



## Clase 09

# Diseño y Programación Web

## Materia:

Aplicaciones para Dispositivos Móviles

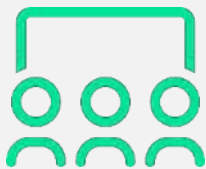
**Docente contenidista:** GARCIA, Mabel

**Revisión:** Coordinación

# Contenido

Entorno de trabajo.....	04
Node.js .....	04
npm.....	09
Crear proyecto con Vite .....	09
Eslint .....	11
Prettier.....	11
Estructura del proyecto generado .....	14
Archivos SFC (Single File Components) .....	18
Diferencias Sintácticas .....	19
Slots .....	23
Bibliografía .....	26

# Clase 9



¡Te damos la bienvenida a la materia  
**Aplicaciones para Dispositivos Móviles!**

**En esta clase vamos a ver los siguientes temas:**

- Entorno de Trabajo.
- Node.Js
- npm,
- Crear proyecto con Vite
- Eslint
- Prettier
- Estructura del proyecto generado.
- Archivos SFC (Single File Components).
- Diferencias sintácticas.
- Slots.

# Entorno de trabajo

En esta etapa vamos a explorar las ventajas y el uso práctico del entorno de consola para el desarrollo de aplicaciones Vue 3. Descubriremos cómo este enfoque **optimiza el flujo de trabajo, mejora la organización del código y nos prepara para adentrarnos en el desarrollo web moderno.**

La consola permite establecer **la estructura base del proyecto**, configurar aspectos como el enrutamiento, la gestión del estado y la integración con bibliotecas, de forma rápida y automatizada. Esto **ahorra tiempo y reduce la posibilidad de errores manuales.**

Dentro de las ventajas de este enfoque nos permite **automatizar tareas repetitivas** como la **creación de nuevos archivos, la configuración de componentes y la instalación de dependencias.**

Este nuevo entorno de trabajo nos prepara para **conocer entornos profesionales, donde estas prácticas son la norma.**

## Node.js

Node.js juega un papel fundamental en el entorno de consola de Vue 3, actuando como la base sobre la cual se ejecutan las herramientas y scripts que facilitan el desarrollo de aplicaciones web modernas.

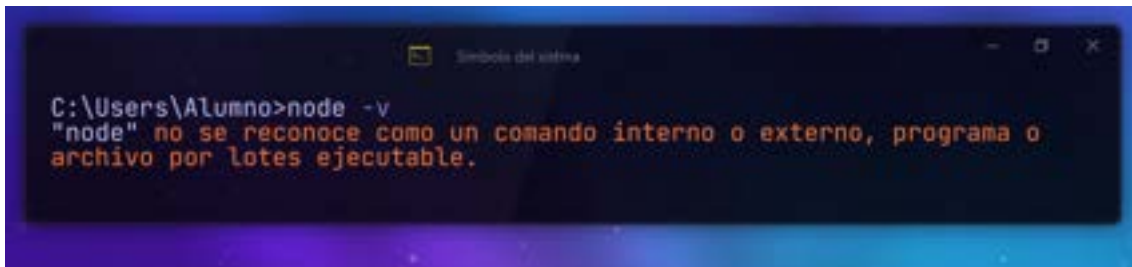
Es un entorno de ejecución de JavaScript que permite ejecutar código JavaScript fuera del navegador web. Esto lo convierte en una herramienta ideal para crear herramientas de línea de comandos, servidores web y aplicaciones de red.

Las herramientas de línea de comandos (CLI) para Vue 3, como **vue create** y **vite**, se basan en **Node.js** para funcionar. **Estas herramientas se ejecutan en la consola y utilizan el motor JavaScript de Node.js** para procesar comandos, generar código entre otras tareas.

Por lo general, deberíamos **verificar desde la consola si tenemos alguna versión de nodejs instalada previamente.**

Esto se hace ejecutando el comando: **node -v**

Si no tenemos node nos aparecerá en consola:



```
C:\Users\Alumno>node -v
"node" no se reconoce como un comando interno o externo, programa o
archivo por lotes ejecutable.
```

Tendremos que descargarlo de la página oficial :  
<https://nodejs.org/es/download/>,



Habitualmente descargamos la versión LTS (Long Term Support) que ofrecen **estabilidad a largo plazo**, soporte extendido y menor riesgo de problemas de compatibilidad.

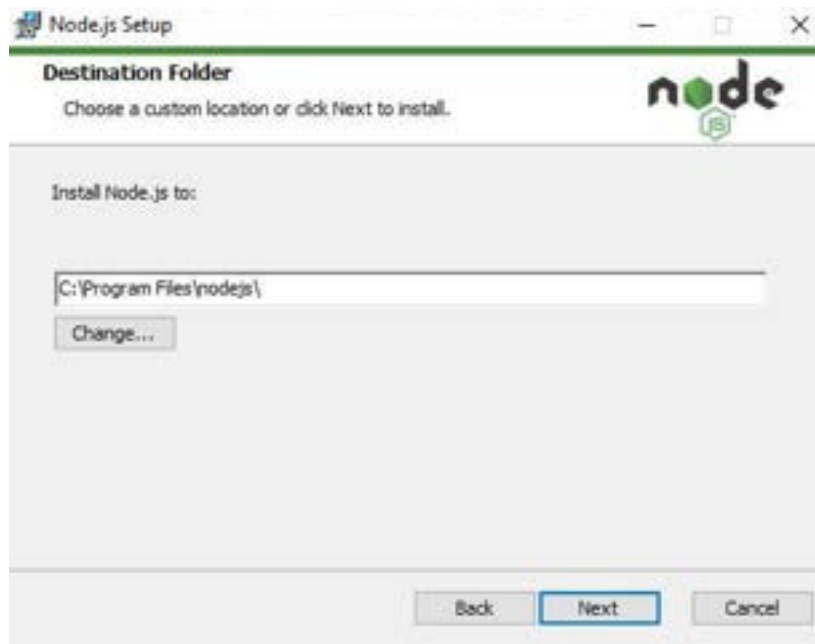
Una vez descargado, si estamos trabajando con el sistema operativo de Windows, haremos click sobre Instalador Windows. Es un archivo con extensión "msi" mostrará el típico asistente de instalación de software.



