

1. Instalación de la herramienta Vue CLI.

Vue CLI es una aplicación de terminal o línea de comandos que te ayudará a desarrollar aplicaciones Vue. Vamos a ver cómo instalarla globalmente, de modo que puedas crear aplicaciones en cualquier lugar de tu sistema. Para ello debes abrir una ventana de terminal y ejecutar uno de los comandos que ves a continuación, dependiendo del gestor de paquetes que utilices:

```
jsersan@iMac-de-Jose ~ % sudo npm i -g @vue/cli
Password:
npm WARN deprecated source-map-url@0.4.1: See https://github.com/lydell/source-map-url#deprecated
npm WARN deprecated urix@0.1.0: Please see https://github.com/lydell/urix#deprecated
npm WARN deprecated resolve-url@0.2.1: https://github.com/lydell/resolve-url#deprecated
npm WARN deprecated source-map-resolve@0.5.3: See https://github.com/lydell/source-map-resolve#deprecated
npm WARN deprecated subscriptions-transport-ws@0.11.0: The `subscriptions-transport-ws` package is no longer recommended you use `graphql-ws` instead. For help migrating Apollo software to `graphql-ws`, see https://www.docs.apollo-server/data/subscriptions/#switching-from-subscriptions-transport-ws For general help use https://github.com/enisdenjo/graphql-ws/blob/master/README.md

added 21 packages, removed 70 packages, changed 827 packages, and audited 849 packages in 17s
!
64 packages are looking for funding
  run `npm fund` for details

4 vulnerabilities (2 moderate, 2 high)
```

2. Creación del proyecto.

Ahora que ya tenemos la aplicación Vue CLI instalada en nuestro sistema, ya podemos usar el comando `vue`. Creamos el proyecto `vue-dom`:

`$ vue create vue-dom`

```
Vue CLI v5.0.8
? Please pick a preset: (Use arrow keys)
> Default ([Vue 3] babel, eslint)
  Default ([Vue 2] babel, eslint)
  Manually select features
```

Seleccionamos la opción Vue 3:

```
added 849 packages, and audited 850 packages in 21s

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

found 0 vulnerabilities
Invoking generators...
Installing additional dependencies...

added 95 packages, and audited 945 packages in 10s

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

found 0 vulnerabilities
Running completion hooks...
Generating README.md...
Successfully created project vue-dom.
Get started with the following commands:

$ cd vue-dom
$ npm run serve
```

Lanzamos el proyecto:

```
jrsersan@iMac-de-Jose ~ % cd vue-dom
jrsersan@iMac-de-Jose vue-dom % npm run serve
```

Resultado:

DONE Compiled successfully in 2698ms

App running at:

- Local: <http://localhost:8080/>
- Network: <http://192.168.0.14:8080/>

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

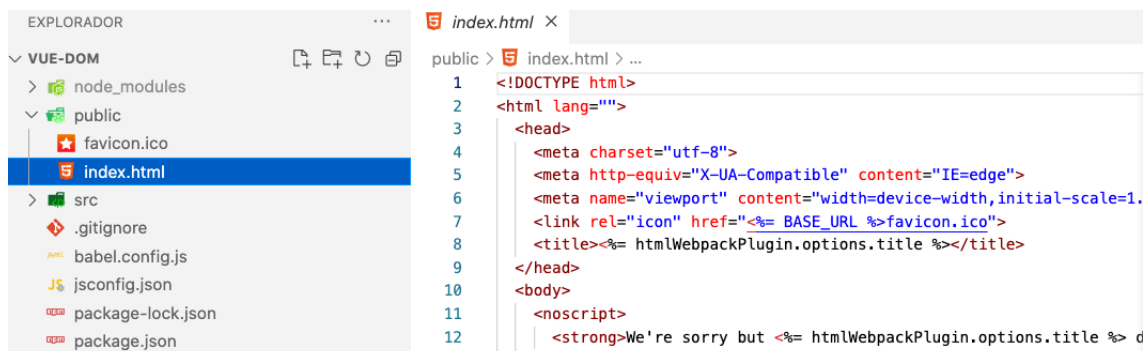
Nos vamos al navegador al puerto seleccionado:



Welcome to Your Vue.js App

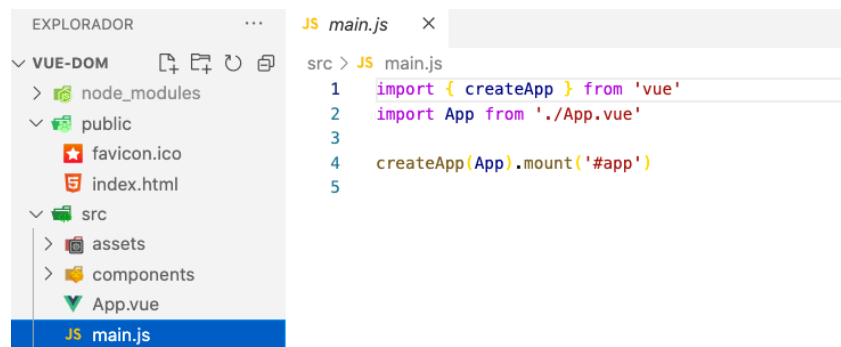
For a guide and recipes on how to configure / customize this project,
check out the [vue-cli documentation](#).

Abrimos el proyecto en vs code:

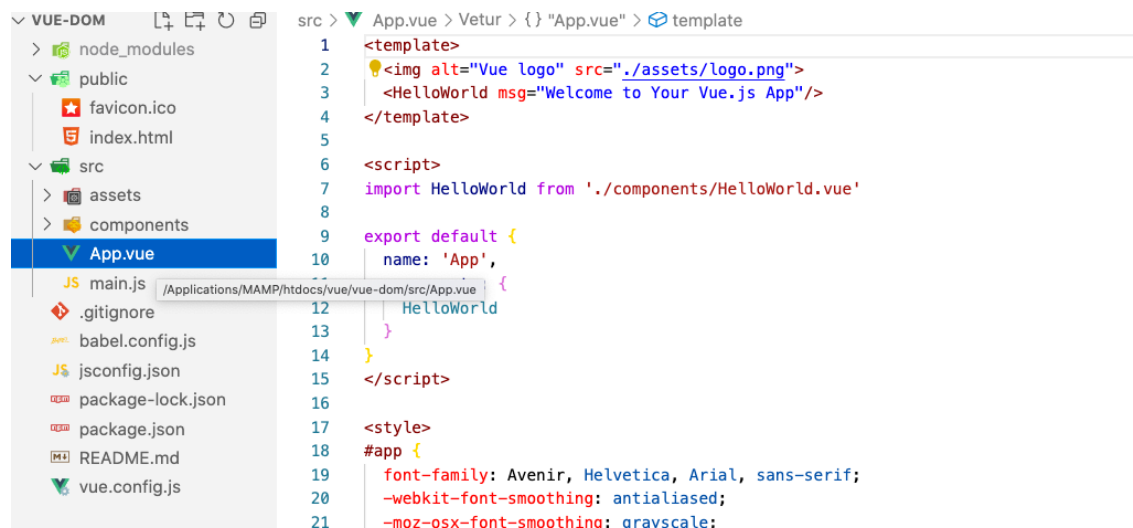


En el directorio que se ha creado verás que hay un directorio llamado `public`, en cuyo interior encontrarás el archivo `index.html`. Este es el directorio en el que se situarán los archivos ya compilados. El directorio en donde estará tu código será el directorio `src`, en donde encontrarás el archivo `main.js`, que es el punto de entrada de la aplicación.

