

EXERCISES QUE TRABAJAREMOS EN EL CUE:

- EXERCISE 1: INSTALANDO VUE Y CONFIGURACIÓN CON WEBPACK.
- EXERCISE 2: INSTALANDO VUE POR VUE/CLI.
- EXERCISE 3: ONE WAY DATA BINDING.
- EXERCISE 4: TWO WAY DATA BINDING.
- EXERCISE 5: DIRECTIVA V-IF.
- EXERCISE 6: DIRECTIVA V-FOR.

EXERCISE 1: INSTALANDO VUE Y CONFIGURACIÓN CON WEBPACK.

REQUERIMIENTOS:

1. Tener instalado node.js versión LTS. [Link Descarga Node](#)
2. Tener instalado gitbash (windows). [Link Descarga gitbash windows](#) (en caso de usar MAC, omitir)
3. Tener instalado Visual StudioCode.
4. Tener enlazado terminal git bash a terminal vscode. [Video explicación vinculación.](#)

INTRODUCCIÓN:

Si estamos trabajando en un proyecto ya más estructurado, donde estemos ocupando webpack, por ejemplo, podríamos incluir Vue para poder trabajar con él.

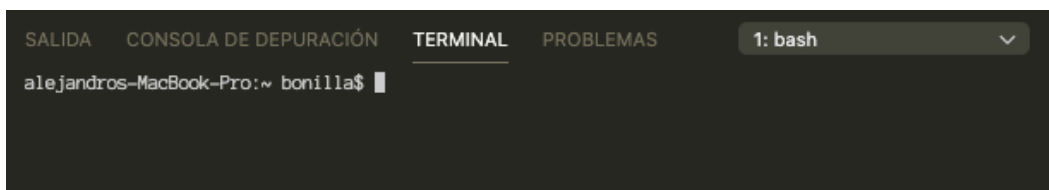
Sabiendo esto, vamos a instalar un proyecto con NPM desde 0, para que puedas observar como agregar Vue de manera correcta, y sin complicaciones.

Ahora, crearemos una carpeta llamada cue2, en el directorio donde deseemos trabajar. Como recomendación, intenta tener todos los proyectos en una carpeta llamada drillings, y dentro de éstas, crearemos las carpetas por CUE'S, en este caso: cue2.

INICIANDO

Abriremos Visual Studio Code, luego de esto, iremos a archivo y abriremos la carpeta cue2.

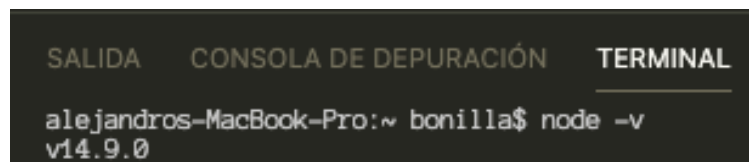
Para comenzar, debemos abrir el terminal de nuestro editor, y para ello iremos a View-> Terminal, que se encuentra en el menú de la aplicación.



```
SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PROBLEMAS  1: bash
alejandros-MacBook-Pro:~ bonilla$
```

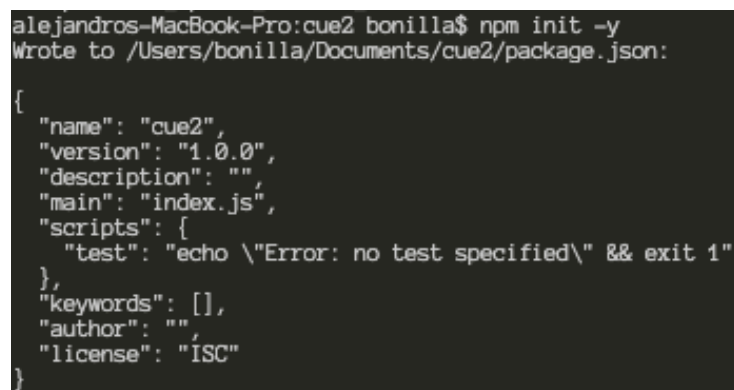
Una vez abierto el terminal, debemos escribir el siguiente comando. Para ello, ya tiene que estar instalado previamente **node.js** en nuestro computador.

Verificaremos que tenemos node, escribiendo el comando: node -v



```
SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL
alejandros-MacBook-Pro:~ bonilla$ node -v
v14.9.0
```

```
npm init -y
```



```
alejandros-MacBook-Pro:cue2 bonilla$ npm init -y
Wrote to /Users/bonilla/Documents/cue2/package.json:

{
  "name": "cue2",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

El comando anterior creará un archivo llamado package.json, el cual debemos modificar. En su interior encontraremos:

```
1 {
2   "name": "cue2",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC"
```

EDITANDO PACKAGE.JSON

Ahora debemos modificar la parte de scripts, y agregar devDependencies¹. El archivo debería quedar de la siguiente forma en las secciones mencionadas.

