App lista de tareas:

ELIAN BENITEZ COGOLLO

La aplicación que estoy desarrollando es una app de lista de tareas que permite a los usuarios gestionar sus pendientes de manera eficiente y colaborativa. Esta app ofrece una serie de características diseñadas para mejorar la productividad y facilitar la organización. Los usuarios pueden agregar nuevas tareas a la lista y eliminar aquellas que ya no sean necesarias, mientras que la interfaz muestra todas las tareas pendientes de manera clara y accesible. Además, los usuarios tienen la opción de chatear con amigos, lo que facilita la comunicación y el intercambio de ideas mientras trabajan en tareas. También pueden realizar tareas en grupo, permitiendo una gestión compartida de proyectos y actividades, lo que fomenta el trabajo en equipo.

La aplicación también permite a los usuarios establecer fechas para sus tareas, asegurando que no se olviden de los plazos importantes. Con la funcionalidad de notificaciones, los usuarios recibirán alertas cuando se acerquen las fechas de sus tareas pendientes, ayudándolos a mantenerse organizados y cumplir con sus responsabilidades a tiempo. Esta característica es especialmente útil para aquellos que manejan múltiples tareas y necesitan un recordatorio para mantenerse al día.

La interfaz de usuario de la aplicación está construida con Vuetify, lo que asegura un diseño moderno y adaptable, compatible con dispositivos móviles y de escritorio. Los componentes visuales, como botones, cuadros de texto y listas, hacen que la interacción sea intuitiva y amigable. El uso de una interfaz atractiva no solo mejora la experiencia del usuario, sino que también facilita la navegación y el acceso a las distintas funcionalidades de la app.

La tecnología base de la aplicación incluye Nuxt.js como framework principal, lo que facilita la estructura de rutas y el desarrollo fluido con Vue.js. Vue.js se encarga de la creación de componentes y la reactividad de la interfaz, mientras que Vuetify proporciona la capa visual necesaria para asegurar una experiencia de usuario efectiva.


Proyecto: https://stackblitz.com/edit/nuxt-starter-rqlapi?file=README.md

# State

En la app de lista de tareas, el **state** se usa para almacenar datos importantes que todos los componentes necesitan compartir, como las tareas creadas por el usuario y la lista de amigos. Este estado global se gestiona con **Vuex** para centralizar estos datos, permitiendo que las tareas o amigos puedan ser agregados, eliminados o actualizados fácilmente.

```
export const state = () => ({
  user: null, // Aquí guardamos los datos del usuario activo
  tasks: [],  // Lista de tareas
  friends: [] // Lista de amigos
});

export const mutations = {
  setUser(state, user) {
    state.user = user; // Mutación para establecer el usuario
  },
  addTask(state, task) {
    state.tasks.push(task); // Mutación para agregar una tarea a la lista
  },
  addFriend(state, friend) {
    state.friends.push(friend); // Mutación para agregar un amigo a la lista
  }
};

export const actions = {
  login({ commit }, user) {
    // Lógica de inicio de sesión simulada
    commit('setUser', user);
  },
  registerTask({ commit }, task) {
    commit('addTask', task);
  },
  registerFriend({ commit }, friend) {
    commit('addFriend', friend);
  }
};
```

- state guarda el estado central, donde user, tasks y friends son variables almacenadas en Vuex.
- **Mutations** (mutations) son métodos que modifican el state. Se llaman con commit.
- **Actions** (actions) son métodos asincrónicos o complejos que despachan mutations.

Se usara el state para administrar el estado global de la aplicación, permitiendo que múltiples componentes accedan y actualicen datos compartidos, como el usuario actual y las tareas.

# Server

El **server** en Nuxt.js, en este caso, se utiliza principalmente para renderizado en el servidor (SSR) y para manejar peticiones a una API. Configuramos una API básica que permite al usuario interactuar con datos como la lista de tareas y amigos, guardándolos y cargándolos según sea necesario, y haciendo que los datos persistan entre sesiones.

El **server** en una aplicación Nuxt.js puede configurarse para manejar tanto el renderizado del lado del servidor (SSR) como la configuración de una API simple.

**Servidor de Renderizado del Lado del Servidor (SSR)**:

- Nuxt.js usa un servidor para renderizar las vistas en el lado del servidor antes de enviarlas al cliente, lo cual mejora la velocidad de carga inicial y la optimización SEO.

- Esto es útil en aplicaciones donde queremos que la información esté lista cuando la página se carga.

**API Backend**:

- El server también puede configurarse para manejar solicitudes a una API.

- Puedes definir una API con Nuxt para manejar datos del usuario o tareas (aunque en proyectos más grandes, usualmente se usa un servidor externo o microservicios).

```
import { NowRequest, NowResponse } from '@vercel/node';

// Este sería el "endpoint" en el servidor frontend de Nuxt para devolver dato
s de ejemplo de tareas.
export default async (req: NowRequest, res: NowResponse) => {
  if (req.method === 'GET') {
    // Datos de ejemplo de tareas
    const tasks = [
      { id: 1, title: 'Tarea de ejemplo 1', completed: false },
      { id: 2, title: 'Tarea de ejemplo 2', completed: true },
      { id: 3, title: 'Tarea de ejemplo 3', completed: false },
    ];
```

```
    res.status(200).json(tasks);
  } else {
    res.setHeader('Allow', ['GET']);
    res.status(405).end(`Method ${req.method} Not Allowed`);
  }
```

```
};
```

# Components

Los **componentes** son las piezas reutilizables de una aplicación de Vue. Cada componente es un bloque de código que representa una parte específica de la interfaz de usuario (UI), como un formulario, una lista, o un botón.

### Componentes Principales en la Aplicación

- Login.vue: Permite al usuario iniciar sesión con solo su nombre.
- AddTask.vue: Muestra el formulario para agregar una tarea nueva.
- TaskList.vue: Lista todas las tareas agregadas y sus estados.
- FriendList.vue: Lista de amigos y sus estados (en línea o fuera de línea).

```
<template>
  <div>
    <input v-model="taskName" placeholder="Nombre de la tarea" />
    <button @click="addTask">Agregar Tarea</button>
  </div>
</template>

<script>
export default {
  data() {
    return {
      taskName: '' // Nombre de la tarea ingresada por el usuario
    };
  },
```

```
  methods: {

    addTask() {

      if (this.taskName) {

        // Llamada a la acción de Vuex para registrar la tarea

        this.$store.dispatch('registerTask', { name: this.taskName, completed:
false });

        this.taskName = ''; // Reiniciar el nombre de la tarea

      }

    }

  }

};

</script>
```

· data define el estado local del componente, como taskName.
· methods define la función addTask, que agrega la tarea al state llamando a una acción de Vuex.

Los componentes dividen la interfaz en piezas reutilizables, cada una con un propósito específico. Los componentes interactúan con el state mediante acciones y computed properties para reflejar el estado global.