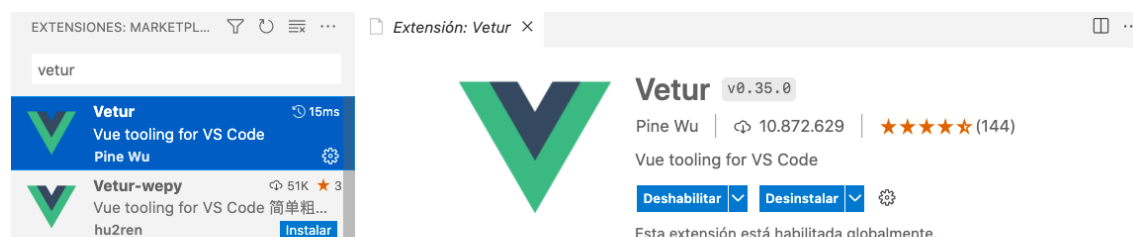
En el archivo `main.js` importamos el método `createApp` y el código principal de la aplicación, localizado en el archivo `App.vue`. Para crear la aplicación sencillamente ejecutamos el método `createApp(App)`. Para incluirla o montarla y que se renderice en un elemento HTML de nuestra página, que en este caso es el `div #app`, usamos el método `mount('#app')`:



Si accedes al archivo `App.vue`, verás que se importa un componente de ejemplo llamado `HelloWorld.vue`, cuyo código está en otro archivo. Explicaremos lo que es un componente y cómo se estructuran los archivos `.vue` más adelante.



Comprobamos que tenemos instalado en vs code el plugin vetur:



3. Incluir Bootstrap.

Como ya hemos hablado, el objetivo de este curso no es el de que aprendas a usar CSS, por lo que vamos a usar un framework. Agregaría el framework **Tailwind** (proyecto MERN), que está en auge. Sin embargo, para que este tutorial resulte más sencillo a programadores noveles, usaremos **Bootstrap**.

Para incluir Bootstrap en el proyecto, abre el archivo `public/index.html` y agrega la siguiente etiqueta en la sección `head`:

```
index.html M X
public > index.html > Vue Language Features (Volar) > { } template
1  <!DOCTYPE html>
2  <html lang="">
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=1.0">
7      <link rel="icon" href="<%= BASE_URL %>favicon.ico">
8      <title><%= htmlWebpackPlugin.options.title %></title>
9      <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/cs
10    </head>
11    <body>
12      <noscript>
13        <strong>We're sorry but <%= htmlWebpackPlugin.options.title %> doesn't work prop
14      </noscript>
15      <div id="app"></div>
16      <!-- built files will be auto injected -->
17    </body>
18  </html>
```

4. Creación del componente personas. Template.

```
1  <template>
2    <div id="tabla-personas">
3      <table class="table">
4        <thead>
5          <tr>
6            <th>Nombre</th><th>Apellido</th><th>Email</th>
7          </tr>
8        </thead>
9        <tbody>
10         <tr>
11           <td>Jon</td><td>Nieve</td><td>jon@email.com</td>
12         </tr>
13         <tr>
14           <td>Tyrion</td><td>Lannister</td><td>tyrion@email.com</td>
15         </tr>
16         <tr>
17           <td>Daenerys</td><td>Targaryen</td><td>daenerys@email.com</td>
18         </tr>
19       </tbody>
20     </table>
21   </div>
22 </template>
```

En el template se puede ver como tenemos una tabla con los personajes de Juego de Tronos donde se puede visualizar los campos nombre, apellido y email.



Además en Template.vue tenemos la parte de script:

```

25 <script>
26   export default {
27     name: 'tabla-personas',
28   }
29 </script>
30
31 <style scoped></style>

```

El archivo completo es el siguiente código, en donde incluiremos una sencilla tabla HTML. Además, verás que incluimos el div `<div id="tabla-personas">` antes de la tabla. El motivo es que en todos los **componentes** en Vue deben contener un **único elemento como raíz**:

src > components >  TablaPersonas.vue > Vue Language Features (Volar) > {} template >  div#tab

```

1  <template>
2    <div id="tabla-personas">
3      <table class="table">
4        <thead>
5          <tr>
6            <th>Nombre</th><th>Apellido</th><th>Email</th>
7          </tr>
8        </thead>
9        <tbody>
10         <tr>
11           <td>Jon</td><td>Nieve</td><td>jon@email.com</td>
12         </tr>
13         <tr>
14           <td>Tyrion</td><td>Lannister</td><td>tyrion@email.com</td>
15         </tr>
16         <tr>
17           <td>Daenerys</td><td>Targaryen</td><td>daenerys@email.com</td>
18         </tr>
19       </tbody>
20     </table>
21   </div>
22 </template>
23
24 <script>
25   export default {
26     name: 'tabla-personas',
27   }
28 </script>
29
30 <style scoped></style>

```

5. Agrega el componente de la aplicación.

Una vez hemos creado y exportado el componente `TablaPersonas`, vamos a importarlo en el archivo `App.vue`. Para importar el archivo escribiremos la siguiente sentencia en la parte superior de la sección `script` del archivo `App.vue`:

```
import TablaPersonas from '@/components/TablaPersonas.vue'
```

Tal y como ves, podemos usar el carácter `@` para referenciar al directorio `src`. Seguidamente, vamos a agregar el componente `°` a la propiedad `components`, de modo que Vue sepa que puede usar este componente. Debes agregar todos los componentes que crees a esta propiedad.

Para renderizar el componente `TablaPersonas`, basta con agregar `<tabla-personas />`, en Kebab case, al código HTML. A continuación puedes ver el código completo que debes incluir en el archivo `App.vue`, reemplazando al código existente:

```
src > ▾ App.vue > Vue Language Features (Volar) > {} style > #app
1  <template>
2  <div id="app" class="container">
3    <div class="row">
4      <div class="col-md-12">
5        <h1>Personas</h1>
6      </div>
7    </div>
8    <div class="row">
9      <div class="col-md-12">
10       <tabla-personas />
11     </div>
12   </div>
13 </div>
14 </template>
15
16 <script>
17 import TablaPersonas from '@/components/TablaPersonas.vue'
18
19 export default {
20   name: 'App',
21   components: {
22     TablaPersonas
23   }
24 }
25 </script>
26
27 <style>
28 #app
29   font-family: Avenir, Helvetica, Arial, sans-serif;
30   -webkit-font-smoothing: antialiased;
31   -moz-osx-font-smoothing: grayscale;
32   text-align: center;
33   color: #2c3e50;
34   margin-top: 60px;
35
36 </style>
```

Si ves de nuevo el proyecto en tu navegador, deberías ver este resultado:

Personas

Nombre	Apellido	Email
Jon	Nieve	jon@email.com
Tyion	Lannister	tyrion@email.com
Daenerys	Targaryen	daenerys@email.com

6. Agrega datos al componente.

Actualmente solamente tenemos texto estático en la tabla de nuestro componente. Antes de continuar, para aquellos que hayan usado React, debéis saber que el método `data` de Vue equivale al estado de los componentes en React.

