GOGODEV / @JJRuizEmpresa

Manual de Fundamentos de Vue.js

Autor: Juan José Ruiz, Alejandro Ruiz
YouTube: https://youtube.com/c/gogodev
Twitter: https://twitter.com/JJRuizEmpresa (@JJRuizEmpresa)

IMPORTANTE
El contenido de este pdf lo tienes también en vídeo en el canal de YouTube https://youtube.com/c/gogodev , dentro de la lista de reproducción: Curso de Vue.

También puedes acceder a todo el contenido de dicho curso desde la plataforma de https://ademass.com/ donde también puedes certificarte sobre dicho curso.

Introducción a Vue.js 3

Vue.js es un marco de trabajo progresivo para la construcción de interfaces de usuario. Su última versión, Vue.js 3, ofrece una sintaxis aún más elegante y un rendimiento mejorado. Este manual proporcionará una introducción a los fundamentos de Vue.js 3 para que puedas comenzar a desarrollar aplicaciones web de manera eficiente y efectiva.

Instalación y Configuración

Instalación con CDN:
Puedes incluir Vue.js directamente desde un CDN en tu HTML utilizando una etiqueta script.

```
<script src="https://cdn.jsdelivr.net/npm/vue@3"></script>
```

Instalación con Vue CLI:
Recomendado para proyectos más grandes, Vue CLI facilita la configuración de proyectos Vue.js con opciones para la creación rápida de proyectos, gestión de dependencias y mucho más.

- Para comenzar, necesitarás tener Node.js y Vue CLI instalados en tu sistema. Si aún no los tienes, puedes instalar Node.js desde su sitio web oficial (https://nodejs.org/) y Vue CLI utilizando npm (Node Package Manager) con el siguiente comando en tu terminal:

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

- Una vez que hayas instalado Vue CLI, puedes crear un nuevo proyecto Vue.js utilizando el siguiente comando en tu terminal:

```
vue create mi-proyecto
```

- Esto iniciará un proceso interactivo que te guiará a través de la configuración de tu nuevo proyecto. Una vez completado, navega al directorio de tu proyecto:

```
cd mi-proyecto
```

- Luego, puedes iniciar tu aplicación Vue.js en modo de desarrollo ejecutando el siguiente comando:

```
npm run serve
```

Esto iniciará un servidor de desarrollo y podrás acceder a tu aplicación en tu navegador visitando la URL proporcionada en la terminal (por lo general, http://localhost:8080/). Este es un ejemplo básico de cómo puedes crear un nuevo proyecto Vue.js utilizando Vue CLI. Si necesitas más ayuda o tienes alguna pregunta específica sobre Vue CLI.

Conceptos Básicos

Componentes: Vue.js utiliza el concepto de componentes para construir interfaces de usuario reutilizables y modulares. Un componente en Vue.js es esencialmente una instancia de Vue con opciones predefinidas que incluyen una plantilla, opciones de datos, métodos, y más.

Los componentes permiten dividir la interfaz de usuario en partes más pequeñas y manejables, lo que facilita la construcción y el mantenimiento de aplicaciones Vue.js, especialmente en proyectos grandes y complejos.

Un componente Vue consta de tres partes principales:

Plantilla (Template): Define la estructura HTML del componente y cómo se renderiza en la página. Puedes utilizar la sintaxis de Vue.js dentro de la plantilla para enlazar datos, controlar eventos y realizar renderizado condicional.

Lógica (Script): Contiene los datos que el componente necesita para renderizarse correctamente. Estos datos pueden ser simples variables, objetos, o incluso funciones que devuelven datos reactivos.

También contiene las funciones que el componente utiliza para realizar acciones específicas, como manejar eventos, realizar operaciones de cálculo, o interactuar con otros componentes.

Estilo (style): Aquí se define el estilo específico del componente utilizando CSS. Para evitar que estos estilos afecten a otros componentes o causen conflictos, utilizamos el atributo `scoped` dentro de la etiqueta <style>, lo que limita su alcance únicamente al componente actual.