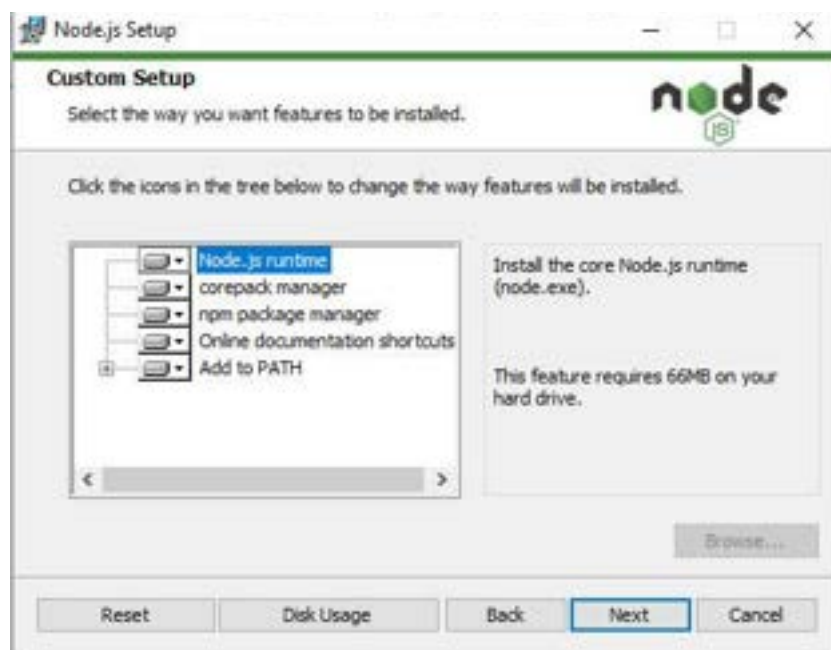
Aceptamos los términos de la licencia para poder utilizarlo. Click en siguiente.



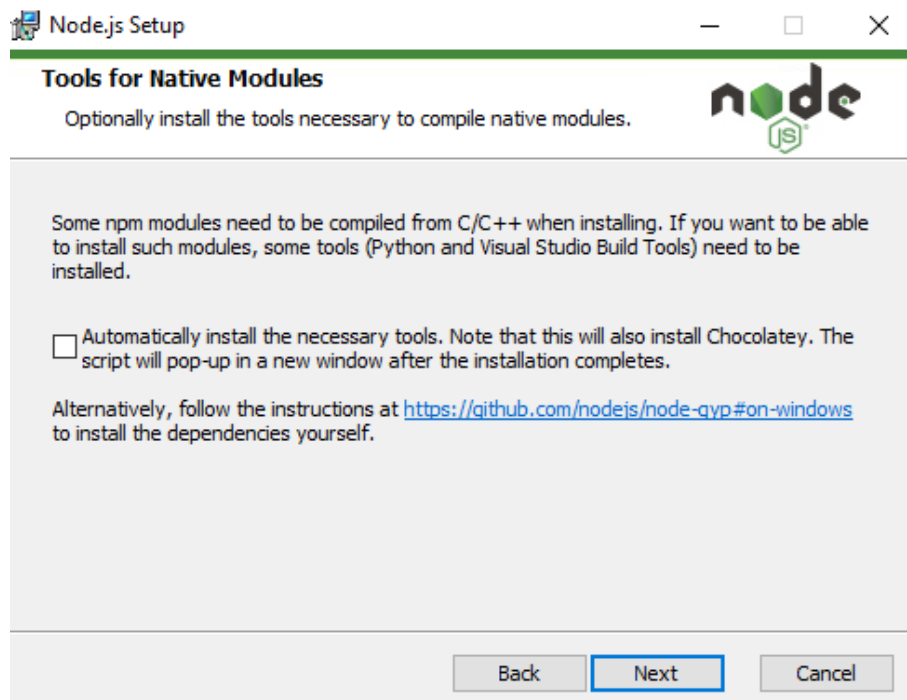
Pasaremos a la siguiente pantalla para elegir dónde queremos instalar, por defecto, nos aparece en C:\Program Files\nodejs.



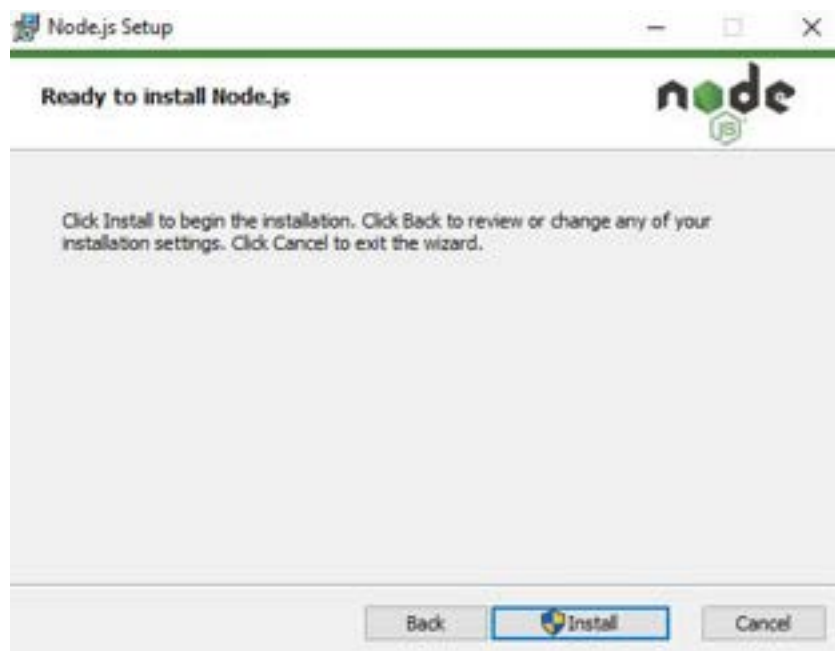
En esta pantalla no elegiremos nada, haremos click en siguiente.



En la pantalla siguiente, haremos click en siguiente. Tampoco necesitaremos herramientas para crear módulos nativos.

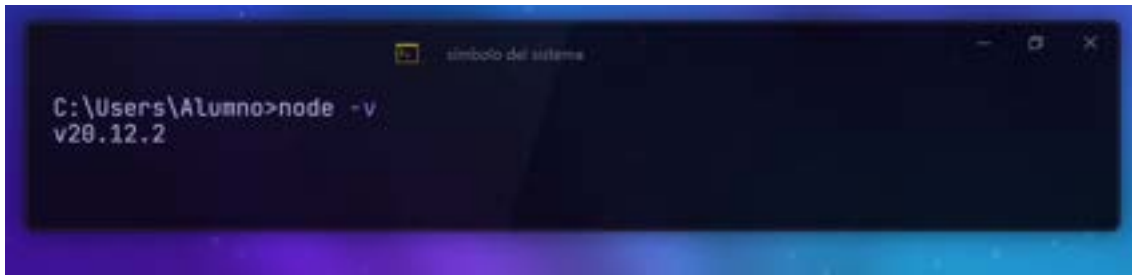


Hacemos click e instalamos.



Una vez finalizado el proceso, haremos click en finish y abriremos la consola para verificar que se ha instalado correctamente **nodejs** con el comando **node -v**. **Nos debe informar que versión tenemos ahora instalada.**





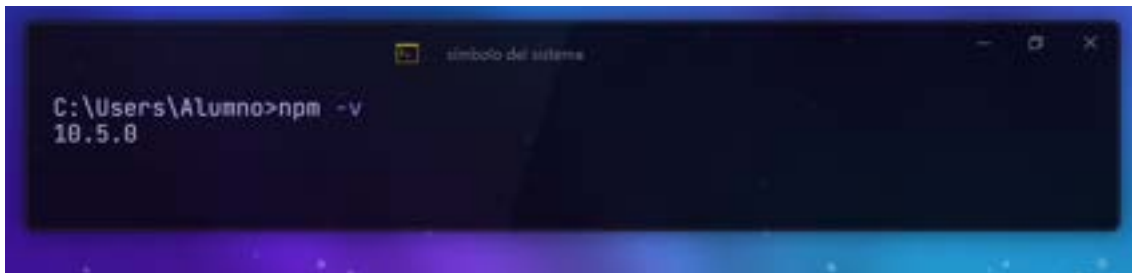
```
C:\Users\Alumno>node -v
v20.12.2
```

## npm

npm (Node Package Manager) es un gestor de paquetes de código abierto para JavaScript que nos permite instalar, actualizar y eliminar bibliotecas y herramientas con Node.js. Funciona como una especie de biblioteca centralizada donde los desarrolladores publican y comparten su código para que otros puedan reutilizarlo.