Vamos a reemplazar los datos de las personas por un array de objetos que contengan los datos de las personas. Por ello, vamos a agregar el método `data` a nuestra aplicación en el archivo `App.vue`, que sencillamente devolverá los datos de los usuarios. Además, cada una tendrá un identificador. En `App.vue`:

```
16 <script>
17 import TablaPersonas from '@components/TablaPersonas.vue'
18
19 export default {
20   name: 'app',
21   components: {
22     TablaPersonas,
23   },
24   data() {
25     return {
26       personas: [
27         {
28           id: 1,
29           nombre: 'Jon',
30           apellido: 'Nieve',
31           email: 'jon@email.com',
32         },
33         {
34           id: 2,
35           nombre: 'Tyion',
36           apellido: 'Lannister',
37           email: 'tyrion@email.com',
38         },
39         {
40           id: 3,
41           nombre: 'Daenerys',
42           apellido: 'Targaryen',
43           email: 'daenerys@email.com',
44         },
45       ],
46     },
47   },
48 }
49 </script>
```

Una vez hemos agregado los datos en el componente `App.js`, debemos pasárselos al componente `TablaPersonas`. Los datos se transfieren como propiedades, y se incluyen con la sintaxis `:nombre="datos"`. Es decir, que se tratan igual que un **atributo** HTML, salvo por el hecho de tener **dos puntos** : delante:

```

9 | <div class="col-md-12">
10 |   <tabla-personas v-bind:personas="personas" />
11 | </div>

```

A continuación debemos aceptar los datos de las personas en el componente `TablaPersonas`. Para ello debemos definir la **propiedad** `personas` en el objeto `props`. El objeto `props` debe contener todas aquellas propiedades que va a recibir el componente, conteniendo pares que contienen el **nombre de la propiedad** y el **tipo** de la misma. En nuestro caso, la propiedad `personas` es un `Array`. En el componente `TablaPersonas.Vue`:

```

24 <script>
25   export default {
26     name: 'tabla-personas',
27     props: {
28       personas: Array,
29     }
30   }
31 </script>

```

Ahora vamos a vincular este origen de datos importado con el template HTML. Una vez hemos agregado los datos al componente `TablaPersonas`, vamos a crear un bucle que recorra los datos de las personas, mostrando una fila de la tabla en cada iteración. Para ello usaremos el atributo `v-for`, que nos permitirá recorrer los datos de una propiedad, que en nuestro caso es la propiedad `personas`:

```

src > components > ▼ TablaPersonas.vue > Vue Language Features (Volar) > {} template > div#tabla-personas
1  <template>
2    <div id="tabla-personas">
3      <table class="table">
4        <thead>
5          <tr>
6            <th>Nombre</th><th>Apellido</th><th>Email</th>
7          </tr>
8        </thead>
9        <tbody>
10         <tr v-for="persona in personas" :key="persona.id">
11           <td>{{persona.nombre}}</td>
12           <td>{{persona.apellido}}</td>
13           <td>{{persona.email}}</td>
14         </tr>
15       </tbody>
16     </table>
17   </div>
18 </template>
19
20 <script>
21   export default {
22     name: 'tabla-personas',
23     props: {
24       personas: Array,
25     }
26   }
27 </script>

```


Vista previa con un nuevo personaje de Juego de Tronos:

Personas

Nombre	Apellido	Email
Jon	Nieve	jon@email.com
Tyrion	Lannister	tyrion@email.com
Daenerys	Targaryen	daenerys@email.com
Sansa	Stark	daenerys@email.com

7. Creación de formularios.

Vamos a crear el archivo `FormularioPersona.vue` en la carpeta `src/componentes`, en el que agregaremos un formulario con un campo `input` para el nombre, otro para el apellido y otro para el email de la persona a agregar. Finalmente también agregaremos un botón de tipo `submit` que nos permita enviar los datos.

En cuanto al código JavaScript, crearemos la propiedad `persona` como una propiedad que será devuelta por el componente, que incluirá el nombre, el apellido y el email de la persona que se agregue. Este sería el código del template:

```
src > components > FormularioPersona.vue > Vetur > {} "FormularioPersona.vue" > style
1  <template>
2    <div id="formulario-persona">
3      <form>
4        <div class="container">
5          <div class="row">
6            <div class="col-md-4">
7              <div class="form-group">
8                <label>Nombre</label>
9                <input type="text" class="form-control" />
10             </div>
11           </div>
12           <div class="col-md-4">
13             <div class="form-group">
14               <label>Apellido</label>
15               <input type="text" class="form-control" />
16             </div>
17           </div>
18           <div class="col-md-4">
19             <div class="form-group">
20               <label>Email</label>
21               <input type="email" class="form-control" />
22             </div>
23           </div>
24         </div>
25         <div class="row">
26           <div class="col-md-4">
27             <div class="form-group">
28               <button class="btn btn-primary">Añadir persona</button>
29             </div>
30           </div>
31         </div>
32       </div>
33     </form>
34   </div>
35 </template>
~
```

El resto del archivo FormularioPersona.vue:

```

37 <script>
38   export default {
39     name: 'formulario-persona',
40     data() {
41       return {
42         persona: {
43           nombre: '',
44           persona: '',
45           apellido: '',
46         },
47       }
48     },
49   }
50 </script>
51
52 <style scoped>
53   form {
54     margin-bottom: 2rem;
55   }
56 </style>

```

Una vez creado el formulario, debemos añadir éste a la App. Vamos a editar el archivo **App.js** y a agregar el componente FormularioPersona que hemos creado:

```

src > App.vue > Vue Language Features (Volar) > {} template > div#app.container
1  <template>
2  <div id="app" class="container">
3    <div class="row">
4      <div class="col-md-12">
5        <h1>Personas</h1>
6      </div>
7    </div>
8    <div class="row">
9      <div class="col-md-12">
10       <formulario-persona />
11       <tabla-personas v-bind:personas="personas" />
12     </div>
13   </div>
14 </div>
15 </template>
16
17 <script>
18
19 import TablaPersonas from '@components/TablaPersonas.vue'
20 import FormularioPersona from '@components/FormularioPersona.vue'
21
22 export default {
23   name: 'app',
24   components: {
25     TablaPersonas,
26     FormularioPersona
27   },
28   data() {}

```

Si ahora accedes al proyecto en tu navegador, verás que se muestra el formulario encima de la tabla:

Personas

Nombre

Apellido

Email

Nombre	Apellido	Email
Jon	Nieve	jon@email.com
Tyrion	Lannister	tyrion@email.com
Daenerys	Targaryen	daenerys@email.com
Sansa	Stark	daenerys@email.com

7.1 Enlazar los campos del formulario con su estado.

A continuación debemos obtener los valores que se introduzcan en el formulario mediante JavaScript, de modo que podamos asignar sus valores al estado del componente.