Ejemplo de un componente :

```vue
<!-- MiComponente.vue -->

<template>
  <div>
    <h1>{{ mensaje }}</h1>
    <button @click="cambiarMensaje">Cambiar Mensaje</button>
  </div>
</template>

<script setup>
import { ref } from 'vue'

const mensaje = ref('¡Hola desde mi componente!')

function cambiarMensaje() {
  mensaje.value = '¡Mensaje cambiado!'
}
</script>

<style scoped>
/* Estilos específicos del componente */
</style>
```

Importar y Usar un Componente en Vue:

Para importar y usar un componente en Vue, primero necesitas importarlo en el archivo donde planeas usarlo. Luego, lo registras en el componente principal o en otro componente donde quieras usarlo. Una vez registrado, puedes usar el componente como si fuera una etiqueta HTML en tu plantilla.

Ejemplo:

Supongamos que tienes un componente llamado MiComponente.vue en el directorio components. Para usar este componente en otro componente, primero lo importarías:

```vue
import MiComponente from '@/components/MiComponente.vue'; // Ruta al componente
```

Finalmente, puedes usar MiComponente en tu plantilla como una etiqueta HTML:

```
<template>
  <div>
    <h1>Mi Aplicación</h1>
    <mi-componente></mi-componente>
  </div>
</template>
```

Directivas: Las directivas son atributos especiales que se utilizan en los elementos HTML para agregar funcionalidad dinámica y reactiva. Estas directivas proporcionan una forma conveniente de manipular el DOM, enlazar datos y controlar el comportamiento de los elementos en respuesta a eventos y cambios en el estado de la aplicación.

Tipos de Directivas:

En Vue.js, hay diferentes tipos de directivas que se utilizan para realizar diversas tareas. Algunas de las directivas más comunes incluyen:

v-bind: La directiva v-bind se utiliza para enlazar dinámicamente los atributos de un elemento HTML a expresiones de datos en el modelo de Vue. Esto permite actualizar los atributos de manera reactiva según cambian los datos en la aplicación.

Ejemplo:

```
<a v-bind:href="url">Enlace</a>
```

v-on: La directiva v-on se utiliza para escuchar eventos en elementos HTML y ejecutar acciones específicas en respuesta a esos eventos. Puedes utilizar esta directiva para manejar eventos del DOM, como clics de ratón, pulsaciones de teclas, cambios de entrada, entre otros.

Ejemplo:

```
<button v-on:click="handleClick">Haz clic</button>
```

v-if: La directiva v-if se utiliza para realizar renderizado condicional en el DOM. Si la expresión vinculada a v-if es verdadera, el elemento se renderizará en el DOM; de lo contrario, se eliminará del DOM.

Ejemplo:

```
<div v-if="mostrarMensaje">Este mensaje se muestra si mostrarMensaje es verdadero</div>
```

Estos son solo algunos ejemplos de las directivas disponibles en Vue.js. Cada directiva tiene su propia función y puede ser utilizada para diferentes propósitos en la construcción de aplicaciones Vue.js.

En resumen, las directivas en Vue.js son poderosas herramientas que te permiten agregar comportamiento dinámico a tus aplicaciones, enlazar datos y controlar el renderizado condicional de elementos en el DOM.

Data Binding: El data binding es un concepto fundamental en Vue.js que permite establecer una conexión bidireccional entre la vista (HTML) y el modelo de datos (JavaScript). Esto significa que cualquier cambio en el modelo de datos se reflejará automáticamente en la vista y viceversa, sin la necesidad de actualizar manualmente el DOM.

Tipos de Data Binding:

En Vue.js, hay diferentes tipos de data binding que se pueden utilizar para establecer esta conexión bidireccional:

Interpolación: Vue.js utiliza la sintaxis de doble llave {{ }} para realizar la interpolación de datos en la vista. Esto permite mostrar valores de propiedades del modelo de datos en el HTML.