```
1 {
2   "name": "clase2_ejercicio1",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "serve": "webpack-dev-server --mode development"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC",
12  "devDependencies": {
13    "@babel/core": "^7.11.6",
14    "@babel/preset-env": "^7.11.5",
15    "babel-loader": "^8.1.0",
```

¹ DevDependencies: o dependencias de desarrollo, son librerías que nos ayudan a crear, desarrollar y empaquetar una aplicación web. Estas nos ayudaran a Instalar Vue, y todo lo necesario para poder trabajar.

Las dependencias son librerías gratuitas, las cuales se manejan a través de un gestor de paquetes llamado NPM.

```
16   "css-loader": "^3.6.0",
17   "html-webpack-plugin": "^4.4.1",
18   "vue": "^2.6.12",
19   "vue-loader": "^15.9.3",
20   "vue-style-loader": "^4.1.2",
21   "vue-template-compiler": "^2.6.12",
22   "webpack": "^4.44.1",
23   "webpack-cli": "^3.3.12",
24   "webpack-dev-server": "^3.11.0"
25 }
26 }
```

Después de editar el archivo y guardarlo (**ctrl+s**), debemos hacer la instalación de las dependencias.

INSTALANDO LAS DEPENDENCIAS AGREGADAS

Para instalar todas las librerías, debemos abrir el terminal y escribir el siguiente comando.

```
npm install
```

Al finalizar la instalación, nos damos cuenta de que se crea una nueva carpeta llamada **node_modules**.



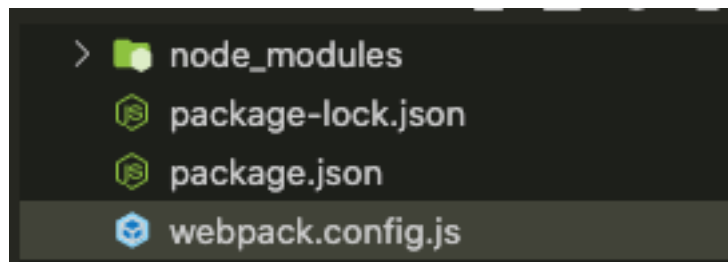
Si detallamos, dentro de **node_modules** quedaron instaladas todas las dependencias escritas en el archivo **package.json**, además de las librerías necesarias para node.

CONFIGURACIÓN WEBPACK

De fábrica, **webpack** no requerirá que se utilice un archivo de configuración. Sin embargo, asumirá que el punto de entrada de su proyecto generará el resultado minificado y optimizado para la producción.

src/index.js y **dist/main.js**

Por lo general, sus proyectos tendrán que ampliar esta funcionalidad, y para ello puede crear un archivo en la carpeta raíz.



webpack.config.js es el nombre del archivo que debemos crear, y éste quedará de la siguiente forma:

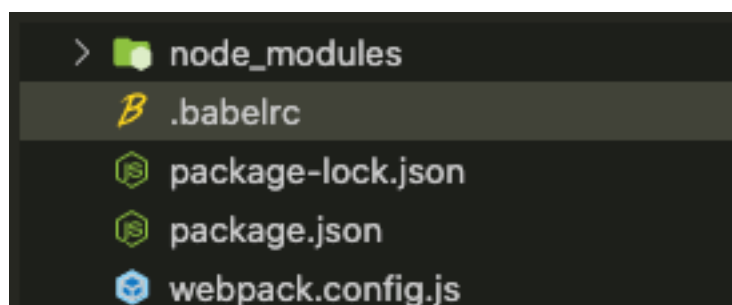
```
1 const VueLoaderPlugin = require('vue-loader/lib/plugin');
2 const HtmlWebpackPlugin = require('html-webpack-plugin');
3
4 module.exports = {
5   entry: './src/index.js',
6   module: {
7     rules: [
8       { test: /\.js$/, exclude: /node_modules/, loader: "babel-
9 loader" },
10      { test: /\.vue$/, use: 'vue-loader' },
11      { test: /\.css$/, use: ['vue-style-loader' , 'css-loader']}
12    ]
13  },
14  plugins: [
15    new HtmlWebpackPlugin({
16      template: './src/index.html'
17    }), new VueLoaderPlugin( ),]
18 };
```

Lo que buscamos, es indicarle la entrada del archivo a procesar por **webpack**, y agregar reglas para reconocer archivos con extensiones **.js**, **.vue** y **.css**, los cuales utilizarán algunas de las librerías que instalamos en las devDependencies.

AGREGANDO .BABELRC

Babel es un transpilador de código JavaScript, que significa transformar sintaxis nueva de JavaScript (ECMA2015 o superior), a un código más antiguo, el cual puede ser leído por navegadores desactualizados o dispositivos análogos.

Ahora, se debe agregar el archivo sin extensiones **.babelrc** en la raíz del proyecto.

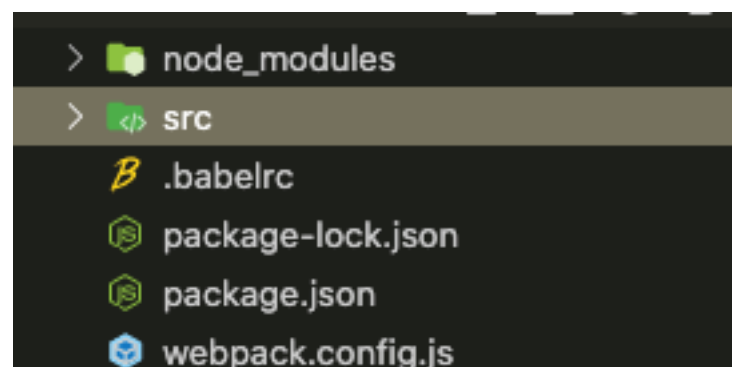