Para ello usamos el atributo `v-model`, que enlazará el valor de los campos con sus respectivas variables de estado, que son las definidas en la propiedad `persona` de la sentencia `return` del componente:

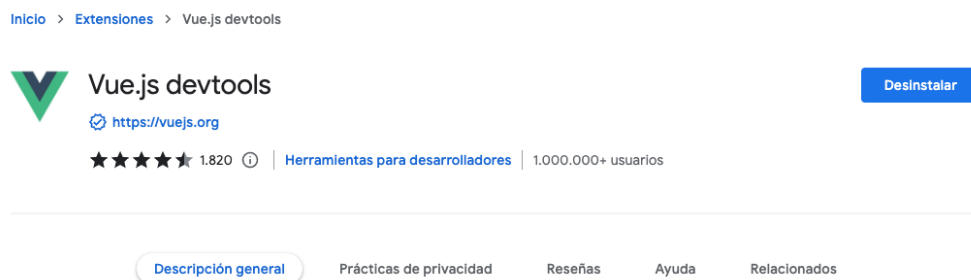
```

4  <div class="container">
5  <div class="row">
6  <div class="col-md-4">
7  <div class="form-group">
8  <label>Nombre</label>
9  <input v-model="persona.nombre" type="text" class="form-control" />
10 </div>
11 </div>
12 <div class="col-md-4">
13 <div class="form-group">
14 <label>Apellido</label>
15 <input v-model="persona.apellido" type="text" class="form-control" />
16 </div>
17 </div>
18 <div class="col-md-4">
19 <div class="form-group">
20 <label>Email</label>
21 <input v-model="persona.email" type="email" class="form-control" />
22 </div>
23 </div>
24 </div>

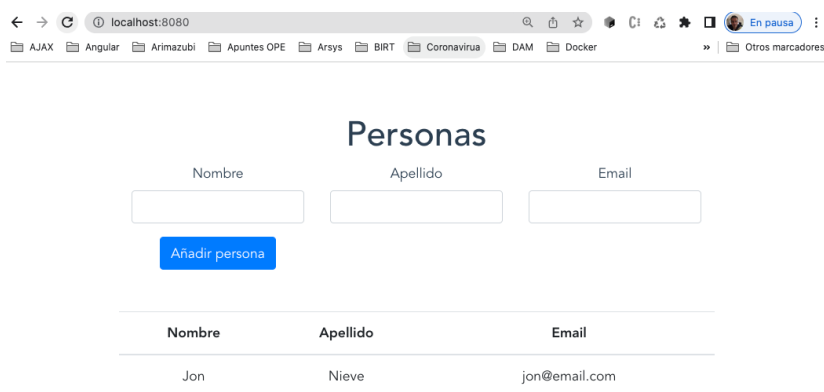
```

Si consultas el resultado en tu navegador y accedes a las **DevTools** de Vue, podrás ver cómo cambia el estado del componente cada vez que modificas un campo del formulario.

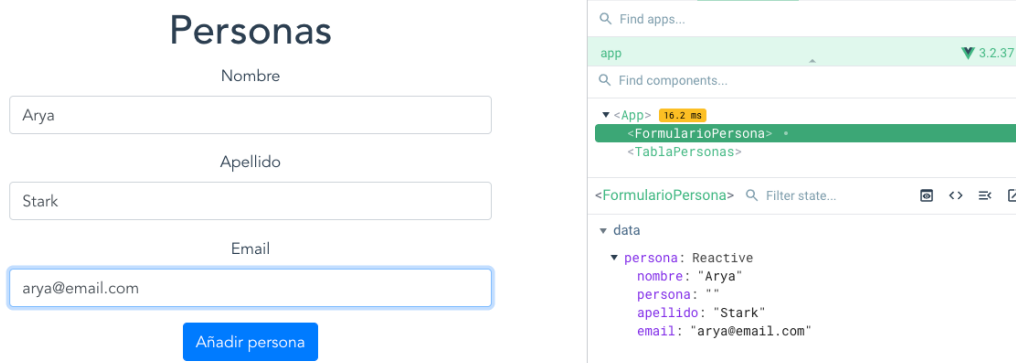
Comprobamos que tenemos instaladas las herramientas de Desarrollo de Vue en el navegador:



Abrimos la app en Google Chrome y vemos el primer registro:



Miramos las herramientas de desarrollo de Vue y agregando un nuevo personaje:



Sin embargo, todavía tenemos que enviar los datos a nuestro componente principal, que es la aplicación App en sí misma, de modo que también se modifique su estado, agregando los datos de una nueva persona a la lista de personas.


7.2 Agrega un método de envío al formulario

Vamos a agregar un **evento**, también conocido como **event listener**, al formulario. En concreto, agregaremos un evento `onSubmit` para que se ejecute un método cuando se haga clic en el botón de envío. Para ello usaremos al atributo `@submit`, que es la forma corta del atributo `v-on:submit`, siendo ambos equivalentes.

Información! En general todos los listeners de eventos en Vue se agregan con los prefijos `@` o `v-on:`. Por ello, podríamos detectar un clic con `@click` o `v-on:click`, y un evento hover con `@mouseover` o `v-on:mouseover`.

Además, también deberíamos ejecutar el método `event.preventDefault()`, para evitar que se refresque la página al enviar el formulario. Para ello, el evento `@submit` cuenta con el modificador `prevent`, que es equivalente a ejecutar el método `event.preventDefault()` en el interior de la función asociada al evento `submit`.

Vamos a asociar el método `enviarFormulario` al evento `@submit` del formulario:

```
src > components >  FormularioPersona.vue > Vetur > {} "FormularioPer:
1  <template>
2    <div id="formulario-persona">
3      <form @submit.prevent="enviarFormulario">
4        <div class="container">
```

Ahora vamos a agregar el método `enviarFormulario` al componente. Los métodos de los componentes de Vue se incluyen en el interior de la propiedad `methods`, que también vamos a crear:

```
37  <script>
38    export default {
39      name: 'formulario-persona',
40      data() {
41        return {
42          persona: {
43            nombre: '',
44            persona: '',
45            apellido: '',
46          },
47        },
48      },
49      methods: {
50        enviarFormulario() {
51          console.log('Funciona!!')
52        },
53      },
54    },
55  </script>
```

Si pruebas el código y envías el formulario, verás que por la consola se muestra el texto `Funciona!!`.

