

INFORME DEL PROYECTO: "APP WEB DE NOTICIAS"

Introducción

Este informe describe el desarrollo de una aplicación web diseñada para proporcionar noticias organizadas en categorías y accesibles a los usuarios de forma dinámica e interactiva. La App busca convertirse en una fuente confiable y accesible de información, destacándose por su diseño intuitivo, estructura eficiente y capacidad para adaptarse a las necesidades de los usuarios.

Objetivos del Proyecto

Objetivo General:

- Desarrollar una aplicación web intuitiva y atractiva que permita a los usuarios acceder a noticias categorizadas de manera eficiente y personalizada.

Objetivos Específicos:

- Crear una interfaz intuitiva que facilite la navegación entre categorías de noticias.
- Implementar funcionalidades avanzadas como búsquedas por categoría, agregar noticias a favoritos y refrescar contenido en tiempo real.
- Diseñar una estructura de datos clara y flexible, optimizada para futuras expansiones.
- Integrar un sistema de navegación eficiente, adaptado a dispositivos móviles y escritorio.
- Incorporar funcionalidades que promuevan la interacción del usuario, como botones de redes sociales y marcadores personalizados.

Descripción del Proyecto

La aplicación web utiliza una estructura de datos JSON organizada para manejar las noticias. Cada noticia incluye propiedades como título, descripción, imágenes, cuerpo, categorías y créditos del autor. Además, cuenta con diversas funcionalidades que hacen de esta app una herramienta interactiva y moderna:

Componentes Principales:

- **Card:** Muestra detalles del clima y las fechas asociadas con las noticias.
- **Buttons:** Facilitan la navegación entre categorías y páginas específicas.
- **List:** Proporciona un menú desplegable con opciones organizadas por categorías. App Bar: Incluye una barra de herramientas con buscador integrado y opciones de navegación.
- **Bottom Navigation:** Permite agregar noticias a favoritos y refrescar contenido.
- **Footer:** Contiene información de contacto, redes sociales y derechos de autor.
- **Speed Dials:** Incluye botones adicionales para funcionalidades especiales.

Funcionalidades adicionales a Futuro

Se espera el futuro integrar, en nuestra página web de noticias características avanzadas para mejorar la experiencia del usuario, y ofrecer un contenido más personalizado y accesible.

- **Modo oscuro:** donde los usuarios puedan alternar entre temas claro y oscuro.
- **Favoritos:** Opciones para guardar noticias preferidas.
- **Filtros:** Búsqueda avanzada por categorías y palabras clave.
- **Interacción social:** Enlaces directos a redes sociales.

Página Principal:

- **Carrusel de Noticias Destacadas:** para presentar noticias relevantes en un formato visual atractivo.
- **Sistema de Publicación:** Las noticias se almacenaran en un formato JSON, lo que facilitara la manipulación y la presentación de datos.
- **Actualización Dinámica:** Los datos se obtendrán de APIs externas y se presentaran en tiempo real

Estructuración.

Su estructura json que se empleara para la aplicación web comienza con un arreglo que contiene varios objetos que contienen las siguientes propiedades o se estructura de esta manera:

```
[
  {
    "title": "string",
    "img": "string",
    "descripcion": "string",
    "categoria": [
      "string"
    ],
    "cuerpo": "string",
    "creditos": {
      "autor": "string",
      "correo": "string",
      "redessociales": {
        "instagram": "string",
        "facebook": "string",
        "x": "string"
      }
    }
  }
]
```

Mockup:

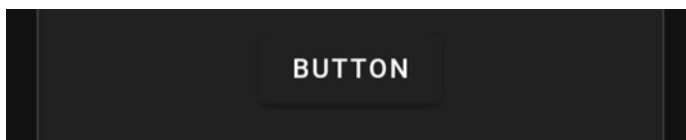


Componentes que se planean usar:

