RETO DOS:

DEPARTAMENTO DE INFORMÁTICA
NIEVES RUIZ NOGUERAS

Curso 2025-2026

Capítulo 1

Vue.js

1.1. Introducción

Convertirse en un desarrollador de javascript es un camino de constante aprendizaje, en el cual después de conocer HTML, CSS y Javascript a un nivel medio-avanzado comienza a abrirse un amplio abanico de posibilidades con todas las herramientas que un desarrollador tiene a su alcance, principalmente el conjunto de **Frameworks y Librerías** que implemente en cada proyecto.

Una librería te permite solucionar un problema concreto, mientras que un framework te brinda un set de herramientas para desarrollar aplicaciones. Un framework, por lo general, contiene librerías, provee buenas prácticas y resulta toda una experiencia de desarrollo.

Una librería es como una herramienta (destornillador, martillo): tú decides cuándo y cómo usarla. Un framework es como una fábrica: tú trabajas dentro de su sistema; la fábrica te dice qué hacer y cuándo.

Las 40 mejores bibliotecas y frameworks de JavaScript https://kinsta.com/es/blog/bibliotecas-javascript/

Resumen comparativo

Aspecto	Librería	Framework
Control	Tú controlas el flujo	El framework controla el flujo
Flexibilidad	Alta (usas solo lo que necesites)	Menor (sigues su estructura)
Curva de aprendizaje	Menor	Mayor
Ejemplo en JS	React, jQuery, D3.js, Lodash	Angular, Vue, Next.js, NestJS
Uso ideal	Pequeños o medianos proyectos, integración libre	Aplicaciones grandes y estructuradas

Figura 1.1: Tabla comparativa generada por ChatGPT

Frontend (Interfaz de usuario)

Framework / Librería	Tipo	Dificultad	Rendimiento	Casos ideales
React	Librería	Media	Alta	Webs dinámicas, SPAs, apps móviles (con React Native)
Vue.js	Framework progresivo	Fácil	Alta	Proyectos medianos, startups, integración rápida
Angular	Framework completo	Alta	Alta	Apps empresariales grandes y estructuradas
Svelte	Compilador	Fácil	Muy alto	Webs ligeras, optimización extrema
SolidJS	Librería reactiva	Media	Muy alto	Apps interactivas con máximo rendimiento

Figura 1.2: Tabla comparativa generada por ChatGPT

Backend (Servidor con Node.js)

Framework	Dificultad	Rendimiento	Casos ideales
Express.js	Fácil	Alta	APIs REST simples o medianas
Fastify	Media	Muy alto	APIs de alto rendimiento
NestJS	Alta	Alta	Backends grandes y modulares (similar a Angular)
AdonisJS	Media	Alta	Aplicaciones full-stack con MVC
Hapi.js	Media	Alta	APIs empresariales seguras y robustas

Figura 1.3: Tabla comparativa generada por ChatGPT

Vue JS es un **framework progresivo** para desarrollar interfaces de usuario creado en el 2014 por Evan You que trabajó como desarrollador frontend en Google.

Vue fue desarrollado buscando obtener una herramienta que pudiera ser de fácil aprendizaje y se adaptara a las diferentes necesidades de proyectos simples y complejos.

¿ Qué es Vue.js y por qué es tan especial? https://www.youtube.com/watch?v=AqesL138vMA (14:57)

Historia de Vue https://es.wikipedia.org/wiki/Vue.js

Vue JS es uno de los Frameworks de mayor popularidad y presenta las siguientes caracterásticas:

- Es **accesible**: Es software Open Source y es posible acceder a él directamente desde https://es.vuejs.org/ en donde se puede encontrar la documentación oficial.
- Es **progresivo**: Vue JS puede ser implementado para proyectos muy básicos o para algo más complejo. Una de las grandes ventajas al ser un framework progresivo es su facilidad para adaptarse al crecimiento del proyecto.
- Es **reactivo**: Esto quiere decir que es posible desarrollar una aplicación que tenga una interacción constante con su entorno, de esta forma los cambios de estado interno se realizan por medio de eventos y generan diferentes reacciones cuando son accionados. Vue rastrea automáticamente los cambios de estado de JavaScript y actualiza eficientemente el DOM cuando ocurren cambios.
- **Representación declarativa**: Vue extiende el HTML estándar con una sintaxis de plantilla que nos permite describir declarativamente la salida HTML en función del estado de JavaScript.
- Utiliza **componentes**: Permite crear componentes y utilizarlos en diferentes secciones de la aplicación.

Un **componente web** (o Web Component) es una pieza reutilizable de interfaz, por ejemplo, un botón, una tarjeta, un formulario o un menú, que encapsula su propio HTML, CSS y JavaScript, funcionando como una unidad independiente dentro de una página web.

Algunas características de los componentes web son:

- **Reutilizabilidad**: Los componentes web se pueden utilizar en múltiples partes de una aplicación o en diferentes aplicaciones, lo que facilita la creación de interfaces coherentes y la reducción de la duplicación de código.
- **Independencia**: Los componentes web deben ser lo más independientes posible, lo que significa que deben funcionar correctamente sin depender demasiado del contexto que los rodea. Esto facilita su uso en diversas situaciones.
- **Encapsulación**: Los componentes web suelen encapsular tanto la presentación visual como la lógica asociada. Esto ayuda a mantener un código mas limpio y modular, ya que los detalles internos de un componente están ocultos al resto de la aplicación.
- **Interactividad**: Los componentes web pueden ser interactivos, permitiendo a los usuarios realizar acciones específicas dentro de la aplicación. Esto se logra mediante la incorporación de eventos y manejo de estados.
- **Personalización**: Los componentes web a menudo ofrecen opciones de personalización para adaptarse a diferentes necesidades y estilos de diseño.

Los marcos de desarrollo web modernos (React, Angular y Vue.js), se centran en la construcción de aplicaciones mediante la creación y utilización de componentes web.

1.2. Componentes web

Figura 1.4: Componentes Web

https://developer.mozilla.org/en-US/docs/Web/Web_Components

https://kinsta.com/es/blog/componentes-web/

Los componentes web son un paquete de diferentes tecnologías que permiten crear **elementos personalizados** reutilizables, con su funcionalidad encapsulada aparte del resto del código, y utilizarlos en las aplicaciones web.