Resultado:

7.3 Emite eventos del formulario a la aplicación.

Ahora necesitamos enviar los datos de la persona que hemos agregado a nuestra aplicación **App**, para ello usaremos el método **\$emit** en el método **enviarFormulario**. El método **\$emit** envía el nombre del evento que definamos y los datos que deseemos al componente en el que se ha renderizado el componente actual.

En nuestro caso, enviaremos la propiedad **persona** y un evento al que llamaremos **add-persona**:

```

49     methods:{
50       enviarFormulario(){
51         console.log(this.persona);
52         this.$emit('add-persona', this.persona);
53       },
54     },
55   }

```

Es importante que tengas en cuenta que el nombre de los eventos debe seguir siempre la sintaxis kebab-case (add-persona). El evento emitido permitirá iniciar el método que reciba los datos en la aplicación App:

7.4 Recibe eventos de la tabla en la aplicación.

El componente **FormularioPersona** de App.vue envía los datos a través del evento **add-persona**. Ahora debemos capturar los datos en la aplicación. Para ello, agregaremos la propiedad **@add-persona** en la etiqueta **formulario-persona** mediante la cual incluimos el componente. En ella, asociaremos un nuevo método al evento, al que llamaremos **agregarPersona**:

```

10 <formulario-persona @add-persona="agregarPersona" />
11 <tabla-personas v-bind:personas="personas" />

```

Seguidamente, creamos el método **agregarPersona** en la propiedad **methods** del archivo **App.vue**, que modificará el array de **personas** que se incluye, agregando un nuevo objeto al mismo:

```
28 | data(){
29 |   return {
30 |     personas: [
31 |       {
32 |         id: 1,
33 |         nombre: 'Jon',
34 |         apellido: 'Nieve',
35 |         email: 'jon@email.com',
36 |       },
37 |       {
38 |         id: 2,
39 |         nombre: 'Tyrion',
40 |         apellido: 'Lannister',
41 |         email: 'tyrion@email.com',
42 |       },
43 |       {
44 |         id: 3,
45 |         nombre: 'Daenerys',
46 |         apellido: 'Targaryen',
47 |         email: 'daenerys@email.com',
48 |       },
49 |       {
50 |         id: 4,
51 |         nombre: 'Sansa',
52 |         apellido: 'Stark',
53 |         email: 'daenerys@email.com',
54 |       },
55 |     ],
56 |   },
57 | },
58 | methods: {
59 |   agregarPersona(persona) {
60 |     this.personas = [...this.personas, persona];
61 |   }
62 | },
```

Hemos usado el operador spread de propagación ..., útil para **combinar objetos y arrays**, para crear un nuevo array que contenga los elementos antiguos del array personas junto con la nueva. Si ahora ejecutas las DevTools y envías el formulario, verás que se agrega un nuevo elemento al array de personas:

The screenshot shows a web application titled "Personas" with a form to add a new person. The form has three input fields: "Nombre" (containing "Txema"), "Apellido" (containing "Serrano"), and "Email" (containing "jsersan@gmail.com"). Below the inputs is a blue button labeled "Añadir persona". Below the form is a table listing existing people.

Nombre	Apellido	Email
Jon	Nieve	jon@email.com
Tyryon	Lannister	tyryon@email.com
Daenerys	Targaryen	daenerys@email.com
Sansa	Stark	daenerys@email.com
Txema	Serrano	jsersan@gmail.com

To the right, the DevTools component inspector shows the component tree. The selected component is <FormularioPersona>, which is a child of <App>. Below it, the props for <TablaPersonas> are shown, including a reactive array named "personas" with 5 elements. The last element (index 4) is a reactive object with the following properties:

```

{
  nombre: "Txema",
  persona: "",
  apellido: "Serrano",
  email: "jsersan@gmail.com"
}

```

Sin embargo, es importante que asignemos un ID único al elemento que acabamos de crear. Habitualmente, insertaríamos la persona creada en una base de datos, que nos devolvería la persona junto con un nuevo ID. Sin embargo, por ahora nos limitaremos a generar un ID basándonos en el ID del elemento inmediatamente anterior al actual:

```

58 |   methods: {
59 |     agregarPersona(persona) {
60 |       let id = 0;
61 |
62 |       if (this.personas.length > 0) {
63 |         id = this.personas[this.personas.length - 1].id + 1;
64 |       }
65 |
66 |       this.personas = [...this.personas, { ...persona, id }];
67 |     },
68 |   },

```

Lo que hemos hecho es aumentar el valor del ID del último elemento agregado en una unidad, o dejarlo en 0 si no hay elementos. Luego insertamos la persona en el array, a la que agregamos el id generado.

8. Validaciones con Vue.

Aunque funciona el formulario, todavía tenemos que **mostrar una notificación** cuando un usuario se inserte correctamente, **reestablecer el foco** en el primer elemento del formulario y **vaciar los campos de datos**. Además, debemos asegurarnos de que se han rellenado todos los campos con **datos válidos**, mostrando un **mensaje de error** en caso contrario.

8.1 Propiedades computadas de Vue

En Vue, los datos se suelen validar mediante **propiedades computadas o computed properties**, que son funciones que se ejecutan automáticamente cuando se modifica el estado de alguna propiedad. De este modo evitamos sobrecargar el código HTML del componente. Las propiedades computadas se agregan en el interior de la propiedad **computed**, que añadiremos justo después de la propiedad **methods** del componente **FormularioPersona**:

```

68 | },
69 | computed: {
70 |   nombreInvalido() {
71 |     return this.persona.nombre.length < 1;
72 |   },
73 |   apellidoInvalido() {
74 |     return this.persona.apellido.length < 1;
75 |   },
76 |   emailInvalido() {
77 |     return this.persona.email.length < 1;
78 |   },
79 | },

```

Hemos agregado una validación muy sencilla que simplemente comprueba que se haya introducido algo en los campos. A continuación vamos a incluir una variable de estado en el componente **FormularioPersona** a la que llamaremos **procesando**, que comprobará si el formulario se está enviando actualmente o no.

También agregaremos las variables **error** y **correcto**. El valor de la variable **error** será **true** si el formulario se ha enviado correctamente o **false** si ha habido algún error. Del mismo modo, el valor de la variable **correcto** será **true** si el formulario se ha enviado correctamente o **false** en caso contrario:

```

40 |   data() {
41 |     return {
42 |       procesando: false,
43 |       correcto: false,
44 |       error: false,
45 |       persona: {
46 |         nombre: '',
47 |         persona: '',
48 |         apellido: '',
49 |       },
50 |     },
51 |   },

```

Seguidamente, debemos modificar el método **enviarFormulario** para que establezca el valor de la variable de estado **procesando** como **true** cuando se esté enviado el formulario, y como **false** cuando se obtenga un resultado. En función del resultado obtenido, se establecerá el valor de la variable **error** como **true** si ha habido algún **error**, o el valor de la variable **correcto** como **true** si se han enviado los datos correctamente. En FormularioPersona.vue:

```
52   methods: {
53     enviarFormulario() {
54       this.procesando = true;
55       this.resetEstado();
56
57       // Comprobamos la presencia de errores
58       if (this.nombreInvalido || this.apellidoInvalido || this.emailInvalido) {
59         this.error = true;
60         return;
61       }
62
63       this.$emit('add-persona', this.persona);
64
65       this.error = false;
66       this.correcto = true;
67       this.procesando = false;
68
69       // Restablecemos el valor de la variables
70       this.persona = {
71         nombre: '',
72         apellido: '',
73         email: '',
74       };
75     },
76     resetEstado() {
77       this.correcto = false;
78       this.error = false;
79     }
80   },
81 }
```

Como ves, hemos agregado también el método **resetEstado** para resetear algunas variables de estado.