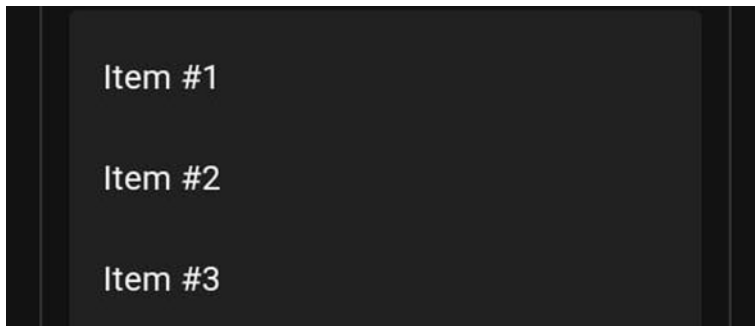
- **Card:** Este componente se utilizará para mostrar detalles sobre el clima y las fechas, aunque estos datos pueden estar sujetos a cambios durante el proceso de desarrollo de la App.



- **Buttons:** Se utilizarán para la navegación entre páginas, como las diferentes categorías, y podrían incluirse en menús desplegables u otros ajustes de navegación.



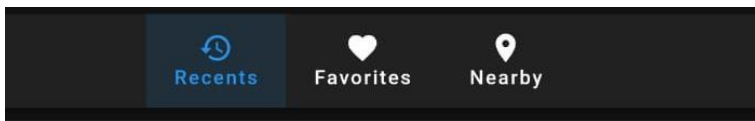
- **List:** Este componente se empleará para una lista de menús desplegables que estará ubicada en el lado izquierdo de la App.



- **App Bar:** En conjunto con el componente "List" y una parte del "App Bar", se creará una lista de categorías. Otras partes del App Bar, como la lupa, se utilizarán exclusivamente para la búsqueda de noticias.



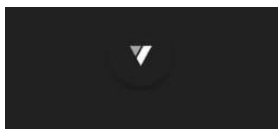
- **Bottom Navigation:** Este componente permitirá al usuario agregar elementos a favoritos y refrescar la página.



- **Footer:** Incluirá los contactos de las redes sociales (Facebook, Instagram y número de WhatsApp), junto con el copyright, logo y nombre de la app. El orden de estos elementos está aún en desarrollo.



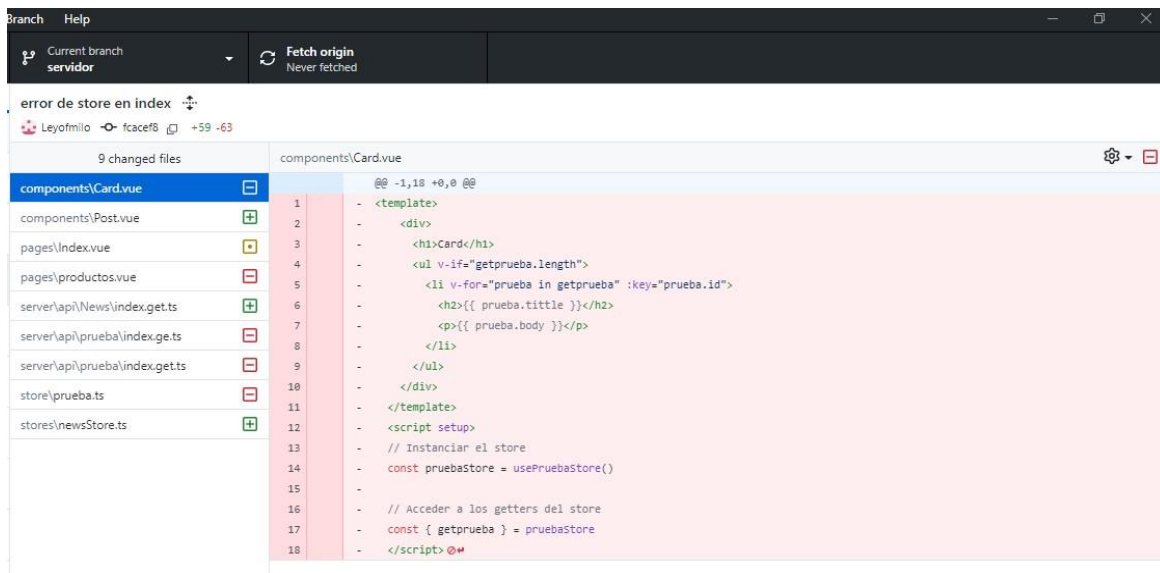
- **Speed Dials:** Se usará para añadir botones adicionales, aunque aún no se ha definido exactamente qué botones se incluirán; esto se decidirá conforme avance el proyecto.