```
1 {  
2   "presets": ["@babel/preset-env"]  
3 }
```

CREAR CARPETA SRC

Se debe crear en la raíz de nuestro proyecto, y ésta contendrá toda nuestra aplicación vue. También podemos agregar carpetas para ordenar nuestro desarrollo, como: assets, components, routes, entre otros.

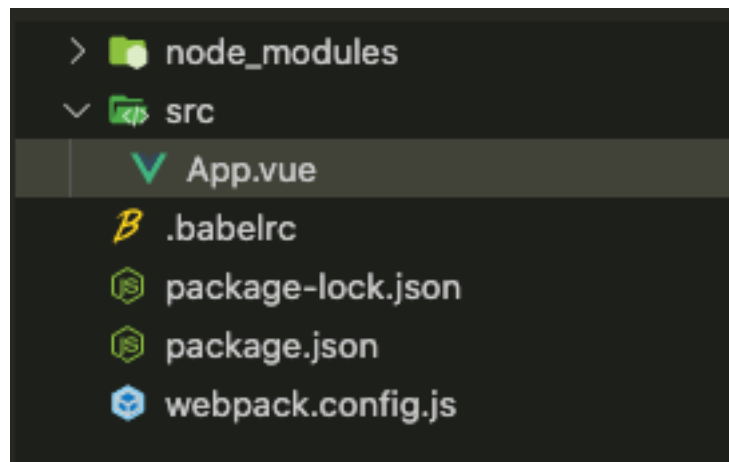
Como es un proyecto de inicio, solo tendremos el archivo principal de vue.



Empezaremos creando el archivo con extensión .vue, llamado App.vue, el cual es un: **SingleFileComponent(SFC).**

CREAR ARCHIVO APP.VUE

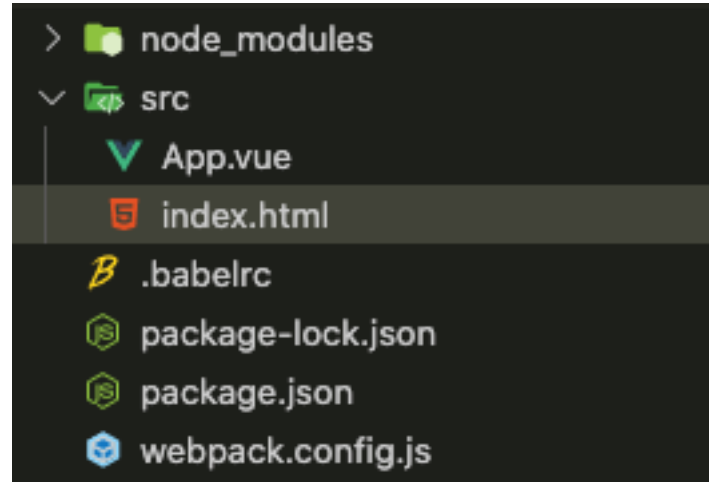
Se hará dentro de la carpeta src. Como recomendación, todos los archivos con extensión .vue, comenzaran con Mayúscula.



```
1 <template>
2   <div id="app">
3     {{mensaje}}
4   </div>
5 </template>
6
7 <script>
8 export default {
9   name: 'app-component',
10  data: function() {
11    return {
12      mensaje: "Hola en vue"
13    }
14  },
15 }
16 </script>
17
18 <style scoped>
19
20 </style>
```

CREAR INDEX.HTML

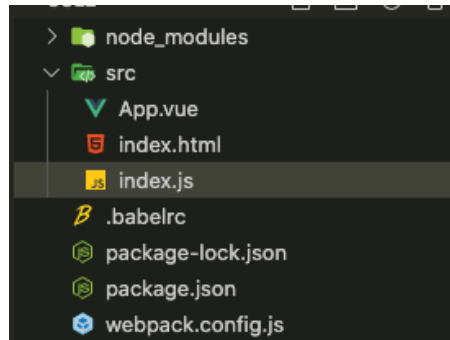
Se creará dentro de la carpeta src.