Cuando instalamos Node.js tenemos esta herramienta a disposición.

En la consola podemos verificar qué versión de npm se ha instalado:



```
C:\Users\Alumno>npm -v
10.5.0
```

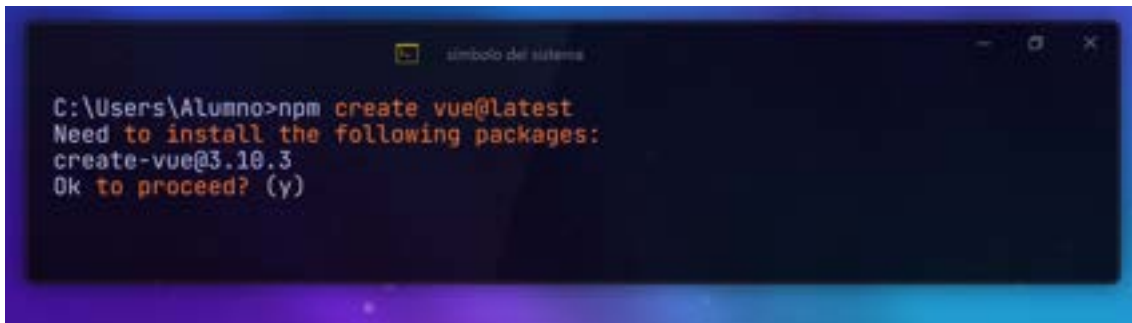
## Crear proyecto con Vite

Vite (palabra en francés para "rápido", pronunciado como /vit/, como veet") es una **herramienta de compilación que tiene como objetivo proporcionar una experiencia de desarrollo más rápida y ágil**. Fue creado por Evan You, creador de Vue. Actualmente la documentación oficial nos aconseja que para nuevos proyectos usemos Vite, en reemplazo de la anterior herramienta llamada @vue/cli.

Esta herramienta nos va a permitir trabajar **por línea de comandos** (cmd) para generar nuestros proyectos.

Para empezar con Vite + Vue3, simplemente ejecutamos en consola el comando **npm create vue@latest**. **Este comando significa:**

- **npm:** El comando para ejecutar el administrador de paquetes de Node.js, para instalar, desinstalar y administrar paquetes en un proyecto.
- **create:** Este es un subcomando de npm que se utiliza para crear un nuevo proyecto a partir de una plantilla predefinida.
- **vue@latest:** Esta es la plantilla que se usará para crear el nuevo proyecto. En este caso, vue es la plantilla oficial de Vue.js y @latest indica que se usará la última versión estable disponible.



```

C:\Users\Alumno>npm create vue@latest
Need to install the following packages:
create-vue@3.10.3
Ok to proceed? (y)

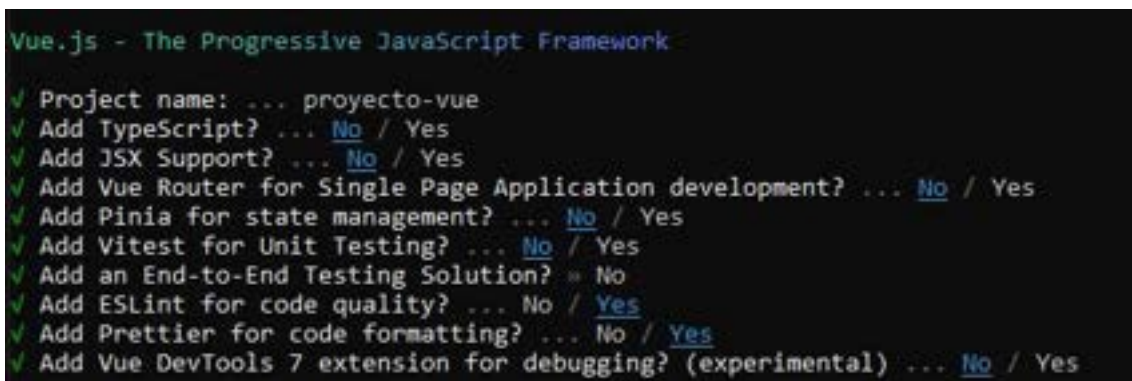
```

Y nos aparecerá que necesitamos esa versión de vue3. Tipeamos en consola **y** (yes) para que empiece la instalación.

Luego, empezaremos con las configuraciones para nuestro nuevo proyecto.

Primero nos pedirá un **nombre para el proyecto**. Daremos enter y nos seguirá ofreciendo distintas opciones de configuración.

Haremos una configuración básica para empezar. Nos moveremos con las teclas del cursor para elegir la opción **YES** para poder usar **Eslint** y **Prettier**. El **resto de las opciones las marcaremos con NO**.



```

Vue.js - The Progressive JavaScript Framework
✓ Project name: ... proyecto-vue
✓ Add TypeScript? ... No / Yes
✓ Add JSX Support? ... No / Yes
✓ Add Vue Router for Single Page Application development? ... No / Yes
✓ Add Pinia for state management? ... No / Yes
✓ Add Vitest for Unit Testing? ... No / Yes
✓ Add an End-to-End Testing Solution? » No
✓ Add ESLint for code quality? ... No / Yes
✓ Add Prettier for code formatting? ... No / Yes
✓ Add Vue DevTools 7 extension for debugging? (experimental) ... No / Yes

```

# Eslint

**ESLint** es una herramienta indispensable para analizar y detectar errores o problemas potenciales en nuestro código JavaScript, TypeScript y Vue.

Funciona como un linter, que revisa nuestro código en busca de violaciones de las reglas de estilo y mejores prácticas.

Nos garantiza que todo el código del proyecto siga las mismas pautas, lo que facilita la lectura y comprensión para todos los desarrolladores.

Se integra con la mayoría de los editores de código populares y nos brinda realimentación en tiempo real sobre los posibles problemas en nuestro código cuando escribimos.