8.2 Sentencias condicionales con Vue

A continuación vamos a modificar el código HTML de nuestro formulario. Tal y como has podido comprobar ya al observar las DevTools, Vue renderiza de nuevo cada componente cada vez que se modifica el estado de un componente. De este modo, los eventos se gestionan con mayor facilidad.

Dicho esto, vamos a configurar el formulario de modo que se agregue la clase CSS **has-error** a los campos del mismo en función de si han fallado o no. También agregaremos el posible mensaje de error al final del formulario.

En FormularioPersona.vue:

```

3  <form @submit.prevent="enviarFormulario">
4    <div class="container">
5      <div class="row">
6        <div class="col-md-4">
7          <div class="form-group">
8            <label>Nombre</label>
9            <input v-model="persona.nombre" type="text" class="form-control"
10              :class="{ 'is-invalid': procesando && nombreInvalido }" @focus="resetEstado" />
11          </div>
12        </div>
13        <div class="col-md-4">
14          <div class="form-group">
15            <label>Apellido</label>
16            <input v-model="persona.apellido" type="text" class="form-control"
17              :class="{ 'is-invalid': procesando && apellidoInvalido }" @focus="resetEstado" />
18          </div>
19        </div>
20        <div class="col-md-4">
21          <div class="form-group">
22            <label>Email</label>
23            <input v-model="persona.email" type="email" class="form-control"
24              :class="{ 'is-invalid': procesando && emailInvalido }" @focus="resetEstado" />
25          </div>
26        </div>
27      </div>
28      <div class="row">
29        <div class="col-md-4">
30          <div class="form-group">
31            <button class="btn btn-primary">Añadir persona</button>
32          </div>
33        </div>
34      </div>
35    </div>
36    <div class="container">
37      <div class="row">
38        <div class="col-md-12">
39          <div v-if="error && procesando" class="alert alert-danger" role="alert">
40            Debes rellenar todos los campos!
41          </div>
42          <div v-if="correcto" class="alert alert-success" role="alert">
43            La persona ha sido agregada correctamente!
44          </div>
45        </div>
46      </div>
47    </div>
48  </form>

```

Tal y como has visto, hemos usado el tributo **:class** para definir las clases, ya que no podemos usar atributos que existan actualmente en HTML. El motivo de no usar el atributo **class** es que acepta únicamente una cadena de texto como valor.

También hemos agregado el evento **@focus** a los campos **input** para que se reseteen los estados cada vez que se seleccionen.

8.3 Referencias con Vue

Cuando envías un formulario, lo normal es que tanto el cursor como el foco, por temas de **accesibilidad web**, se sitúen en el primer elemento del formulario, que en nuestro caso es el campo **nombre**. Para ello podemos usar las **referencias de Vue** descritas en el enlace de la página anterior, ya que permiten hacer referencia a los elementos que las incluyen. Para agregar una referencia debes usar el atributo **ref**.

A continuación vamos a agregar una referencia al campo **nombre**:

```

9      <input
10      ref="nombre"
11      v-model="persona.nombre"
12      type="text"
13      class="form-control"
14      :class="{ 'is-invalid': procesando && nombreInvalido }"
15      @focus="resetEstado"
16      @keypress="resetEstado"
17  />

```

Finalmente, tras enviar el formulario, vamos a usar el método **focus** que incluyen las referencias para que el cursor se sitúe en el campo **nombre**:

Personas

Nombre

Apellido

Email

Añadir persona

La persona ha sido agregada correctamente!

Nombre	Apellido	Email
Jon	Nieve	jon@email.com
Tyrion	Lannister	tyrion@email.com
Daenerys	Targaryen	daenerys@email.com
Sansa	Stark	daenerys@email.com
Txema	Serrano	jsersan@gmail.com

Find apps...

app 3.2.37

Find components...

<App> 15.8.86

<FormularioPersona>

<TablaPersonas>

También hemos agregado un evento **@keypress** al campo **nombre** para que el estado se resetee cuando se pulse una tecla, puesto que el foco ya estará en dicho campo.

Si ahora pruebas a agregar una persona, verás que el cursor se sitúa en el primer elemento.

9. Elimina elementos con Vue.

El formulario ya funciona correctamente, pero vamos a agregar también la opción de poder borrar personas.

9.1 Agrega un botón de borrado a la tabla.

Para ello, vamos a agregar una columna más a la tabla del componente TablaPersonas, que contendrá un botón que permita borrar cada fila. En el fichero TablaPersonas.vue:

```
src > components > ▼ TablaPersonas.vue > Vue Language Features (Volar) > {} template > div#tabla-per
1  <template>
2    <div id="tabla-personas">
3      <table class="table">
4        <thead>
5          <tr>
6            <th>Nombre</th>
7            <th>Apellido</th>
8            <th>Email</th>
9            <th>Acciones</th>
10         </tr>
11       </thead>
12       <tbody>
13         <tr v-for="persona in personas" :key="persona.id">
14           <td>{{ persona.nombre }}</td>
15           <td>{{ persona.apellido }}</td>
16           <td>{{ persona.email }}</td>
17           <td>
18             <button class="btn btn-danger">🗑 Eliminar</button>
19           </td>
20         </tr>
21       </tbody>
22     </table>
23   </div>
24 </template>
```

9.2 Emite un evento de borrado desde la tabla

Ahora, al igual que hemos hecho en el formulario, debemos emitir un evento al que llamaremos **eliminarPersona**. Este evento enviará el **id** de la persona a eliminar al componente padre, que en este caso es la aplicación **App.vue**. Así, en TablaPersonas.vue:

```
12   <tbody>
13     <tr v-for="persona in personas" :key="persona.id">
14       <td>{{ persona.nombre }}</td>
15       <td>{{ persona.apellido }}</td>
16       <td>{{ persona.email }}</td>
17       <td>
18         <button class="btn btn-danger"
19           @click="$emit('delete-persona', persona.id)">
20           🗑 Eliminar
21         </button>
22       </td>
23     </tr>
24   </tbody>
```

9.3 Recibe el evento de borrado en la aplicación.

Ahora, en la aplicación **App.vue** debes agregar una acción que ejecute un método que elimina a la **persona** correspondiente con el **id** recibido:

```

8      <div class="row">
9        <div class="col-md-12">
10         <formulario-persona @add-persona="agregarPersona" />
11         <tabla-personas v-bind:personas="personas" @delete-persona="eliminarPersona" />
12       </div>
13     </div>