```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-
6 scale=1.0">
7   <title>Intro vue</title>
8 </head>
9 <body>
10   <div id="app"></div>
11 </body>
12 </html>
```


CREAR INDEX.JS

Por su parte, éste irá dentro de la carpeta src.



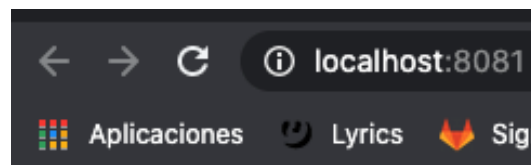
El archivo creado contiene a la librería de vue, y el archivo principal App.vue

```
1 import Vue from 'vue';
2 import App from './App.vue';
3
4 new Vue({
5   el: "#app",
6   render: h => h(App),
7 })
```

La instancia new vue, contiene como argumento un objeto, el cual integra dos elementos.

- El `el: "#app"`: es el div en el archivo App.vue, donde podremos utilizar todas las propiedades de vue.
- `render`: es la función que renderiza el componente principal App.vue

Este es el resultado que obtendremos al correr el siguiente comando en el terminal `npm run serve`



Hola en vue

EXERCICE 2: INSTALANDO VUE POR VUE/CLI

INTRODUCCIÓN:

Vue nos provee una manera de instalar un proyecto, a través de su CLI (Command Line Interface), y con esta opción, podemos agregar ciertas dependencias que se ocupan en sus proyectos, tales como: Vuex, vue-router, testl, entre otros, con solo pulsar unos comandos.

Para empezar a **crear** proyectos con Vue, previamente debemos instalar vue cli, a través del terminal.

INSTALANDO VUE/CLI

Necesitamos abrir el terminal de nuestro Visual Studio Code, o entorno bash.

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

Esto solo debemos hacerlo una vez, ya que estamos instalando vue/cli de forma global, lo que significa que, en cualquier parte del sistema, puedes invocarlo.

VERIFICANDO VERSION:

Si la instalación fue correcta, ahora podremos ver la versión instalada.

```
vue --version
```

UTILIZANDO VUE/CLI PARA CREAR PROYECTO:

Ahora, solo debes posicionarte en la carpeta que se desee trabajar **nombre_proyecto**, y debes remplazarlo por el nombre de tu proyecto.

Luego, abrimos terminal:

```
vue create nombre_proyecto
```

Vamos a seleccionar **Manually select features**, y damos enter.

```
Vue CLI v4.5.6
? Please pick a preset:
  Default ([Vue 2] babel, eslint)
  Default (Vue 3 Preview) ([Vue 3] babel, eslint)
> Manually select features
```

En esta fase debemos movernos con las flechas del teclado arriba y abajo, y para seleccionar un elemento a instalar, presionamos la tecla **espacio**. En este caso, agregaremos **Router**, una vez seleccionado le dan **enter**

```
Vue CLI v4.5.6

New version available 4.5.6 → 4.5.9
Run npm i -g @vue/cli to update!

? Please pick a preset: Manually select features
? Check the features needed for your project: (Press <space>
> to select, <a> to toggle all, <i> to invert selection)
  ● Choose Vue version
  ● Babel
  ○ TypeScript
  ○ Progressive Web App (PWA) Support
  ○ Router
  ○ Vuex
  ○ CSS Pre-processors
> ● Linter / Formatter
  ○ Unit Testing
  ○ E2E Testing
```

Versión vue: escogeremos la versión 2.x

```
Vue CLI v4.5.6

New version available 4.5.6 → 4.5.9
Run npm i -g @vue/cli to update!

? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue ve
rsion, Babel, Linter
? Choose a version of Vue.js that you want to start the pro
ject with (Use arrow keys)
> 2.x
  3.x (Preview)
```

Posterior a eso, solo dar aceptar a las preguntas de vue, ya que necesitamos lo básico.

Vue CLI v4.5.6

New version available 4.5.6 → 4.5.9
Run `npm i -g @vue/cli` to update!

? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel, Linter
? Choose a version of Vue.js that you want to start the project with 2.x
? Pick a linter / formatter config: (Use arrow keys)
> ESLint with error prevention only
ESLint + Airbnb config
ESLint + Standard config
ESLint + Prettier

Vue CLI v4.5.6

New version available 4.5.6 → 4.5.9
Run `npm i -g @vue/cli` to update!

? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel, Linter
? Choose a version of Vue.js that you want to start the project with 2.x
? Pick a linter / formatter config: Basic
? Pick additional lint features: (Press <space> to select, <a> to toggle all, <i> to invert selection)
> ☒ Lint on save
 ☐ Lint and fix on commit

Vue CLI v4.5.6

New version available 4.5.6 → 4.5.9
Run `npm i -g @vue/cli` to update!

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel, Linter
? Choose a version of Vue.js that you want to start the project with 2.x
? Pick a linter / formatter config: Basic
? Pick additional lint features: Lint on save
? Where do you prefer placing config for Babel, ESLint, etc.? (Use arrow keys)
> In dedicated config files
  In package.json
```

Vue CLI v4.5.6

New version available 4.5.6 → 4.5.9
Run `npm i -g @vue/cli` to update!

```
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel, Linter
? Choose a version of Vue.js that you want to start the project with 2.x
? Pick a linter / formatter config: Basic
? Pick additional lint features: Lint on save
? Where do you prefer placing config for Babel, ESLint, etc.? In dedicated config files
? Save this as a preset for future projects? (y/N) N
```

