
7      v-model='criteria'
8    />
9  </template>
10
11 <script>
12 export default {
13   name: 'CoSearch',
14   data () {
15     return {
16       criteria: ''
17     }
18   },
19   watch: {
20     criteria () {
21       this.$emit('search', this.criteria)
22     }
23   }
24 }
25 </script>
26
27 <style lang='css'>
28 </style>
```

En **data** devuelvo un objeto que va a tener una variable criterio de búsqueda. Si nos fijamos, en el **input** del template podemos ver el código **v-model='criteria'**. Con esto conseguimos que cuando ese input cambie, automáticamente se almacene ese valor en el data correspondiente. Es parecido a un watcher.

Lo que queremos es que cuando el usuario escriba algo en el input, se modifique ese valor y se envíe un evento hacia arriba.

Vamos a crear un watch y vamos a hacer que cuando cambie **criteria** se emita un evento hacia arriba.

Los evento reciben dos parámetros, el nombre y un payload, que en nuestro caso el dato que queremos enviar es **criteria**. Esto no es lo más óptimo porque cada vez que el usuario teclee estaremos enviando eventos hacia arriba, pero lo modificaremos después.

Vamos a ver cómo hacer que el padre sepa que su hijo ha enviado un evento hacia arriba. Para ello, tenemos una directiva **v-on** que indica qué evento queremos escuchar e indicaremos qué método vamos a ejecutar cuando detecte dicho evento. Esto lo tenemos que añadir en el padre (**App.vue**).

```
▼ App.vue ×
codit > src > ▼ App.vue > {} "App.vue"
1  <template lang='html'>
2    <div class='codit'>
3      <div class='container'>
4        <nav class='menu'>
5          <co-search
6            v-on:search='onSearch'></co-search>
7          <co-bookmarks></co-bookmarks>
8        </nav>
9        <main class='content'>
10         <header class='content__header'>
11           <co-logo></co-logo>
12         </header>
13         <co-developers></co-developers>
14       </main>
15     </div>
16   </div>
17 </template>
```

Estamos indicando que si recibimos un evento de tipo **search**, sea manejado por un método **onSearch** que tenemos que implementar. Por ahora, dicho método sólo mostrará un mensaje por consola.

Vamos a probar todo esto, para ello tenemos que añadir los componentes que nos faltan, es decir: CoBookmarks y CoLogo.

The image shows two code editors side-by-side. The left editor contains the code for `CoBookmarks.vue` and the right one contains the code for `CoLogo.vue`. Both files are located in the `codit/src/components` directory.

```
codit > src > components > ▼ CoBookmark
1   <template lang='html'>
2   |
3   </template>
4
5   <script>
6   |   export default {
7   |       name: 'CoBookmarks'
8   |   }
9   </script>
10
11  <style lang='css'>
12  </style>
```

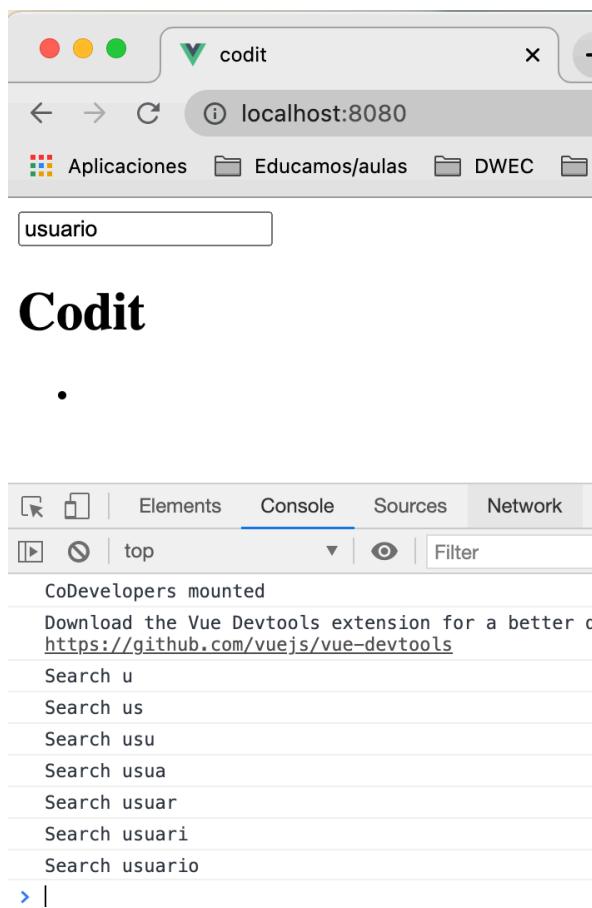


```
codit > src > components > ▼ CoLogo.vue >
1   <template lang='html'>
2   |   <h1 class='logo'>Codit</h1>
3   </template>
4
5   <script>
6   |   export default {
7   |       name: 'CoLogo'
8   |   }
9   </script>
10
11  <style lang='css'>
12  </style>
```

Para probarlo tenemos que ejecutar:

A screenshot of a terminal window. At the top, there are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL, with TERMINAL being the active tab. Below the tabs, the command `inma@MacBook-Pro-de-Inmaculada codit % npm start` is visible, with the cursor at the end of the line.

Y como no tenemos estilos aún ni ningún usuario que se pueda mostrar en el componente **CoDevelopers** pues lo vemos así:



- Tenemos el componente **co-search**, en el cual si tecleamos un texto, podemos ver en la consola que sí ha emitido el evento correspondiente hacia el padre (**co-app**) y por tanto muestra el mensaje cada vez que se modifica esa variable **criteria** asociada al input en **co-search**.

```

V CoSearch.vue X V CoDevelopers.vue V App.vue
codit > src > components > V CoSearch.vue > {} "CoSearch.vu
1   <template lang='html'>
2     <input
3       name='search'
4       type='text'
5       placeholder='Write an user!'
6       class='search'
7       v-model='criteria'
8     />
9   </template>
10
11  <script>
12    import bus from '@/busData.js'
13    export default {
14      name: 'CoSearch',
15      data () {
16        return {
17          criteria: ''
18        }
19      },
20      watch: {
21        criteria () {
22          // this.$emit('search', this.criteria)
23          bus.$emit('search', this.criteria)
24        }
25      }
26    }
27  </script>
28
29  <style lang='css'>
30  </style>

```

Bus de eventos

Si queremos comunicarnos con otro ancestro que no sea inmediato o con un hermano, ... surgen los buses de eventos.

Es un canal en el que todos los componentes están suscritos y reaccionan según lo que se publique.

Siguiendo con nuestro ejemplo, ahora vamos a hacer que el evento **search** que emitía **CoSearch** lo escuche **CoDevelopers**.

Para hacer esto, nos vamos a crear un fichero **busData.js**, en el cual nos vamos a importar Vue y vamos a crearnos nuestro bus de eventos.

Vamos a crear una constante en la que vamos a instanciar Vue sin ningún tipo de datos sin que se renderice en el DOM y vamos a exportarlo.

Ahora, en **CoSearch** vamos a importar dicho bus.



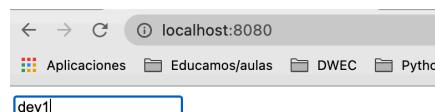
```
JS busData.js ×
codit > src > JS busData.js > [x] default
1 import Vue from 'vue'
2
3 const bus = new Vue()
4
5 export default bus
```

Nota: cuando importamos, @ es una abreviatura que tiene Webpack para indicarnos el directorio src.

Ahora, lo que vamos a hacer es que **CoDevelopers** se subscriba. Para ello, importamos el bus de datos también.

Por otro lado, implementamos el hook **created**, para subscribirnos cuando el componente se haya creado y lo que hacemos es mostrar el evento por consola.

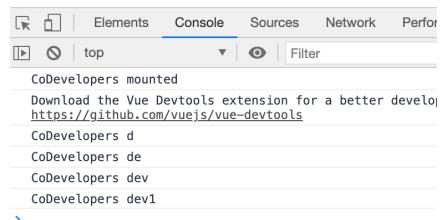
Ya podemos eliminar la gestión del evento por parte de **App.vue**.



Codit

•

Si lo probamos, el resultado se puede ver en la consola.



```

codit > src > components > ▼ CoDevelopers.vue > {} "CoDevelopers.vue"
1   <template lang='html'>
2     <ul class='developers'>
3       |   <li class="developers__item">
4       |     |   <!--co-developer></co-developer-->
5       |   </li>
6     </ul>
7   </template>
8
9   <script>
10  import CoDeveloper from '@/components/CoDeveloper'
11  import bus from '@/busData.js'
12
13  export default {
14    name: 'CoDevelopers',
15    data () {
16      return {
17        // Va a contener un listado de usuarios, inicialmente comienza vacío
18        users: []
19      }
20    },
21    mounted () {
22      console.log('CoDevelopers mounted')
23    },
24    created () {
25      bus.$on('search', criteria => {
26        console.log('CoDevelopers', criteria)
27      })
28    },
29    components: {
30      CoDeveloper
31    }
32  }
33  </script>
34
35  <style lang='css'>
36  </style>

```



```

codit > src > ▼ App.vue > {} "App.vue"
1   <template lang='html'>
2     <div class='codit'>
3       <div class='container'>
4         <nav class='menu'>
5           |   <co-search></co-search>
6           |   <co-bookmarks></co-bookmarks>
7         </nav>
8         <main class='content'>
9           <header class='content__header'>
10          |   <co-logo></co-logo>
11        </header>
12        <co-developers></co-developers>
13      </main>
14    </div>
15  </template>
16
17  <script>
18  import CoLogo from '@/components/CoLogo'
19  import CoSearch from '@/components/CoSearch'
20  import CoBookmarks from '@/components/CoBookmarks'
21  import CoDevelopers from '@/components/CoDevelopers'
22
23  export default {
24    name: 'CoApp',
25    components: {
26      CoLogo,
27      CoSearch,
28      CoBookmarks,
29      CoDevelopers
30    }
31  }
32  </script>
33
34  <style lang='css'>
35  </style>

```

Un bus de datos es una buena solución para comunicar elementos, pero cuando la aplicación va creciendo no es la más óptima, ya veremos otras opciones.

Componentes asíncronos

A nivel de performance, las aplicaciones muy grandes puede ser recomendable dividirlas en pequeñas partes y cargar desde el servidor sólo los componentes que necesitemos, por ejemplo, si tenemos un layout con 5 componentes, que no cargue del servidor 17, sino sólo los 5 que necesite. En proyectos grandes es muy interesante.

En nuestro ejemplo, vamos a hacer que nuestro componente **CoDevelopers** se cargue de forma asíncrona (esto sólo lo podemos hacer con templates