```





Ahora definimos también en **App.vue** el método **eliminarPersona** justo debajo del método **agregarPersona** que hemos creado anteriormente:

```


58     methods: {
59       agregarPersona(persona) {
60         let id = 0;
61
62         if (this.personas.length > 0) {
63           id = this.personas[this.personas.length - 1].id + 1;
64         }
65
66         this.personas = [...this.personas, { ...persona, id }];
67       },
68       eliminarPersona(id) {
69         this.personas = this.personas.filter(
70           persona => persona.id !== id
71         );
72       }
73     }

```

Hemos usado el método **filter**, que conservará aquellos elementos del array **personas** cuyo **id** no sea el indicado. Con esto, si consultas el navegador, podrás comprobar que las personas se eliminan de la tabla al pulsar el botón de su respectiva fila. De la tabla siguiente, si le damos al botón Eliminar de todos menos Sansa:

Nombre	Apellido	Email	Acciones
Jon	Nieve	jon@email.com	 Eliminar
Tyrion	Lannister	tyrion@email.com	 Eliminar
Daenerys	Targaryen	daenerys@email.com	 Eliminar
Sansa	Stark	daenerys@email.com	 Eliminar

Tenemos:

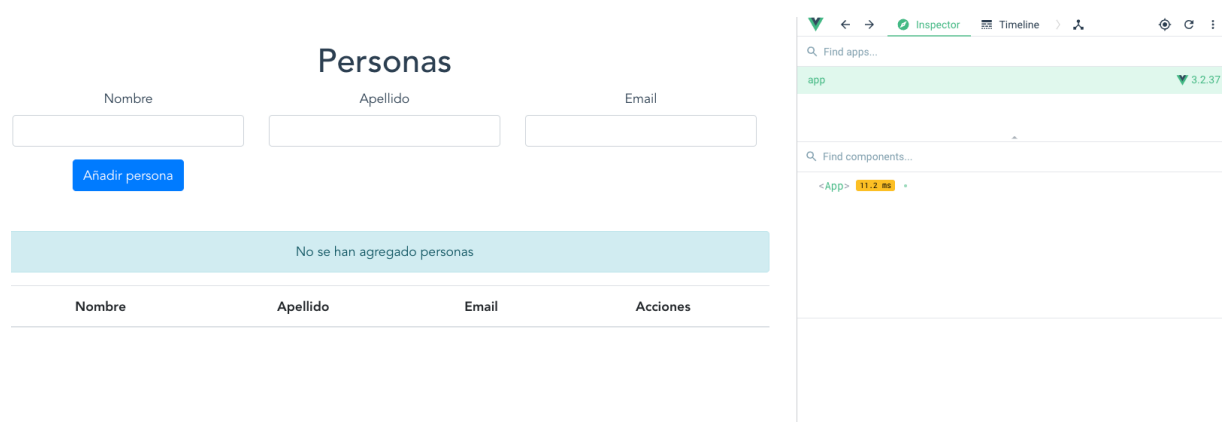
Nombre	Apellido	Email	Acciones
Sansa	Stark	daenerys@email.com	 Eliminar

9.4 Mostrar un mensaje informativo.

Para que la aplicación sea más usable, vamos a agregar un mensaje que se mostrará justo antes de la etiqueta **table** de apertura cuando ésta no contenga personas. En **tablaPersonas.vue** hacemos:

```
1 <template>
2   <div id="tabla-personas">
3     <div v-if="!personas.length" class="alert alert-info" role="alert">
4       No se han agregado personas
5     </div>
6     <table class="table">
```

Este es el mensaje que se mostrará cuando no queden usuarios:



10. Edita elementos con Vue.

Ahora que podemos eliminar elementos, tampoco estaría mal poder editarlos, que es lo que haremos en este apartado. Vamos a agregar la posibilidad de editar los elementos en la propia tabla, por lo que no necesitaremos componentes adicionales.

10.1 Agrega un botón de edición a la tabla.

Comenzaremos agregando un botón edición a la tabla del componente **TablaPersonas**, justo al lado del botón de borrado. En **TablaPersonas.vue**:

```

15     <tbody>
16       <tr v-for="persona in personas" :key="persona.id">
17         <td>{{ persona.nombre }}</td>
18         <td>{{ persona.apellido }}</td>
19         <td>{{ persona.email }}</td>
20         <td>
21           <button class="btn btn-danger" @click="$emit('delete-persona', persona.id)">
22              Eliminar
23           </button>
24           <button class="btn btn-info ml-2" @click="editarPersona(persona)">
25              Editar
26           </button>
27         </td>
28       </tr>
29     </tbody>

```

El botón ejecutará el método **editarPersona**, al que pasará la persona que seleccionada.

10.2 Agrega un método de edición a la tabla.

A continuación vamos a agregar el método **editarPersona** a nuestra lista de métodos, en el interior de la propiedad **methods** de nuestro componente **TablaPersonas.vue**:

```

35 export default {
36   name: 'tabla-personas',
37   props: {
38     personas: Array,
39   },
40   data() { },
41   methods: {
42     editarPersona(persona) {
43       this.personaEditada = Object.assign({}, persona);
44       this.editando = persona.id;
45     },
46   },
47 }

```

Lo que hemos hecho ha sido almacenar los datos originales de la persona que ese está editando actualmente en el objeto **personaEditada**, de forma que podemos recuperar los datos si se cancela la edición.

Además, hemos cambiado el valor de la variable de estado **editando**, que como no podría ser de otro modo, debemos agregar a nuestro componente. En **TablaPersonas.vue**:

```

35 export default {
36   name: 'tabla-personas',
37   props: {
38     personas: Array,
39   },
40   data() {
41     return {
42       editando: null,
43     },
44   },
45   methods: {
46     editarPersona(persona) {
47       this.personaEditada = Object.assign({}, persona);
48       this.editando = persona.id;
49     },
50   }
51 }

```

Vista previa:

Nombre

Apellido

Email

Añadir persona

Nombre	Apellido	Email	Acciones
Jon	Nieve	jon@email.com	<div>Eliminar</div> <div>Editar</div>
Tyion	Lannister	tyion@email.com	<div>Eliminar</div> <div>Editar</div>
Daenerys	Targaryen	daenerys@email.com	<div>Eliminar</div> <div>Editar</div>
Sansa	Stark	daenerys@email.com	<div>Eliminar</div> <div>Editar</div>

Inspector

Timeline

Find apps...

app

Find components...

