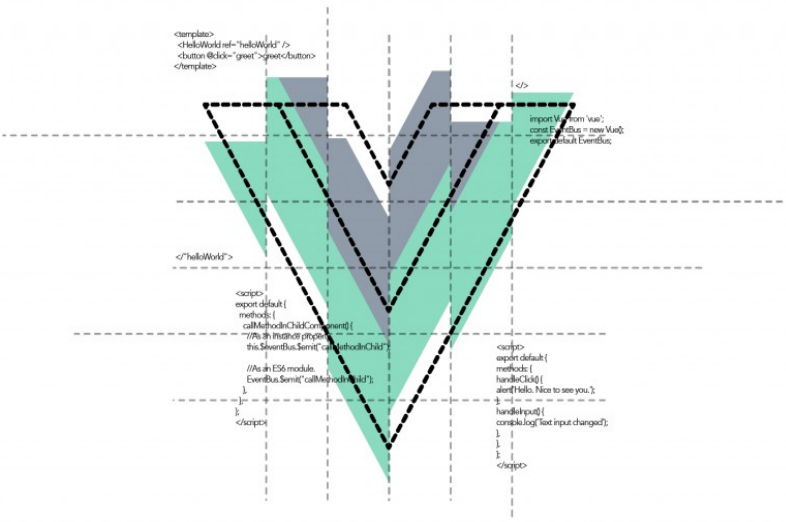


Vue.js 3

Indice Vue.js

¿Que son los Componentes en Vue?

En Vue.js, los componentes son bloques reutilizables de código que encapsulan la lógica y la interfaz de usuario de una parte específica de una aplicación web. Los componentes permiten dividir una aplicación en partes más pequeñas y manejables, lo que facilita el desarrollo, el mantenimiento y la reutilización del código.

A continuación, se presentan los conceptos clave relacionados con los componentes en Vue.js:

1. Definición de Componentes: Un componente en Vue.js se define como un objeto que contiene opciones de configuración, incluyendo la plantilla (HTML), la lógica (JavaScript) y los estilos (CSS) asociados. Puedes definir componentes de una sola instancia (usando `Vue.component`) o componentes de instancia múltiple (utilizando el sistema de componentes de Vue, como `components` en la instancia Vue).

2. Plantilla: La plantilla de un componente define la estructura HTML y la presentación visual del componente. Puedes usar la sintaxis de plantilla de Vue.js para incluir directivas, interpolaciones y enlace de datos en la plantilla.

3. Lógica: La lógica de un componente se implementa en JavaScript. Esto incluye la definición de datos, métodos, propiedades computadas, eventos y ciclo de vida del componente. La lógica define cómo se comporta y responde el componente a las interacciones del usuario y los cambios en los datos.

4. Estilos: Los estilos asociados a un componente se pueden incluir directamente en el componente utilizando CSS o preprocesadores como SASS o LESS. Los estilos se aplican solo al ámbito del componente, lo que evita colisiones de estilos en toda la aplicación.

5. Reactividad: Los componentes en Vue.js son reactivos, lo que significa que se actualizan automáticamente cuando los datos cambian. Vue.js se encarga de mantener la interfaz de usuario sincronizada con el estado de los datos.

6. Props: Los props (propiedades) son atributos personalizados que se pueden pasar a un componente desde su componente padre. Los props permiten la comunicación entre componentes y la reutilización de componentes con diferentes datos.

7. Eventos: Los componentes pueden emitir eventos personalizados para notificar a sus componentes padres sobre acciones o cambios importantes. Los eventos permiten la comunicación ascendente de datos.

8. Ciclo de Vida: Los componentes en Vue.js tienen un ciclo de vida que consta de una serie de ganchos (hooks) que se ejecutan en momentos específicos durante la creación, actualización y destrucción del componente. Estos ganchos permiten realizar acciones personalizadas en cada etapa del ciclo de vida.

9. Registro y Uso: Los componentes deben registrarse antes de que puedan ser utilizados. Pueden ser registrados de manera global o local en una instancia Vue. Luego, los componentes se pueden utilizar en la plantilla de otros componentes.

Los componentes son una parte fundamental de la arquitectura de aplicaciones Vue.js y facilitan la construcción de aplicaciones web modulares y mantenibles. Al dividir la interfaz de usuario en componentes más pequeños y reutilizables, Vue.js promueve buenas prácticas de desarrollo y mejora la organización del código.

Options API

El Options API es la forma tradicional de definir componentes en Vue.js. En el Options API, los componentes se definen como objetos que contienen diversas opciones, como `data`, `methods`, `computed`, `props`, y más. Cada opción define una parte específica del componente, lo que facilita la organización y estructuración del código. El Options API es especialmente adecuado para proyectos pequeños o medianos y es fácil de entender para desarrolladores que están familiarizados con Vue.js.

Estructura

```
export default {
      name: ‘ComponentName’,
      props: { … },     //propiedades
      data: { … },      //reactividad, variables y logica
      computed: { … },  //propiedades computadas
      methods: { … },   //funciones
      mounted: { … },   //hooks, ciclo de vida
}
```

Composition API

La Composition API es una nueva forma de definir componentes introducida en Vue.js 3. A diferencia del Options API, que divide las opciones del componente en secciones, la Composition API permite definir la lógica del componente en términos de funciones reutilizables. Esto se logra mediante el uso de funciones `setup()` que devuelven un objeto de opciones que describe el estado y el comportamiento del componente. La Composition API es más flexible y escalable, lo que la hace adecuada para proyectos grandes y complejos, así como para compartir lógica entre componentes.

Estructura

```
Export default {
      name: ‘ComponentName’,
      props: { … }, //propiedades
      setup() {
            // init logic, licecycle hooks, etc.
            return {
                  //Data, methods, computed, etc.
            }
      }
}
```

Diferencias entre el Options API y el Composition API

1. Estructura: La principal diferencia es la estructura. El Options API organiza las opciones del componente en un objeto, mientras que la Composition API utiliza funciones `setup()` que devuelven un objeto de opciones.

2. Reactividad: En el Options API, la reactividad se declara utilizando el objeto `data`. En la Composition API, la reactividad se logra mediante el uso de la función `ref()` o `reactive()` y declarando el elemento.

3. Reutilización de Lógica: La Composition API facilita la reutilización de lógica entre componentes mediante la creación de funciones reutilizables, lo que la hace más modular y escalable.