Los componentes web son un conjunto de estándares para crear elementos HTML personalizados con sus propiedades y métodos, DOM encapsulado y estilos.

Una vez que se define un elemento personalizado, podemos usarlo con los elementos HTML existentes.

Los componentes web surgieron como propuesta por parte de Google a la W3C. Los procedimientos del W3C para permitir la evolución de la web son un poco lentos. Los desarrolladores detectan necesidades mucho antes de que la W3C realice un estándar para poder cubrirlas. De hecho, pueden pasar años desde que algo comienza a ser usado en el mundo de la web hasta que se presenta el estándar. Es por ello que han decidido no crear nuevas etiquetas HTML sino que han generado unos estándares para que los desarrollares creen sus propias etiquetas.

Con los componentes web se puede hacer casi cualquier cosa que pueda ser hecha con HTML, CSS y JavaScript, y puede ser un componente portable reutilizado fácilmente.

Un componente tiene:

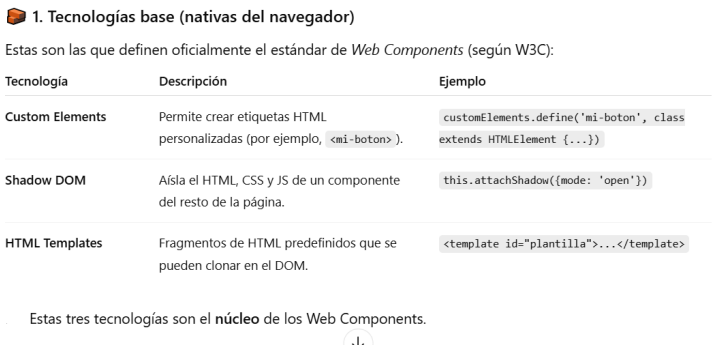
W3C World Wide Web Consortium es un comité que se dedica a implementar tecnologías uniformes en el uso y desarrollo de Internet.

- Su propia clase de JavaScript.
- La estructura DOM, gestionada únicamente por su clase, el código externo no accede a ella (principio de encapsulación).
- Estilos CSS, aplicados al componente.
- API: eventos, métodos de clase, etc... para interactuar con otros componentes.

Los repositorios de componentes web pre-construidos están disponibles para que todos podamos echar un vistazo aunque también podemos crearlos.

- https://www.webcomponents.org/
- https://component.gallery/
- https://genericcomponents.netlify.app/
- https://github.com/mdn/web-components-examples
- https://github.com/davatron5000/awesome-standalones
- https://github.com/scottaohara/accessible_components
- https://kickstand-ui.com/

Los componentes web constan de diferentes tecnologías o estándares que pueden ser usados conjuntamente o por separado.



1. Tecnologías base (nativas del navegador)

Estas son las que definen oficialmente el estándar de *Web Components* (según W3C):

Tecnología	Descripción	Ejemplo
Custom Elements	Permite crear etiquetas HTML personalizadas (por ejemplo, `<mi-boton>`).	`customElements.define('mi-boton', class extends HTMLElement {...})`
Shadow DOM	Aísla el HTML, CSS y JS de un componente del resto de la página.	`this.attachShadow({mode: 'open'})`
HTML Templates	Fragmentos de HTML predefinidos que se pueden clonar en el DOM.	`<template id="plantilla">...</template>`

Estas tres tecnologías son el **núcleo** de los Web Components.

Figura 1.5: Tecnologías

1.2.1. Custom elements

Para crear un elemento personalizado, debemos decirle al navegador cómo mostrarlo, qué hacer cuando el elemento se agrega o elimina de la página, etc... Todo esto se hace creando una clase en javascript que hereda de HTMLElement con métodos especiales. La mayoría de estos métodos son opcionales.

```
1
2  class MiElemento extends HTMLElement {
3
4    constructor() {
5      super();
6
7      // Crea el elemento. Se ejecuta cuando el componente se inicializa
           por primera vez.
8
9      // Debe llamar a super() y puede establecer cualquier valor por
           defecto.
10   }
11
12   connectedCallback() {
13
14     // Método que se ejecuta cuando el elemento es añadido a un documento.
15   }
16
17   disconnectedCallback() {
18
19     // Método que se ejecuta cuando el elemento es borrado en un
           documento.
20   }
21
```

```
22    static get observedAttributes() {
23
24      return [/* array con el nombre de los atributos. */];
25
26      // Se le dice al navegador que cuando cambie uno de estos atributos
            nos avise para que se ejecute el método attributeChangeCallback.
27    }
28
29    attributeChangedCallback(name, oldValue, newValue) {
30
31      // Se ejecuta cuando se modifican los atributos que observamos con el
            método anterior. Los parámetros son nombre del atributo, valor
            viejo y nuevo valor.
32    }
33
34    adoptedCallback() {
35
36      // Se ejecuta cuando un elemento es movido a un nuevo documento
37    }
38
39    // Los métodos que nosotros decidamos.
40  }
```

Después de eso, necesitamos registrar el elemento:

```
1
2  // Nombre de la etiqueta y nombre de la clase
3
4  window.customElements.define("mi-elemento", MiElemento);
```

El **nombre de la etiqueta** para que sea válido debe seguir unas reglas muy concretas. Debe empezar por una letra entre la a y la z en minúsculas, debe contener un guión - y no puede utilizar letras mayúsculas. No hay ni habrá etiquetas HTML que contengan un guión, así que cuando veamos una etiqueta con un guión es un componente.