Ejemplo:

```
<p>{{ mensaje }}</p>
```

v-bind: Como se mencionó anteriormente, la directiva v-bind se utiliza para enlazar dinámicamente los atributos de los elementos HTML a expresiones de datos en el modelo de Vue. Esto es útil para establecer atributos HTML dinámicamente en función del estado de la aplicación.

Ejemplo:

```
<img v-bind:src="imagenUrl">
```

v-model: La directiva v-model proporciona una forma conveniente de enlazar datos de entrada (como inputs, selects, textareas, etc.) a propiedades del modelo de datos. Con v-model, cualquier cambio en el valor de entrada se reflejará automáticamente en el modelo de datos y viceversa.

Ejemplo:

```
<input v-model="nombre" type="text">
```

Estos son solo algunos ejemplos de cómo Vue.js proporciona una sintaxis fácil de usar para establecer el enlace bidireccional entre la vista y el modelo de datos. Esto hace que el desarrollo de aplicaciones Vue.js sea más eficiente y fácil de mantener, ya que los cambios en los datos se reflejan automáticamente en la vista y viceversa.

3. Ciclo de Vida de los Componentes

El ciclo de vida de un componente en Vue.js se refiere al conjunto de etapas por las que pasa un componente desde su creación hasta su destrucción. Durante este ciclo, Vue.js proporciona diversos "hooks" o ganchos de ciclo de vida que te permiten ejecutar código en diferentes momentos clave del ciclo de vida del componente. Esto te brinda la flexibilidad necesaria para realizar acciones específicas en cada etapa del ciclo de vida.

Principales Etapas del Ciclo de Vida:

beforeCreate: Este gancho se ejecuta antes de que se cree la instancia del componente. En este punto, el componente aún no tiene acceso a las opciones de datos (data) ni a los métodos (methods) definidos en el componente.

created: Se ejecuta después de que se haya creado la instancia del componente, pero antes de que se monte en el DOM. En este punto, el componente ha inicializado sus datos y métodos, pero aún no se ha agregado al DOM.

beforeMount: Este gancho se ejecuta justo antes de que el componente se monte en el DOM. En este punto, el componente ya ha sido creado y ha renderizado su plantilla, pero aún no se ha adjuntado al DOM.

mounted: Se ejecuta después de que el componente se haya montado en el DOM. En este punto, el componente está visible en la página y se pueden acceder a sus elementos en el DOM.

beforeUpdate: Este gancho se ejecuta antes de que el componente se actualice debido a cambios en los datos. En este punto, el componente ha recibido nuevas propiedades o ha actualizado sus datos, pero aún no se ha vuelto a renderizar en el DOM.

updated: Se ejecuta después de que el componente se haya actualizado en el DOM debido a cambios en los datos. En este punto, el componente ha sido re-renderizado con los nuevos datos y el DOM ha sido actualizado en consecuencia.

beforeUnmount: Este gancho se ejecuta justo antes de que el componente se elimine del DOM. En este punto, el componente aún es accesible y funcional, pero está a punto de ser eliminado.

unmounted: Se ejecuta después de que el componente se haya eliminado del DOM. En este punto, el componente ya no es accesible y se ha limpiado de la memoria.

Uso de los Hooks del Ciclo de Vida:

Puedes utilizar estos ganchos del ciclo de vida para realizar diversas acciones, como inicializar datos, realizar llamadas a la API, suscribirse a eventos, limpiar recursos y más. Esto te permite controlar el comportamiento del componente en cada etapa del ciclo de vida y garantizar que se ejecute el código necesario en el momento adecuado.

4. Eventos y Métodos

En Vue.js, los eventos y los métodos son componentes clave para la interactividad de tus aplicaciones. Te permiten responder a acciones del usuario, como clics de ratón, pulsaciones de teclas, cambios de entrada, entre otros, y ejecutar código en respuesta a estos eventos.