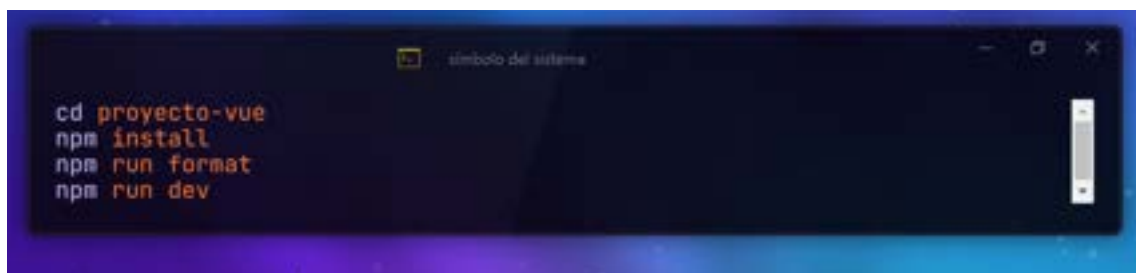
# Prettier

**Prettier** es una herramienta de **formateo de código** que nos ayuda a mantener una **codificación consistente y legible** en nuestro proyecto Vue. Analiza nuestro código JavaScript, TypeScript o CSS y lo **re-formatea automáticamente** de acuerdo a un conjunto de reglas predefinidas o personalizadas.

Esto nos asegura que el código esté formateado de manera uniforme, con sangrías consistentes, espacios en blanco y alineaciones, lo que facilita nuestra lectura y comprensión.

**El uso de ESLint junto con Prettier en nuestro proyecto Vue 3 con Vite es una práctica altamente recomendada.** Estas herramientas nos ayudarán a mantener un código limpio, consistente y libre de errores, mejorando la calidad y mantenibilidad del proyecto a largo plazo.

Una vez terminada la configuración, la consola nos mostrará los comandos necesarios para poder entrar al proyecto y poder correr en el servidor.

A screenshot of a terminal window with a dark background and light blue text. The window title is 'símbolo del sistema'. The commands entered are: 'cd proyecto-vue', 'npm install', 'npm run format', and 'npm run dev'.

```
cd proyecto-vue
npm install
npm run format
npm run dev
```

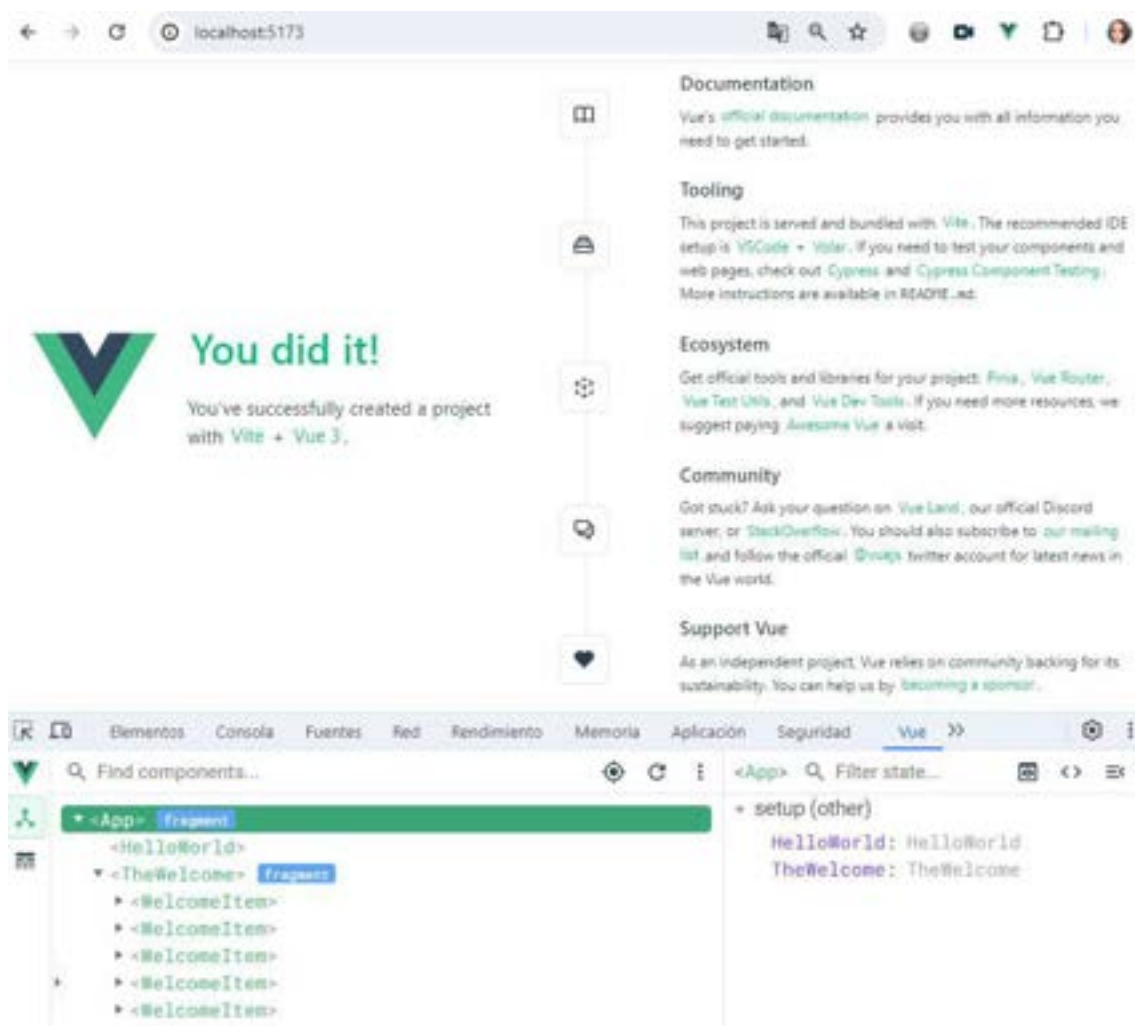
- **cd proyecto-vue** : Nos lleva a la carpeta del proyecto.
- **npm install** : Instala las dependencias del proyecto.
- **npm run format** : Formatea el código automáticamente con Prettier.
- **npm run dev** : Inicia el servidor de desarrollo para nuestra aplicación Vue.

Dependiendo de las características del proyecto generado pueden variar los comandos a utilizar.

El último comando nos permitirá visualizar el proyecto base con las características de configuración que asignamos para el proyecto.

Abrimos el navegador y en la barra de direcciones pondremos la dirección:

**localhost:5173.**

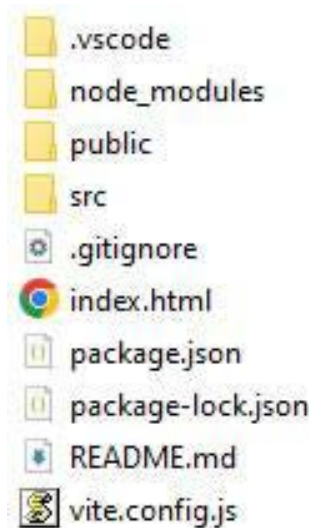


Todo cambio que hagamos en nuestro editor de código actualizará el navegador en tiempo real, esto se debe a la Api HMR (Hot Module Replacement) de Vite.

Si abrimos la consola el panel de Vue Devtools, veremos cómo se organiza el árbol de componentes que posee el proyecto base generado.

# Estructura del proyecto generado

Al abrir el explorador de archivos, nos ubicaremos en la carpeta en donde hemos creado el proyecto y examinaremos el código proporcionado.



