1. Primeros pasos con Vue 3

Instalación de Vue 3

Hay varias formas de empezar a usar Vue 3, pero la más común es usando Vue CLI o con `vite` para una configuración más rápida y moderna.

Opción 1: Usando Vue CLI
Instala Vue CLI

```
<script src="https://unpkg.com/vue@3/dist/vue.global.js"></script>
<div id="app">{{ message }}</div>

<script>
  const { createApp, ref } = Vue

  createApp({
    setup() {
      const message = ref('Hello vue!')
      return {
        message
      }
    }
  }).mount('#app')
</script>
```

Opción 2: Usando node
Instalar nodeJS:
Debemos de instalar node js desde su pagina oficial, https://nodejs.org/en esto dependerá de nuestro sistema operativo, en windows solo se descarga el instalador desde su web y se siguen los pasos de este, para linux y mac vienen las instrucciones dentro de la web https://nodejs.org/en/download/package-manager

Instala desde node:

```
npm create vue@latest
```

Instala las dependencias:
✔ Project name: … <your-project-name>
✔ 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 / Cypress / Nightwatch / Playwright

✔ Add ESLint for code quality? … No / Yes
✔ Add Prettier for code formatting? … No / Yes
✔ Add Vue DevTools 7 extension for debugging? (experimental) … No / Yes

Scaffolding project in ./<your-project-name>...
Done.

```
cd my-vue-app
npm install
```

Inicia el servidor:
```
npm run dev
```

2. Estructura básica de un componente en Vue 3

Vue se organiza principalmente en componentes:

```
<template>
  <div>
    <h1>{{ message }}</h1>
    <button @click="changeMessage">Cambiar mensaje</button>
  </div>
</template>

<script>
export default {
  data() {
    return {
      message: "¡Hola, Vue 3!"
    };
  },
  methods: {
    changeMessage() {
      this.message = "¡Mensaje actualizado!";
    }
  }
};
</script>

<style scoped>
h1 {
  color: blue;
}
```

```
</style>
```

3. Usar Axios en Vue 3 para hacer solicitudes HTTP

Instalación de Axios

Primero necesitas instalar Axios en tu proyecto:

```
npm install axios
```

Usar Axios dentro de un componente:

Aquí te muestro un ejemplo sencillo de cómo hacer una solicitud GET utilizando Axios:

```
<template>
  <div>
    <h1>Datos desde la API</h1>
    <div v-if="loading">Cargando...</div>
    <ul v-else>
      <li v-for="item in items" :key="item.id">{{ item.name }}</li>
    </ul>
  </div>
</template>

<script>
import axios from 'axios';

export default {
  data() {
    return {
      items: [],
      loading: true
    };
  },
  created() {
    this.fetchData();
  },
  methods: {
    async fetchData() {
      try {
        const response = await
axios.get('https://ejemplo.com/users');
        this.items = response.data;
```

```
      } catch (error) {
        console.error('Error al obtener los datos:', error);
      } finally {
        this.loading = false;
      }
    }
  }
};
</script>

<style scoped>
h1 {
  color: green;
}
</style>
```

4. Vue Router: Navegación en tu Aplicación

El **Vue Router** es una librería oficial de Vue.js para gestionar el enrutamiento en aplicaciones de una sola página (SPA).

Instalación de Vue Router:

Si no lo tienes instalado, puedes agregarlo a tu proyecto:

```
npm install vue-router
```

Configuración Básica de Vue Router:

Crea un archivo `router.js` en la carpeta `src` de tu proyecto:

```
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;
```

Modifica `main.js` para usar el router:

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

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

Uso en los componentes: Ahora puedes usar las rutas en los componentes con enlaces:

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

5. Directivas en Vue 3

Las directivas son muy importantes para manipular el DOM en Vue. Aquí te explico algunas de las más usadas:

`v-if` y `v-else`: Renderizado condicional

```
<template>
  <div>
    <h1 v-if="isLoggedIn">Bienvenido de nuevo</h1>
    <h1 v-else>Por favor, inicia sesión</h1>
  </div>
</template>

<script>
export default {
  data() {
    return {
```

```
      isLoggedIn: false
    };
  }
};
</script>
```

- **`v-if`**: Renderiza el elemento si la condición es `true`.
- **`v-else`**: Se muestra si la condición del `v-if` es `false`.