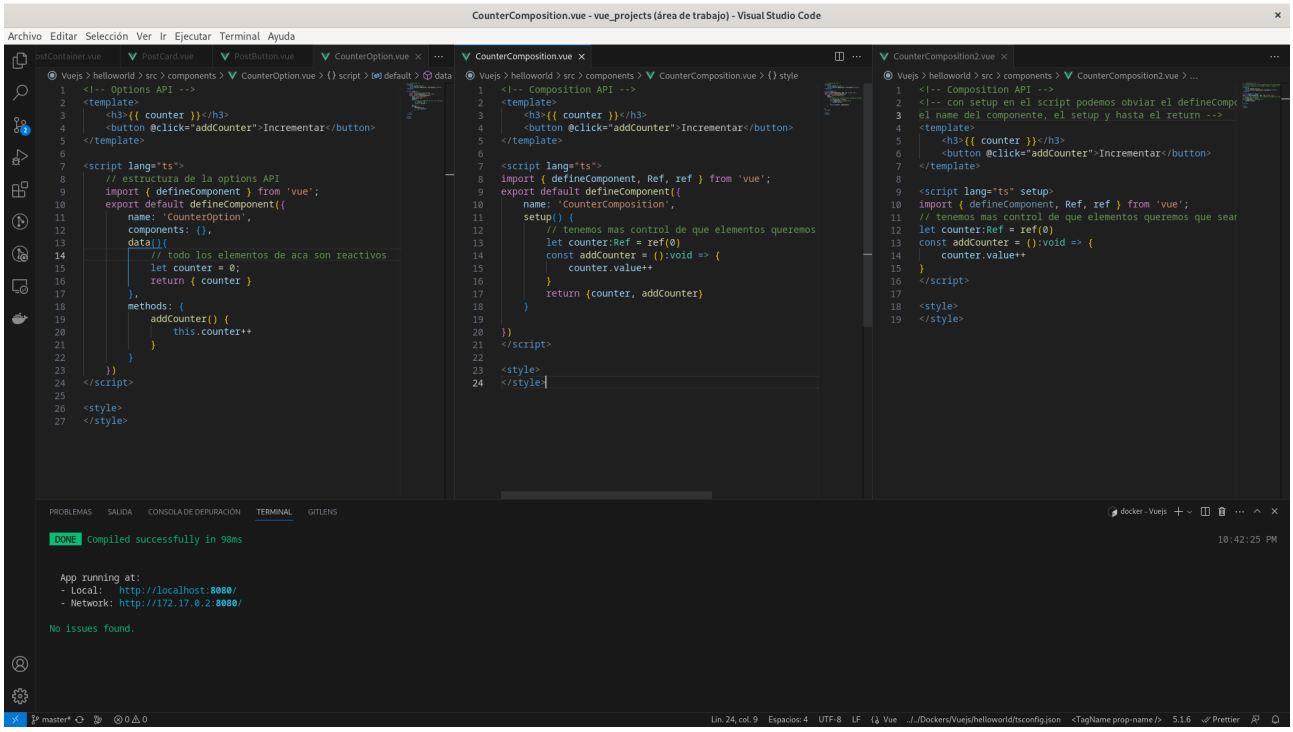
4. Legibilidad: El Options API es más legible para desarrolladores que están familiarizados con Vue.js desde versiones anteriores, ya que sigue una estructura más clara y tradicional. La Composition API puede ser más poderosa, pero puede ser menos intuitiva para los recién llegados.

5. Tamaño de Bundle: En proyectos grandes, la Composition API puede generar un tamaño de bundle más pequeño debido a su capacidad para importar solo las funciones necesarias.

6. Migración: La Composition API permite una migración gradual desde el Options API, lo que significa que puedes utilizar ambas en el mismo proyecto.

La elección entre el Options API y la Composition API depende de las necesidades de tu proyecto y de tu preferencia personal. Para proyectos más pequeños o equipos que ya están familiarizados con el Options API, es posible que el Options API siga siendo una opción viable. Para proyectos grandes o complejos, la Composition API proporciona ventajas en términos de modularidad y escalabilidad. Además, Vue.js permite la coexistencia de ambos en un mismo proyecto para una migración gradual.

Diferencias en código



CounterOption.vue

```
<!-- Options API -->
<template>
<h3>{{ counter }}</h3>
<button @click="addCounter">Incrementar</button>
```

```
</template>

<script lang="ts">
// estructura de la options API
import { defineComponent } from 'vue';
export default defineComponent({
name: 'CounterOption',
components: {},
data(){
// todo los elementos de aca son reactivos
let counter = 0;
return { counter }
},
methods: {
addCounter() {
this.counter++
}
}
})
</script>

<style>
</style>
```

CounterComposition.vue

```
<!-- Composition API -->
<template>
    <h3>{{ counter }}</h3>
    <button @click="addCounter">Incrementar</button>
</template>

<script lang="ts">
import { defineComponent, Ref, ref } from 'vue';
export default defineComponent({
    name: 'CounterComposition',
    setup() {
        // tenemos mas control de que elementos queremos que sean reactivos,
ref()
        let counter:Ref = ref(0)
        const addCounter = ():void => {
            counter.value++
        }
        return {counter, addCounter}
    }

})
</script>

<style>
</style>
```

Exportando el defineComponent, agregando el name del componente, agregar el setup() y el return.

Setup

CounterComposition2.vue

```
<!-- Composition API setup como parametro -->
<template>
```

```
    <h3>{{ counter }}</h3>
    <button @click="addCounter">Incrementar * 2</button>
</template>

<script lang="ts" setup>
import { defineComponent, Ref, ref } from 'vue';
// tenemos mas control de que elementos queremos que sean reactivos, ref()
let counter:Ref = ref(0)
const addCounter = ():void => {
    counter.value+=2
}
</script>

<style>
</style>
```

Diferencias notables

1. Ubicación de `setup`:

- En el primer código, la función `setup()` se coloca dentro del objeto `defineComponent`. Es la forma tradicional de utilizar `setup` y es adecuada para componentes más complejos donde necesitas exponer propiedades y métodos de manera más organizada.

- En el segundo código, se utiliza `<script lang="ts" setup>`, lo que significa que `setup` se utiliza directamente en el ámbito del componente sin necesidad de definir un objeto `defineComponent`. Esto hace que el código sea más conciso y es adecuado para componentes más simples o pequeños.

2. Exportación del Componente:

- En el primer código, el componente se exporta utilizando `export default defineComponent(...)`, lo que es común en el Options API y la Composition API cuando se utiliza `setup`.

- En el segundo código, no es necesario exportar el componente de esta manera, ya que `setup` se utiliza directamente en el ámbito del componente. No hay una exportación explícita del componente.

3. Acceso a las Propiedades y Métodos:

- En ambos casos, se define una variable `counter` reactiva y una función `addCounter` que incrementa el valor de `counter`.

En resumen, ambas implementaciones son válidas y utilizan la Composition API. La principal diferencia radica en cómo se estructura el componente y cómo se utiliza `setup`. El primer código sigue una estructura más tradicional, mientras que el segundo código es más conciso y adecuado para componentes más simples. Además, el comportamiento de `addCounter` es diferente en ambos códigos. La elección entre uno u otro dependerá de tus necesidades y preferencias de codificación.

¿Que son las directivas?