- **package.json:**

Este archivo contiene información esencial sobre el proyecto, incluyendo su nombre, versión, dependencias, scripts y otros metadatos. Funciona como un archivo de manifiesto para paquetes Node.js y proporciona una forma estandarizada de gestionar las dependencias y configuraciones del proyecto.

- **vite.config.js:**

Posee la configuración del servidor de desarrollo Vite y la herramienta de compilación. Define varios aspectos del proceso de desarrollo y compilación, como el directorio raíz del proyecto, los plugins y las opciones de compilación. Vite es una herramienta altamente performante y versátil que facilita el desarrollo y la empaquetación eficiente de proyectos.

- **README.md:**

Sirve como documentación del proyecto, nos ofrece una descripción general del mismo, instrucciones de instalación, guías de uso y cualquier otra información relevante. Es una

herramienta importante para comunicar los detalles del proyecto y guiar a los usuarios.

- **Carpeta src:**

Esta carpeta contiene el código fuente para la aplicación de Vue. Contiene todos los componentes, scripts, estilos y assets que conforman la funcionalidad y la interfaz de usuario. Es el directorio central donde escribiremos y organizaremos nuestro código.

- **App.vue:**

Representa el componente Vue raíz de nuestra aplicación. Define la estructura y el diseño general de tu aplicación, incluyendo el contenedor principal y cualquier componente anidado. Es el punto de partida para construir la interfaz de usuario.

- **Carpeta assets:**

Se usa para almacenar recursos estáticos como estilos CSS, imágenes, fuentes y cualquier otro archivo multimedia que requiera nuestro sistema. Nos da la posibilidad de organizar y gestionar estos recursos de forma independiente del código fuente principal.

- **Carpeta components:**

Dentro estarán todos los componentes de Vue reutilizables que generemos. Estos componentes pueden importarse y utilizarse en toda la aplicación para mejorar la modularidad y la reutilización del código. Promueve una base de código estructurada y mantenible.

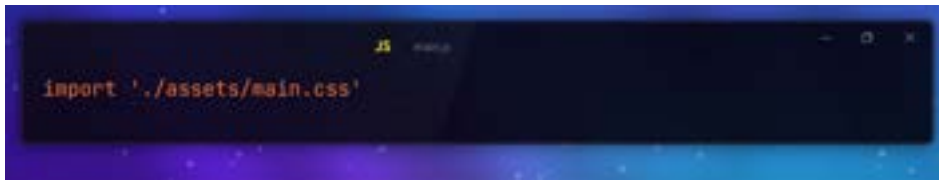
- **index.html:**

Sirve como punto de entrada para el código JavaScript de la aplicación. Normalmente importa el componente App.vue y lo monta en un elemento HTML, haciéndolo visible para el usuario. Inicializa la aplicación y pone en marcha el proceso de renderizado.

- **main.js:**

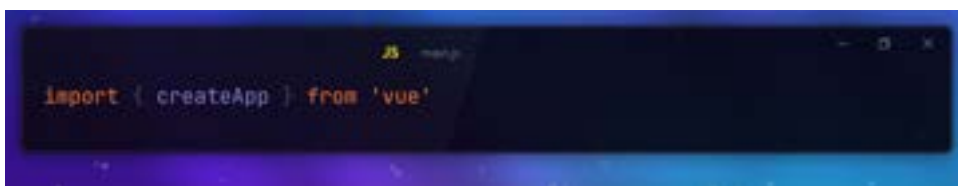
Es otro punto de entrada para la aplicación. Crea una instancia de Vue y monta el componente App.vue, similar a index.js. Ambos archivos proporcionan diferentes formas de inicializar la

aplicación.  
Veamos su código:



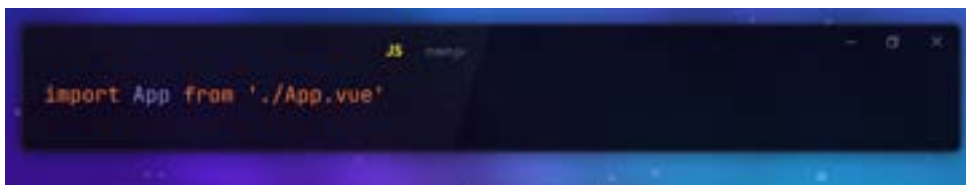
```
import './assets/main.css'
```

- Importa el archivo CSS principal de la aplicación (main.css), que se encuentra dentro de la carpeta assets. Este archivo CSS contiene los estilos globales que se aplicarán a toda la aplicación.



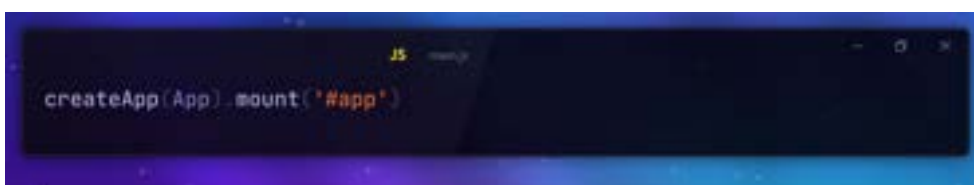
```
import { createApp } from 'vue'
```

- Importa la **función createApp desde la librería Vue (dentro de node\_modules)**. Esta función se usa para crear **una nueva instancia de Vue**.



```
import App from './App.vue'
```

- Importa el componente raíz de la aplicación (App.vue). Este componente será el punto de montaje principal de la aplicación Vue y contendrá todos los demás componentes que se renderizarán.



```
createApp(App).mount('#app')
```

- Crea una nueva instancia de Vue, pasando el componente raíz **App como argumento a la función createApp**. Luego, usa el método mount para montar la aplicación Vue en el elemento HTML con el id #app. Esto significa que **todo el contenido de nuestra aplicación se renderizará dentro del elemento <div id="app">...</div> del archivo index.html**.

- **Carpeta node\_modules:**



Se genera al instalar dependencias cuando usamos npm.  
Contiene los paquetes instalados y sus respectivos archivos.

- **.gitignore:**

Se usa para especificar archivos o directorios que deben ser ignorados por el sistema de control de versiones Git.  
Habitualmente la carpeta node\_modules se ignora. Ésta se regenera cuando se hace el npm install en consola.

- **Carpeta public:**

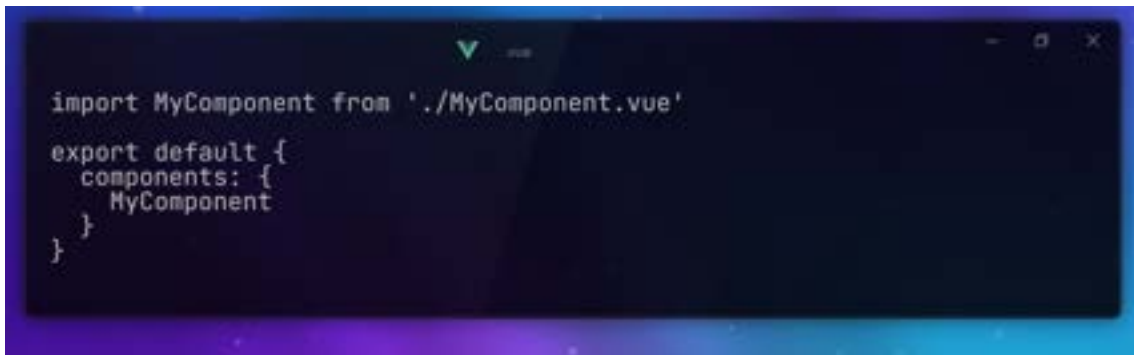
Se usa normalmente para almacenar recursos estáticos que se servirán directamente al navegador durante la implementación.