Ejemplos:

Hola mundo

```
1   Contenido del fichero index.html
2
3   <!DOCTYPE html>
4   <html lang="en">
5   <head>
6       <meta charset="UTF-8">
7       <title>Title</title>
8       <link rel=stylesheet type="text/css" href="estilo.css">
9   </head>
10  <body>
11  <hola-mundo></hola-mundo>
12  <script src="funciones.js"></script>
13  </body>
14  </html>
```

```
Contenido del fichero funciones.js

Aquí está la declaración del componente web.

class HolaMundo extends HTMLElement {
    constructor(){
        super();
        console.log("Se ha creado la etiqueta hola-mundo");
    }

    connectedCallback() {
        this.innerHTML = "<p>Hola, mundo</p>";
        console.log("Se ha utilizado la etiqueta hola-mundo");
    }

}

customElements.define( 'hola-mundo', HolaMundo );
```

```
Contenido del fichero css

body {
    font-family: sans-serif;
  }

  hola-mundo {
    font-weight: bold;
    color: red;
  }
```

Hola y el nombre de una persona.

```

Contenido del fichero index.html

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link rel=stylesheet type="text/css" href="estilo.css">

</head>
<body>
    <script src="funciones.js"></script>
    <hola-nombre id="nombre" nombre="Pepe"></hola-nombre>
    <hola-nombre></hola-nombre>
</body>
</html>
```

```

let vnombre;

class HolaNombre extends HTMLElement {

    constructor(){
        super();
        console.log("Componente construido");
        // Atributo nombre
        this.nombre;
        // Atributo de tipo función.
        this.fSaludar = this.fSaludar.bind(this);
    }

    // Esta función es llamada cuando el componente se usa en el
        documento html
    connectedCallback() {
        console.log("Se usa la etiqueta");
        // Usamos el valor del atributo si está. Si no se lo pedimos con
            un prompt
        vnombre = this.getAttribute("nombre") || prompt("Dime tu nombre");
        this.innerHTML = `Hola variable ${ vnombre }!`;
        this.innerHTML = `Hola atributo ${  this.nombre }!`;
        // Llamamos a la función cuando hagan click
        this.addEventListener('click',this.fSaludar);
    }

    // Estamos pendientes a los cambios en el atributo name
  static get observedAttributes() {
    return ['nombre'];
  }

    // Se ejecuta cuando cambiar el valor del atributo name
  attributeChangedCallback(property, oldValue, newValue) {
    console.log("El atributo " + property +" tenía el valor " +  oldValue
        + " y ahora tiene " + newValue)
    this.nombre = newValue;

  }
```

```
37
38    fSaludar()
39    {
40      // Código de la función saludar
41      alert("Hola atributo " + this.nombre);
42      alert("Hola variable " + vnombre);
43    }
44
45  }
46
47  customElements.define( 'hola-nombre', HolaNombre );
```

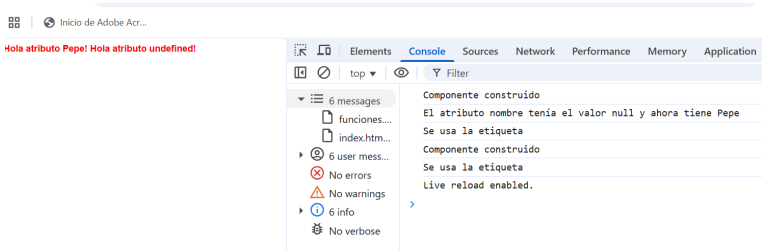


Figura 1.6: Orden de ejecución

1.2.2. Shadow DOM

El Shadow DOM (DOM en la sombra) es una característica del navegador que permite a un elemento HTML tener su propio árbol DOM interno, completamente aislado del DOM principal de la página. Es como si el elemento tuviera un mini-documento HTML privado dentro de sí.

Ejemplo:

```
1
2   <!DOCTYPE html>
3   <html lang="es">
4   <head>
5     <meta charset="UTF-8">
6     <title>Comparativa Shadow DOM</title>
7     <style>
8       button {
9         background: green;
10        color: white;
11        border: none;
12        padding: 10px 20px;
13        border-radius: 8px;
14        font-size: 1em;
15        cursor: pointer;
16        margin: 10px;
17      }
18    </style>
```

```
</head>
<body>
  <h2>Comparación: sin Shadow DOM vs con Shadow DOM</h2>

  <boton-normal></boton-normal>
  <boton-shadow></boton-shadow>

  <script>
    // 1. Componente SIN Shadow DOM. Se le aplicará el estilo definido.
    // Verde
    class BotonNormal extends HTMLElement {
      connectedCallback() {
        this.innerHTML = `<button>Botón SIN Shadow DOM</button>`;
      }
    }
    customElements.define("boton-normal", BotonNormal);

    // 2. Componente CON Shadow DOM
    // Da igual el estilo definido. Este será rojo
    class BotonShadow extends HTMLElement {
      constructor() {
        super();
        // Creamos el shadow root
        const shadow = this.attachShadow({ mode: "open" });

        // Estilos encapsulados dentro del shadow
        shadow.innerHTML = `
          <style>
            button {
              background: red;
              color: white;
              border: none;
              padding: 10px 20px;
              border-radius: 8px;
              font-size: 1em;
              cursor: pointer;
            }
            button:hover {
              background: darkred;
            }
          </style>
          <button>Botón CON Shadow DOM</button>
        `;
      }
    }
    customElements.define("boton-shadow", BotonShadow);
  </script>
</body>
</html>
```

1.2.3. HTML Templates

Un HTML Template es una plantilla de código HTML predefinida que el navegador no renderiza automáticamente, sino que espera a que tú la clones o actives desde JavaScript.

Se define con la etiqueta template:

```
1
2  <template id="miPlantilla">
3    <!-- HTML oculto -->
4  </template>
```

Todo lo que está dentro de la etiqueta template no se muestra en la página hasta que lo clones, se puede usar para crear contenido repetitivo o dinámico.

Son plantillas reutilizables. Se define una vez la estructura y se puede clonar muchas veces. Separan estructura y lógica. El HTML está en la etiqueta template y la lógica en el JS.

Ejemplo sin template (html en el código):

```
1
2  <!DOCTYPE html>
3  <html lang="es">
4  <head>
5    <meta charset="UTF-8" />
6    <title>Ejemplo sin Template</title>
7  </head>
8  <body>
9    <mi-carta></mi-carta>
10
11   <script>
12     class MiCarta extends HTMLElement {
13       constructor() {
14         super();
15         // Creamos Shadow DOM para aislar estilos y contenido
16         this.attachShadow({ mode: "open" });
17       }
18
19       connectedCallback() {
20         // Definimos el contenido directamente desde JS
21         this.shadowRoot.innerHTML = `
22           <style>
23             .carta {
24               border: 2px solid #333;
25               padding: 10px;
26               border-radius: 6px;
27               background-color: #f2f2f2;
28               width: 200px;
29               text-align: center;
30               font-family: Arial, sans-serif;
31             }
32             button {
```

```
              margin-top: 8px;
            }
          </style>
          <div class="carta">
            <h3>Hola sin template</h3>
            <p>Este contenido se creó con <code>innerHTML</code>.</p>
            <button>Haz clic</button>
          </div>
        `;

        // Añadimos un comportamiento
        const boton = this.shadowRoot.querySelector("button");
        boton.addEventListener("click", () => alert("Clic en la carta sin
            template"));
      }
    }

    customElements.define("mi-carta", MiCarta);
  </script>
</body>
</html>
```