`v-for`: Iteraciones

```
<template>
  <ul>
    <li v-for="(item, index) in items" :key="index">
      {{ index + 1 }}. {{ item.name }}
    </li>
  </ul>
</template>

<script>
export default {
  data() {
    return {
      items: [
        { name: 'Item 1' },
        { name: 'Item 2' },
        { name: 'Item 3' }
      ]
    };
  }
};
</script>
```

- **`v-for`**: Permite iterar sobre arrays y objetos.
- **`:key`**: Es importante agregar una clave única para evitar problemas de renderizado.

`v-show`: Mostrar u ocultar un elemento (sin quitarlo del DOM)

```
<template>
  <p v-show="isVisible">Este texto es visible</p>
  <button @click="toggleVisibility">Alternar Visibilidad</button>
</template>
```

```
<script>
export default {
  data() {
    return {
      isVisible: true
    };
  },
  methods: {
    toggleVisibility() {
      this.isVisible = !this.isVisible;
    }
  }
};
</script>
```

- **`v-show`**: Muestra u oculta el elemento manipulando el atributo `display`.

6. `v-model`: Enlace Bidireccional de Datos

El uso de `v-model` permite sincronizar el valor de un campo de formulario con los datos del componente:

```
<template>
  <div>
    <input v-model="username" placeholder="Ingresa tu nombre" />
    <p>Hola, {{ username }}!</p>
  </div>
</template>

<script>
export default {
  data() {
    return {
      username: ''
    };
  }
};
</script>
```

- **`v-model`**: Crea un enlace reactivo entre el valor del input y la variable `username`. Cualquier cambio en el campo del input actualizará `username`, y viceversa.

7. Acciones y Eventos

En Vue, puedes capturar eventos y realizar acciones usando `@` seguido del evento que quieres escuchar.

Captura de Eventos con `@click`:

```
<template>
  <button @click="incrementCounter">Haz clic aquí</button>
  <p>Contador: {{ counter }}</p>
</template>

<script>
export default {
  data() {
    return {
      counter: 0
    };
  },
  methods: {
    incrementCounter() {
      this.counter++;
    }
  }
};
</script>
```

- **`@click`**: Escucha el evento de clic y ejecuta la función `incrementCounter`.

Paso de parámetros en eventos:

```
<template>
  <button @click="incrementBy(5)">Incrementar por 5</button>
</template>

<script>
export default {
  data() {
    return {
      counter: 0
    };
  },
  methods: {
    incrementBy(value) {
```

```
      this.counter += value;
    }
  }
};
</script>
```

- Puedes pasar parámetros a las funciones en las directivas de evento como en `@click`.

8. Formularios Complejos y Validación con `v-model`

Puedes usar `v-model` para manejar formularios más complejos, como checkboxes, radio buttons y selectores. Aquí te muestro un ejemplo de checkbox y selectores:

```
<template>
  <div>
    <label>
      <input type="checkbox" v-model="accepted" />
      Acepto los términos y condiciones
    </label>
    <p>{{ accepted ? 'Términos aceptados' : 'Términos no aceptados'
}}</p>

    <select v-model="selectedOption">
      <option disabled value="">Selecciona una opción</option>
      <option value="opcion1">Opción 1</option>
      <option value="opcion2">Opción 2</option>
    </select>
    <p>Opción seleccionada: {{ selectedOption }}</p>
  </div>
</template>

<script>
export default {
  data() {
    return {
      accepted: false,
      selectedOption: ''
    };
  }
};
</script>
```

9. Configurar el entorno para producción

En Vue, puedes definir diferentes entornos, como desarrollo y producción, usando un archivo `.env`. Asegúrate de que tienes configurados correctamente los valores en un archivo `.env.production` para definir las variables específicas del entorno de producción.

Ejemplo de `.env.production`:

```
VUE_APP_API_URL=https://api.ejemplo.com
VUE_APP_ENV=production
```

10. Compilar el proyecto

Vue CLI proporciona una manera muy sencilla de generar los archivos optimizados para producción. En tu terminal, ejecuta el siguiente comando:

```
npm run build
```

Este comando hace lo siguiente:

- **Minifica**: Reduce el tamaño de los archivos JavaScript, CSS y HTML eliminando espacios en blanco y comentarios.
- **Optimiza**: Organiza los recursos y agrupa los módulos de forma más eficiente.
- **Crea un directorio `dist/`**: El resultado final es una carpeta llamada `dist`, que contiene los archivos listos para ser desplegados.

11. Inspeccionar el contenido del directorio `dist`

Después de ejecutar `npm run build`, el directorio `dist/` tendrá archivos como:

- **`index.html`**: El archivo principal HTML.
- **`css/` y `js/`**: Directorios que contienen los archivos CSS y JavaScript minificados y optimizados.

Es importante no modificar manualmente estos archivos después del build.

12. Desplegar en un servidor web