# Archivos SFC (Single File Components)

Los archivos SFC (Single File Components) son una característica fundamental en Vue 3 que nos permite organizar y modularizar nuestro código de manera eficiente. Un archivo SFC generalmente tiene **la extensión .vue y contiene tres secciones principales:**

1. **Template:** La sección **<template>** donde definimos la estructura y el marcado HTML de nuestro componente Vue, incluyendo enlaces a datos y directivas de Vue.
2. **Script:** La sección **<script>** donde escribimos la lógica JavaScript de nuestro componente, incluyendo la definición del componente, la lógica de datos, métodos, computed, watch, y hooks del ciclo de vida.
3. **Estilos:** La sección **<style>** donde definimos los estilos CSS o preprocesados (como SCSS o LESS) específicos de nuestro componente, asegurando la encapsulación de estilos y evitando la contaminación del estilo global.

Un SFC compilado es un módulo estándar de JavaScript (ES); lo que significa que con una configuración de construcción adecuada, podemos importar un SFC como un módulo:



```
import MyComponent from './MyComponent.vue'

export default {
  components: {
    MyComponent
  }
}
```

Las etiquetas **<style>** dentro de los SFC se inyectan normalmente como etiquetas **<style>** nativas durante el desarrollo para soportar las actualizaciones en "caliente". (Vista en vivo desde el navegador).

# Diferencias Sintácticas

Cuando creamos un nuevo proyecto Vue 3 usando la **herramienta Vite**, la instalación por defecto se realiza bajo la filosofía de la **API Composition**.

Esto significa que el enfoque principal para definir la funcionalidad de nuestros componentes es mediante **la composición de funciones y lógica reutilizable**, en lugar de depender exclusivamente de la API de Options.

Para **trabajar con la API de Options**, debemos **realizar algunas modificaciones específicas en nuestro código, ya que la estructura y el enfoque de desarrollo difieren ligeramente**.

En lugar de utilizar la composición de funciones con **setup()** en nuestros componentes, **emplearemos las opciones tradicionales como data, methods, computed, props, etc directamente en el objeto de opciones del componente**.

Veamos qué archivos y partes del código debemos modificar para trabajar con la API de Options:

Para la **instancia principal (App.vue)** utilizando Composition API, el código sólo tiene la importación de los componentes que requiere la instancia. Esta importación está dentro de la etiqueta **<script setup>** que **permite combinar la lógica de data, computed, methods y watch en un solo bloque de código**.

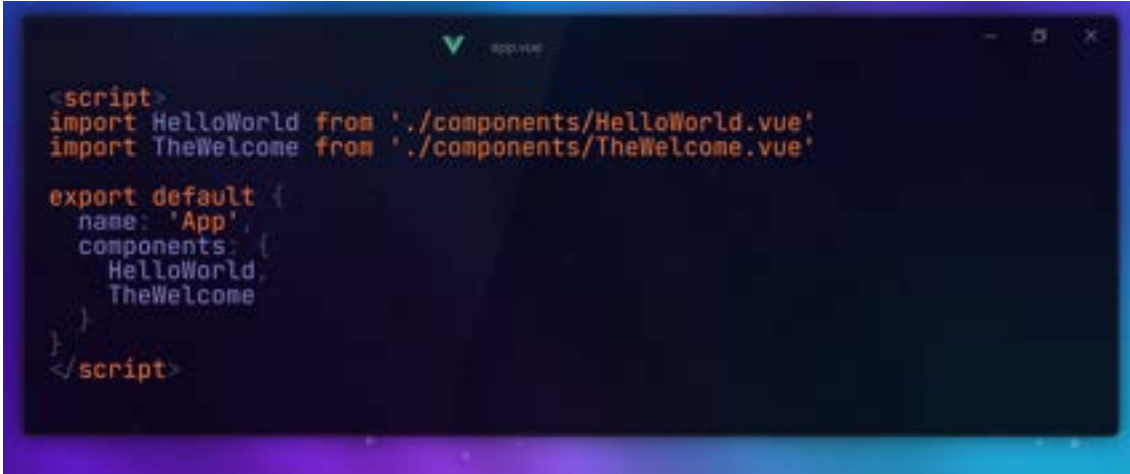
Al importar un componente dentro de otro archivo .vue, es común otorgarle un **nombre significativo y descriptivo** para referirse al componente dentro de ese archivo.



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

En el **archivo App.vue usando la API Options**, usaremos la sintaxis de **export default**. El componente principal (App) se **exporta como un objeto que contiene la configuración del componente**, incluyendo la asignación de su nombre y la lista de componentes que se utilizarán en su interior. La propiedad **name**, nos permite otorgarle un nombre a la instancia o


componente. Mientras que la propiedad **components** nos permite **registrar los componentes hijos que serán utilizados en el componente principal App**. Si este elemento tuviera más lógica deberíamos declarar las distintas propiedades para App (methods, props, etc), tal cual como lo hacíamos en la primera etapa que usábamos archivos js.