Ejemplo con template (html en la plantilla y el código en js):

```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <title>Ejemplo con Template</title>
</head>
<body>
  <!-- Plantilla reutilizable -->
  <template id="plantilla-carta">
    <style>
      .carta {
        border: 2px solid #333;
        padding: 10px;
        border-radius: 6px;
        background-color: #e3f2fd;
        width: 200px;
        text-align: center;
        font-family: Arial, sans-serif;
      }
      button {
        margin-top: 8px;
      }
    </style>
    <div class="carta">
      <h3>Hola con template</h3>
      <p>Este contenido viene de un &lt;template&gt;.</p>
      <button>Haz clic</button>
    </div>
  </template>

  <!-- Usamos el componente -->
  <mi-carta-template></mi-carta-template>

  <script>
```

```
36      class MiCartaTemplate extends HTMLElement {
37        constructor() {
38          super();
39          this.attachShadow({ mode: "open" });
40        }
41
42        connectedCallback() {
43          // Buscamos el template en el documento
44          const template = document.getElementById("plantilla-carta");
45          // Clonamos su contenido
46          const contenido = template.content.cloneNode(true);
47          // Lo añadimos al Shadow DOM
48          this.shadowRoot.appendChild(contenido);
49
50          // Añadimos comportamiento JS (igual que antes)
51          const boton = this.shadowRoot.querySelector("button");
52          boton.addEventListener("click", () => alert("Clic en la carta con
                template"));
53        }
54      }
55
56      customElements.define("mi-carta-template", MiCartaTemplate);
57    </script>
58  </body>
59  </html>
```

1.3. Empezando a trabajar con Vuejs

VueJs se puede usar de maneras diferentes:

1. **Archivo CDN incluido directamente.**

```
1
2  <!DOCTYPE html>
3  <html lang="en">
4  <head>
5    <meta charset="UTF-8">
6    <meta name="viewport" content="width=device-width,
         initial-scale=1.0">
7    <title>Hola Mundo con Vue 3</title>
8    <!-- Incluye Vue.js 3 desde CDN -->
9    <script src="https://cdn.jsdelivr.net/npm/vue@3"></script>
10 </head>
11 <body>
12
13   <!-- Contenedor de Vue -->
14   <div id="app">
15     <!-- Mostrar el mensaje de Vue -->
16     {{ mensaje }}
17   </div>
18
19   <!-- Tu script de Vue -->
20   <script>
21     // Instancia de Vue
22     const app = Vue.createApp({
23       data() {
24         return {
25           mensaje: 'Hola Mundo con Vue 3!'
26         };
27       }
28     });
29
30     // Montar la aplicación en un elemento existente (por ID)
31     app.mount('#app');
32   </script>
33
34 </body>
35 </html>
```

2. **Creando un proyecto Vue**

   Requisitos para crear un proyecto Vue 3:

   - Visual Studio Code.
   - Node.js en versión 16 o superior.

```
PS D:\Curso23-24\DWEC\CursoVue3\DominandoVue3-main\DominandoVue3-main> node -v
v20.9.0
PS D:\Curso23-24\DWEC\CursoVue3\DominandoVue3-main\DominandoVue3-main>
```

Figura 1.7: Versión de node

   - Las extensiones Vue Languaje Features y Vue Vscode Snippets.