Por último, le pondremos **No** a la pregunta: "a save this as a preset for future projects?" N

Ya con eso, vue cli comenzará a crear el proyecto, y si todo sale correctamente, debería verse así:

```
found 0 vulnerabilities

🚀 Invoking generators...
📦 Installing additional dependencies...

added 59 packages from 39 contributors and audited 1314 packages in 22.937s

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

found 0 vulnerabilities

⚓ Running completion hooks...

📄 Generating README.md...

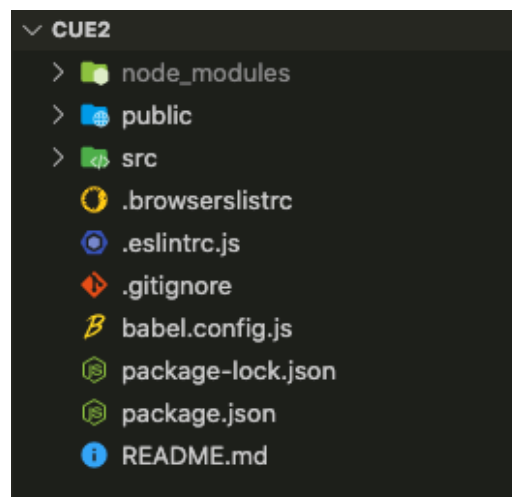
🎉 Successfully created project clase_router.
👉 Get started with the following commands:

$ cd clase_router
$ npm run serve

bonilla@alejandros-MacBook-Pro Documents %
```

Al terminar la instalación, notaremos que vue/cli ha creado y configura, el entorno inicial para comenzar.

Si abrimos el proyecto con Visual Studio Code:



Si abrimos la consola, y ejecutamos “**npm run serve**”:

```
> cue2@0.1.0 serve /Users/bonilla/Documents/cue2/cue2
> vue-cli-service serve

INFO Starting development server...
98% after emitting CopyPlugin

DONE Compiled successfully in 9603ms

App running at:
- Local:   http://localhost:8081/
- Network: http://192.168.0.2:8081/

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

Si todo se desarrolla como lo estimado, se podrá observar que la app ha sido levantada en un localhost, esto dependerá de nuestros puertos disponibles.

Así que, si vamos al navegador y ponemos la anterior dirección, se verá así:



Welcome to Your Vue.js App

For a guide and recipes on how to configure / customize this project,
check out the [vue-cli documentation](#).

Installed CLI Plugins

[babel](#) [eslint](#)

Essential Links

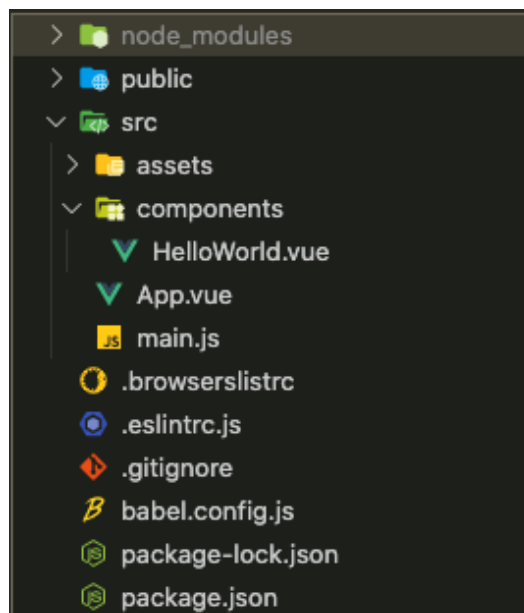
[Core Docs](#) [Forum](#) [Community Chat](#) [Twitter](#) [News](#)

EXERCISE 3: ONE WAY DATA BINDING.

INSTRUCCIONES:

Para comenzar, utilizaremos el proyecto desarrollado en el ejercicio número 2, en el cual creamos un proyecto con vue/cli.

Éste lo abriremos con Visual Studio Code, y deberíamos tener las siguientes carpetas:



Ahora, lo que haremos en primer lugar, es eliminar el component HelloWorld.vue, que se encuentra dentro de la carpeta componentes.

Una vez eliminado el archivo, editaremos App.vue.

ONE WAY DATA BINDING:

Se basa en poder mostrar al usuario datos que tengamos en nuestro objeto data, dentro del template. Puede hacerse de dos formas:

1. Renderizando los datos a través de los *mustaches* ({{}})
2. Utilizando la propiedad v-bind.

1-RENDERIZADO:

```

1 <template>
2   <div>
3     <p>{{info}}</p>
4   </div>
5 </template>
6 <script>
7 export default {
8   name: "One-way-component",
9   data: function() {
10    return {
11      info: "Enlace por una sola dirección",
12    }
13  }
14 }
15 </script>

```

Si observamos, el enlace se produce desde los datos, hacia el template.

2-PROPIEDAD V-BIND:

Podemos enlazar datos a cualquier atributo html que necesitemos. En el siguiente ejemplo, se muestra la propiedad v-bind , en su forma corta, solo anteponiendo los dos puntos (:)