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

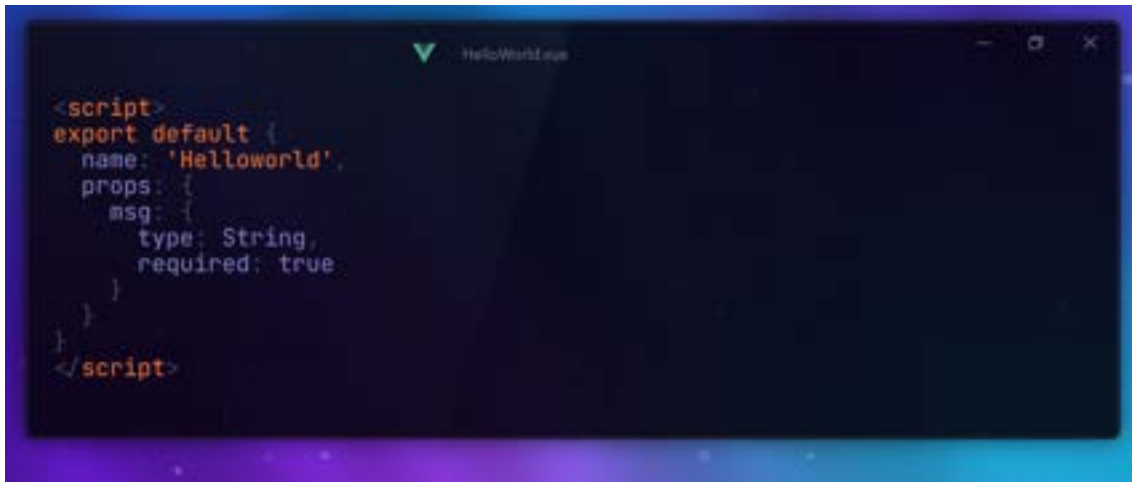
export default {
  name: 'App',
  components: {
    HelloWorld,
    TheWelcome
  }
}
</script>
```

En el **componente HelloWorld**, usando la Api Composition observamos que tenemos una **función llamada defineProps** propia de Vue, que se usa para definir las props que recibirá el componente. Es similar a definir las props en el objeto props del componente en la API Options.



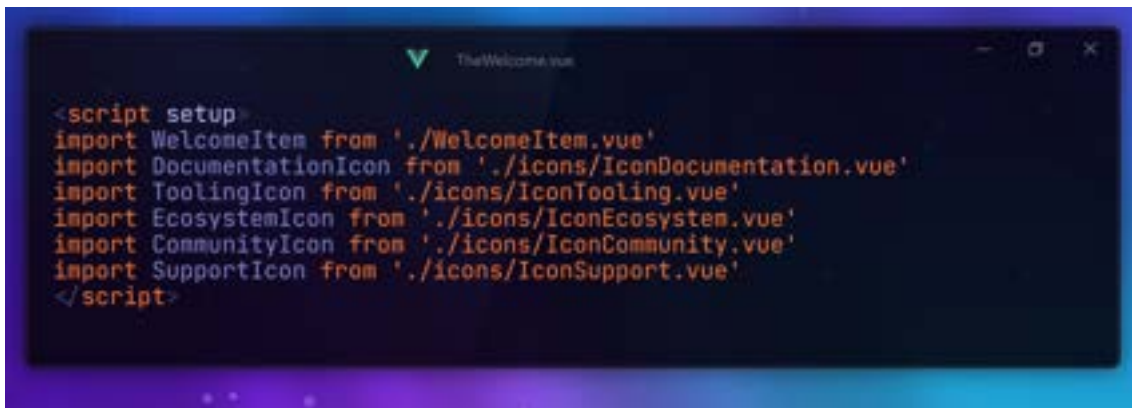
```
<script setup>
defineProps({
  msg: {
    type: String,
    required: true
  }
})
</script>
```

Este sería el caso para el reemplazo del código si usamos la Api Options:



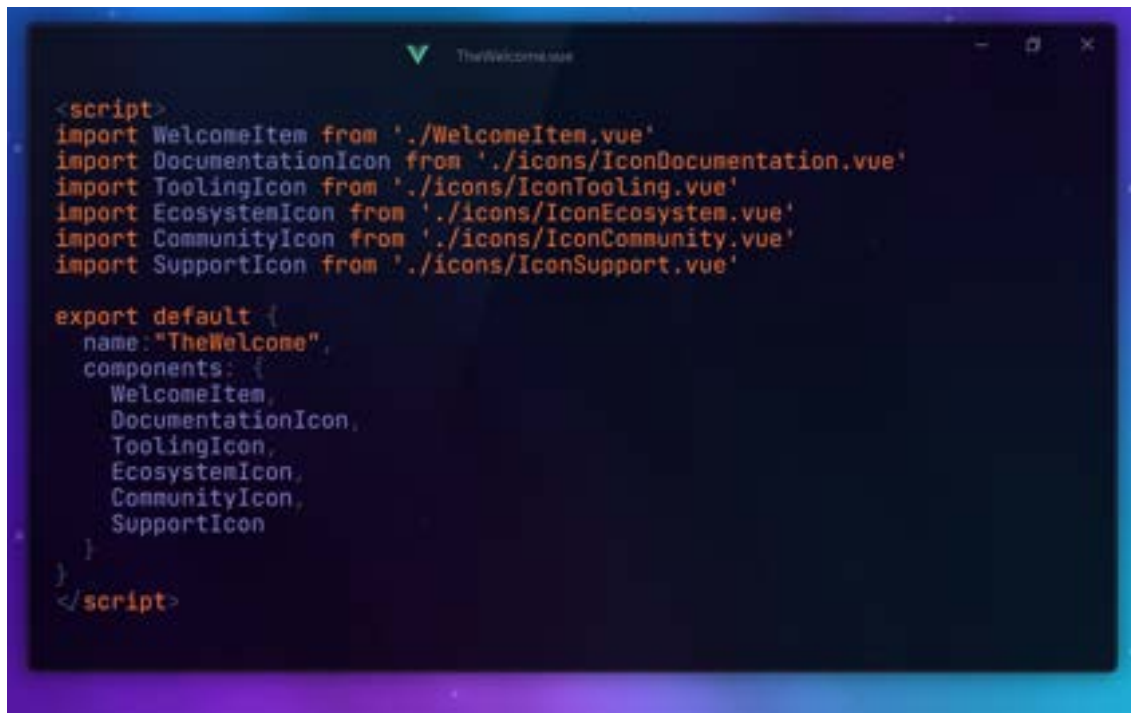
```
<script>
export default {
  name: 'Helloworld',
  props: {
    msg: {
      type: String,
      required: true
    }
  }
}
</script>
```

En el componente **TheWelcome.vue** usando la Api Composition, observamos una sintaxis más simplificada, sólo importando los componentes que requiere este componente para poder usarlos:



```
<script setup>
import WelcomeItem from './WelcomeItem.vue'
import DocumentationIcon from './icons/IconDocumentation.vue'
import ToolingIcon from './icons/IconTooling.vue'
import EcosystemIcon from './icons/IconEcosystem.vue'
import CommunityIcon from './icons/IconCommunity.vue'
import SupportIcon from './icons/IconSupport.vue'
</script>
```

En el caso del componente **TheWelcome.vue** usando la Api Options, tenemos una sintaxis más declarativa.



```
<script>
import WelcomeItem from './WelcomeItem.vue'
import DocumentationIcon from './icons/IconDocumentation.vue'
import ToolingIcon from './icons/IconTooling.vue'
import EcosystemIcon from './icons/IconEcosystem.vue'
import CommunityIcon from './icons/IconCommunity.vue'
import SupportIcon from './icons/IconSupport.vue'