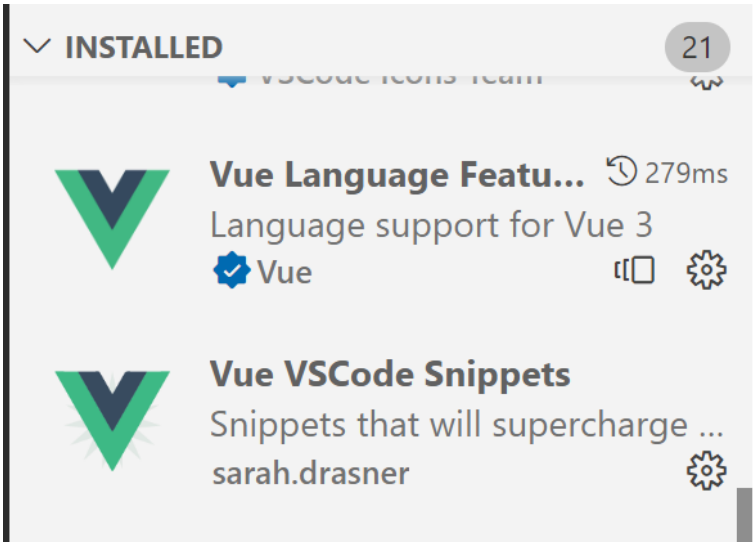


Figura 1.8: Extensiones de Vue añadidas a VSC

   Desde el terminal podemos ejecutar el comando npm init para crear un proyecto Vuejs

```
1
2  npm init vue@latest
```

```
1
2  PS C:\Users\nieves\Desktop\EjemplosVue> npm init vue@latest
3  ?  Vue.js - The Progressive JavaScript Framework
4  ?
5  ?  Project name (target directory):
```

```
?  EjemploUno
?
?  Package name:
?  ejemplouno
?
?  Select features to include in your project: (?/? to navigate,
space to select, a to toggle all, enter to confirm)
?  none
?
?  Select experimental features to include in your project: (?/? to
navigate, space to select, a to toggle all, enter to confirm)
?  none
?
?  Skip all example code and start with a blank Vue project?
?  No

Scaffolding project in
    C:\Users\nieves\Desktop\EjemplosVue\EjemploUno...
?
?  Done. Now run:

   cd EjemploUno
   npm install
   npm run dev

| Optional: Initialize Git in your project directory with:

   git init && git add -A && git commit -m "initial commit"

PS C:\Users\nieves\Desktop\EjemplosVue> cd EjemploUno
PS C:\Users\nieves\Desktop\EjemplosVue\EjemploUno> npm install
npm WARN tar TAR_BAD_ARCHIVE: Unrecognized archive format
npm WARN tarball cached data for
    @vue/babel-plugin-resolve-type@https://registry.npmjs.org/@vue/babel-plugin-resolve-type/-
    (sha512-Wm/60o+53JwJODm4Knz47dxJnLDJ9FnKnGZJbUUf8nQRAtt6P+undLUAVU3Ha33LxOJe6IPoifRQ6F/0Rr
    seems to be corrupted. Refreshing cache.
npm WARN tar TAR_BAD_ARCHIVE: Unrecognized archive format
npm WARN tarball cached data for
    @babel/plugin-syntax-typescript@https://registry.npmjs.org/@babel/plugin-syntax-typescript
    (sha512-xfYCBMxveHrRMnAWl1ZlPXOZjzkN82THFvLhQhFXFt81Z5HnN+EtUkZhv/zcKpmT3fzmWZB0ywiBrbC3vo
    seems to be corrupted. Refreshing cache.
npm WARN tar TAR_BAD_ARCHIVE: Unrecognized archive format
npm WARN tarball cached data for
    @babel/helper-optimise-call-expression@https://registry.npmjs.org/@babel/helper-optimise-c
    (sha512-URMGH08NzYFhubNSGJrpUEphGKQwMQYBySzat5cAByY1/YgIRkULnIy3tAMeszlL/so2HbeilYloUmSpd7
    seems to be corrupted. Refreshing cache.
npm WARN tar TAR_BAD_ARCHIVE: Unrecognized archive format
npm WARN tarball cached data for
    @babel/helper-annotate-as-pure@https://registry.npmjs.org/@babel/helper-annotate-as-pure/-
    (sha512-fXSwMQqitTGeHLBC08Eq5yXz2m37E4pJX1qAU1+2cNedz/ifv/bVXft90VeSav5nFO61EcNgwr0aJxbyPa
    seems to be corrupted. Refreshing cache.

added 123 packages, and audited 124 packages in 12s

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

found 0 vulnerabilities
PS C:\Users\nieves\Desktop\EjemplosVue\EjemploUno> npm run dev
```

```
52
53  > ejemplouno@0.0.0 dev
54  > vite
55
56
57    VITE v7.2.2  ready in 2015 ms
58
59    ?  Local:   http://localhost:5173/
60    ?  Network: use --host to expose
61    ?  Vue DevTools: Open http://localhost:5173/__devtools__/ as a
          separate window
62    ?  Vue DevTools: Press Alt(?)+Shift(?)+D in App to toggle the Vue
          DevTools
63    ?  press h + enter to show help
```

En primer lugar nos pide el **nombre** del proyecto (primero en la imagen).

Después nos ha preguntando si queremos añadir una serie de elementos y no hemos seleccionado ninguno.

```
|
◆  Select features to include in your project: (↑/↓ to navigate,
space to select, a to toggle all, enter to confirm)
|  ☐ TypeScript
|  ☐ JSX Support
|  ☐ Router (SPA development)
|  ☐ Pinia (state management)
|  ☐ Vitest (unit testing)
|  ☐ End-to-End Testing
|  ☐ ESLint (error prevention)
|  ☐ Prettier (code formatting)
└
```

TypeScript es un lenguaje de programación de código abierto desarrollado y mantenido por Microsoft. Es un superconjunto de JavaScript, lo que significa que cualquier código JavaScript válido también es código TypeScript válido. Sin embargo, TypeScript agrega características adicionales a JavaScript como la definición de tipos de datos para variables, parámetros de funciones y valores de retorno. Esto ayuda a detectar errores de tipo durante el desarrollo en lugar de en tiempo de ejecución.

```
1
2  // Dato de tipo string
3  let name: string = 'Tu nombre';
4  name = 'Otro nombre'; // Es correcto
5  name = 2 // Es incorrecto
6
7  // Dato de tipo number
8  let age: number = 29;
9  age = 0xf00d; // Es hexadecimal y es correcto
10 age = '3'; // Es un string e incorrecto
```

JSX (JavaScript XML) es una extensión de sintaxis que permite escribir código HTML dentro de JavaScript de una manera concreta. Nosotros vamos a usar Templates.

Vue Router es una biblioteca para el enrutamiento en aplicaciones de una sola página (SPA) con Vue.js. Las aplicaciones de una sola página son aquellas en las que la navegación entre diferentes secciones o vistas de la aplicación ocurre sin la necesidad de recargar la página completa. En lugar de cargar una nueva página, la aplicación actualiza dinámicamente la interfaz de usuario para mostrar la vista deseada. Lo usaremos más adelante.

Pinia es una biblioteca de estado para Vue.js que proporciona una gestión de estado simple. El estado se refiere a los datos y la información que define el estado actual de la aplicación en un momento dado. **Vitest y end-to-end** tienen que ver con el testeo.

ESLint es una herramienta de linting (herramienta que realiza un análisis estático del código fuente para encontrar patrones problemáticos o código que no cumple con ciertas convenciones y estaandares) para JavaScript que ayuda a identificar y corregir problemas en el código fuente. En Vue, ESLint es utilizado para mantener un código consistente y de alta calidad.

Prettier es una herramienta de formateo de código que se utiliza para mantener la consistencia en la apariencia del código fuente en un proyecto. A diferencia de las herramientas de linting Prettier se enfoca en formatear automáticamente el código según un conjunto de reglas predefinidas.

Una vez creado el proyecto, nos indican que es lo que tenemos que hacer para ejecutarlo:

```
1
2  cd primero // Colocarlos en el directorio que lleva el nombre del
       proyecto.
3
4  npm install
5
6  npm run dev
```

npm install es un comando que se utiliza en Node.js para instalar las dependencias de un proyecto en el archivo package.json.

Sobre npm hablaremos en FAFS

npm run dev inicia un servidor de desarrollo. Este comando está configurado a través de scripts definidos en el archivo package.json.