En Vue.js, las directivas son atributos especiales que comienzan con el prefijo "v-" y se utilizan para aplicar comportamientos dinámicos a los elementos HTML y controlar la manipulación del DOM (Documento Objeto Modelo) de manera declarativa. Las directivas son una parte fundamental de Vue.js y permiten enlazar datos del modelo de Vue con la vista, manipular el DOM de manera reactiva y aplicar lógica condicional y repetitiva en la interfaz de usuario. Algunas de las directivas más comunes en Vue.js incluyen:

1. v-bind o (:): Se utiliza para enlazar atributos HTML a expresiones de Vue, lo que permite la interpolación de datos en los atributos HTML. Por ejemplo, :src="imagenURL" enlaza el atributo src de una imagen con la variable imagenURL del modelo de Vue.

2. v-model: Se utiliza para establecer una comunicación bidireccional entre un elemento de formulario HTML (como input, textarea o select) y una variable del modelo de Vue. Permite que los cambios en el elemento del formulario actualicen automáticamente el modelo y viceversa.

3. v-if, v-else, v-else-if: Estas directivas se utilizan para controlar la visibilidad de elementos HTML en función de condiciones lógicas. Por ejemplo, v-if="mostrarElemento" mostrará un elemento si la variable mostrarElemento es verdadera.

4. v-for: Se utiliza para realizar bucles y repetir elementos HTML en función de una matriz o un objeto en el modelo de Vue. Por ejemplo, v-for="(item, index) in listaDeItems" repetirá elementos HTML para cada elemento en listaDeItems.

5. v-on o @: Se utiliza para escuchar eventos DOM y ejecutar métodos o expresiones de Vue cuando ocurren esos eventos. Por ejemplo, @click="hacerAlgo" ejecutará la función hacerAlgo cuando se haga clic en un elemento.

6. v-show: Similar a v-if, se utiliza para controlar la visibilidad de elementos HTML, pero a diferencia de v-if, los elementos siempre están presentes en el DOM, solo que se ocultan o muestran mediante CSS.

7. v-pre y v-cloak: Se utilizan para optimizaciones y evitar que Vue compile o interprete ciertos elementos o partes del DOM. v-pre evita la compilación de un elemento y sus descendientes, mientras que v-cloak se utiliza para ocultar temporalmente elementos hasta que Vue termine de compilar.

8. v-once: Indica que un elemento y su contenido deben representarse solo una vez y no se actualizarán en futuras actualizaciones del modelo.

Estas son algunas de las directivas más comunes en Vue.js, pero Vue ofrece muchas otras para realizar diversas operaciones en la vista y controlar la interacción entre la interfaz de usuario y el modelo de datos. Las directivas son una parte esencial de la filosofía de Vue de hacer que el desarrollo de aplicaciones web sea más declarativo y fácil de entender.

Ejemplos de directivas

Vue.js utiliza directivas para agregar funcionalidad y comportamiento a elementos HTML en la vista. Aquí tienes una breve descripción de algunas de las directivas más comunes en Vue.js junto con ejemplos de código:

1. v-model: Esta directiva permite enlazar bidireccionalmente datos de entrada del usuario a una propiedad en el modelo de datos. Es comúnmente utilizada con campos de entrada como `<input>`, `<textarea>` y `<select>`.

```
<input v-model="mensaje">
<p>{{ mensaje }}</p>
```

2. v-bind: Esta directiva se utiliza para enlazar atributos o propiedades de un elemento HTML a una expresión o propiedad en el modelo de datos, se abrevia con el (:).

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

3. v-for: Permite iterar sobre una lista y renderizar elementos HTML repetidamente. Se usa para mostrar una lista de elementos en la vista.

```
<ul>
  <li v-for="item in lista">{{ item }}</li>
</ul>
```

4. v-if / v-else-if / v-else: Estas directivas se utilizan para mostrar u ocultar elementos en función de una expresión condicional.

```
<div v-if="mostrar">Este elemento se muestra si mostrar es verdadero</div>
<div v-else>Este elemento se muestra si mostrar es falso</div>
```

5. v-on: Esta directiva se utiliza para escuchar eventos en elementos HTML y ejecutar métodos o expresiones en respuesta a esos eventos, se abrevia con el (@).

```
<button v-on:click="hacerAlgo">Haz algo</button>
<button @click="hacerAlgo">Haz algo</button>
```

6. v-show: Similar a `v-if`, pero en lugar de agregar o quitar elementos del DOM, simplemente cambia la propiedad CSS `display` para mostrar u ocultar un elemento.

```
<div v-show="mostrar">Este elemento se muestra si mostrar es verdadero</div>
```

7. v-pre: Esta directiva indica a Vue.js que ignore completamente el elemento y sus descendientes durante la compilación. Útil para elementos estáticos.

```
<div v-pre>{{ mensaje }}</div>
```

8. v-cloak: Esta directiva se usa junto con CSS para ocultar elementos antes de que Vue.js compile la plantilla. Se utiliza principalmente para evitar que los usuarios vean la plantilla sin compilar.

```
<div v-cloak>{{ mensaje }}</div>
```

9. v-once: Esta directiva hace que un elemento y todos sus descendientes se rendericen una sola vez y luego se queden inmutables.

```
<p v-once>{{ mensaje }}</p>
```

10. v-html: Esta directiva permite renderizar HTML dinámico desde una propiedad en el modelo de datos. Se debe usar con precaución para evitar ataques de inyección de código.

```
<div v-html="contenidoHTML"></div>
```

Estas son algunas de las directivas más comunes en Vue.js. Cada una de ellas se utiliza para controlar aspectos específicos de la interfaz de usuario y el comportamiento de la vista en una aplicación Vue.js.

Ciclo de vida de un componente

El ciclo de vida en Vue.js se refiere a una serie de etapas predefinidas que un componente Vue pasa a lo largo de su existencia. Estas etapas permiten a los desarrolladores controlar y gestionar el comportamiento del componente en momentos específicos, como la creación, actualización y destrucción.

El ciclo de vida en Vue.js se puede dividir en tres fases principales:

1. Creación (Creation): En esta fase, se inicializa el componente y se establecen las relaciones con otros componentes. Durante esta etapa, se ejecutan ganchos de ciclo de vida como `beforeCreate` y `created`.

- `beforeCreate`: Se ejecuta antes de la creación del componente.
- `created`: Se ejecuta después de la creación del componente.

2. Actualización (Update): Durante esta fase, el componente se actualiza en respuesta a cambios en los datos o propiedades. Aquí se ejecutan ganchos de ciclo de vida como `beforeUpdate` y `updated`.

- `beforeMount`: Se ejecuta antes de que el componente se monte en el DOM.
- `mounted`: Se ejecuta después de que el componente se monte en el DOM.
- `beforeUpdate`: Se ejecuta antes de que los datos reactivos se actualicen en el componente.
- `updated`: Se ejecuta después de que los datos reactivos se actualicen en el componente.