```

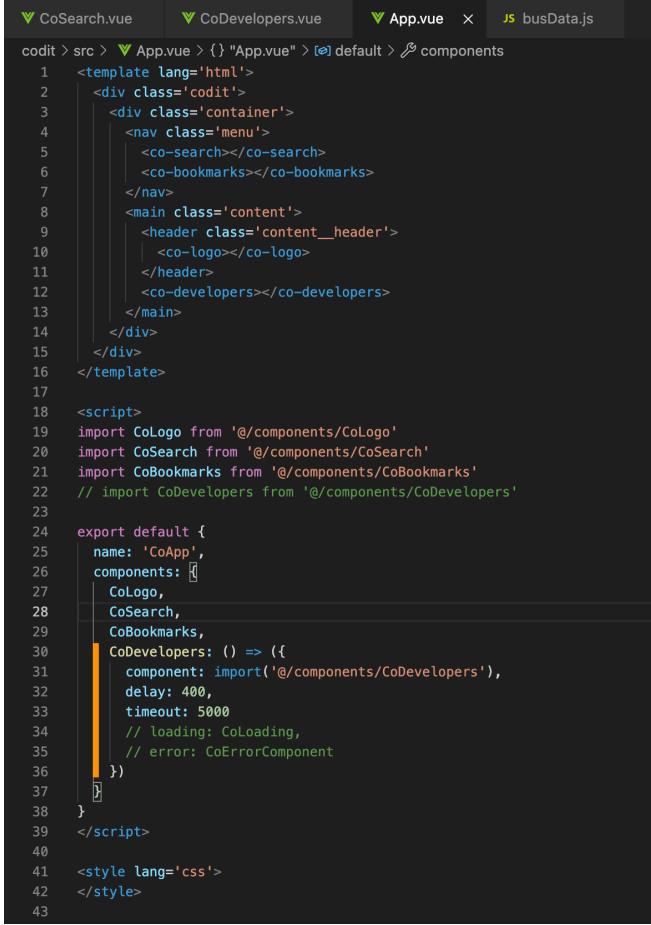
codit > src > ▼ App.vue > {} "App.vue"
1   <template lang='html'>
2     <div class='codit'>
3       <div class='container'>
4         <nav class='menu'>
5           <co-search></co-search>
6           <co-bookmarks></co-bookmarks>
7         </nav>
8         <main class='content'>
9           <header class='content__header'>
10             | <co-logo></co-logo>
11           </header>
12           <co-developers></co-developers>
13         </main>
14       </div>
15     </div>
16   </template>
17
18   <script>
19     import CoLogo from '@/components/CoLogo'
20     import CoSearch from '@/components/CoSearch'
21     import CoBookmarks from '@/components/CoBookmarks'
22     // import CoDevelopers from '@/components/CoDevelopers'
23
24     export default {
25       name: 'CoApp',
26       components: {
27         CoLogo,
28         CoSearch,
29         CoBookmarks,
30         CoDevelopers: () => import('@/components/CoDevelopers')
31       }
32     }
33   </script>
34
35   <style lang='css'>
36   </style>
37

```

basadas en Webpack, con templates basadas en Browserify no es posible).

Nuestro compilador haría todos los imports y traería esos componentes, pero en el caso de CoDevelopers lo traerá cuando corresponda según la función.

También puedo indicar con qué tiempo de **delay** quiero que se cargue y las opciones que podemos ver comentadas porque no tenemos los componentes a los que referencia creados (**loading**, cargaría ese componente mientras está cargando y error, cuando hay un error por ejemplo por timeout).



```

codit > src > ▼ App.vue > {} "App.vue" > ⚡ default > ⚡ components
1   <template lang='html'>
2     <div class='codit'>
3       <div class='container'>
4         <nav class='menu'>
5           <co-search></co-search>
6           <co-bookmarks></co-bookmarks>
7         </nav>
8         <main class='content'>
9           <header class='content__header'>
10             | <co-logo></co-logo>
11           </header>
12           <co-developers></co-developers>
13         </main>
14       </div>
15     </div>
16   </template>
17
18   <script>
19     import CoLogo from '@/components/CoLogo'
20     import CoSearch from '@/components/CoSearch'
21     import CoBookmarks from '@/components/CoBookmarks'
22     // import CoDevelopers from '@/components/CoDevelopers'
23
24     export default {
25       name: 'CoApp',
26       components: [
27         CoLogo,
28         CoSearch,
29         CoBookmarks,
30         CoDevelopers: () => ({
31           component: import('@/components/CoDevelopers'),
32           delay: 400,
33           timeout: 5000
34           // loading: CoLoading,
35           // error: CoErrorComponent
36         })
37       ]
38     }
39   </script>
40
41   <style lang='css'>
42   </style>
43

```

Sintaxis en los templates

En cada componente, indicamos mediante **lang** en el template por ejemplo, en qué lenguaje vamos a implementar el componente para Webpack, por ahora estamos indicando que la template está en HTML.

También como dijimos, podemos pasarlo a un fichero externo mediante el uso de **src**. Por ejemplo, sacando el html de **App.vue** obtenemos:

```

src > App.vue > {} "App.vue"
1   <template lang='html' src='./App.html'>
2   </template>
3
4   <script>
5     import CoLogo from '@/components/CoLogo'
6     import CoSearch from '@/components/CoSearch'
7     import CoBookmarks from '@/components/CoBookmarks'
8     // import CoDevelopers from '@/components/CoDevelopers'
9
10    export default {
11      name: 'CoApp',
12      components: {
13        CoLogo,
14        CoSearch,
15        CoBookmarks,
16        CoDevelopers: () => ({
17          component: import('@/components/CoDevelopers'),
18          delay: 400,
19          timeout: 5000
20          // loading: Coloading,
21          // error: CoErrorComponent
22        })
23      }
24    }
25  </script>
26
27  <style lang='css'>
28  </style>
29

```



```

src > <> App.html > <> div.codit
1   <div class='codit'>
2     <div class='container'>
3       <nav class='menu'>
4         <co-search></co-search>
5         <co-bookmarks></co-bookmarks>
6       </nav>
7       <main class='content'>
8         <header class='content__header'>
9           <co-logo></co-logo>
10        </header>
11        <co-developers></co-developers>
12      </main>
13    </div>
14

```

Podemos hacer binding en el template con variables de nuestro componente. Desde nuestro template podemos acceder a distintas variables de nuestro componente: data, props, computed, ...

Para ello vamos a utilizar la sintaxis de interpolaciones:

```

src > components > CoLogo.vue > {} "CoLogo.vue"
1   <template lang='html'>
2     <h1 class='logo'>{{ appName }}</h1>
3   </template>
4
5   <script>
6     export default {
7       name: 'CoLogo',
8       data () {
9         return {
10           appName: 'Codit'
11         }
12       }
13     }
14   </script>
15
16   <style lang='css'>
17   </style>

```

HTML y PUG

Vamos a hablar de los sistemas de templating que podemos utilizar. Hasta ahora hemos utilizado html, pero también podemos utilizar **pug**. Pug nos permite realizar plantillas de forma más corta que html, no requiere etiquetas de cierre y tiene muchas posibilidades: <https://pugjs.org/api/getting-started.html>

pug

Attributes

Tag attributes look similar to HTML (with optional commas), but their values are just regular JavaScript.

(NOTE: Examples on this page use the pipe character (|) for whitespace control.)

```
a(href='//google.com') Google  
|  
|  
a(class='button' href='//google.c  
|  
|  
a(class='button', href='//google.
```

```
<a href="//google.com">Google</a>  
<a class="button" href="//google.  
<a class="button" href="//google.
```

Normal JavaScript expressions work fine, too:

```
- var authenticated = true  
body(class=authenticated ? 'authed' : 'anon')
```

```
<body class="authed"></body>
```

Multiline Attributes

If you have many attributes, you can also spread them across many lines:

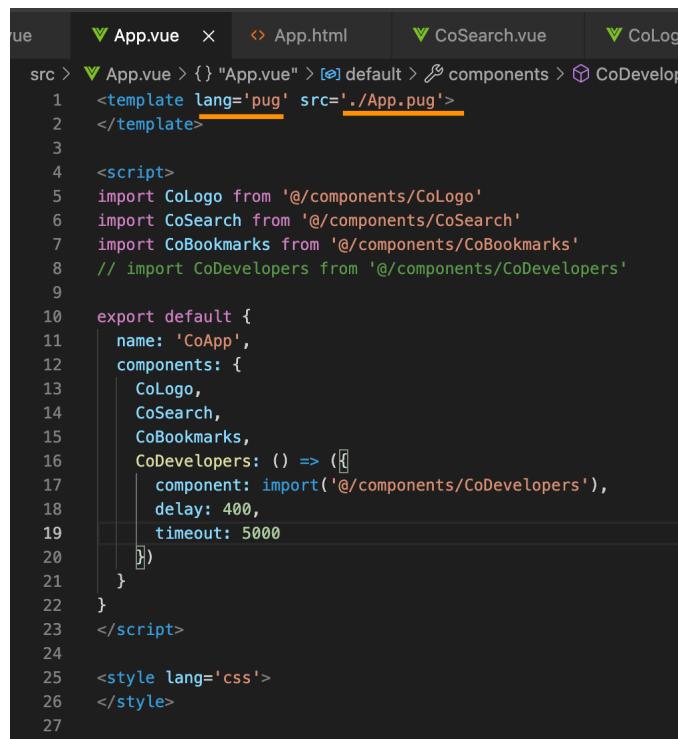
```
input(  
  type='checkbox'  
  name='agreement'  
  checked  
)
```

```
<input type="checkbox" name="agre
```

Vamos a probar **pug** y para ello, lo primero que haremos será instalar el paquete.

```
inma@MacBook-Pro-de-Inmaculada codit % sudo npm install --save pug
```

Lo primero que haremos será indicar en **App.vue** que ahora la plantilla va a ser pug y la extensión del template habrá cambiado. También tendremos que cambiar el nombre del fichero **App.html** por **App.pug** y transformar el código html en código pug.



```

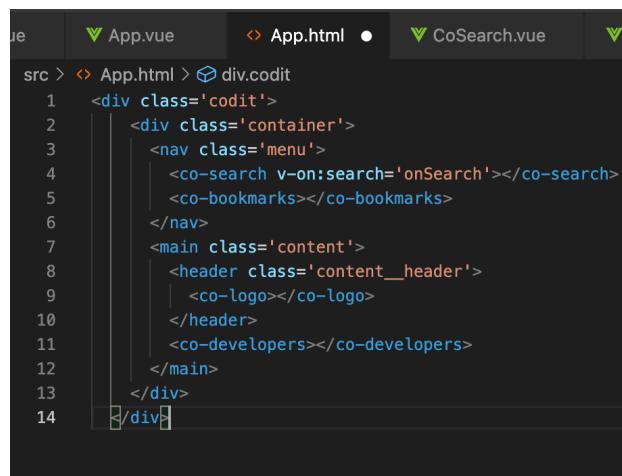
<template lang='pug' src='./App.pug'>
</template>

<script>
import CoLogo from '@/components/CoLogo'
import CoSearch from '@/components/CoSearch'
import CoBookmarks from '@/components/CoBookmarks'
// import CoDevelopers from '@/components/CoDevelopers'

export default {
  name: 'CoApp',
  components: {
    CoLogo,
    CoSearch,
    CoBookmarks,
    CoDevelopers: () => [
      component: import('@/components/CoDevelopers'),
      delay: 400,
      timeout: 5000
    ]
  }
}
</script>
<style lang='css'>
</style>

```

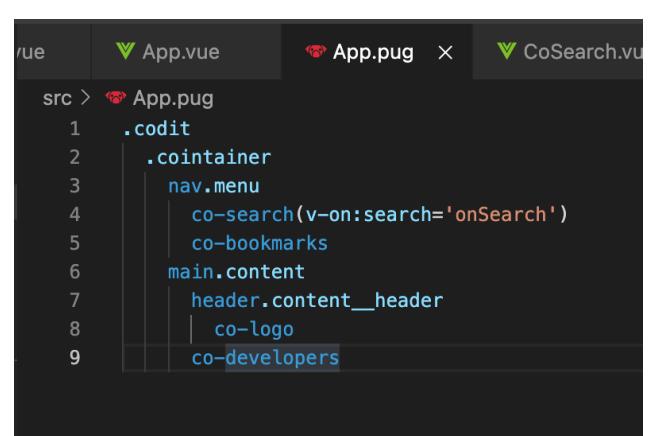
La conversión del código inicial html en pug quedaría:



```

<div class='codit'>
  <div class='container'>
    <nav class='menu'>
      <co-search v-on:search='onSearch'></co-search>
      <co-bookmarks></co-bookmarks>
    </nav>
    <main class='content'>
      <header class='content__header'>
        <co-logo></co-logo>
      </header>
      <co-developers></co-developers>
    </main>
  </div>
</div>

```



```

.codit
  .cointainer
    nav.menu
      co-search(v-on:search='onSearch')
      co-bookmarks
    main.content
      header.content__header
        co-logo
      co-developers

```

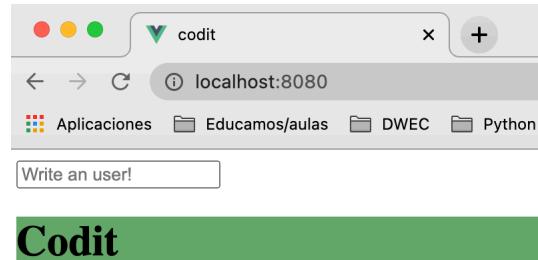
CSS, SASS o POSTCSS

Vue-loader y nuestro template PWA nos permite CSS, SASS o POSTCSS.

Nos vamos a ir a nuestro componente **CoLogo** y ahí vamos a hacer una serie de modificaciones en **style**.

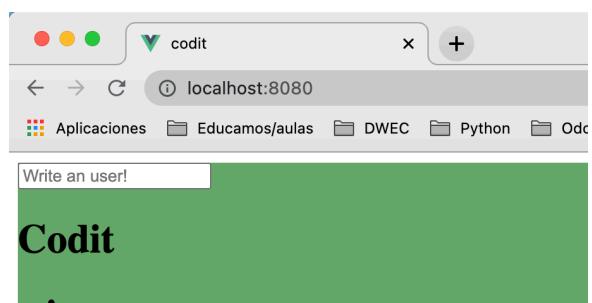
Si yo añado una clase **.logo** en la que defino un color de fondo, obtengo lo siguiente:

```
▼ CoLogo.vue ×
src > components > ▼ CoLogo.vue > {} "CoLogo.vue"
1  <template lang='html'>
2  | <h1 class='logo'>{{ appName }}</h1>
3  </template>
4
5  <script>
6    export default {
7      name: 'CoLogo',
8      data () {
9        return {
10          | appName: 'Codit'
11        }
12      }
13    }
14  </script>
15
16  <style lang='css'>
17  .logo {
18    background-color: #62ab67;
19  }
20  </style>
```



Si yo asigno esa clase logo a un componente superior, por ejemplo, en **App.pug**.

```
▼ CoLogo.vue      ↗ App.pug ×
src > ↗ App.pug
1  .codit.logo
2  .container
3  nav.menu
4  |   co-search(v-on:search='onSearch')
5  |   co-bookmarks
6  main.content
7  |   header.content__header
8  |   |   co-logo
9  |   co-developers
10 |
```



Nos damos cuenta de que el CSS no está encapsulado. Estamos generando un CSS global accesible desde cualquier componente de nivel superior. Pero

```

<template lang='html'>
  <h1 class='logo'>{{ appName }}</h1>
</template>

<script>
  export default {
    name: 'CoLogo',
    data () {
      return {
        appName: 'Codit'
      }
    }
  }
</script>

<style lang='css' scoped>
  .logo {
    background-color: #62ab67;
  }
</style>

```

la mayoría de las veces queremos un componente encapsulado que tenga su CSS accesible sólo por él y sus hijos, para esto tenemos que establecer un **scope**. De esta manera el scope de nuestros estilos sería desde nuestro componente hasta abajo. De este modo, en App no se cargaría esta clase.

```

:root {
  --color-black: #000000;
  --color-light-black: #444444;
  --color-white: #FCFCFC;

  --color-light-green: #7FCB70;
  --color-green: #62AB67;

  --color-blue: #283D55;
  --color-light-blue: #2C7AFD;

  --color-light-grey: #D0D4D8;
  --color-lighter-grey: #F5F5F5;
  --color-grey: #8F8F98;

  --color-brown: #CBBB84;

  --color-error: #ff9898;
  --color-bookmark: #FFE498;
}

```

Vamos a crear en **assets/css** un fichero para definir colores en nuestro proyecto. Creamos el fichero **colors.css**.

Estas variables están añadidas a nivel global.

Vamos a añadir css a nuestro código. Como no es objeto de este módulo, pegaré el código a añadir y los plugins que vamos a instalar para que funcionen correctamente los estilos.

Necesitamos instalar mediante **npm**:

npm install –save-dev postcss

npm install –save-dev postcss-import@3.0.2

npm install –save-dev postcss-cssnext@10.0.0

```

module.exports = {
  "plugins": {
    'postcss-import': {},
    'postcss-cssnext': {
      'browsers': ['last 2 versions']
    }
  }
}

```

Además, añadimos al fichero **.postcssrc.js** que exporte los siguientes módulos.

No es objeto de este módulo centrarnos en los estilos, así que añadimos aquí los estilos que tiene este ejemplo y en qué ficheros están incorporados. Los ficheros **colors.css**, **global.css**, **mixins.css** y **reset.css** los

incorporo en el aula virtual para que los incorporéis al proyecto.

Además añadimos estilos en los siguientes ficheros:

App.vue:

```
▼ App.vue ×  🔍 App.pug      # App.css      # colo
codit > src > App.vue > {} "App.vue"
23   </script>
24
25   <style lang='css'>
26     @import 'assets/css/reset.css';
27     @import 'assets/css/global.css';
28   </style>
29
30   <style lang='css' scoped src='./App.css'>
31   </style>
32 |
```

CoSearch.vue:

```
▼ CoSearch.vue ×  🔍 package.json  ▼ CoDevelopers.vu
codit > src > components > ▼ CoSearch.vue > {} "CoSearch.vue"
20
21   watch: {
22     criteria () {
23       // this.$emit('search', this.criteria)
24       bus.$emit('search', this.criteria)
25     }
26   }
27 </script>
28
30   <style lang='css' scoped>
31     .search {
32       margin: 1rem;
33     }
34   </style>
```

CoLogo.vue

```
▼ CoLogo.vue ×  ▼ CoSearch.vue  🔍 package.json  ▼ CoDevelopers.vue
codit > src > components > ▼ CoLogo.vue > {} "CoLogo.vue"
9     return {
10       appName: 'Codit'
11     }
12   }
13 </script>
14
16   <style lang='css' scoped>
17     @import '../assets/css/colors.css';
18     .logo {
19       margin: 0;
20       color: var(--color-lighter-grey);
21       line-height: 2rem;
22       font-size: 2rem;
23       font-weight: 800;
24       letter-spacing: -2px;
25       text-shadow: 0 0 20px color(var(--color-blue) blackness(50%));
26     }
27 </style>
```

CoDevelopers.vue

```
▼ CoLogo.vue ×  ▼ CoSearch.vue  🔍 package.json  ▼ CoDevelopers.vue
codit > src > components > ▼ CoLogo.vue > {} "CoLogo.vue"
9     return {
10       appName: 'Codit'
11     }
12   }
13 </script>
15
16   <style lang='css' scoped>
17     @import '../assets/css/colors.css';
18     .logo {
19       margin: 0;
20       color: var(--color-lighter-grey);
21       line-height: 2rem;
22       font-size: 2rem;
23       font-weight: 800;
24       letter-spacing: -2px;
25       text-shadow: 0 0 20px color(var(--color-blue) blackness(50%));
26     }
27 </style>
```

CoDeveloper.vue

```

codit > src > components > CoDeveloper.vue > {} "CoDeveloper"
81  <style lang='css' scoped>
82  @import "../assets/css/colors.css";
83  @import "../assets/css/mixins.css";
84  .developer {
85    @apply --flex-row;
86    &.developer__avatar {
87      width: 4rem;
88      height: 4rem;
89      border-radius: 50%;
90    }
91    &.developer__info {
92      @apply --flex-col;
93      justify-content: space-around;
94      flex: 1;
95      margin: 0 1rem;
96    }
97    &.developer__name,
98    &.developer__login {
99      color: var(--color-black);
100   }
101  &.developer__login {
102    font-size: 0.7rem;
103  }
104  &.developer__name {
105    font-weight: 700;
106  }
107  &.developer__metadata {
108    font-size: 0.6rem;
109  }
110  &.developer__data::after {
111    content: " -";
112  }
113  &.developer__data:last-child::after {
114    content: "";
115  }
116  &.developer__stats {
117    @apply --flex-col;
118    justify-content: center;
119  }
120  &.developer__stat {
121    @apply --flex-row;
122    font-size: 0.6rem;
123  }
124  &.developer__icon {
125    width: 1rem;
126    fill: var(--color-grey);
127    margin-right: 0.2rem;
128  }
129
130 </style>

```

CoBookmarks.vue

```

codit > src > components > CoBookmarks.vue > {} "CoBookmarks"
14  </script>
15
16  <style lang='css' scoped>
17  @import "../assets/css/colors.css";
18  .bookmarks__empty {
19    padding: 1rem;
20    text-align: center;
21    font-size: .8rem;
22    color: var(--color-error);
23  }
24  .bookmarks__bookmark {
25    transition: all .6s;
26  }
27  .bookmarks__bookmark:hover {
28    background-color: var(--color-lighter-grey);
29  }
30  .bookmark__link {
31    display: block;
32    width: 100%;
33    padding: 1rem;
34    color: var(--color-lighter-grey);
35    background-color: transparent;
36    border: none;
37    text-transform: uppercase;
38  }
39  .bookmarks__bookmark:hover .bookmark__link {
40    color: var(--color-blue);
41  }
42  .bookmark__small {
43    font-size: .8rem;
44    text-transform: none;
45  }
46 </style>

```

La sintaxis es postcss que es la evolución de CSS y permite transformar CSS con plugins de JavaScript.

Bindings

Para mostrar unos datos de ejemplo, en **CoDevelopers** vamos a poner:

```

▼ CoDevelopers.vue X ▼ CoBookmarks.vue ▼ CoDeveloper.vue ▼ CoS
codit > src > components > ▼ CoDevelopers.vue > {} "CoDevelopers.vue"
1   <template lang='html'>
2     <ul class="developers">
3       <li class="developers__item">
4         <co-developer
5           avatar='https://avatars2.githubusercontent.com/u/25254?v=4'
6           name='TJ Holowaychuk'
7           login='tj'
8           email='tj@apex.sh'
9           location='Victoria, BC, Canada'
10          company='Apex'
11        ></co-developer>
12      </li>
13    </ul>
14  </template>
15

```

De este modo cargamos un desarrollador, con sus propiedades.

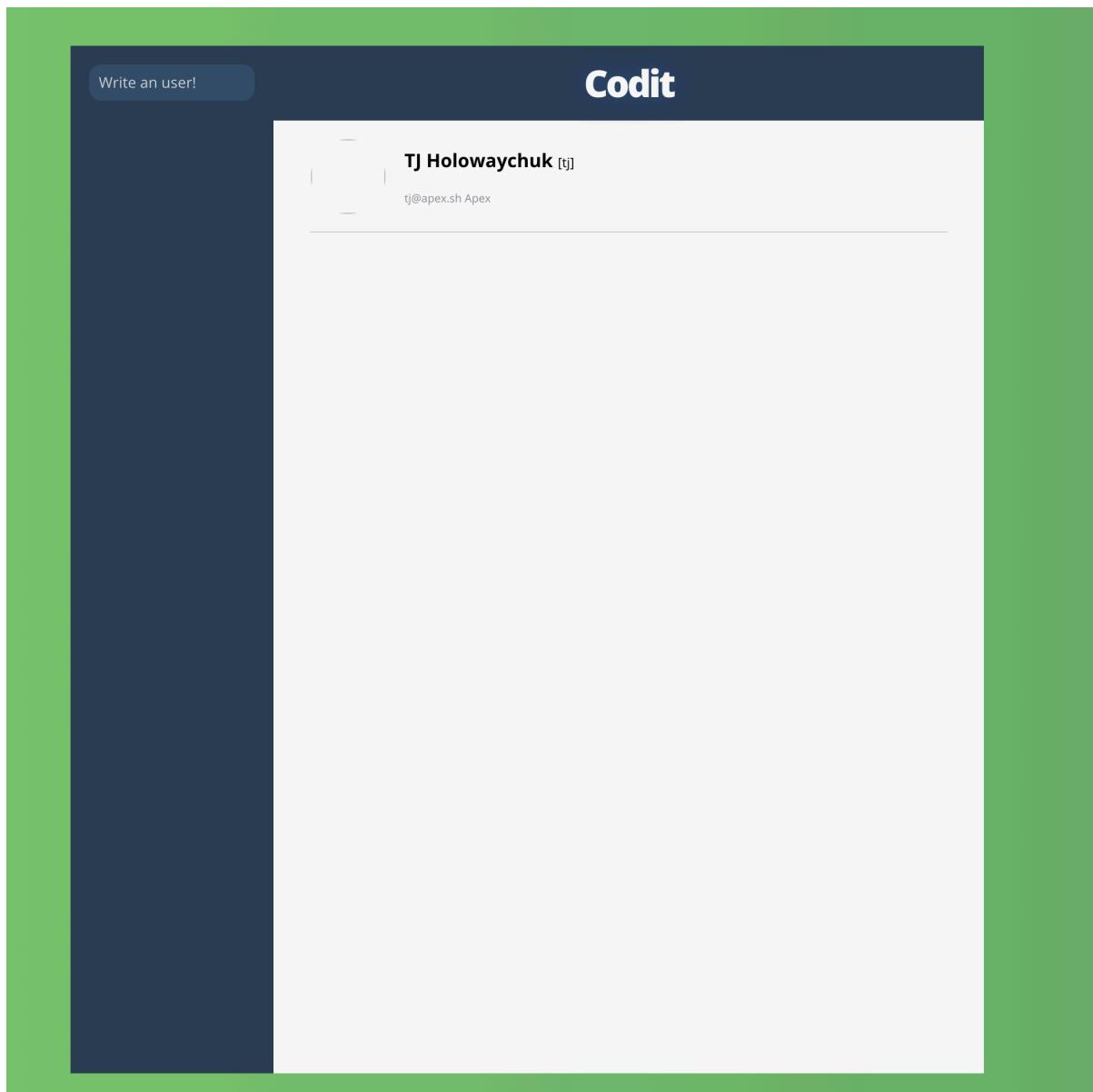
Ahora mismo, si vamos al componente **CoDeveloper**, tenemos lo siguiente:

```

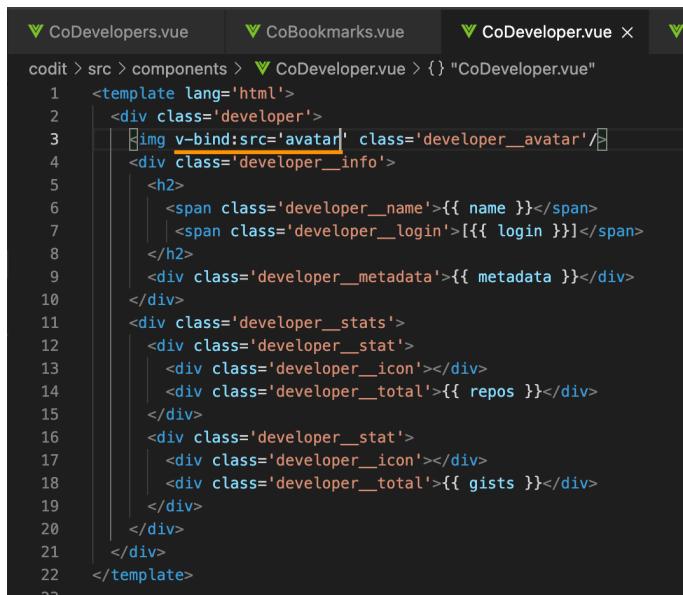
▼ CoDevelopers.vue ▼ CoBookmarks.vue ▼ CoDeveloper.vue X ▼ CoS
codit > src > components > ▼ CoDeveloper.vue > {} "CoDeveloper.vue" > .developer
1   <template lang='html'>
2     <div class='developer'>
3       <img class='developer__avatar' />
4       <div class='developer__info'>
5         <h2>
6           <span class='developer__name'>{{ name }}</span>
7           <span class='developer__login'>[{{ login }}]</span>
8         </h2>
9         <div class='developer__metadata'>{{ metadata }}</div>
10        </div>
11        <div class='developer__stats'>
12          <div class='developer__stat'>
13            <div class='developer__icon'></div>
14            <div class='developer__total'>{{ repos }}</div>
15          </div>
16          <div class='developer__stat'>
17            <div class='developer__icon'></div>
18            <div class='developer__total'>{{ gists }}</div>
19          </div>
20        </div>
21      </div>
22    </template>
23

```

La imagen, de momento no aparece, y el estilo de nuestra aplicación es:



Para esto, vamos a hacer un binding, es decir, decirle que utilice una variable dentro de un atributo, no podemos utilizar la interpolación `{}{}` dentro de un atributo, por tanto hacemos:

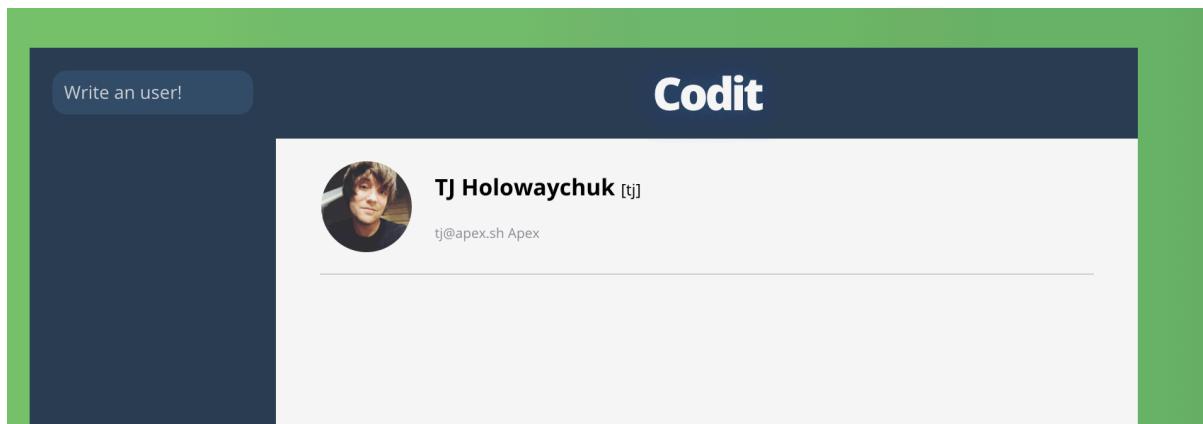
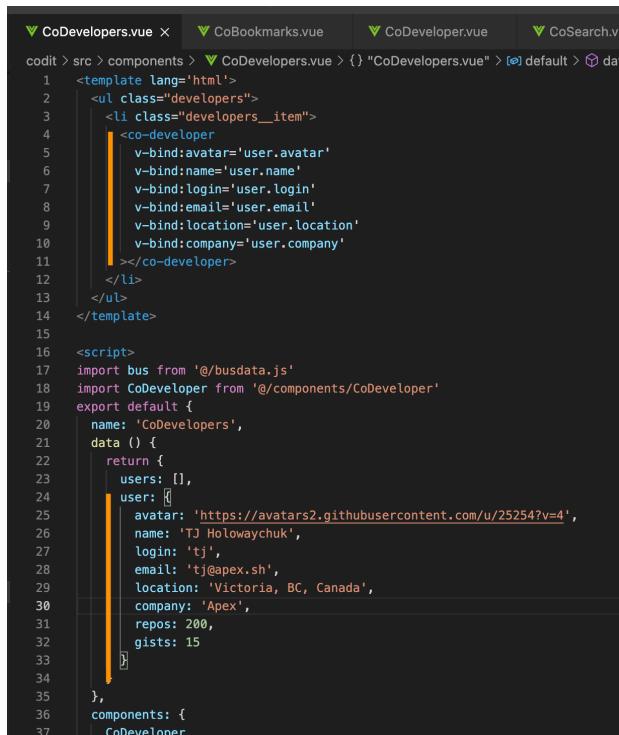


```

codit > src > components > ▼ CoDeveloper.vue > {} "CoDeveloper.vue"
1  <template lang='html'>
2    <div class='developer'>
3      <img v-bind:src='avatar' class='developer__avatar'/>
4      <div class='developer__info'>
5        <h2>
6          <span class='developer__name'>{{ name }}</span>
7          <span class='developer__login'>{{ login }}</span>
8        </h2>
9        <div class='developer__metadata'>{{ metadata }}</div>
10       </div>
11       <div class='developer__stats'>
12         <div class='developer__stat'>
13           <div class='developer__icon'></div>
14           <div class='developer__total'>{{ repos }}</div>
15         </div>
16         <div class='developer__stat'>
17           <div class='developer__icon'></div>
18           <div class='developer__total'>{{ gists }}</div>
19         </div>
20       </div>
21     </div>
22   </template>
23

```

De este modo ya funcionaría:

```

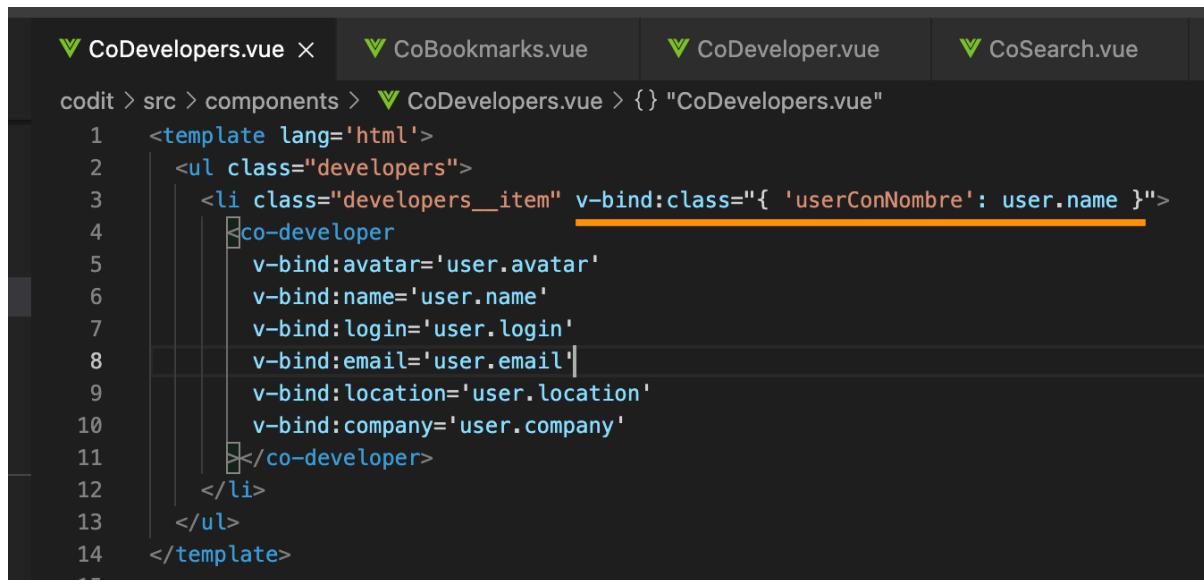
codit > src > components > ▼ CoDevelopers.vue > {} "CoDevelopers.vue" > ⏺ default > ⏺ da
1  <template lang='html'>
2    <ul class="developers">
3      <li class="developers__item">
4        <co-developer
5          v-bind:avatar='user.avatar'
6          v-bind:name='user.name'
7          v-bind:login='user.login'
8          v-bind:email='user.email'
9          v-bind:location='user.location'
10         v-bind:company='user.company'
11       ></co-developer>
12     </li>
13   </ul>
14 </template>
15
16 <script>
17 import bus from '@busdata.js'
18 import CoDeveloper from '@/components/CoDeveloper'
19 export default {
20   name: 'CoDevelopers',
21   data () {
22     return {
23       users: [],
24       user: []
25     }
26     user: [
27       {
28         avatar: 'https://avatars2.githubusercontent.com/u/25254?v=4',
29         name: 'TJ Holowaychuk',
30         login: 'tj',
31         email: 'tj@apex.sh',
32         location: 'Victoria, BC, Canada',
33         company: 'Apex',
34         repos: 200,
35         gists: 15
36       }
37     ],
38     components: {
39       CoDeveloper
40     }
41   }
42 }

```

Ahora, si queremos pasar todos los datos desde **CoDevelopers** a **CoDeveloper** mediante un objeto podemos hacer lo que vemos en la captura de pantalla.

Los bindings se utilizan para más cosas, como por ejemplo para añadir clases o estilos de forma condicional.

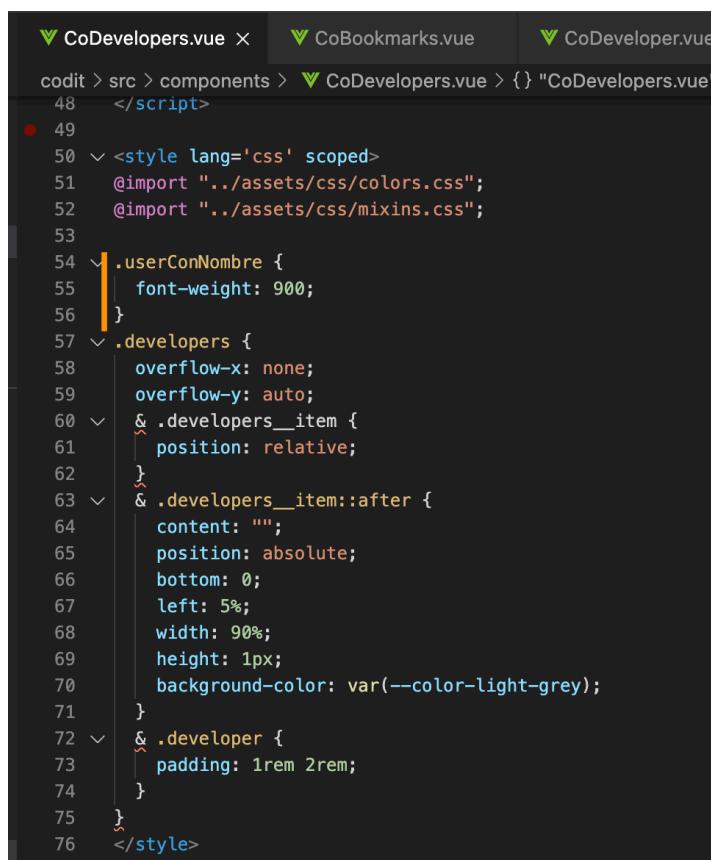
Por ejemplo, si quiero aplicar la clase **userConNombre** sólo cuando la variable **user.name** tenga valor puedo hacer lo siguiente.



```

codit > src > components > CoDevelopers.vue > {} "CoDevelopers.vue"
1   <template lang='html'>
2     <ul class="developers">
3       <li class="developers__item" v-bind:class="{ 'userConNombre': user.name }">
4         <co-developer
5           v-bind:avatar='user.avatar'
6           v-bind:name='user.name'
7           v-bind:login='user.login'
8           v-bind:email='user.email'
9           v-bind:location='user.location'
10          v-bind:company='user.company'
11        </co-developer>
12      </li>
13    </ul>
14  </template>
15

```



```

codit > src > components > CoDevelopers.vue > {} "CoDevelopers.vue"
48  </script>
49
50  <style lang='css' scoped>
51    @import "../assets/css/colors.css";
52    @import "../assets/css/mixins.css";
53
54  .userConNombre {
55    font-weight: 900;
56  }
57  .developers {
58    overflow-x: none;
59    overflow-y: auto;
60  &.developers__item {
61    position: relative;
62  }
63  &.developers__item::after {
64    content: "";
65    position: absolute;
66    bottom: 0;
67    left: 5%;
68    width: 90%;
69    height: 1px;
70    background-color: var(--color-light-grey);
71  }
72  &.developer {
73    padding: 1rem 2rem;
74  }
75
76 </style>

```

De este modo sólo se aplica cuando esa variable tiene valor, también podríamos haber puesto: **user.name != "**

También podríamos hacer binding a estilos o a cualquier atributo del html.

Directivas

Una directiva es un atributo especial de Vue que permite hacer operaciones en nuestro template.

```

codit > src > mocks > JS user.js      JS users.js < .editorconfig   .eslintignore
codit > src > mocks > JS users.js > ...
1  export default [
2  {
3    'login': 'tj',
4    'id': 2524,
5    'avatar_url': 'https://avatars2.githubusercontent.com/u/2524?v=4',
6    'gravatar_id': '',
7    'url': 'https://api.github.com/users/tj',
8    'html_url': 'https://github.com/tj',
9    'followers_url': 'https://api.github.com/users/tj/followers',
10   'following_url': 'https://api.github.com/users/tj/following{/other_user}',
11   'gists_url': 'https://api.github.com/users/tj/gists{/gist_id}',
12   'starred_url': 'https://api.github.com/users/tj/starred{/owner}{/repo}',
13   'subscriptions_url': 'https://api.github.com/users/tj/subscriptions',
14   'organizations_url': 'https://api.github.com/users/tj/orgs',
15   'repos_url': 'https://api.github.com/users/tj/repos',
16   'events_url': 'https://api.github.com/users/tj/events{/privacy}',
17   'received_events_url': 'https://api.github.com/users/tj/received_events',
18   'type': 'User',
19   'site_admin': false,
20   'name': 'TJ Holowaychuk',
21   'company': 'Apex',
22   'blog': 'https://apex.sh',
23   'location': 'Victoria, BC, Canada',
24   'email': 'tj@apex.sh',
25   'hireable': null,
26   'bio': 'Founder of Apex\r\nhttps://apex.sh, a real company.\r\n@tjholowaychuk on Twitter',
27   'followers': 30722,
28   'following': 48,
29   'created_at': '2008-09-18T22:37:28Z',
30   'updated_at': '2017-09-12T09:54:17Z'
31 },
32 {
33   'login': 'ruanyf',
34   'id': 905434,
35 }
36 ]
37 
```

Todas comienzan por **v-**, como por ejemplo **v-bind**, **v-on**, ...

Para seguir con nuestro ejemplo, vamos a crear un fichero de **mocks** que van a ser datos fijos para poder consumirlos. Estos ficheros los tenéis disponible en el aula virtual para que los incorporéis al proyecto. Estos ficheros tienen datos que posteriormente obtendremos de un api externa.

Crearemos una carpeta **Mocks** dentro de **src** que contendrá los ficheros **user.js** y **users.js**.

Estos ficheros básicamente tienen datos fijos sobre usuarios.

Estos mocks, los hemos asignado a la propiedad **data** para que coja esos usuarios como datos iniciales.

```

codit > src > components > CoDevelopers.vue > {} "CoDevelopers.vue"
codit > src > components > CoDevelopers.vue > {}
1  <template lang='html'>
2    <ul class="developers">
3      <li v-for='user in users' class="developers__item">
4        <co-developer
5          v-bind:avatar='user.avatar_url'
6          v-bind:name='user.name'
7          v-bind:login='user.login'
8          v-bind:email='user.email'
9          v-bind:location='user.location'
10         v-bind:company='user.company'
11         v-bind:repos='user.public_repos'
12         v-bind:gists='user.public_gists'
13       ></co-developer>
14     </li>
15   </ul>
16 </template>
17
18 <script>
19 import bus from '@/busdata.js'
20 import CoDeveloper from '@/components/CoDeveloper'
21 import mocks from '@/mocks/users.js'
22 export default {
23   name: 'CoDevelopers',
24   data () {
25     return {
26       users: mocks
27     }
28   },
29   components: {
30     CoDeveloper
31   }
32 }
33 
```

Ahora tenemos que hacer que cargue en el template de **CoDevelopers**, tantos **CoDeveloper** como desarrolladores tenga en **users**. Para ello usamos la directiva **v-for**.

El resultado ya muestra una lista de desarrolladores con la información que necesitamos.

Developer	Login	Email	Statistics
TJ Holowaychuk [tj]	tj@apex.sh Apex		47 24
Ruan YiFeng [ruanyf]	yifeng.ruan@gmail.com		
Addy Osmani [addyosmani]	addyosmani@gmail.com Google		295 168
Paul Irish [paulirish]	Google Chrome, ❤z		240 109
Evan You [yyx990803]			129 44
TJ Holowaychuk [tj]	tj@apex.sh Apex		269 542
Ruan YiFeng [ruanyf]	yifeng.ruan@gmail.com		47 24
Addy Osmani [addyosmani]	addyosmani@gmail.com Google		295 168

Otra directiva que podemos ver es la condicional **v-if**. Vamos a mostrarla en el ejemplo en el que no llegase valor de las propiedades no obligatorias y no quisiésemos renderizar el componente. También existe **v-show**. **v-if** no renderiza nada del html cuando no se cumple la condición y **v-show** lo que hace es mostrarlo o no.

```

▼ CoDeveloper.vue ×
codit > src > components > ▼ CoDeveloper.vue > {} "CoDeveloper.vue"
  1  <template lang='html'>
  2    <div class='developer'>
  3      <img v-bind:src='avatar' class='developer__avatar'/>
  4      <div class='developer__info'>
  5        <h2>
  6          <span class='developer__name'>{{ name }}</span>
  7          <span class='developer__login'>{{ login }}</span>
  8        </h2>
  9        <div class='developer__metadata'>{{ metadata }}</div>
 10      </div>
 11      <div class='developer__stats'>
 12        <div class='developer_stat' v-if='repos'>
 13          <div class='developer__icon'></div>
 14          <div class='developer__total'>{{ repos }}</div>
 15        </div>
 16        <div class='developer_stat' v-show='gists'>
 17          <div class='developer__icon'></div>
 18          <div class='developer__total'>{{ gists }}</div>
 19        </div>
 20      </div>
 21    </div>
 22  </template>

```

```

▼ CoSearch.vue ×

codit > src > components > ▼ CoSearch.vue > {} "CoSearch.vue"
1   <template lang='html'>
2     <input
3       name='search'
4       type='text'
5       placeholder='Write an user!'
6       class='search'
7       v-on:keyup.enter='onSearch'
8     />
9
10    </template>
11
12    <script>
13      import bus from '@/busdata.js'
14      export default {
15        name: 'CoSearch',
16        data () {
17          return [
18            criteria: ''
19          ]
20        },
21        methods: {
22          onSearch () {
23            // this.$emit('search', this.criteria)
24            bus.$emit('search', this.criteria)
25          }
26        }
27      }
28    </script>
29

```

Ahora vamos a modificar el componente **CoSearch**, porque tenemos que lance eventos cada vez que detecta un cambio en **criteria**. Ahora vamos a hacer que realmente reaccione cuando el usuario pulse enter.

De este modo hemos eliminado el **watch** y ahora reaccionamos cuando el usuario pulsa enter, mediante un método en lugar de un watch.

Vamos a modificar **CoBookmarks**:

```

▼ CoSearch.vue      ▼ CoBookmarks.vue ×

codit > src > components > ▼ CoBookmarks.vue > {} "CoBookmarks.vue" > ⓘ default > ⚏ data
1   <template lang='html'>
2     <div class="co-bookmarks">
3       <ul v-if="bookmarks.length" class='bookmarks'>
4         <li v-for="bookmark in bookmarks" class='bookmarks__bookmark'>
5           {{ bookmark.name }}
6           <span class='bookmark__small'>{{ bookmark.login }}</span>
7         </li>
8       </ul>
9       <p v-else class="bookmarks__empty">You dont have any bookmark selected!</p>
10      </div>
11    </template>
12
13    <script>
14      import mocks from '@/mocks/users.js'
15
16      export default {
17        name: 'CoBookmarks',
18        data () [
19          return {
20            bookmarks: mocks
21          }
22        ]
23      }
24    </script>
25

```

Además de las directivas de **Vue**, también podemos crear directivas propias. Como ejemplo vamos a crear una directiva propia para controlar el foco de un elemento input.