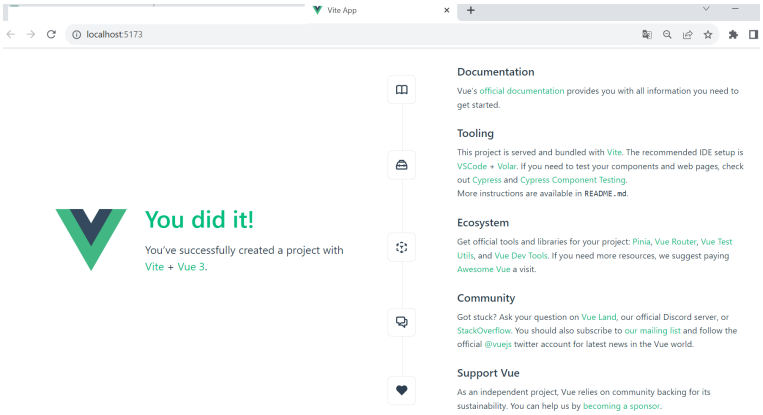


Figura 1.9: Ejecución del primer proyecto.

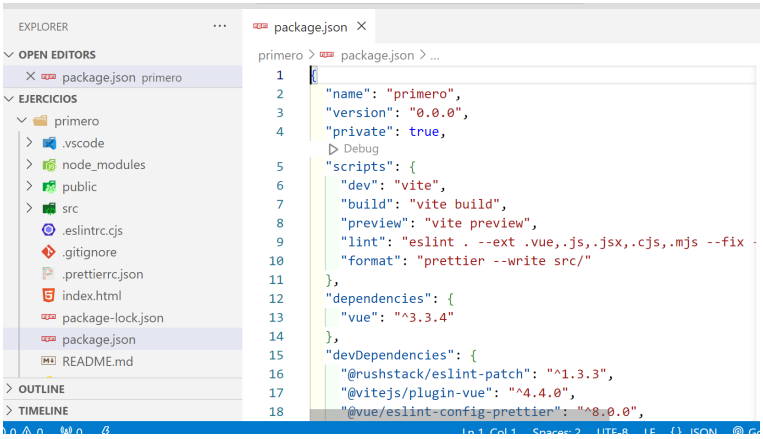


Figura 1.10: Estructura del primer proyecto.

https://lenguajejs.com/vuejs/introduccion/estructura-carpetas/

Los principales elementos de la estructura del proyecto creado en vue son:

1. La carpeta **.vscode** contiene configuraciones específicas de Visual Studio Code para el proyecto. Estas configuraciones son específicas de la herramienta de desarrollo y no son esenciales para la ejecución del proyecto en sí.
2. La **carpeta node-modules** contiene las dependencias del proyecto. Cuando se instala un paquete o módulos Node.js utilizando npm, estas dependencias se descargan y se almacenan en esta carpeta.
3. La **carpeta public** contiene archivos estáticos que deben ser accesibles directamente por el navegador. En nuestro caso solo hay un fichero denominado favicon.ico que es el icono que aparece al lado del nombre de la pestaña en el navegador, por defecto es el logo de Vue.

4. La **carpeta src** contiene el código fuente de la aplicación. Aquí es donde se encuentran los archivos que componen los componentes, estilos, scripts y otros recursos que conforman la lógica y la interfaz de usuario de una aplicación.

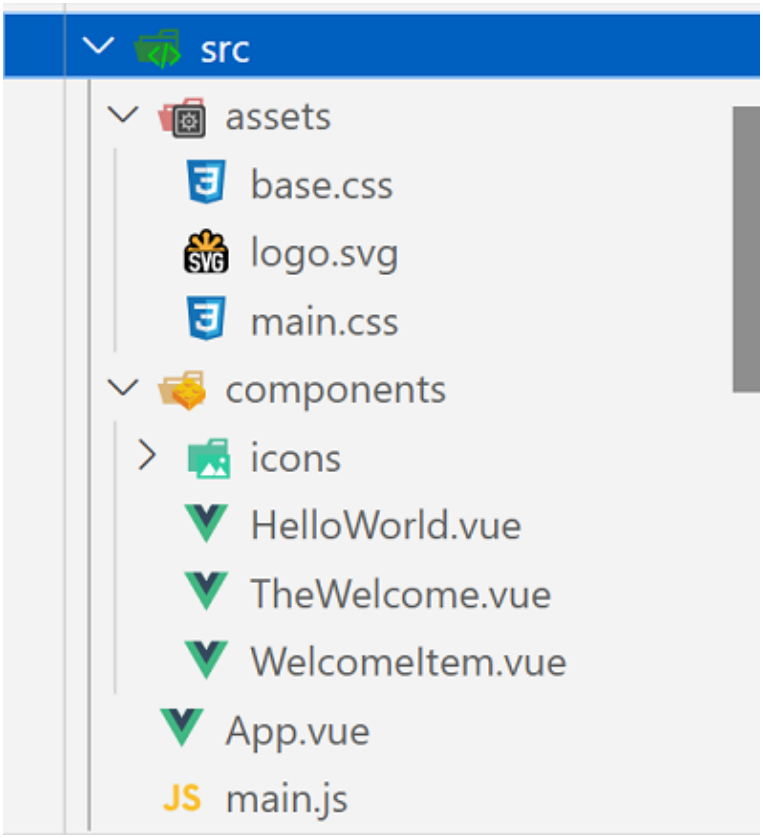


Figura 1.11: Contenido de la carpeta src.

assets: Aquí se pueden almacenar recursos estáticos como imágenes, archivos de estilo (CSS, SCSS) y otros recursos multimedia. Los archivos que tenemos en esta carpeta se utilizan en nuestro código, importándolos, y muchas veces no queremos que se pueda acceder directamente a ellos mediante una URL concreta como los de public.

components: Este directorio contiene los componentes que se van a utilizar en la aplicación.

```
1
2  Componente HelloWorld.vue
3
```

```
<script setup>
defineProps({
  msg: {
    type: String,
    required: true
  }
})
</script>

<template>
  <div class="greetings">
    <h1 class="green">{{ msg }}</h1>
    <h3>
      You?ve successfully created a project with
      <a href="https://vitejs.dev/" target="_blank"
          rel="noopener">Vite</a> +
      <a href="https://vuejs.org/" target="_blank"
          rel="noopener">Vue 3</a>.
    </h3>
  </div>
</template>

<style scoped>
h1 {
  font-weight: 500;
  font-size: 2.6rem;
  position: relative;
  top: -10px;
}

h3 {
  font-size: 1.2rem;
}

.greetings h1,
.greetings h3 {
  text-align: center;
}

@media (min-width: 1024px) {
  .greetings h1,
  .greetings h3 {
    text-align: left;
  }
}
</style>
```

main.js: Este archivo es la entrada principal de la aplicación. Aquí se inicializa y configura la instancia de Vue, y puede contener la lógica principal de inicialización. Se trata del fichero principal que arranca el proyecto Vue y que se cargará desde la plantilla index.html que se incluye en el raíz del proyecto.

```

import './assets/main.css'

import { createApp } from 'vue'
```

```
5  import App from './App.vue'
6
7  createApp(App).mount('#app')
```