```

1 <template>
2   <div>
3     
4   </div>
5 </template>
6
7 <script>
8 export default {
9   name: "One-way-component",
10  data: function() {
11    return {
12      logo_src: "https://vuejs.org/images/logo.png",
13    }
14  }
15 }
16 </script>

```

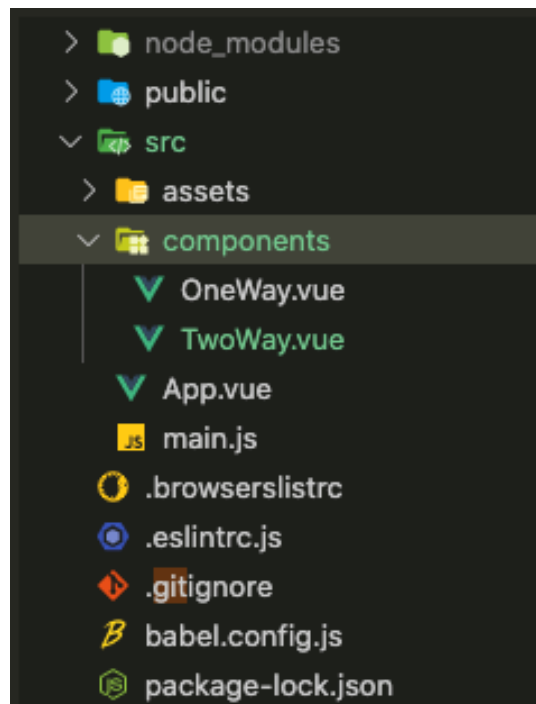
EXERCISE 4: TWO WAY DATA BINDING

INSTRUCCIONES:

Para seguir trabajando, utilizaremos el mismo proyecto desarrollado en los ejercicios 2 y 3.

Ahora, vamos a crear un nuevo SFC (Single File Component), dentro de la carpeta components.

A éste le llamaremos TwoWay.vue.



Una vez listo, procederemos a llamarlo dentro del archivo principal App.vue. En éste, vamos a comentar el componente creado en el ejercicio anterior.

```
1 <template>
2   <div id="app">
3     <!-- <OneWayComponent></OneWayComponent> -->
4     <TwoWayComponent></TwoWayComponent>
5   </div>
6 </template>
```

```
7
8 <script>
9
10 // import OneWayComponent from '@components/OneWay.vue'
11 import TwoWayComponent from '@components/TwoWay.vue'
12
13 export default {
14   name: 'App',
15   components: {
16     // OneWayComponent,
17     TwoWayComponent,
18   }
19 }
20 </script>
21
22 <style>
23 #app {
24   font-family: Avenir, Helvetica, Arial, sans-serif;
25   -webkit-font-smoothing: antialiased;
26   -moz-osx-font-smoothing: grayscale;
27   text-align: center;
28   color: #2c3e50;
29   margin-top: 60px;
30 }
31 </style>
```

TWO WAY DATA BINDING:

Se basa en poder mostrar datos desde la data hacia el template, y mutarlos desde el template hacia la data.

DIRECTIVA V-MODEL:

Nos sirve para poder hacer el enlace en dos vías (Two Way Data Binding). No hay una contraparte directa de v-model en las funciones de renderizado, para esto, tendrá que implementar la lógica usted mismo; esta directiva solo puede ser aplicada a los siguientes elementos HTML:

1. Input.
2. Text área.
3. Select option.

EJEMPLO: TWO WAY DATA BINDING

Utilizaremos la directiva v-model en un input HTML. Éste podrá mostrar el dato “nombre” en el input, y mutarlo cuando el usuario ingrese datos.

El siguiente script será el contenido de nuestro componente TwoWay.vue.