export default {
  name: "TheWelcome",
  components: {
    WelcomeItem,
    DocumentationIcon,
    ToolingIcon,
    EcosystemIcon,
    CommunityIcon,
    SupportIcon
  }
}
</script>
```

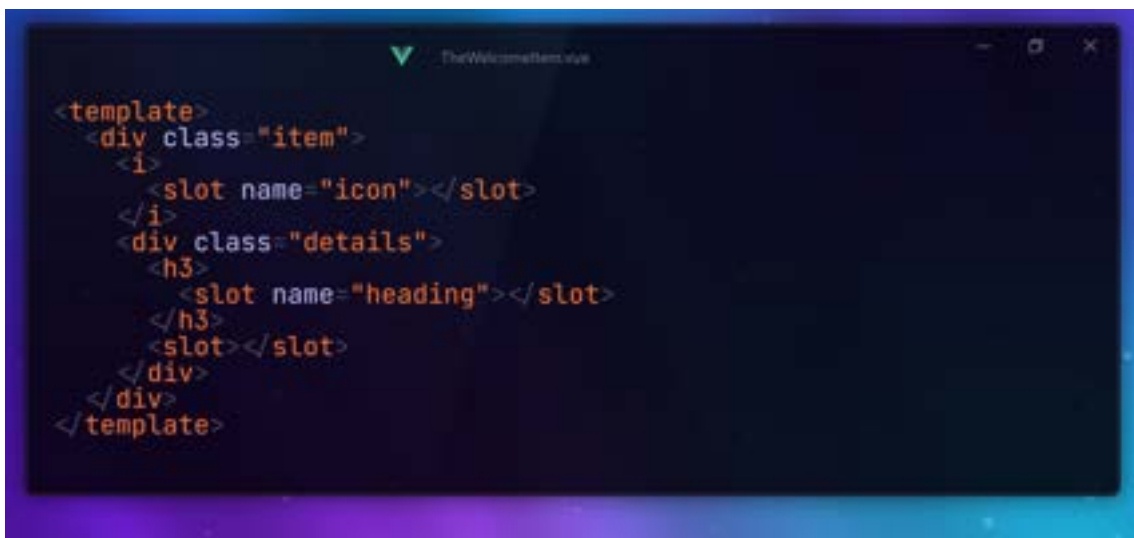
El componente **TheWelcomeItem.vue** está diseñado sin lógica específica declarada. En su lugar, hace uso de un **concepto propio de Vue llamado "slots" dentro de su plantilla (<template>)**.

Los **slots** permiten la **inserción dinámica de contenido** proporcionado **desde el componente padre**, lo que brinda una gran flexibilidad y reutilización en la estructura del componente.

# Slots

Los **slots** son una característica que permite a los componentes **padres renderizar contenido dentro de los componentes hijos**, de una manera flexible y dinámica.

En este archivo **TheWelcomeItem.vue**, se implementa el concepto de **slots** para permitir que **el componente padre decida qué contenido se renderizará dentro de ciertas áreas específicas del componente hijo**.



```
<template>
  <div class="item">
    <i>
      <slot name="icon"></slot>
    </i>
    <div class="details">
      <h3>
        <slot name="heading"></slot>
      </h3>
      <slot></slot>
    </div>
  </div>
</template>
```

En este caso tenemos 3 instancias diferentes de **<slot>**:

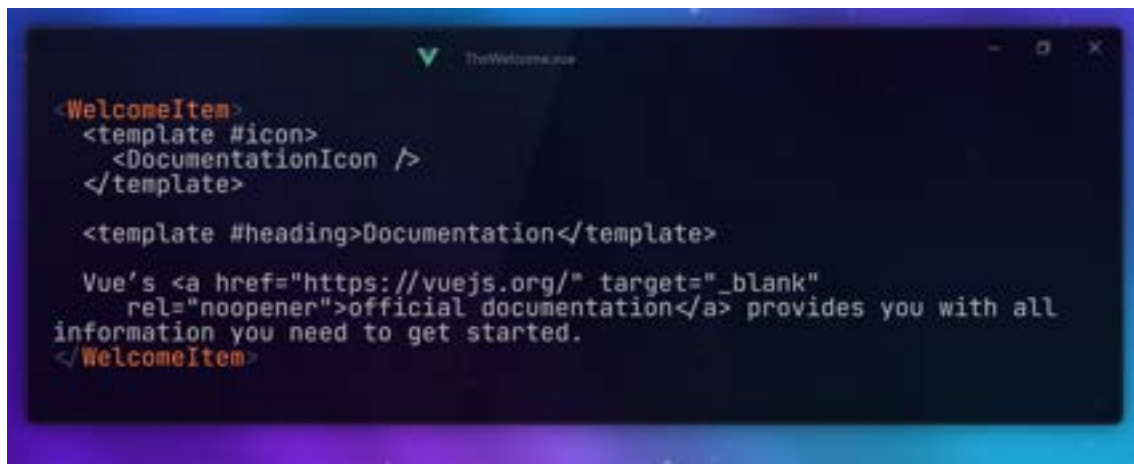
- **<slot name="icon"></slot>**: Este slot con el **nombre "icon"** es un **área reservada dentro del elemento <i>** donde **el componente padre puede renderizar un contenido específico**, como un ícono o una imagen.
- **<slot name="heading"></slot>**: En este caso, el slot con el **nombre "heading"** es un **área reservada dentro del elemento <h3>** donde el **componente padre puede renderizar un encabezado**.
- **<slot></slot>**: Este **slot sin nombre** es un **área reservada** donde el componente **padre puede renderizar cualquier otro contenido adicional que desee dentro del componente hijo**.

El componente padre es **TheWelcome.vue**, y en su template tiene definido cada componente **WelcomeItem** y dentro, con la etiqueta **template**, el **nombre de cada área**.

En el caso de área para el icono, incorpora otro componente que posee un svg en su template.

La información que no está definida dentro de un template con un nombre, corresponde a lo que renderizará dentro como **<slot>** sin nombre.

Por ejemplo:



```
<WelcomeItem>
  <template #icon>
    <DocumentationIcon />
  </template>

  <template #heading>Documentation</template>

  Vue's <a href="https://vuejs.org/" target="_blank"
    rel="noopener">official documentation</a> provides you with all
  information you need to get started.
</WelcomeItem>
```

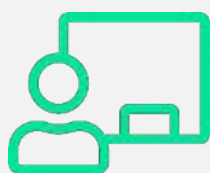
*¡Ya estamos preparados para nuestro nuevo entorno de trabajo!*





Hemos llegado así al final de esta clase en la que vimos:

- Entorno de Trabajo.
- Node.js
- npm,
- Crear proyecto con Vite
- Eslint
- Prettier
- Estructura del proyecto generado.
- Archivos SFC (Single File Components).
- Diferencias sintácticas.
- Slots



Te esperamos en la **clase en vivo** de esta semana.  
No olvides realizar el **desafío semanal**.

**¡Hasta la próxima clase!**

# Bibliografía

Node.js. (s.f.). *Documentación Oficial: Introducción a Node.js*.

Recuperado de <https://nodejs.org/en/learn/getting-started/introduction-to-nodejs>

Node.js. (s.f.). *Documentación Oficial: Introducción a npm*.

Recuperado de <https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager>

Vite.js. (s.f.). *Documentación Oficial: Guía de Vite*. Recuperado de

<https://vitejs.dev/guide/>

ESLint. (s.f.). *Documentación Oficial: Conceptos básicos*. Recuperado de

<https://eslint.org/docs/latest/use/core-concepts/>

Prettier. (s.f.). *Documentación Oficial: Guía*. Recuperado de

<https://prettier.io/docs/en/>

Vue.js. (s.f.). *Documentación Oficial: SFC*. Recuperado de

<https://vuejs.org/guide/scaling-up/sfc.html>

Vue.js. (s.f.). *Documentación Oficial: Slots*. Recuperado de

<https://vuejs.org/guide/components/slots.html#slots>