3. Destrucción (Destruction): En esta fase, el componente se destruye y se libera de la memoria. Los ganchos de ciclo de vida como `beforeUnmount` y `unmounted` se utilizan para realizar limpieza antes de que el componente desaparezca por completo.

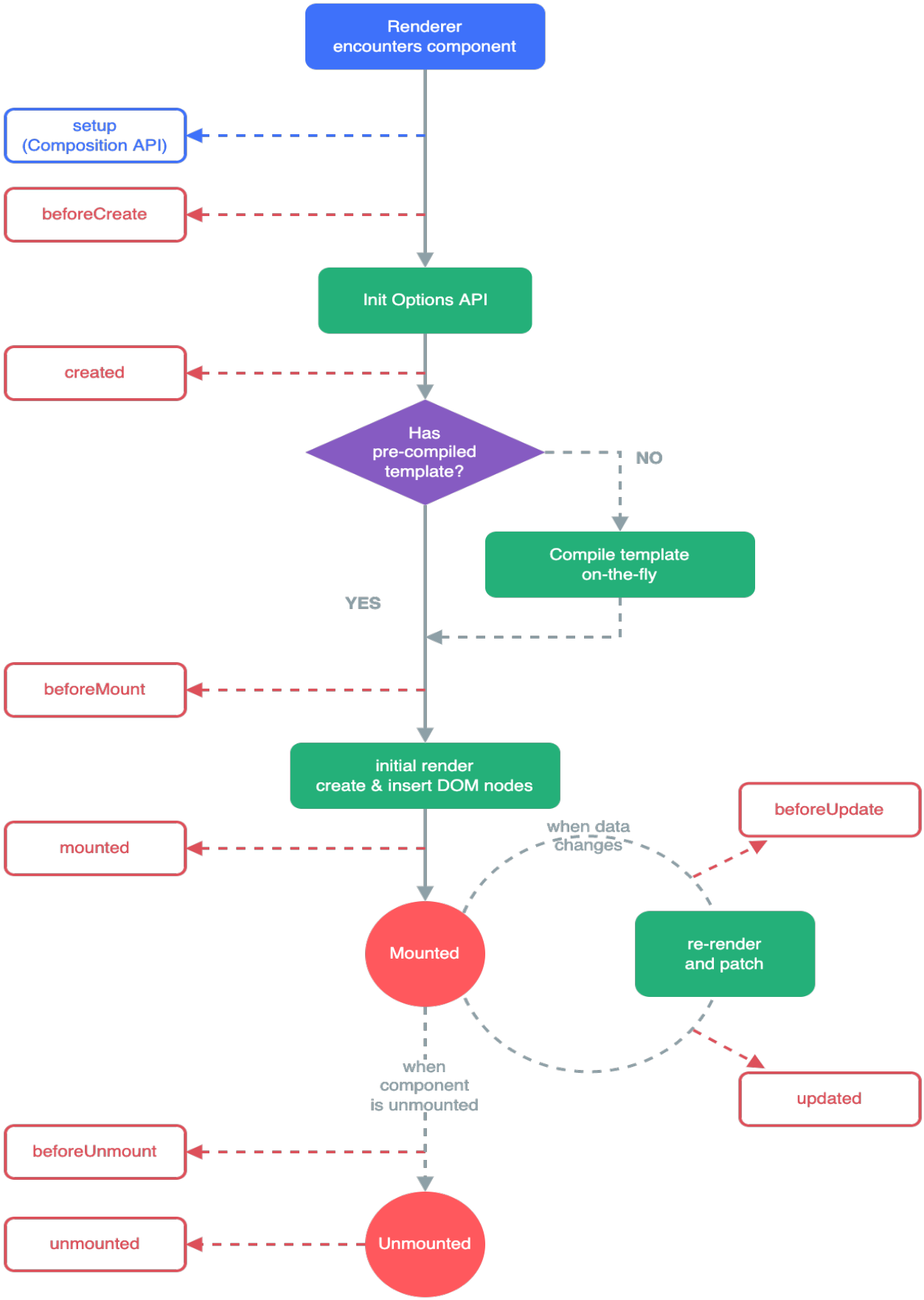
- `beforeUnmount`: Se ejecuta antes de que el componente se desmonte del DOM. (antes llamado `beforeDestroy` en Vue.js 2)

- `unmounted`: Se ejecuta después de que el componente se desmonte del DOM. (antes llamado `destroyed` en Vue.js 2)

Estos ganchos de ciclo de vida brindan oportunidades para realizar tareas específicas en momentos específicos, como la inicialización de datos, la suscripción a eventos, la liberación de recursos y la interacción con componentes padres e hijos.

Es importante comprender el ciclo de vida de Vue.js para escribir componentes eficientes y gestionar adecuadamente el estado y los recursos. El conocimiento de estas etapas permite a los desarrolladores optimizar el rendimiento, depurar problemas y garantizar una experiencia de usuario fluida en aplicaciones Vue.js.

Diagrama del ciclo de vida



Los ciclos de vida de un componente en Vue.js son una serie de etapas que un componente pasa durante su ciclo de vida, desde su creación hasta su destrucción (unmounted). Cada etapa tiene un propósito específico y permite que nosotros como desarrolladores controlemos el comportamiento y la lógica del componente en momentos clave.

beforeCreate

En esta etapa, se crea la instancia del componente, pero aún no se ha inicializado. Los datos y eventos personalizados aún no están disponibles en esta etapa. Es útil para configuraciones iniciales y tareas que deben realizarse antes de que el componente esté listo.

created

Después de la etapa "beforeCreate", los datos y eventos personalizados se han configurado, pero el componente aún no se ha agregado al DOM. Es una buena etapa para realizar llamadas a API y configuraciones iniciales basadas en datos.

beforeMount

En esta etapa, el componente está a punto de ser insertado en el DOM, pero aún no ha sido renderizado. Es útil para realizar acciones antes de que el componente se visualice en la pantalla, como modificaciones finales en los datos.

mounted

Después de la etapa "beforeMount", el componente se ha agregado al DOM y ha sido renderizado. En esta etapa, es seguro interactuar con el DOM, realizar solicitudes de red y realizar tareas de inicialización que requieren la presencia de elementos en el DOM.

beforeUpdate

Cuando los datos del componente cambian, Vue.js ejecuta esta etapa antes de volver a renderizar el componente. Es útil para realizar tareas de limpieza, validaciones o ajustes antes de que se reflejen los cambios en la interfaz de usuario.

updated

Después de la etapa "beforeUpdate", el componente se ha vuelto a renderizar con los nuevos datos. Es un buen lugar para realizar acciones posteriores a la actualización en el DOM o ejecutar código después de que se hayan aplicado los cambios.

beforeUnmount

Esta etapa se activa cuando el componente está a punto de ser eliminado del DOM, pero aún no se ha eliminado. Es útil para realizar tareas de limpieza o liberación de recursos antes de que el componente se desmonte.

unmounted

Después de la etapa "beforeUnmount", el componente se ha desmontado y eliminado del DOM. En esta etapa, el componente ya no existe y no se pueden realizar interacciones con él.