NOTA: Todos los componentes mencionados pueden estar sujetos a cambios durante el desarrollo de la app.

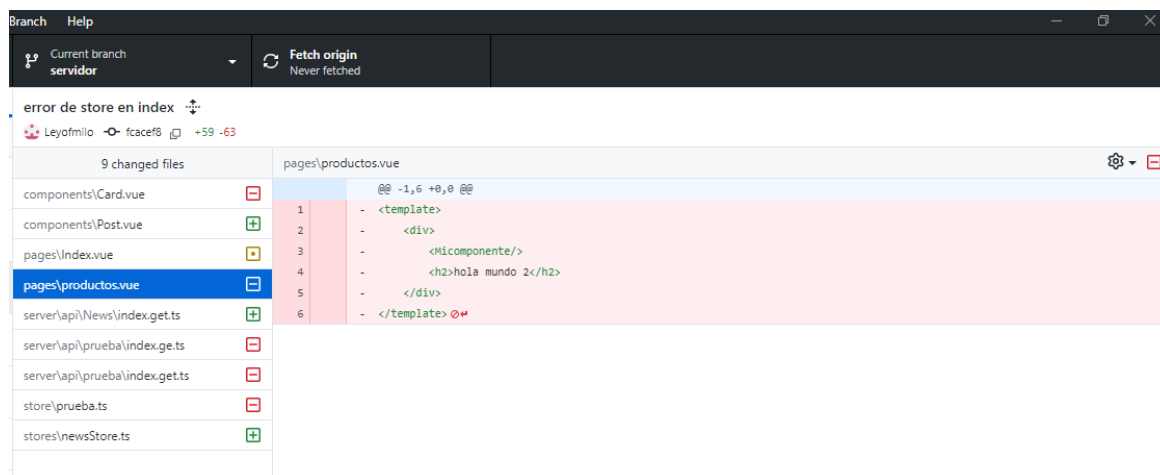
ALGUNAS MODIFICACIONES.

- Este componente obtiene una lista de objetos prueba del store pruebaStore. Si la lista contiene elementos, el componente muestra cada uno en una tarjeta con un título y un cuerpo. La lógica de mostrar u ocultar depende de la cantidad de elementos en getprueba.



```
@@ -1,18 +0,0 @@
1 - <template>
2 -   <div>
3 -     <h1>Card</h1>
4 -     <ul v-if="getprueba.length">
5 -       <li v-for="prueba in getprueba" :key="prueba.id">
6 -         <h2>{{ prueba.title }}</h2>
7 -         <p>{{ prueba.body }}</p>
8 -       </li>
9 -     </ul>
10 -   </div>
11 - </template>
12 - <script setup>
13 -   // Instanciar el store
14 -   const pruebaStore = usePruebaStore()
15 -
16 -   // Acceder a los getters del store
17 -   const { getprueba } = pruebaStore
18 - </script> @*
```

- Este código tiene una estructura básica y solo renderizará el contenido de MiComponente junto con el texto. No tiene ninguna lógica adicional ni comportamiento interactivo, solo sirve para estructurar y mostrar estos elementos.