Eventos:

Los eventos en Vue.js son acciones que ocurren en la interfaz de usuario, como hacer clic en un botón, ingresar texto en un campo de entrada, o desplazarse por una lista. Puedes escuchar estos eventos utilizando la directiva ‘v-on’ y ejecutar funciones de manejo de eventos en respuesta a ellos.

Ejemplo:

```
<button v-on:click="handleClick">Haz clic</button>
```

En este ejemplo, estamos utilizando la directiva v-on para escuchar el evento de clic en el botón y ejecutar la función handleClick cuando se produce el evento.

En Vue.js 3, puedes definir tanto eventos como métodos dentro del bloque script setup. Aquí tienes un ejemplo de cómo hacerlo:

Ejemplo:

```
<template>
  <button @click="incrementarContador">Haz clic</button>
  <p>{{ contador }}</p>
</template>

<script setup>
import { ref } from 'vue'

// Definimos una variable reactiva para almacenar el contador
const contador = ref(0)

// Definimos un método para incrementar el contador
function incrementarContador() {
  contador.value++
}
</script>
```

En este ejemplo:

- Importamos la función ref de Vue para crear una referencia reactiva que almacena el contador.
- Definimos una variable contador utilizando ref para almacenar el estado del contador.
- Definimos un método incrementarContador() que incrementa el valor del contador cada vez que se llama.
- En la plantilla, usamos @click para escuchar el evento de clic en el botón y llamar al método incrementarContador().

Con 'script setup', el código es más conciso y legible. Tanto los eventos como los métodos están disponibles directamente en la plantilla y pueden acceder a las variables reactivas definidas en el mismo bloque 'script setup'. Esto simplifica la estructura del componente y mejora la claridad del código.

5. Renderizado Condicional y Listas

Vue.js ofrece métodos simples y poderosos para realizar renderizado condicional y trabajar con listas en tus componentes. Estas características te permiten controlar dinámicamente

qué elementos se muestran en tu interfaz de usuario y cómo se presentan los datos de tus listas.

Renderizado Condicional:

El renderizado condicional en Vue.js te permite mostrar o ocultar elementos en función del estado de tus datos o de ciertas condiciones lógicas. Puedes utilizar las directivas 'v-if', 'v-else', 'v-else-if' y 'v-show' para lograr esto.

Ejemplo v-if :

```
<p v-if="mostrarMensaje">Este mensaje se muestra si mostrarMensaje es verdadero</p>
```

Ejemplo con v-show:

```
<p v-show="mostrarMensaje">Este mensaje se muestra si mostrarMensaje es verdadero</p>
```

La diferencia entre 'v-if' y 'v-show' radica en cómo afectan al DOM. 'v-if' agrega o elimina completamente el elemento del DOM según la evaluación de la expresión, mientras que 'v-show' solo cambia el estilo display del elemento para mostrar u ocultar.

Listas:

Para renderizar listas de datos en Vue.js, puedes utilizar la directiva 'v-for', que te permite iterar sobre una matriz u objeto y generar contenido dinámicamente basado en cada elemento.

Ejemplo con v-for:

```
<ul>
  <li v-for="item in lista" :key="item.id">{{ item.texto }}</li>
</ul>
```

En este ejemplo, 'lista' es una matriz de objetos, y para cada objeto en la matriz, se genera un elemento <li> en la lista.

Estas características de renderizado condicional y listas en Vue.js te permiten crear interfaces de usuario dinámicas y receptivas, donde los elementos se muestran o se ocultan según las condiciones de tus datos, y donde puedes mostrar listas de datos de manera eficiente y sencilla.

6. Comunicación entre Componentes

La comunicación entre componentes es fundamental en cualquier aplicación Vue.js, ya que te permite compartir datos, eventos y funcionalidades entre diferentes partes de tu aplicación.

Emisión de Eventos:

Una forma común de comunicarse entre componentes en Vue.js es a través de la emisión de eventos. Un componente puede emitir un evento personalizado utilizando el método '$emit', y otros componentes pueden escuchar ese evento utilizando la directiva 'v-on'.