Estas etapas del ciclo de vida proporcionan un control preciso sobre el comportamiento de un componente en diferentes momentos de su vida útil, lo que nos permite como desarrolladores implementar lógica personalizada según sea necesario.

Comunicación Vertical

La comunicación vertical en Vue.js 3 se refiere a la forma en que los componentes se comunican entre sí en la jerarquía de padres e hijos. Esto significa que los datos fluyen desde un componente padre a uno o varios componentes hijos. Aquí están las principales técnicas de comunicación vertical en Vue.js 3:

Descendente

La comunicación vertical descendente en Vue.js se refiere a la transferencia de datos desde un componente padre a sus componentes hijos en la jerarquía de la aplicación. Esta es la forma más común de comunicación en Vue.js, y se logra principalmente a través del uso de props y eventos personalizados.

props

Los props en Vue son propiedades personalizadas que se pueden pasar de un componente padre a un componente hijo. El componente padre define los props y los pasa al componente hijo como atributos en su instancia. El componente hijo utiliza estos props como datos de solo lectura. Los props son valores inmutables en el componente hijo, lo que significa que el componente hijo no puede modificar directamente los valores de los props; en cambio, los utiliza como datos de solo lectura. Los props son una parte fundamental de la comunicación vertical en Vue y son esenciales para la construcción de aplicaciones basadas en componentes reutilizables.

Ascendente

La comunicación vertical ascendente en Vue.js se refiere a la transferencia de datos o eventos desde un componente hijo a su componente padre en la jerarquía de la aplicación. A diferencia de la comunicación vertical descendente, donde los datos fluyen desde el padre hacia los hijos, en la comunicación vertical ascendente, los datos fluyen desde los hijos hacia el padre. Para lograr esto, se utilizan principalmente los eventos personalizados.

emit

Emitir en Vue es un mecanismo que permite a un componente hijo comunicarse con su componente padre. Utiliza la función ‘emit’ para enviar eventos personalizados desde el componente hijo al componente padre. Estos eventos personalizados pueden llevar datos y desencadenar funciones en

el componente padre, lo que facilita la interacción y la comunicación entre componentes en una aplicación Vue. El componente hijo emite un evento y el componente padre lo "escucha" y responde según sea necesario, lo que permite una comunicación efectiva y flexible entre componentes en la jerarquía de Vue.

El componente padre escucha estos eventos utilizando la directiva @ ó v-on, y puede responder a ellos ejecutando funciones en respuesta a los eventos emitidos por los hijos.

Propiedades Computadas

Las propiedades computadas (computed properties) en Vue.js son propiedades que se calculan automáticamente en función de una o varias propiedades de datos existentes en un componente Vue. Estas propiedades se almacenan en caché y se recalculan solo cuando alguna de las dependencias cambia. Las propiedades computadas son útiles para realizar cálculos basados en datos y mantener el código limpio y organizado.

Características clave de las propiedades computadas:

1. **Cálculos Reactivos:** Las propiedades computadas se actualizan automáticamente cuando sus dependencias cambian. Esto significa que si alguna de las propiedades de datos en las que se basa una propiedad computada cambia, la propiedad computada se recalcula automáticamente.
2. **Cacheo Automático:** Vue.js almacena en caché el resultado de una propiedad computada y solo la recalcula cuando alguna de las dependencias cambia. Esto mejora el rendimiento, especialmente en propiedades computadas costosas en términos de cálculos.
3. **Sintaxis Simple:** Definir una propiedad computada es simple y se asemeja a la definición de una propiedad de datos en un componente. Se utiliza la palabra clave `computed` seguida de una función que devuelve el valor calculado.
4. **Acceso como Propiedad:** Las propiedades computadas se utilizan en las plantillas de la misma manera que las propiedades de datos. Esto facilita su uso en la interfaz de usuario.

Las propiedades computadas se pueden utilizar en diversas situaciones, incluidas:

- **Filtros y Transformaciones de Datos:** Para realizar transformaciones en los datos antes de mostrarlos en la interfaz de usuario. Por ejemplo, formatear una fecha o aplicar filtros a una lista.

- **Cálculos Derivados:** Para calcular valores derivados o resúmenes basados en datos existentes. Por ejemplo, calcular el total de una lista de compras o determinar si ciertas condiciones se cumplen.

- **Validaciones Dinámicas:** Para realizar validaciones dinámicas en función de datos del usuario. Por ejemplo, habilitar o deshabilitar un botón según ciertas condiciones.
- **Ordenación Personalizada:** Para ordenar listas de elementos de acuerdo con reglas personalizadas.
- **Operaciones Matemáticas:** Para realizar operaciones matemáticas complejas en los datos y mostrar los resultados en la interfaz de usuario.

En resumen, las propiedades computadas son una poderosa característica de Vue.js que permite realizar cálculos reactivos de manera eficiente y mantener un código más limpio y comprensible al separar la lógica de presentación de los cálculos subyacentes.

```
<!-- example -->
<template>
    <div>
        <label for="">SubTotal</label>
        <input type="number" v-model="subTotal">
        <h3>Total de impuestos a pagar: {{ totalVat }} ({{ vat }}%)</h3>
    </div>
</template>

<script lang="ts" setup>
import { Ref, ref, computed} from 'vue'

    const vat = 21
    let subTotal:Ref<number> = ref(0)
    const totalVat:Ref<number> = computed(() =>
        (vat*subTotal.value) / 100
    )

    /*
        1000---------subtotal
        21-----------x

        x = (21 * subtotal) / 100
    */
</script>
```

Directivas Personalizadas

En Vue 3, puedes crear directivas personalizadas para extender la funcionalidad de Vue.js según tus necesidades específicas. Las directivas personalizadas te permiten interactuar directamente con el DOM, modificar elementos, escuchar eventos y aplicar comportamientos personalizados a elementos HTML. Aquí hay una descripción general de cómo puedes crear una directiva personalizada en Vue 3:

1. Definición de la Directiva Personalizada:

Para crear una directiva personalizada, puedes usar el método `app.directive()` proporcionado por la instancia de Vue. Por ejemplo, en el archivo principal de tu aplicación Vue, puedes definir una directiva personalizada de la siguiente manera:

```
app.directive('mi-directiva', {
  // Configuración de la directiva
});
```

2. Configuración de la Directiva:

La configuración de la directiva es un objeto que contiene varios ganchos (hooks) que te permiten controlar el comportamiento de la directiva. Algunos de los ganchos comunes son:

- bind: Se ejecuta una vez cuando la directiva se adjunta al elemento.
- inserted: Se ejecuta una vez cuando el elemento se inserta en el DOM.
- update: Se ejecuta cuando el valor de la expresión vinculada cambia.
- componentUpdated: Se ejecuta después de que el componente y sus hijos se actualizan.
- unbind: Se ejecuta una vez cuando la directiva se elimina del elemento.