```
codit > src > components > ▼ CoSearch.vue > {} "CoSearch.vue" > [x] default > ↗ direct
  1   <template lang='html'>
  2     <input
  3       name='search'
  4       type='text'
  5       placeholder='Write an user!'
  6       class='search'
  7       v-focus
  8       v-on:keyup.enter='onSearch'
  9     />
10
11   </template>
12
13   <script>
14     import bus from '@/busdata.js'
15
16     export default {
17       name: 'CoSearch',
18       data () {
19         return {
20           criteria: ''
21         }
22       },
23       methods: {
24         onSearch () {
25           // this.$emit('search', this.criteria)
26           bus.$emit('search', this.criteria)
27         }
28       },
29       directives: {
30         focus: [
31           inserted: function (el) {
32             el.focus()
33           }
34         ]
35       }
36     }
37   
```

Para ello en **CoSearch**, creamos la directiva de nombre **focus** que cuando se inserte en un elemento concreto le da el foco. Para utilizarla tenemos que emplear la nomenclatura **v-nombreDirectiva**, que en este caso sería **v-focus**.

Cuando reiniciamos la aplicación vemos que el elemento input de **CoSearch** tiene el foco.

Filtros

Son parecidos a las directivas pero nos permiten desde el template, filtrar el contenido de los datos que nos llegan.

Como ejemplo, vamos a crear un filtro para que siempre se vea el título de la aplicación en mayúscula.

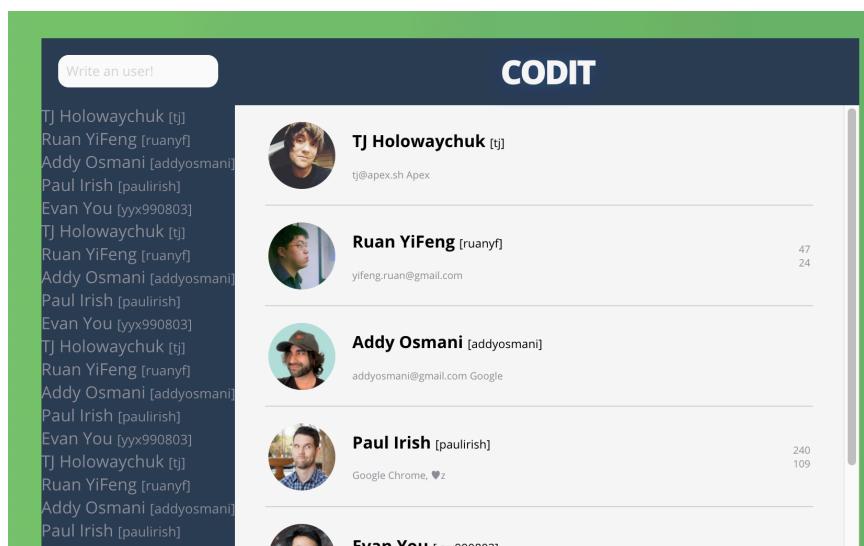
```

CoSearch.vue   CoLogo.vue ×   main.js

codit > src > components > CoLogo.vue > {} "CoLogo.vue"
1   <template lang="html">
2   | <h1 class="logo">{{ appName | upper }}</h1>
3   </template>
4
5   <script>
6     export default {
7       name: 'CoLogo',
8       data () {
9         return {
10           appName: 'Codit'
11         }
12       },
13       filters: {
14         upper (value) {
15           if (!value) {
16             return ''
17           }
18           return value.toUpperCase()
19         }
20       }
21     }
22   </script>
23

```

Hemos creado un filtro que lo que hace es que el valor que recibe si no es vacío lo pasa a mayúscula.



También podemos concatenar filtros añadiéndolos a la implementación y luego utilizando más tuberías para su concatenación.

Transiciones

Vue nos permite gestionar efectos desde la propia librería.

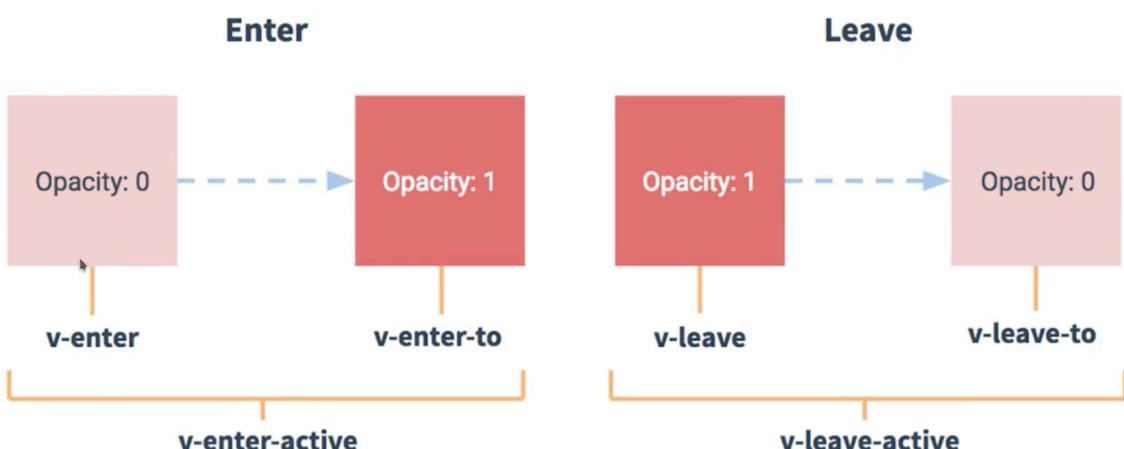
```

50 border-radius: 10px;
51 }
52
53 input:focus {
54 background-color: var(--color-white);
55 color: var(--color-blue);
56 }
57
58 .fade-enter-active, .fade-leave-active {
59 transition: opacity 4s
60 }
61
62 .fade-enter, .fade-leave-to {
63 opacity: 0
64 }

```

Vamos a añadir en **global.css** un código para que haga un fade, ahora veremos dónde lo vamos a aplicar.

Para las transiciones, Vue nos permite unirnos a cada estado en los que se produce la transición.



En Vue, tenemos 4 estados en los que podemos engancharnos en una transición.

En el caso de Enter y Leave le estamos diciendo que la opacidad debe ser 0 y cuando esté en v-enter-active o v-leave-active que haga una transición de opacidad de 4s.

Vamos a ver la implementación en nuestro ejemplo. Lo que haremos será que cuando el usuario en **CoSearch** pulse enter al buscar, vuelva a emitir un

```

codit > src > components > CoSearch.vue > {} "CoSearch.vue"
  9   />
10
11  </template>
12
13  <script>
14
15  export default {
16    name: 'CoSearch',
17    data () {
18      return {
19        criteria: ''
20      }
21    },
22    methods: {
23      onSearch () {
24        this.$emit('search', this.criteria)
25      }
26    },
27    directives: {
28      focus: {
29        inserted: function (el) {
30          el.focus()
31        }
32      }
33    }
34  }
35  </script>
36
37  <style lang='css' scoped>

```

Es importante que el nombre coincida con la primera palabra de las clases establecidas en el fichero global.css, es decir, fade.

De esta forma tenemos el efecto codificado y sólo se realizará el fade cuando se haga una búsqueda y se pulse intro.

NOTA: Para seguir con el ejercicio quitamos de App.pug la transición creada y de App.vue la variable show ya que ahora después vamos a conectar con un servicio externo y queremos renderizar siempre el componente.

Servicios externos. Fetch.

Vamos a ver cómo conectarnos con un servicio externo, primero lo haremos con el API fetch que proporciona HTML5.

Para poder trabajar con el api de Github que es el servicio externo que vamos a utilizar, necesitamos un token. Para ello, vamos a Github, y en **Settings**, accedemos a la opción **Developer Settings / Personal access tokens**.

Generamos un token y elegimos un nombre y seleccionamos todos los scope:

evento hacia el padre, no a todos, tal y como estaba al principio.

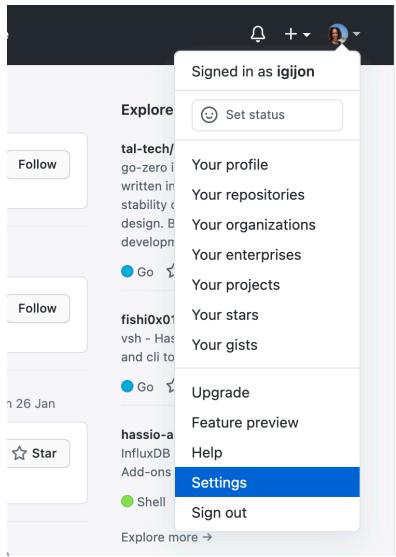
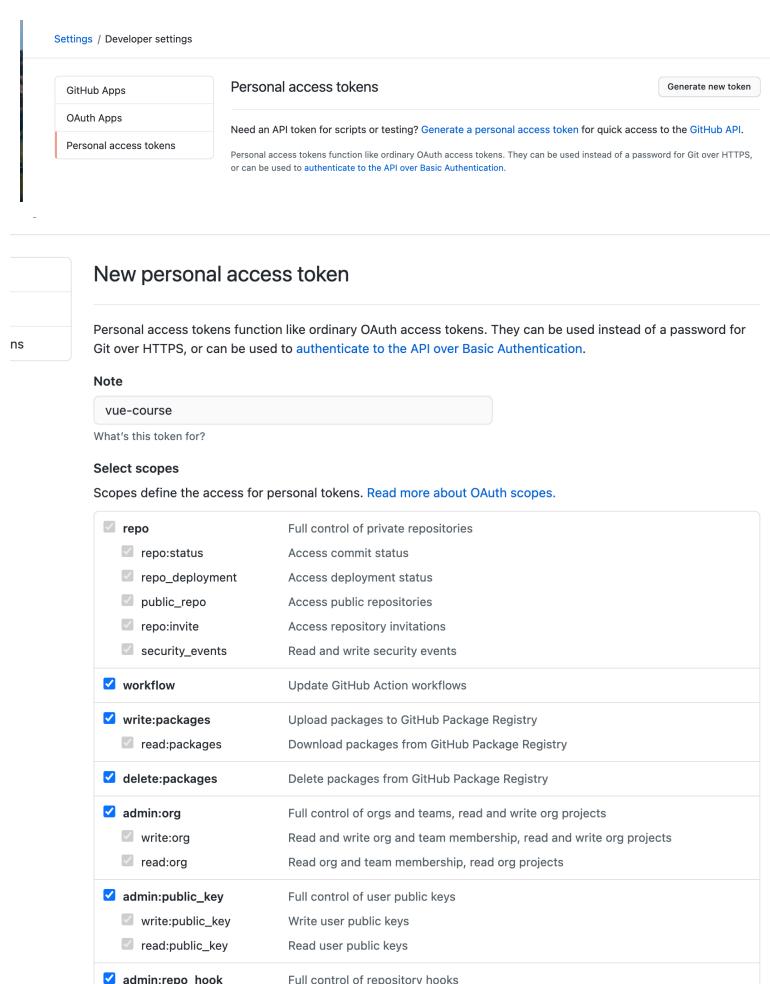
En el fichero **App.vue**, voy a crear una variable **show** inicialmente a **false** que se vuelva **true** cuando se detecte el evento del intro en el input de **CoSearch**.

En el fichero **App.pug**:

```

codit > src > App.pug
  1 .codit
  2 .container
  3   nav.menu
  4     co-search(v-on:search='onSearch')
  5     co-bookmarks
  6     main.content
  7       header.content__header
  8         co-logo
  9         transition(name='fade' v-if='show')
10         co-developers

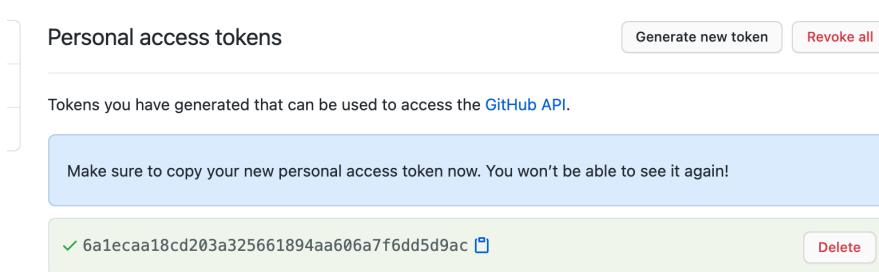
```

The screenshot shows the GitHub developer settings for generating a personal access token. The 'Personal access tokens' section is active. A note explains that personal access tokens function like OAuth tokens and can be used instead of a password for Git over HTTPS or for API authentication. A 'Generate new token' button is visible. Below it, a table lists various OAuth scopes with checkboxes. Most scopes have detailed descriptions next to them.

Scope	Description
<input checked="" type="checkbox"/> repo	Full control of private repositories
<input type="checkbox"/> repo:status	Access commit status
<input type="checkbox"/> repo_deployment	Access deployment status
<input type="checkbox"/> public_repo	Access public repositories
<input type="checkbox"/> repo:invite	Access repository invitations
<input type="checkbox"/> security_events	Read and write security events
<input checked="" type="checkbox"/> workflow	Update GitHub Action workflows
<input checked="" type="checkbox"/> write:packages	Upload packages to GitHub Package Registry
<input type="checkbox"/> read:packages	Download packages from GitHub Package Registry
<input checked="" type="checkbox"/> delete:packages	Delete packages from GitHub Package Registry
<input checked="" type="checkbox"/> admin:org	Full control of orgs and teams, read and write org projects
<input type="checkbox"/> write:org	Read and write org and team membership, read and write org projects
<input type="checkbox"/> read:org	Read org and team membership, read org projects
<input checked="" type="checkbox"/> admin:public_key	Full control of user public keys
<input type="checkbox"/> write:public_key	Write user public keys
<input type="checkbox"/> read:public_key	Read user public keys
<input checked="" type="checkbox"/> admin:repo_hook	Full control of repository hooks

Una vez hecho esto, generamos el token:



The screenshot shows the GitHub personal access tokens page. A message at the top says to copy the token now as it won't be shown again. Below is a table of tokens. One token is listed with a green checkmark and a copy icon, with a 'Delete' button to its right.

Token	Action
6a1ecaa18cd203a325661894aa606a7f6dd5d9ac	

A note at the bottom explains that personal access tokens function like OAuth tokens and can be used instead of a password for Git over HTTPS or for API authentication.

Ahora vamos a configurar este token en nuestra aplicación.

En nuestro proyecto Vue tenemos un directorio **config** con 4 ficheros JavaScript. Por

defecto, **prod.env.js** son las variables que vamos a utilizar, y en test o en desarrollo, trae las variables de producción y las sobreescribe por las de desarrollo.

En **prod**, vamos a tener la ruta del api de GitHub porque la utilizaríamos tanto en desarrollo como en producción.

```

JS dev.env.js      ▼ CoDevelopers.vue      JS prod.env.js ×
codit > config > JS prod.env.js > [?] <unknown> > 🔗 API
1   module.exports = [
2     ...
3       NODE_ENV: '"production"',
4       API: '"https://api.github.com"'
5

```

En **dev**, vamos a tener el token que acabamos de crear. Tenemos este token en el porque podríamos tener un token en producción y otro en desarrollo. En este caso como sólo tenemos uno realmente daría igual.

```

JS dev.env.js ×
codit > config > JS dev.env.js > ...
1   'use strict'
2
3   const merge = require('webpack-merge')
4   const prodEnv = require('./prod.env')
5
6   module.exports = merge(prodEnv, {
7     NODE_ENV: '"development"',
8     TOKEN: '"6a1ecaa18cd203a325661894aa606a7f6dd5d9ac"'
9   })
10

```

Una vez hecho esto, vamos a ver cómo conectarnos a un servicio externo.

Nuestro componente **CoDevelopers** va a ser el que se encargue de obtener los usuarios. Vamos a eliminar todo lo referente a la carga mediante mocks.

```

JS dev.env.js      ▼ CoDevelopers.vue ×      JS prod.env.js      App.pug      App.vue
codit > src > components > ▼ CoDevelopers.vue > () "CoDevelopers.vue" > [?] default > 🔗 methods > ⌂ getTopUsers
20   export default {
21     name: 'CoDevelopers',
22     data () {
23       return {
24         users: []
25       }
26     },
27     components: {
28       CoDeveloper
29     },
30     mounted () {
31       this.getTopUsers()
32     },
33     methods: {
34       getTopUsers () {
35         return fetch(`${process.env.API}search/users?language=javascript&order=desc&per_page=15`, {
36           headers: {
37             'Authorization': `token ${process.env.TOKEN}`
38           }
39         })
40       }
41     }
42   }
43 </script>
44
45   <style lang='css' scoped>
46     @import "../assets/css/colors.css";
47     @import "../assets/css/mixins.css";
48   </style>
49   .userConNombre {
50     font-weight: 900;
51   }

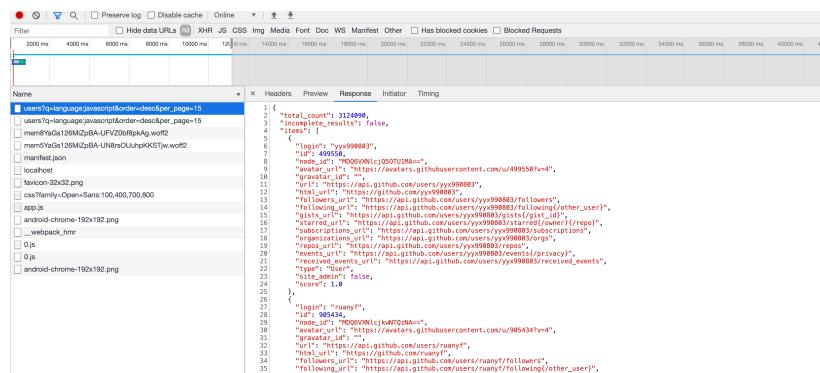
```

Vamos a crear un método **getTopUsers** para obtener los mejores desarrolladores. Vamos a utilizar **fetch** que es una petición AJAX que necesita introducir la ruta y las cabeceras. Github nos limita el número de peticiones y por tanto necesitamos el token.

Como la url la tenemos en las variables en el entorno de producción, podemos obtenerla mediante **process.env.API**.

La petición la vamos a hacer en el hook **mounted**.

Si todo ha ido bien, obtendremos un resultado de como máximo 15 desarrolladores en la response:



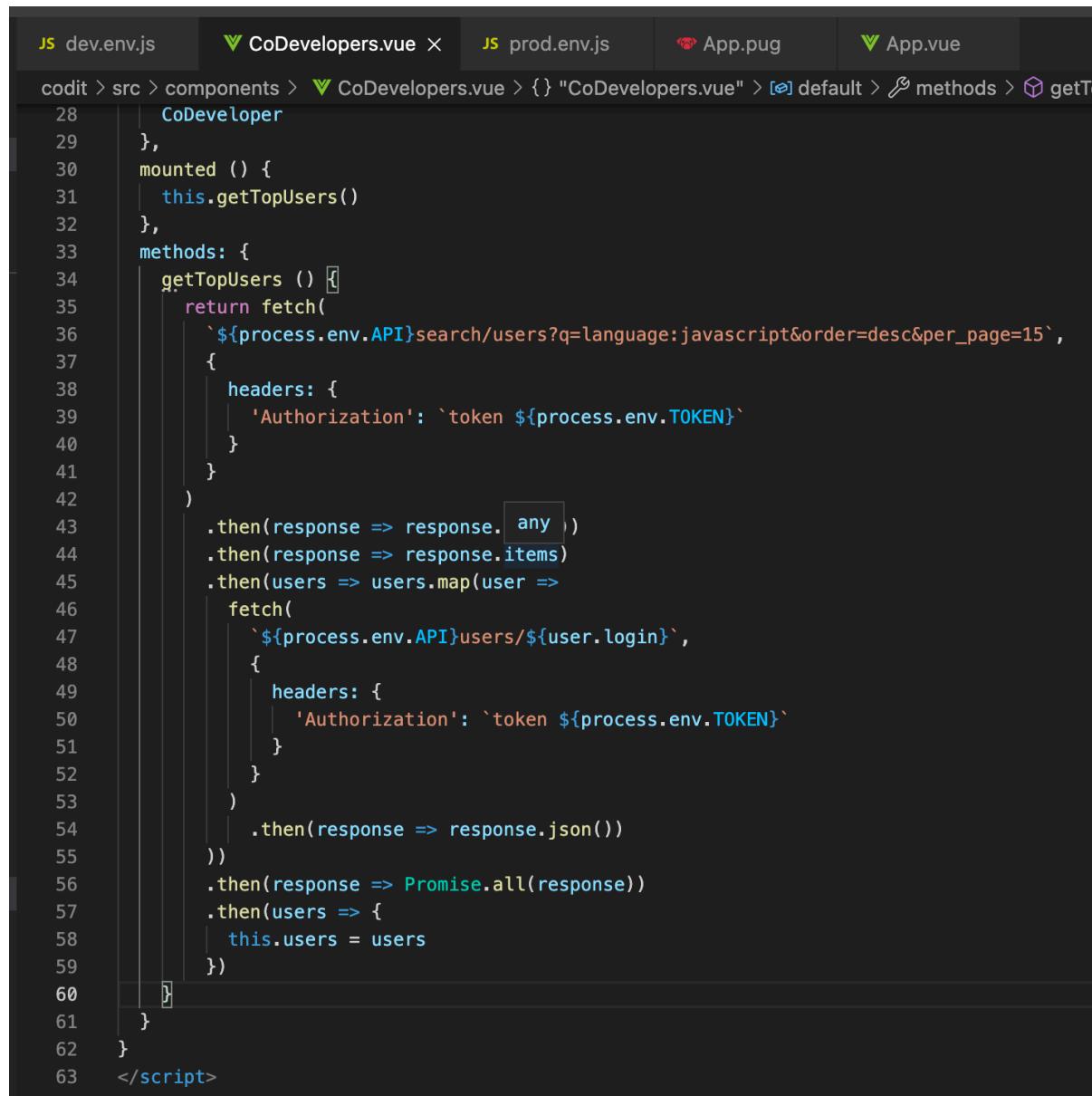
Fetch devuelve una promesa (objeto de javascript que representa la terminación con éxito o el fracaso de una operación asíncrona), y podemos obtener la respuesta que queremos tratar como json.

De esa promesa queremos obtener los items, nos faltaría el número de repos y el número de gists. Por tanto debemos concatenar peticiones, para cada user, necesitamos conseguir estos datos.

Lo que estamos haciendo es lo siguiente: estamos haciendo un fetch inicial en el que hacemos una petición **get** para obtener los usuarios más top de GitHub con lenguaje javascript y que me devuelva 15.

Una vez hecho esto, `.then` (promesa) me permite tratar esa respuesta que obtengo como json. A este resultado le decimos que sólo queremos los ítems y a dichos ítems (que serán los usuarios) vamos a tratarlos de nuevo porque para cada usuario nos faltan los datos de los repos y gists.

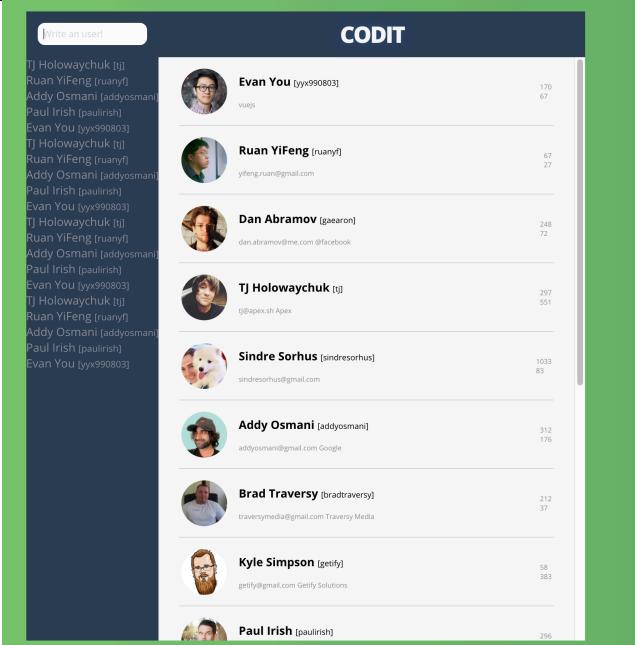
Para cada usuario, tendremos que hacer un nuevo fetch para obtener la información que me falta y esto lo vuelvo en un array. Obtengo dicha información, la trato también como un JSON. Como en este caso lo que tengo es un array de fetch, puedo gestionar todas las promesas con **Promise.all**, una vez hecho esto, obtengo un array de usuarios que asigno directamente a nuestra variable **this.users** y como resultado:



```

codit > src > components > ▼ CoDevelopers.vue > {} "CoDevelopers.vue" > default > ⚡ methods > ⌂ getTopUsers
28   },
29   },
30   mounted () {
31     this.getTopUsers()
32   },
33   methods: {
34     getTopUsers () {
35       return fetch(
36         `${process.env.API}search/users?q=language:javascript&order=desc&per_page=15`,
37       {
38         headers: {
39           'Authorization': `token ${process.env.TOKEN}`
40         }
41       }
42     )
43       .then(response => response. any )
44       .then(response => response.items)
45       .then(users => users.map(user =>
46         fetch(
47           `${process.env.API}users/${user.login}`,
48         {
49           headers: {
50             'Authorization': `token ${process.env.TOKEN}`
51           }
52         }
53       )
54         .then(response => response.json())
55       ))
56       .then(response => Promise.all(response))
57       .then(users => {
58         this.users = users
59       })
60     }
61   }
62 }
63 </script>

```



The screenshot shows a web application interface titled "CODIT". At the top, there is a search bar with the placeholder "Write an user!". Below the search bar is a list of developer profiles. Each profile includes a small circular profile picture, the developer's name, their GitHub handle, and some numerical statistics.

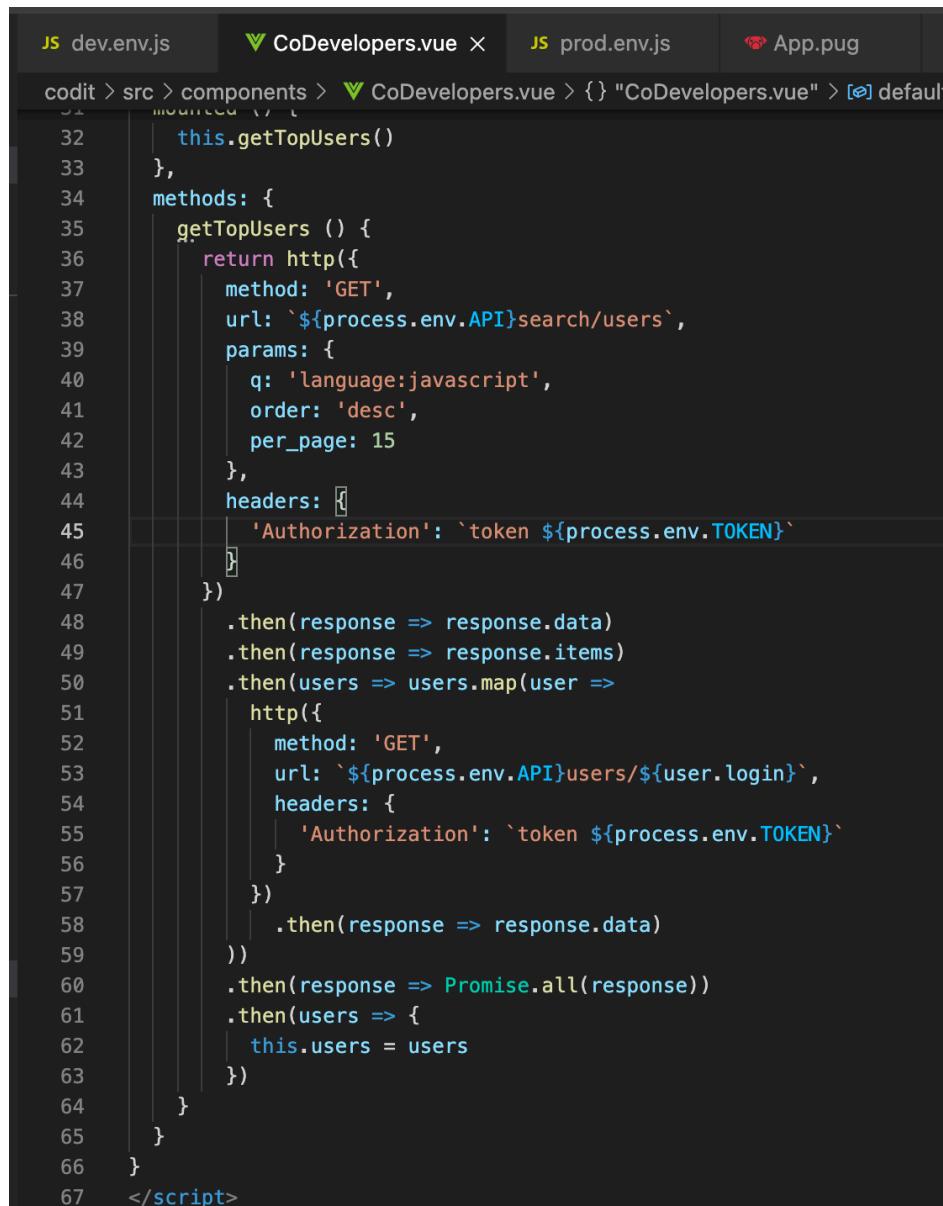
Developer	Handle	Profile Picture	Statistics
Evan You	yyx990803		170 67
Ruan YiFeng	ruanyf		67 27
Dan Abramov	gaearon		248 72
TJ Holowaychuk	tj		297 551
Sindre Sorhus	sindresorhus		1033 83
Addy Osmani	addyosmani		312 176
Brad Traversy	bradtraversy		212 37
Kyle Simpson	getify		58 383
Paul Irish	paulirish		296

Axios

Es un cliente HTTP que está basado en promesas y es ultraconfigurable.

Vamos a instalar axios.

```
inma@MacBook-Pro-de-Inmaculada codit % npm install --save axios
```



The screenshot shows a code editor with several tabs at the top: dev.env.js, CoDevelopers.vue (which is the active tab), prod.env.js, and App.pug. The CoDevelopers.vue tab contains the following code:

```
32     this.getTopUsers()
33   },
34   methods: {
35     getTopUsers () {
36       return http({
37         method: 'GET',
38         url: `${process.env.API}search/users`,
39         params: {
40           q: 'language:javascript',
41           order: 'desc',
42           per_page: 15
43         },
44         headers: [
45           'Authorization': `token ${process.env.TOKEN}`
46         ]
47       })
48       .then(response => response.data)
49       .then(response => response.items)
50       .then(users => users.map(user =>
51         http({
52           method: 'GET',
53           url: `${process.env.API}users/${user.login}`,
54           headers: [
55             'Authorization': `token ${process.env.TOKEN}`
56           ]
57         })
58         .then(response => response.data)
59       ))
60       .then(response => Promise.all(response))
61       .then(users => {
62         this.users = users
63       })
64     }
65   }
66 }
67 </script>
```

El resultado sería este.

The screenshot shows a user interface for a social network or directory service. At the top, there is a search bar with the placeholder text "Write an user!". Below the search bar, the word "CODIT" is displayed in large, bold, white letters. To the left of the main content area, there is a sidebar containing a list of user names and their corresponding profile pictures. The main content area displays a list of users with their names, profiles, and contact information. Each user entry includes a small profile picture, the user's name, a link to their profile, and two numerical values on the right side representing some form of activity or statistics.

User	Profile Picture	Name	Link	Value 1	Value 2
TJ Holowaychuk [tj]		Evan You [yx990803]	yx990803	170	67
Ruan YiFeng [ruanyf]		Ruan YiFeng [ruanyf]	ruanyf	67	27
Addy Osmani [adduosman]		Addy Osmani [adduosman]	adduosman		
Paul Irish [paulirish]		Paul Irish [paulirish]	paulirish		
Evan You [yx990803]		Evan You [yx990803]	yx990803		
TJ Holowaychuk [tj]		TJ Holowaychuk [tj]	tj		
Ruan YiFeng [ruanyf]		Ruan YiFeng [ruanyf]	ruanyf		
Addy Osmani [adduosman]		Addy Osmani [adduosman]	adduosman		
Paul Irish [paulirish]		Paul Irish [paulirish]	paulirish		
Evan You [yx990803]		Evan You [yx990803]	yx990803		
TJ Holowaychuk [tj]		TJ Holowaychuk [tj]	tj	297	551
Ruan YiFeng [ruanyf]		Ruan YiFeng [ruanyf]	ruanyf	67	27
Addy Osmani [adduosman]		Addy Osmani [adduosman]	adduosman		
Paul Irish [paulirish]		Paul Irish [paulirish]	paulirish		
Evan You [yx990803]		Evan You [yx990803]	yx990803		
Sindre Sorhus [sindresorhus]		Sindre Sorhus [sindresorhus]	sindresorhus@gmail.com	1033	83
Addy Osmani [adduosman]		Addy Osmani [adduosman]	adduosman@gmail.com	312	176
Brad Traversy [bradtraversy]		Brad Traversy [bradtraversy]	bradtraversy	212	37
Kyle Simpson [getify]		Kyle Simpson [getify]	getify	58	383
Paul Irish [paulirish]		Paul Irish [paulirish]	paulirish	296	