Ejemplo de emisión de eventos sin <script setup>:

```
<!-- Componente A -->
<template>
  <button @click="enviarMensaje">Enviar Mensaje</button>
</template>

<script>
export default {
  methods: {
    enviarMensaje() {
      this.$emit('mensaje-enviado', 'Hola desde el componente A');
    }
  }
}
</script>
```

```vue
<!-- Componente B -->
<template>
  <div>
    <p>{{ mensaje }}</p>
  </div>
</template>

<script>
export default {
  data() {
    return {
      mensaje: ''
    };
  },
  mounted() {
    this.$on('mensaje-enviado', mensaje => {
      this.mensaje = mensaje;
    });
  }
}
</script>
```

En este ejemplo, el componente A emite un evento llamado 'mensaje-enviado' cuando se hace clic en el botón, y el componente B escucha ese evento y actualiza su estado en consecuencia.

Propiedades Personalizadas:

Otra forma de comunicación entre componentes en Vue.js es a través de propiedades personalizadas. Puedes pasar datos de un componente padre a un componente hijo utilizando propiedades, y los componentes hijos pueden acceder a estas propiedades utilizando la sintaxis de interpolación en sus plantillas.

Ejemplo de propiedades personalizadas sin <script setup> :

```vue
<!-- Componente Padre -->
<template>
  <h1>{{ titulo }}</h1>
  <hijo :mensaje="mensaje"></hijo>
</template>

<script>
import Hijo from './Hijo.vue';

export default {
  components: {
    Hijo
  },
  data() {
    return {
      titulo: 'Comunicación entre Componentes',
      mensaje: 'Hola desde el componente padre'
    };
  }
}
</script>
```

```vue
<!-- Componente Hijo -->
<template>
  <p>{{ mensaje }}</p>
</template>

<script>
export default {
  props: ['mensaje']
}
</script>
```

En este ejemplo, el componente padre pasa la propiedad ‘mensaje’ al componente hijo utilizando la sintaxis ‘:mensaje="mensaje"’, y el componente hijo accede a esta propiedad utilizando’ props’.

Estas son algunas de las formas más comunes de comunicación entre componentes en Vue.js. Utilizando la emisión de eventos y las propiedades personalizadas, puedes construir aplicaciones Vue.js altamente modularizadas y reutilizables, donde los componentes pueden interactuar entre sí de manera efectiva.

7. Vue Router

Vue Router es una biblioteca oficial de Vue.js que proporciona una solución para la navegación en aplicaciones Vue.js. Permite definir rutas en tu aplicación y asociarlas con componentes específicos, lo que te permite cambiar dinámicamente las vistas de tu aplicación en respuesta a la navegación del usuario.

Principales Características:

Enrutamiento Declarativo: Con Vue Router, puedes definir las rutas de tu aplicación utilizando una sintaxis declarativa similar a la de Vue.js. Esto te permite especificar las rutas y las correspondientes vistas de manera clara y legible.

Navegación Basada en Componentes: Cada ruta en Vue Router se asocia con un componente Vue específico. Cuando el usuario navega a una determinada ruta, Vue Router carga automáticamente el componente correspondiente y lo renderiza en el lugar adecuado de la aplicación.

Navegación Programática: Además de la navegación basada en la URL, Vue Router también proporciona métodos programáticos para cambiar de ruta desde el código JavaScript. Esto te permite navegar entre rutas y manipular el historial del navegador de manera programática.

Gestión de Estado de la Aplicación: Vue Router facilita la gestión del estado de la aplicación en función de las rutas. Puedes acceder a los parámetros de la ruta, a la consulta de la URL y a otros datos relacionados con la ruta desde cualquier componente de tu aplicación.

Uso de Vue Router:

Para utilizar Vue Router en tu aplicación Vue.js, primero debes instalarlo utilizando npm o yarn. Luego, debes configurar las rutas de tu aplicación utilizando la API proporcionada por Vue Router. Finalmente, debes integrar el enrutador en la instancia de Vue de tu aplicación principal.