Aquí tienes un ejemplo básico de una directiva personalizada que cambia el fondo de un elemento a rojo cuando se le aplica:

```
app.directive('fondo-rojo', {
  // Se ejecuta una vez cuando la directiva se adjunta al elemento.
  mounted(el) {
    el.style.backgroundColor = 'red';
  }
});
```

3. Uso de la Directiva en la Plantilla:

Después de definir la directiva personalizada, puedes usarla en las plantillas de tus componentes Vue utilizando la notación `v-` seguida del nombre de la directiva. Por ejemplo:

```
<div v-fondo-rojo>
```

```
    Este div tendrá un fondo rojo.
  </div>
```

En este caso, cuando se renderiza el componente, la directiva `v-fondo-rojo` se aplicará al elemento `div`, y su fondo se establecerá en rojo según la configuración de la directiva.

4. Parámetros y Modificadores:

Puedes definir parámetros y modificadores personalizados para tus directivas, lo que permite una mayor flexibilidad y personalización en su comportamiento. Por ejemplo:

```
  app.directive('mi-directiva', {
    mounted(el, binding) {
      // Acceder a los parámetros y modificadores
      const { value, arg, modifiers } = binding;
      // Hacer algo con los valores
    }
  });
```

Luego, puedes usar la directiva en la plantilla con parámetros y modificadores:

```
  <div v-mi-directiva:parametro.modificador>
    <!-- Contenido -->
  </div>
```

Los parámetros y modificadores se pueden acceder en la función `mounted` de la directiva personalizada.

En el contexto de una directiva personalizada en Vue.js, `el` se refiere al elemento DOM al que se aplica la directiva, mientras que `binding` es un objeto que contiene información adicional sobre la directiva y su uso en la plantilla. Aquí hay una explicación más detallada de cada uno:

1. `el` (Elemento DOM): Este es el primer parámetro en el gancho `mounted` de una directiva personalizada. Representa el elemento HTML al cual se aplica la directiva. Puedes acceder y manipular las propiedades y el contenido del elemento utilizando la variable `el`. Por ejemplo, puedes cambiar su estilo, agregar clases, modificar atributos, etc.

Ejemplo:

```
  mounted(el) {
    el.style.backgroundColor = 'red'; // Cambia el fondo del elemento a rojo
  }
```

2. `binding` (Objeto de Vinculación): Este es el segundo parámetro en el gancho `mounted` de una directiva personalizada. Es un objeto que contiene información sobre cómo se utiliza la directiva en la plantilla y proporciona acceso a varios valores útiles. Algunas de las propiedades más comunes de `binding` incluyen:

- `value`: Representa el valor de la expresión que se pasa a la directiva. Es útil cuando se desea obtener el valor de una expresión vinculada a la directiva.

- `arg`: Representa el argumento de la directiva, que es un valor opcional que se puede proporcionar cuando se utiliza la directiva en la plantilla.

- `modifiers`: Es un objeto que contiene modificadores que se aplican a la directiva. Los modificadores se especifican en la plantilla después de un punto (`.`) cuando se usa la directiva.

Ejemplo:

```
  mounted(el, binding) {
    const valorExpresion = binding.value; // Obtener el valor de la expresión
vinculada
    const argumento = binding.arg; // Obtener el argumento de la directiva
    const modificadores = binding.modifiers; // Obtener los modificadores
  }
```

Ejemplo de uso en la plantilla:

```
  <div v-mi-directiva:argumento.modificador="expresion">Contenido</div>
```

En resumen, `el` te permite acceder al elemento al que se aplica la directiva para realizar manipulaciones en el DOM, mientras que `binding` te proporciona información adicional sobre cómo se utiliza la directiva en la plantilla, incluyendo el valor de la expresión, el argumento y los modificadores. Esto te permite personalizar el comportamiento de la directiva en función de estos datos.

Estos son los conceptos básicos para crear directivas personalizadas en Vue 3. Puedes utilizar estas directivas para extender la funcionalidad de tus componentes Vue y manipular el DOM de acuerdo con las necesidades específicas.

Nota:

Las directivas personalizadas en Vue.js no se limitan únicamente a trabajar con CSS. De hecho, las directivas personalizadas son una poderosa característica que te permite agregar comportamientos personalizados a elementos HTML en tus plantillas Vue. Aunque es común usar directivas

personalizadas para manipular el estilo CSS de los elementos, también se pueden utilizar para realizar una amplia variedad de tareas, incluyendo:

1. Manipulación del DOM: Puedes usar directivas personalizadas para agregar, eliminar o modificar elementos y atributos del DOM.

2. Gestión de Eventos: Puedes crear directivas personalizadas que manejen eventos de elementos y realicen acciones específicas en respuesta a esos eventos.

3. Integración con bibliotecas externas: Si estás trabajando con bibliotecas o frameworks externos que no son directamente compatibles con Vue.js, puedes usar directivas personalizadas para integrar esas funcionalidades de manera más natural en tus componentes Vue.

4. Validación de Datos: Puedes crear directivas personalizadas para validar datos de entrada de usuarios y proporcionar retroalimentación visual en función de esa validación.

5. Control de Visibilidad: Puedes usar directivas personalizadas para mostrar u ocultar elementos en función de ciertas condiciones o lógica.

Las directivas personalizadas son una forma versátil de extender las capacidades de Vue.js para adaptarse a tus necesidades específicas, y no se limitan únicamente al trabajo con estilos CSS. Puedes utilizarlas para agregar comportamientos personalizados y manipular el DOM, gestionar eventos, validar datos, integrar con bibliotecas externas y mucho más.

Pre Procesadores CSS

Los preprocesadores CSS son herramientas que extienden la funcionalidad de CSS estándar, permitiéndote utilizar características adicionales como variables, anidamiento, mixins y más. A continuación, te menciono algunos de los preprocesadores CSS más populares que puedes utilizar en proyectos Vue.js:

1. **Sass (Syntactically Awesome Style Sheets):** Sass es uno de los preprocesadores CSS más utilizados y ampliamente adoptados. Permite el uso de variables, anidamiento de selectores, mixins y funciones, lo que facilita la escritura y el mantenimiento de estilos complejos. Puedes usar la sintaxis `.scss` o `.sass` con Vue.js.

2. **Less:** Less es otro preprocesador CSS que también es bastante popular. Ofrece características similares a Sass, como variables, anidamiento y mixins. La sintaxis de Less utiliza la extensión `.less`.
3. **Stylus:** Stylus es un preprocesador CSS que se destaca por su sintaxis minimalista y su facilidad de escritura. Utiliza una sintaxis similar a la de Python, lo que puede resultar en archivos de estilo más limpios y legibles.