```
1 <template>
2   <div>
3     <h1>Two way data binding (Enlace en dos direcciones)</h1>
4     <input type="text" v-model="nombre">
5     <h2>{{nombre}}</h2>
6   </div>
7 </template>
8
9 <script>
10 export default {
11   name: "Two-way-component",
12   data: function() {
13     return {
14       nombre: ""
15     }
16   }
17 }
18 </script>
```

Si guardamos y levantamos el Proyecto sin haber hecho “npm run serve”, veremos en el navegador.

Two Way data binding (enlace en dos direcciones)

Si nos fijamos en el template, el dato nombre está 2 veces, el primero a través a de un input con un v-model, y el segundo a través de un h2.

Al momento de ir rellenando el input, podremos ver la reactividad del dato nombre.

Two Way data binding (enlace en dos direcciones)

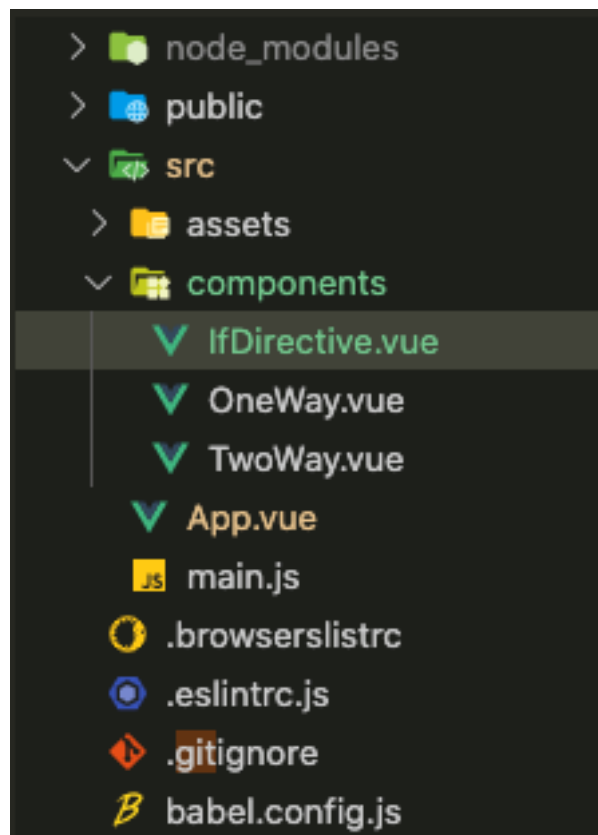
hola mundo

EXERCISE 5: DIRECTIVA V-IF

INSTRUCCIONES:

Para continuar aprendiendo sobre la directiva v-if, lo seguiremos haciendo en el mismo proyecto utilizado en los ejercicios 2, 3 y 4.

Vamos a crear un SFC (Single File Component) dentro de la carpeta components, llamado IfDirective.vue.



Luego, iremos a App.vue, para agregar el nuevo componente, y comentar el que creamos en el ejercicio anterior.

```
1 <template>
2   <div id="app">
3     <!-- <OneWayComponent></OneWayComponent> -->
4     <!-- <TwoWayComponent></TwoWayComponent> -->
5     <IfDirectiveComponent></IfDirectiveComponent>
6   </div>
7 </template>
8
9 <script>
10
11 // import OneWayComponent from '@components/OneWay.vue'
12 // import TwoWayComponent from '@components/TwoWay.vue'
13 import IfDirectiveComponent from '@components/IfDirective.vue'
14
15 export default {
16   name: 'App',
17   components: {
18     // OneWayComponent,
19     // TwoWayComponent,
20     IfDirectiveComponent
21   }
22 }
23 </script>
24
25 <style>
26 #app {
27   font-family: Avenir, Helvetica, Arial, sans-serif;
28   -webkit-font-smoothing: antialiased;
29   -moz-osx-font-smoothing: grayscale;
30   text-align: center;
31   color: #2c3e50;
32   margin-top: 60px;
33 }
34 </style>
```

DIRECTIVA V-IF:

Nos permite condicionar el renderizado de ciertos elementos HTML. Para utilizarlo, necesitamos crear una condición de verdad, con algún dato que tengamos en la data.

DIRECTIVAS ASOCIADAS:

V-ELSE-IF:

Nos da la opción, en caso de cumplirse el if, de poner un segundo condicional.

V-ELSE:

Se puede recurrir a éste, en caso de ningún condicional cumplido.

EJEMPLO

Ahora, veremos como a través de un dato, podemos crear variadas condiciones para mostrar distintos mensajes.

En este caso, observaremos como la propiedad v-if toma una condición de verdad, y si el div cumple con la directiva, mostrará la información. De no cumplirse el v-if, se pasará al v-else-if siguiente, y si la condición no se cumple, nuevamente se pasará a otro.

```
1 <template>
2   <div>
3     <h2>Contador de likes {{likes}}</h2>
4     <div v-if="likes === 0">No hay likes</div>
5     <div v-else-if="likes < 10">Vamos por los 10 likes</div>
6     <div v-else-if="likes < 20">Vamos por los 20 likes</div>
7     <div v-else>Eres una estrella de los likes</div>
8     <button v-on:click="addLike">Agregar</button>
9     <button @click="removeLike">Quitar</button>
10  </div>
11 </template>
```

```
1 <script>
2 export default {
3   name: "If-component",
4   data: function() {
5     return {
6       likes: 0,
7     }
8   },
9   methods: {
10    addLike() {
```