App.vue: Este archivo es el componente raíz de una aplicación Vue.js. Contiene la estructura básica de la aplicación y puede incluir el diseño general y estilos.

```
1
2  <script setup>
3  import HelloWorld from './components/HelloWorld.vue'
4  import TheWelcome from './components/TheWelcome.vue'
5  </script>
6
7  <template>
8    <header>
9      <img alt="Vue logo" class="logo" src="./assets/logo.svg"
           width="125" height="125" />
10
11     <div class="wrapper">
12       <HelloWorld msg="You did it!" />
13     </div>
14   </header>
15
16   <main>
17     <TheWelcome />
18   </main>
19 </template>
20
21 <style scoped>
22 header {
23   line-height: 1.5;
24 }
25
26 .logo {
27   display: block;
28   margin: 0 auto 2rem;
29 }
30
31 @media (min-width: 1024px) {
32   header {
33     display: flex;
34     place-items: center;
35     padding-right: calc(var(--section-gap) / 2);
36   }
37
38   .logo {
39     margin: 0 2rem 0 0;
40   }
41
42   header .wrapper {
43     display: flex;
44     place-items: flex-start;
45     flex-wrap: wrap;
46   }
47 }
48 </style>
```

5. **Index.html**

```
1
2  <!DOCTYPE html>
3  <html lang="en">
4    <head>
5      <meta charset="UTF-8">
6      <link rel="icon" href="/favicon.ico">
7      <meta name="viewport" content="width=device-width,
           initial-scale=1.0">
8      <title>Vite App</title>
9    </head>
10   <body>
11     <div id="app"></div>
12     <script type="module" src="/src/main.js"></script>
13   </body>
14 </html>
```

6. **package.json y package-lock.json**: Ficheros donde se establecen las versiones de la librerías requeridas. Además se pueden establecer otros parámetros como el nombre de la aplicación, versión, autores, etc.

```
1
2  {
3    "name": "primero",
4    "version": "0.0.0",
5    "private": true,
6    "scripts": {
7      "dev": "vite",
8      "build": "vite build",
9      "preview": "vite preview",
10     "lint": "eslint . --ext .vue,.js,.jsx,.cjs,.mjs --fix
           --ignore-path .gitignore",
11     "format": "prettier --write src/"
12   },
13   "dependencies": {
14     "vue": "^3.3.4"
15   },
16   "devDependencies": {
17     "@rushstack/eslint-patch": "^1.3.3",
18     "@vitejs/plugin-vue": "^4.4.0",
19     "@vue/eslint-config-prettier": "^8.0.0",
20     "eslint": "^8.49.0",
21     "eslint-plugin-vue": "^9.17.0",
22     "prettier": "^3.0.3",
23     "vite": "^4.4.11"
24   }
25 }
```

7. **Otros archivos**

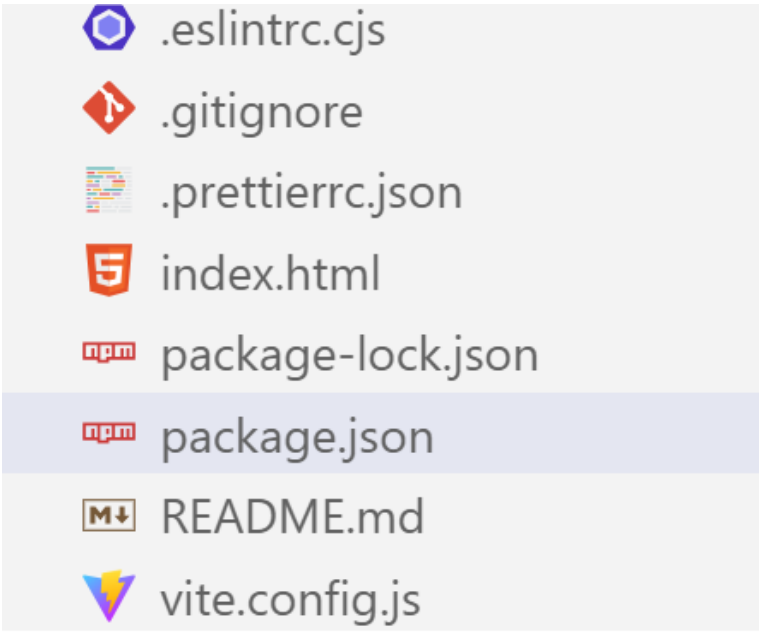


Figura 1.12: Otros ficheros creados por vue

1.4. Componentes en Vue 3

La **función setup** es el punto de entrada a la COMPOSITION API. La API de Composición es la manera de construir componentes en Vue.

La estructura básica de un componente es:

```
<script setup>

  Código js

</script>

<template>

  HTML

</template>

<style scoped>

  CSS

</style>
```

```
Contenido del fichero App.vue (Segundo ejemplo)

<script setup>
  const nombre = 'Nieves';
</script>

<template>
  <h1> Hola {{ nombre }}</h1>
</template>

<style scoped>
  h1{
    font-family: Arial;
  }
</style>
```