Para utilizar un preprocesador CSS en un proyecto Vue.js, debes configurar el entorno de desarrollo adecuado. Por lo general, esto implica instalar una dependencia relacionada con el preprocesador y configurar tu entorno de desarrollo para compilar el código del preprocesador en CSS estándar que pueda ser interpretado por el navegador.

Por ejemplo, si estás utilizando Vue CLI para crear tu proyecto Vue.js, puedes agregar soporte para preprocesadores CSS durante la configuración inicial del proyecto. Vue CLI admite Sass, Less y Stylus, lo que facilita la integración de tu preprocesador CSS preferido.

Una vez configurado, puedes escribir tus estilos utilizando la sintaxis del preprocesador elegido en los archivos `.vue` de tus componentes Vue. El entorno de desarrollo se encargará de compilar automáticamente el código del preprocesador en CSS válido cuando construyas tu aplicación.

Recuerda que la elección del preprocesador CSS depende de tus preferencias personales y las necesidades de tu proyecto. Cualquiera de las opciones mencionadas (Sass, Less o Stylus) es una elección sólida y ampliamente aceptada en la comunidad de desarrollo web.

Instalación

Si es un proyecto desde cero, como se mencionó anteriormente se puede especificar desde el inicio, pero en el caso de un proyecto existente podemos hacer la instalación con NPM

```
npm install -D sass-loader sass
npm install -D less-loader less
npm install -D stylus-loader stylus
```

En el caso del componente se define de la siguiente manera

<style scoped lang=”scss”>

Configuración

Este sera el directorio donde agregaremos nuestros ficheros, con la extención especifica según lo requiera nuestro proyecto.

```
./src/scss/_variables.scss en el caso de Sass
./src/less/_variables.scss en el caso de Lees
```

El archivo `vue.config.js` es un archivo de configuración utilizado en proyectos Vue.js que se crean con Vue CLI (Command Line Interface) o proyectos personalizados basados en Vue.js. Este archivo permite personalizar la configuración de tu proyecto Vue de diversas maneras. A continuación, te explico algunas de las cosas que puedes hacer con `vue.config.js`:

1. Personalizar la configuración del Webpack: Puedes utilizar `vue.config.js` para modificar la configuración de Webpack, que es la herramienta de construcción que se utiliza en proyectos Vue.js para compilar y empaquetar el código. Esto te permite ajustar parámetros de construcción, como configurar alias de rutas, agregar reglas personalizadas para procesar archivos y realizar muchas otras personalizaciones avanzadas.

2. Configurar el servidor de desarrollo: Puedes configurar el servidor de desarrollo que se utiliza para ejecutar tu aplicación Vue durante el desarrollo. Esto incluye la especificación del puerto, la habilitación de HTTPS y la configuración de proxy para redirigir solicitudes a otros servidores.

3. Personalizar el proceso de empaquetado: Puedes modificar la forma en que se empaqueta tu aplicación para producción. Esto incluye la configuración de opciones de minificación, la generación de nombres de archivos, la inclusión de metadatos y otros ajustes relacionados con la construcción de la aplicación para producción.

4. Definir variables de entorno: Puedes definir variables de entorno específicas de tu proyecto en el archivo `vue.config.js`. Esto te permite establecer valores predeterminados para variables de entorno que se pueden utilizar en tu aplicación, lo que facilita la gestión de diferentes configuraciones para entornos de desarrollo, prueba y producción.

5. Personalizar el título de la página: Puedes establecer el título de la página HTML generada por tu aplicación Vue a través de la propiedad `chainWebpack`. Esto es útil para asegurarte de que cada página tenga un título único y significativo.

6. Habilitar la compatibilidad con navegadores antiguos: Puedes configurar `vue.config.js` para que tu aplicación sea compatible con navegadores antiguos mediante el uso de herramientas como Babel y polyfills.

7. Personalizar la ruta de salida: Puedes especificar la ruta de salida de los archivos generados al construir tu aplicación. Esto es útil si deseas que los archivos se generen en un directorio específico en lugar del directorio predeterminado.

Para utilizar `vue.config.js`, simplemente debes crear un archivo con ese nombre en la raíz de tu proyecto Vue y definir tus configuraciones personalizadas en él. Cuando ejecutas comandos como `vue-cli-service build` o `vue-cli-service serve`, Vue CLI leerá automáticamente este archivo y aplicará las configuraciones personalizadas que hayas definido.

En resumen, `vue.config.js` es una herramienta poderosa para personalizar y configurar tu proyecto Vue.js de acuerdo con tus necesidades específicas de desarrollo y producción.

Importación global

Vue.config.js

```
const { defineConfig } = require('@vue/cli-service')
module.exports = defineConfig({
  transpileDependencies: true,
  css: {
    loaderOptions: {
      sass: {
        additionalData: `
          @import "@/scss/_variables.scss";
        `
      }
    }
  }
})
```

De esta manera le decimos a vue que considere el preprocesador css que le indiquemos y sus ficheros.

Router

El enrutamiento en Vue 3 se logra comúnmente utilizando una biblioteca llamada Vue Router. Vue Router es una biblioteca oficial que permite la navegación basada en componentes en una aplicación Vue.js. Te permite definir rutas y vincularlas a componentes específicos, lo que facilita la creación de aplicaciones de una sola página (SPA) y la navegación entre vistas de manera declarativa.

A continuación, se muestra una guía básica para configurar y utilizar Vue Router en una aplicación Vue 3:

1. Instalación de Vue Router:

Asegúrate de que tienes Vue.js y Vue Router instalados en tu proyecto. Si no los has instalado previamente, puedes hacerlo mediante npm o yarn:

```
npm install vue@next
npm install vue-router@4
```

o

```
yarn add vue@next
yarn add vue-router@4
```

2. Configuración de Vue Router:

En tu archivo principal de la aplicación (por lo general, main.js ó ./src/router/index,js), configura Vue Router e instálalo en la instancia de Vue:

en el caso de hacerlo modular, separado de main.js, en este se debe importar:

./src/main.js

```
import router from ‘./router’
```

./src/router/index.js

```
import { createRouter, createWebHistory, RouteRecordRaw } from 'vue-router';
import HomeView from './views/Home.vue';
import AboutView from './views/About.vue';

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

const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes
});

export default router
```

3. Creación de componentes de vista:

Crea los componentes que se utilizarán como vistas en tu aplicación. En el ejemplo anterior, se crean los componentes `Home` y `About`. Estos se crean en el directorio ./src/views/AboutView.vue

4. Uso de rutas en componentes:

Para usar rutas en tus componentes, puedes utilizar la directiva `router-link` para crear enlaces y el componente `router-view` para mostrar las vistas correspondientes. Por ejemplo:

```
<template>
  <div>
    <router-link to="/">Home</router-link>
    <router-link to="/about">About</router-link>
    <router-view/>
  </div>
</template>
```