```

11     this.likes++;
12 },
13     removeLike() {
14         if(this.likes > 0) {
15             this.likes--;
16         }
17     }
18 }
19 }
20 </script>

```

Al partir el componente, el **v-if** cumple su condición, ya que la variable **"likes"** inicia su valor en 0.



Por último, si damos clic en el botón **"Agregar Like"**, veremos que la condición de **v-if** no se cumple, pero si la del **v-else-if** siguiente.

Contador de likes 1

Vamos por los 10 likes

Agregar Like

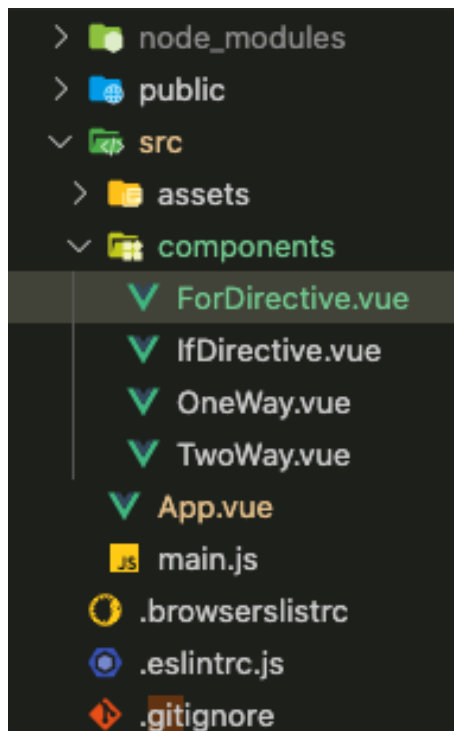
Quitar Like

EXERCISE 6: DIRECTIVA V-FOR

INSTRUCCIONES:

Para continuar aprendiendo sobre la directiva v-for, lo haremos en el mismo proyecto utilizado en los ejercicios 2, 3, 4 y 5.

Nuevamente vamos a crear un SFC (Single File Component) dentro de components, esta vez llamado ForDirective.vue.



Como lo hemos hecho en los ejercicios anteriores, luego de crear el componente, vamos a ir a App.vue, para poder instanciarlo.

```

1 <template>
2   <div id="app">
3     <!-- <OneWayComponent></OneWayComponent> -->
4     <!-- <TwoWayComponent></TwoWayComponent> -->
5     <!-- <IfDirectiveComponent></IfDirectiveComponent> -->
6     <ForDirective></ForDirective>
7   </div>
8 </template>
9
10 <script>
11
12 // import OneWayComponent from '@components/OneWay.vue'
13 // import TwoWayComponent from '@components/TwoWay.vue'
14 // import IfDirectiveComponent from
15 '@components/IfDirective.vue'
16 import ForDirective from '@components/ForDirective.vue'
17 export default {
18   name: 'App',
19   components: {
20     // OneWayComponent,
21     // TwoWayComponent,
```

```
22     // IfDirectiveComponent
23     ForDirective
24   }
25 }
26 </script>
27
28 <style>
29 #app {
30   font-family: Avenir, Helvetica, Arial, sans-serif;
31   -webkit-font-smoothing: antialiased;
32   -moz-osx-font-smoothing: grayscale;
33   text-align: center;
34   color: #2c3e50;
35   margin-top: 60px;
36 }
37 </style>
```

DIRECTIVA V-FOR

Nos puede ser útil al momento de trabajar con Array; estos pueden ser Array simple de datos (string, números, booleanos), a Array de objetos. La propiedad de v-for, es la que nos permite iterar entre datos de un Array, y poder mostrarlo de manera sencilla a través del template.

EJEMPLO

v-for: “Esta directiva nos permite iterar sobre un elemento creado; en este caso, tomamos el Array skills, y por cada resultado nos devolverá un skill en específico”.

:key= cuando realizamos un v-for por obligación, Vue nos pide una llave única para cada dato. En este caso, sería el mismo dato iterado.

```
1 <template>
2
3   <section>
4     <h1>Mis conocimientos en programación web son:</h1>
5     <div class="container_skill">
6       <ul>
7         <li v-for="skill in skills" :key="skill">{{skill}}</li>
8       </ul>
9       <input type="text" v-model="skill">
10      <button @click="addSkill" :disabled="skill == ''">Agregar
11 skill</button>
12    </div>
13  </section>
14
15</template>
```

```
1 <script>
2 export default {
3   name: "For-component",
4   data: function() {
5     return {
6       skills: ['HTML', 'CSS', 'JS'],
7       skill: "",
8     }
9   },
10  methods: {
11    addSkill() {
12      //agregar una nueva skill
13      this.skills.push(this.skill)
14      //limpiar input
15      this.skill = ""
16    },
17  }
18 }
19 </script>
```

Si levantamos nuestra aplicación con “npm run serve”, y luego abrimos el Puerto asociado, deberíamos poder observar lo siguiente:

Mis conocimientos en programación son:

- HTML
- CSS
- JS

 Agrega skill

Ahora, al agregar una nueva skill, se integrará un nuevo dato al Array ya renderizado, pero vue, al detectar un cambio de estado, vuelve a renderizar los datos.

Mis conocimientos en programación son:

- HTML
- CSS
- JS

 Agrega skill

Por último, al momento de hacer clic en el botón, éste automáticamente gatilla el método asociado “addSkill”, el cual toma el valor del input y lo agrega al Array skills; permitiendo que v-for itere sobre un nuevo dato.

Mis conocimientos en programación son:

- HTML
- CSS
- JS
- php

 Agrega skill