1.4.1. Métodos y propiedades computadas

En varias ocasiones se ha dicho que Vue es un framework reactivo, es decir, que los cambios se ven reflejados inmediatamente en el DOM. Esto se consigue a través de los **métodos y propiedades computadas.**

ref es una función que envuelve el valor inicial de la propiedad y devuelve un objeto con propiedad value. ref es una característica de Vue que se utiliza

para conseguir la reactividad. Para poder usarla hay que importarla. ref se puede utilizar con tipos de datos primitivos y con objetos.

```
1
2  Ejemplo tercero:
3
4  En el contexto de Vue.js 3 y su nueva sintaxis <script setup>, se utiliza
       const para declarar variables reactivas que no van a ser reasignadas,
       mientras que ref se encarga de proporcionar reactividad.
5
6  Cuando se usa const, no se puede reasignar a la variable a un nuevo
       valor, pero eso no significa que el valor contenido dentro del objeto
       ref (como en servicio1 y servicio2) no pueda cambiar. La reactividad
       en Vue.js se gestiona mediante el uso de métodos proporcionados por
       ref, reactive y otros.
7
8  <script setup>
9  // La función ref hay que importarla.
10 import {ref} from "vue";
11
12 const taller = "Taller Arriaga";
13 // ref envuelve el string en un objeto cuya principal propiedad es value.
14 const servicio1 = ref('Basico');
15 const servicio2 = ref("Especial");
16
17 // Modifico el valor de servicio dos.
18 servicio2.value = "Premiun";
19 </script>
20
21 <template>
22  <h1>{{ taller }}</h1>
23  <h3>Servicios disponibles</h3>
24  <ul>
25   <li>{{ servicio1 }}</li>
26   <li>{{ servicio2 }}</li>
27  </ul>
28 </template>
29
30 <style scoped>
31 h1,h3,li{
32   font-family: Arial;
33 }
34 section{
35   display:flex;
36 }
37 </style>
```

reactive es otra función para gestionar la reactividad. Se puede utilizar con objetos y también hay que importarla.

```
1
2  Ejemplo tercero.
3
4  <script setup>
5  // La función ref hay que importarla. También reactive
6  import {ref, reactive} from "vue";
7
8  const taller = "Taller Arriaga";
9  // ref envuelve el string en un objeto cuya principal propiedad es value.
10 const servicio1 = ref('Basico');
11 const servicio2 = ref("Especial");
12
13 // Modifico el valor de servicio dos.
14 servicio2.value = "Premiun";
15
16 // Objeto, no un tipo de dato primitivo.
17 const respuesto1 = reactive({
18   id:1,
19   nombre: 'Filtro de aceite',
20   precio: 100,
21   disponibilidad:true
22 });
23
24 // Modifico uno de los campos. Ya no value.
25 respuesto1.disponibilidad = false;
26 </script>
27
28 <template>
29  <h1>{{ taller }}</h1>
30  <h3>Servicios disponibles</h3>
31  <ul>
32   <li>{{ servicio1 }}</li>
33   <li>{{ servicio2 }}</li>
34  </ul>
35  <h3>Repuestos</h3>
36  <p>{{ respuesto1.nombre }} ... {{ respuesto1.disponibilidad }}</p>
37 </template>
38
39 <style scoped>
40 h1,h3,li,p{
41   font-family: Arial;
42 }
43 section{
44   display:flex;
45 }
46 </style>
```

Las **propiedades computadas** nos permiten transformar los datos o realizar cálculos sobre los mismos y rápidamente actualizar el resultado en el template.

```
1
2  <script setup>
3
4  import {ref, reactive, computed} from "vue";
5
6  const taller = "Taller Arriaga";
7
8  // ref envuelve el string en un objeto cuya principal propiedad es value.
9  const servicio1 = ref('Basico');
10 const servicio2 = ref("Especial");
11
12 // Modifico el valor de servicio dos.
13 servicio2.value = "Premiun";
14
15 // reactive
16 // Objeto, no un tipo de dato primitivo.
17 const repuesto1 = reactive({
18   id:1,
19   nombre: 'Filtro de aceite',
20   precio: 100,
21   disponibilidad:true
22 });
23
24 // Modifico uno de los campos. Ya no value.
25 repuesto1.disponibilidad = false;
26
27 // Propiedad computada
28 const evaluarPrecio = computed(()=>{
29     return (repuesto1.precio < 120) ? 'Oferta !!!' : 'Precio regular';
30   });
31
32 repuesto1.precio = 150;
33
34 </script>
35
36 <template>
37   <h1>{{ taller }}</h1>
38   <h3>Servicios disponibles</h3>
39   <ul>
40     <li>{{ servicio1 }}</li>
41     <li>{{ servicio2 }}</li>
42   </ul>
43   <h3>Repuestos</h3>
44   <section>
45       <p>{{ repuesto1.nombre }} ... {{ repuesto1.disponibilidad }} </p>
46       <p>{{ evaluarPrecio }}</p>
47   </section>
48 </template>
49
50 <style scoped>
51 h1,h3,li,p{
52   font-family: Arial;
53 }
54 section{
55   display:flex;
```

```
}
</style>
```

Lo que acabamos de hacer con una propiedad computada, también se puede hacer con un **método**.

```

<script setup>
// La función ref hay que importarla. También reactive
import {ref, reactive, computed} from "vue";

// ref
const taller = "Taller Arriaga";
// ref envuelve el string en un objeto cuya principal propiedad es value.
const servicio1 = ref('Basico');
const servicio2 = ref("Especial");

// Modifico el valor de servicio dos.
servicio2.value = "Premiun";

// reactive
// Objeto, no un tipo de dato primitivo.
const repuesto1 = reactive({
  id:1,
  nombre: 'Filtro de aceite',
  precio: 100,
  disponibilidad:true
});

// Modifico uno de los campos. Ya no value.
repuesto1.disponibilidad = false;

// Propiedad computada
const evaluarPrecio = computed(()=>{
    return (repuesto1.precio < 120) ? 'Oferta !!!' : 'Precio regular';
  });



// Método
const funcionEvaluarPrecio = ()=>{
    return (repuesto1.precio < 120) ? 'Oferta !!!' : 'Precio regular';
  };

  repuesto1.precio = 150;
</script>

<template>
  <h1>{{ taller }}</h1>
  <h3>Servicios disponibles</h3>
  <ul>
    <li>{{ servicio1 }}</li>
    <li>{{ servicio2 }}</li>
  </ul>
  <h3>Repuestos</h3>
  <section>
      <p>{{ repuesto1.nombre }} ... {{ repuesto1.disponibilidad }} </p>
      <p>{{ evaluarPrecio }}</p>
```

```
53        <p>{{ funcionEvaluarPrecio() }}</p>
54    </section>
55  </template>
56
57  <style scoped>
58  h1,h3,li,p{
59    font-family: Arial;
60  }
61  section{
62    display:flex;
63  }
64  </style>
```

La diferencia entre las propiedades computadas y los métodos, es que las propiedades computadas se almacenan en caché en función de sus dependencias reactivas. Una propiedad computada solo se volverá a evaluar cuando alguna de sus dependencias reactivas hayan cambiado por lo que consume menos recursos que un método.