Nota: Es buena practica hacer uso en los componentes del name de la ruta en el router-link el mismo que definimos en el fichero de configuración de rutas, de esta manera es mas fácil y rápido en caso de cambiar la dirección de la ruta.

```
<template>
  <div>
    <router-link to="/">Home</router-link>
    <router-link :to="{name: ‘about’}">About</router-link>
    <router-view/>
  </div>
</template>
```

5. Navegación programática:

Puedes realizar la navegación programáticamente en tus componentes utilizando el objeto `$router`. Por ejemplo:

```
// Navegar a la página de About
this.$router.push('/about');

// Navegar hacia atrás
this.$router.back();
```

6. Rutas con parámetros:

Vue Router también admite rutas con parámetros dinámicos. Puedes definir rutas con parámetros en tu configuración de rutas y acceder a esos parámetros desde tus componentes.

```
const routes = [
  { path: '/user/:id', component: UserProfile },
];
```

Luego, puedes acceder a `this.$route.params.id` en tu componente para obtener el valor del parámetro.

Este es un resumen básico de cómo configurar y usar Vue Router en una aplicación Vue 3. Vue Router ofrece muchas más características avanzadas, como rutas anidadas, rutas con nombres, rutas con guardias de navegación y más. Puedes consultar la documentación oficial de Vue Router https://router.vuejs.org/ para obtener información detallada y ejemplos adicionales.

Instalación de vue-router

La instalación de `vue-router` en un proyecto nuevo y en uno existente se realiza de manera similar, pero los pasos específicos pueden variar ligeramente según tu entorno y configuración. A continuación, te mostraré cómo instalar `vue-router` en ambos casos:

Instalación en un Proyecto Nuevo

Si estás creando un proyecto Vue.js desde cero, puedes seguir estos pasos para instalar `vue-router`:

1. Crear un Proyecto Vue.js:

Utiliza Vue CLI para crear un nuevo proyecto Vue.js si aún no lo has hecho. Si aún no tienes Vue CLI instalado, puedes instalarlo globalmente utilizando el siguiente comando:

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

Luego, crea un nuevo proyecto Vue.js:

```bash
vue create my-vue-project
```

Sigue las instrucciones para configurar tu proyecto.

2. **Instalar Vue Router:**

   Una vez que tengas tu proyecto Vue.js configurado, ingresa al directorio del proyecto:

   ```bash
   cd my-vue-project
   ```

   Luego, instala `vue-router` utilizando npm o yarn:

   ```bash
   npm install vue-router
   ```

   o

   ```bash
   yarn add vue-router
   ```

3. **Configurar y Utilizar Vue Router:**

   En el archivo principal de tu aplicación (por lo general, `src/main.js`), configura y utilice `vue-router`, como se muestra en el ejemplo en la respuesta anterior.

Instalación en un Proyecto Existente

Si ya tienes un proyecto Vue.js existente y deseas agregar `vue-router`, sigue estos pasos:

1. **Instalar Vue Router:**

En el directorio raíz de tu proyecto, instala `vue-router` utilizando npm o yarn:

```bash
npm install vue-router
```

o

```bash
yarn add vue-router
```

2. **Configurar y Utilizar Vue Router:**

Luego de instalar `vue-router`, realiza los siguientes cambios en tu proyecto:

- En el archivo principal de tu aplicación (por lo general, `src/main.js`), configura y utilice `vue-router`, como se muestra en el ejemplo en la respuesta anterior.

- Crea componentes para tus vistas si aún no los tienes. Por ejemplo, puedes crear un componente llamado `Home.vue` para la página de inicio y un componente llamado `About.vue` para la página "Acerca de".

- Define las rutas en un archivo separado, como `router.js`, y luego importa y utilízalo en tu archivo principal.

- Agrega los componentes de vista a las rutas y utiliza la directiva `router-link` y el componente `router-view` en tus componentes.

- Asegúrate de que las rutas y los componentes de vista estén configurados correctamente para que la navegación funcione según tus necesidades.

Después de realizar estos pasos, deberías tener `vue-router` instalado y configurado en tu proyecto existente, lo que te permitirá implementar la navegación en tu aplicación Vue.js.

Estados Globales

Vuex

Vuex es un administrador de estado para aplicaciones Vue.js. Proporciona un almacén centralizado donde puedes mantener y gestionar el estado compartido de tu aplicación de manera coherente. En otras palabras, Vuex es una biblioteca que ayuda a administrar los datos que necesitas compartir entre componentes en una aplicación Vue.js, lo que facilita el control del flujo de datos y el mantenimiento de la coherencia en toda la aplicación.

En resumen, Vuex es una solución de gestión de estado que simplifica la tarea de mantener y sincronizar datos compartidos entre componentes de Vue.js, lo que mejora la previsibilidad y el mantenimiento de las aplicaciones complejas.

./src/store/index.ts

```
import { createStore } from 'vuex'

export default createStore({
      state: {
      },
      getters: {
      },
      mutations: {
      },
      actions: {
      },
      modules: {
      }
})
```

El código es una plantilla básica de una tienda (store) Vuex en una aplicación Vue.js. Vuex es un sistema de gestión de estado centralizado que se utiliza comúnmente en aplicaciones Vue.js para administrar y compartir datos entre componentes de manera eficiente. Aquí está una descripción de cada sección en la plantilla:

1. state: Esta sección es donde se definen las propiedades de estado de tu aplicación. Aquí se almacenan los datos que necesitas compartir y administrar en toda la aplicación. Puedes inicializar estas propiedades con valores iniciales.

2. getters: Los "getters" son funciones que te permiten acceder y calcular propiedades derivadas basadas en el estado actual. Son útiles para obtener datos del estado de una manera transformada o filtrada.

3. mutations: Las "mutations" son funciones que se utilizan para modificar el estado. Son responsables de realizar cambios en el estado de forma síncrona. Cada mutación debe ser una función pura que toma el estado actual como argumento y realiza una modificación en él.

4. actions: Las "actions" son funciones que se utilizan para realizar operaciones asíncronas o para realizar mutaciones con lógica más compleja. A menudo, las acciones se utilizan para manejar solicitudes HTTP, temporizadores y otras operaciones asincrónicas antes de realizar una mutación.

5. modules: Vuex permite dividir la tienda en módulos para mantener el código organizado y modular. Cada módulo puede tener su propio estado, getters, mutaciones y acciones. Los módulos se definen aquí y se pueden anidar para crear una estructura jerárquica de almacenamiento.

Esta plantilla es un punto de partida común para una tienda Vuex, y a medida que desarrollas tu aplicación, puedes agregar propiedades de estado, getters, mutaciones y acciones según sea necesario para administrar y compartir datos de manera efectiva en tu aplicación Vue.js.