Ejemplo de configuración de Vue Router:

```
import { createRouter, createWebHistory } from 'vue-router';
import Home from './components/Home.vue';
import About from './components/About.vue';

const routes = [
  { path: '/', component: Home },
  { path: '/about', component: About }
];

const router = createRouter({
  history: createWebHistory(),
  routes
});

export default router;
```

En este ejemplo, hemos definido dos rutas: una para la página de inicio (‘/’) y otra para la página "Acerca de" (‘/about’). Cada ruta se asocia con un componente específico.

Luego, integramos el enrutador en la instancia de Vue de nuestra aplicación principal:

```
import { createApp } from 'vue';
import App from './App.vue';
import router from './router';

createApp(App).use(router).mount('#app');
```

Con esto, hemos configurado correctamente Vue Router en nuestra aplicación Vue.js, lo que nos permite realizar la navegación entre las diferentes vistas de la aplicación de manera fluida y eficiente.

8. Pinia.js

Pinia.js es una biblioteca de gestión del estado para aplicaciones Vue.js que proporciona una alternativa eficiente y reactiva para el manejo del estado de la aplicación. Está diseñada para simplificar y mejorar la experiencia de desarrollo al proporcionar una arquitectura simple y reactiva para el manejo del estado.

Características Principales:

Arquitectura Reactiva: Pinia.js utiliza una arquitectura reactiva basada en Vue 3, lo que significa que el estado de la aplicación se actualiza de manera reactiva en respuesta a cambios en los datos. Esto garantiza que la interfaz de usuario se actualice automáticamente para reflejar el estado más reciente de la aplicación.

Gestión Centralizada del Estado: Pinia.js adopta un enfoque de gestión centralizada del estado, donde el estado de la aplicación se almacena en un único almacén global. Esto facilita el acceso y la manipulación del estado desde cualquier parte de la aplicación, lo que mejora la modularidad y la mantenibilidad del código.

API Intuitiva: Pinia.js proporciona una API intuitiva y fácil de usar para definir el estado de la aplicación, mutar el estado y acceder a él desde componentes Vue. Esto hace que sea rápido y sencillo implementar la gestión del estado en tu aplicación y reduce la curva de aprendizaje para los desarrolladores.

Soporte para DevTools: Pinia.js ofrece soporte integrado para las herramientas de desarrollo de Vue, como Vue DevTools. Esto facilita la depuración y la inspección del estado de la aplicación durante el desarrollo, lo que ayuda a identificar y solucionar problemas de manera más eficiente.

Uso de Pinia.js:

Para utilizar Pinia.js en tu aplicación Vue.js, primero debes instalarlo mediante npm o yarn. Luego, puedes definir y configurar tus almacenes de estado utilizando la API proporcionada por Pinia.js. Finalmente, puedes integrar los almacenes de estado en tu aplicación Vue.js para comenzar a gestionar el estado de manera eficiente y reactiva.

Con Pinia.js, puedes crear aplicaciones Vue.js más escalables, intuitivas y mantenibles al proporcionar una solución robusta y eficiente para el manejo del estado de la aplicación.

Conclusión

Con este manual, has adquirido una base sólida en los fundamentos de Vue.js 3. Al comprender y practicar estos conceptos, estás preparado para desarrollar aplicaciones web modernas y dinámicas utilizando este poderoso framework.

Vue.js 3 ofrece una sintaxis intuitiva, una arquitectura flexible y herramientas poderosas que te permiten crear aplicaciones web escalables y mantenibles. Con su enfoque en la reactividad, la modularidad y la facilidad de uso, Vue.js se ha convertido en una opción popular para desarrolladores de todo el mundo.

A medida que continúes explorando Vue.js y construyendo aplicaciones, te animo a seguir aprendiendo y experimentando con nuevas características y técnicas. La documentación oficial de Vue.js, la comunidad en línea y los recursos educativos están disponibles para ayudarte en tu viaje de aprendizaje.

¡A PROGRAMAR!