<App>

13:4 85

<FormularioPersona>





<TablaPersonas>

10.3 Agrega campos de edición a la tabla.

Vamos a comprobar el valor de la variable **editando** en cada fila, mostrando campos **input** en lugar de los valores de cada persona en la fila que se esté editando:

```
components > ▼ TablaPersonas.vue > Vue Language Features (Volar) > { } template > div#tabla-personas > table.table
1  <template>
2    <div id="tabla-personas">
3      <div v-if="!personas.length" class="alert alert-info" role="alert">
4        No se han agregado personas
5      </div>
6      <table class="table">
7        <thead>
8          <tr>
9            <th>Nombre</th>
10           <th>Apellido</th>
11           <th>Email</th>
12           <th>Acciones</th>
13         </tr>
14       </thead>
15       <tbody>
16         <tr v-for="persona in personas" :key="persona.id">
17           <td v-if="editando === persona.id">
18             <input type="text" class="form-control" v-model="persona.nombre" />
19           </td>
20           <td v-else>
21             {{ persona.nombre }}
22           </td>
23           <td v-if="editando === persona.id">
24             <input type="text" class="form-control" v-model="persona.apellido" />
25           </td>
26           <td v-else>
27             {{ persona.apellido }}
28           </td>
29           <td v-if="editando === persona.id">
30             <input type="email" class="form-control" v-model="persona.email" />
31           </td>
32           <td v-else>
33             {{ persona.email }}
34           </td>
35           <td>
36             <button class="btn btn-info" @click="editarPersona(persona)">
37               Editar
38             </button>
39             <button class="btn btn-danger ml-2" @click="$emit('delete-persona', persona.id)">
40               Eliminar
41             </button>
42           </td>
43         </tr>
44       </tbody>
45     </table>
46   </div>
47 </template>
```





Ahora, si pruebas la aplicación y editas una fila, podrás ver que se muestran campos de edición en las filas de la persona que estás editando. Por ejemplo, Jon Nieve:

Nombre	Apellido	Email	Acciones	
<input type="text" value="Jon"/>	<input type="text" value="Nieve"/>	<input type="text" value="jon@email.com"/>	 Editar	 Eliminar
Tyrion	Lannister	tyrion@email.com	 Editar	 Eliminar

10.4 Agrega un botón de guardado a la tabla.





Sin embargo, todavía necesitamos un **botón de guardado** y otro que permita **cancelar el estado de edición**. Estos botones se mostrarán únicamente en la fila que se está editando. En **TablaPersonas.vue** después del último campo:

```

32 |         <td v-else>
33 |           {{ persona.email }}
34 |         </td>
35 |         <td v-if="editando === persona.id">
36 |           <button class="btn btn-success" @click="guardarPersona(persona)">
37 |              Guardar
38 |           </button>
39 |           <button class="btn btn-secondary ml-2" @click="cancelarEdicion(persona)">
40 |              Cancelar
41 |           </button>
42 |         </td>
43 |         <td v-else>
44 |           <button class="btn btn-info" @click="editarPersona(persona)">
45 |              Editar
46 |           </button>
47 |           <button class="btn btn-danger ml-2" @click="$emit('delete-persona', persona.id)">
48 |              Eliminar
49 |           </button>
50 |         </td>
51 |       </tr>

```

Como ves, hacemos referencia al método **guardarPersona**, que todavía tenemos que agregar. Por ahora, este sería el resultado cuando haces clic en el botón de edición de alguna fila:

Nombre	Apellido	Email	Acciones	
Jon	Nieve	jon@email.com	 Guardar	 Cancelar
Tyrion	Lannister	tyrion@email.com	 Editar	 Eliminar

10.5 Emite el evento de Guardado.

Ahora, vamos a agregar el método **guardarPersona** a la lista de métodos. Este método enviará un evento de guardado a la aplicación **App.vue**, que se encargará de actualizar los datos de la persona que hemos editado. En **TablePersonas.vue**:

```

60 |     methods: {
61 |       editarPersona(persona) {
62 |         this.personaEditada = Object.assign({}, persona);
63 |         this.editando = persona.id;
64 |       },
65 |       guardarPersona(persona) {
66 |         if (!persona.nombre.length || !persona.apellido.length || !persona.email.length) {
67 |           return;
68 |         }
69 |         this.$emit('actualizar-persona', persona.id, persona);
70 |         this.editando = null;
71 |       },

```

10.6 Agrega un método que cancele la edición.









Vamos a agregar también el método **cancelarEdicion** en **TablaPersonas.vue** que también hemos referenciado en el apartado anterior, que permite **cancelar** el estado de edición de una persona:

```









68   methods: {
69     editarPersona(persona) {
70       this.personaEditada = Object.assign({}, persona);
71       this.editando = persona.id;
72     },
73     cancelarEdicion(persona) {
74       Object.assign(persona, this.personaEditada);
75       this.editando = null;
76     }
77   }

```

Si le damos al botón Editar tenemos:

Nombre	Apellido	Email	Acciones	
<input type="text" value="Jon"/>	<input type="text" value="Nieve"/>	<input type="text" value="jon@email.com"/>	 Guardar	 Cancelar
Tyrion	Lannister	tyrion@email.com	 Editar	 Eliminar
Daenerys	Targaryen	daenerys@email.com	 Editar	 Eliminar
Sansa	Stark	daenerys@email.com	 Editar	 Eliminar

Si, ahora, le damos a Cancelar:

Nombre	Apellido	Email	Acciones	
Jon	Nieve	jon@email.com	 Editar	 Eliminar
Tyrion	Lannister	tyrion@email.com	 Editar	 Eliminar
Daenerys	Targaryen	daenerys@email.com	 Editar	 Eliminar
Sansa	Stark	daenerys@email.com	 Editar	 Eliminar

Tal y como ves, cuando cancelamos la edición de una persona, recuperamos su valor original, almacenado en el objeto **personaEditada**.

10.7 Recibe el evento de actualización en la aplicación.

Todavía tenemos que modificar el código de la aplicación **App.vue** para que reciba el evento de guardado y actualice los datos de la persona indicada. Primero agregaremos el evento **actualizar-persona**:

```

1  <template>
2    <div id="app" class="container">
3      <div class="row">
4        <div class="col-md-12">
5          <h1>Personas</h1>
6        </div>
7      </div>
8      <div class="row">
9        <div class="col-md-12">
10         <formulario-persona @add-persona="agregarPersona" />
11         <tabla-personas
12           :personas="personas"
13           @delete-persona="eliminarPersona"
14           @actualizar-persona="actualizarPersona" />
15         </div>
16       </div>
17     </div>
18   </template>

```

Ahora vamos a agregar el método **actualizarPersona** a la lista de métodos de la aplicación, justo después del método **eliminarPersona**. En la zona de código:

```

59   methods: {
60     actualizarPersona(id, personaActualizada) {
61       this.personas = this.personas.map(persona =>
62         persona.id === id ? personaActualizada : persona
63       )
64     },

```

En el método anterior hemos usado la estructura **map** para recorrer el array de personas, actualizando aquella que coincida con el **id** de la persona que queremos actualizar.

Personas

Nombre
Apellido
Email

Añadir persona

Nombre	Apellido	Email	Acciones
Jon	Nieve	jon@email.com	✎ Editar 🗑 Eliminar
Tyrion	Lannister	tyrion@email.com	✎ Editar 🗑 Eliminar
Daenerys	Targaryen	daenerys@email.com	✎ Editar 🗑 Eliminar