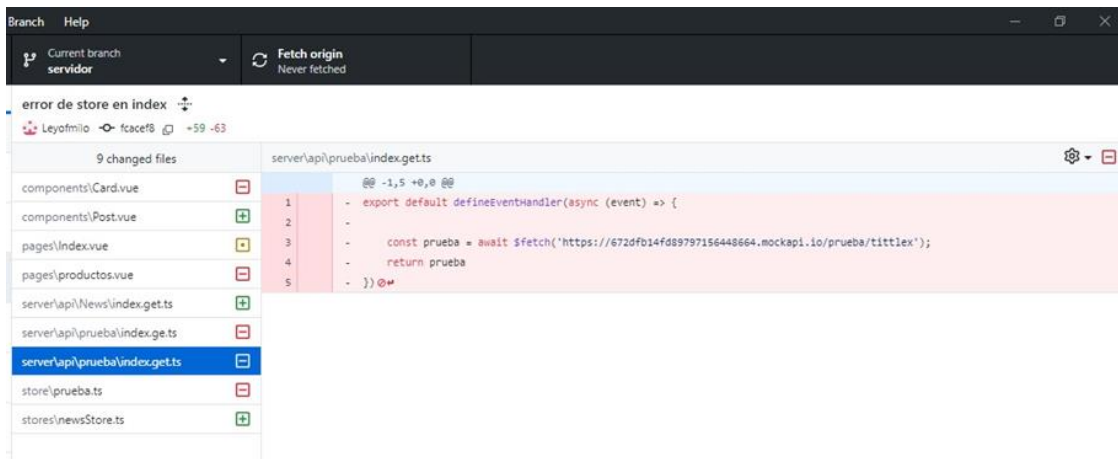


```
@@ -1,6 +0,0 @@
1 - <template>
2 -   <div>
3 -     <MiComponente/>
4 -     <h2>hola mundo 2</h2>
5 -   </div>
6 - </template> @*
```

Este código define un endpoint de API en Nuxt que:

- Realiza una solicitud a una API externa para obtener datos.
- Espera la respuesta de esa API.
- Devuelve los datos obtenidos al cliente que hizo la solicitud.

Cuando alguien accede al endpoint `/api/prueba/index`, esta función se ejecuta, obtiene los datos de la API externa y devuelve esos datos al cliente. Esto es útil cuando se necesitan datos de otra fuente y se quiere encapsular esa lógica en un endpoint del servidor.



```
Branch  Help
Current branch: servidor
Fetch origin: Never fetched

error de store en index
Leyofmilo fcaef8 +59 -63

9 changed files
components/Card.vue
components/Post.vue
pages/Index.vue
pages/productos.vue
server/api/News/index.get.ts
server/api/prueba/index.get.ts
store/prueba.ts
stores/newsStore.ts

server/api/prueba/index.get.ts
@@ -1,5 +0,0 @@
1 - export default defineEventHandler(async (event) => {
2 -
3 -   const prueba = await $fetch('https://672dfb14fd89797156448664.mockapi.io/prueba/tittlex');
4 -   return prueba
5 - }) @#
```

EXPLICACION DEL CÓDIGO

Este store proporciona una forma centralizada de manejar y almacenar datos de tipo Prueba en la aplicación. Las características clave son:

- Estado reactivo para almacenar la lista pruebas y el objeto prueba.
- Persistencia en localStorage para conservar los datos.
- Getters para acceder al estado.
- Acción `fetchPruebas` para obtener datos de una API y actualizar el estado.

Este store facilita la gestión de datos prueba en los componentes Vue, ya que permite acceder y modificar el estado desde cualquier componente que lo use.

```

1  import { defineStore } from "pinia";
2
3  interface Prueba {
4    id: number;
5    title: string;
6    body: string;
7  }
8
9  export const usePruebasStore = defineStore({
10    id: "pruebas",
11    state: () => ({
12      pruebas: [] as Prueba[],
13      pruebas1: [] as Prueba[],
14    }),
15    persist: {
16      storage: persistedState.localStorage,
17    },
18    getters: {
19      getPruebas: (state) => state.pruebas,
20      getPruebas1: (state) => state.pruebas1,
21    },
22    actions: {
23      async fetchPruebas() {
24        const pruebas = await $fetch("/api/pruebas");
25        this.pruebas = pruebas;
26      },
27    },
28  });

```

EXPLICACION DEL CÓDIGO.

Utiliza el componente v-list con el modelo de datos opened="open", lo cual indica un control reactivo de apertura/cierre.

Primer elemento: v-list-item con el icono mdi-home y título Home. Grupos de listas (v-list-group):

Grupo "Users":

Usa v-slot:activator="{ props }" para manejar la activación del grupo.

Incluye un elemento v-list-item con ícono mdi-account-circle y título Users. Grupo "Admin":

Mismo enfoque que "Users", pero con el título Admin.

Usa una iteración v-for para recorrer un arreglo admins que renderiza los títulos e íconos dinámicamente.

- **Conclusión:** Este componente List.vue está diseñado para mostrar listas organizadas en grupos desplegables, con iconos e información dinámica, probablemente usando Vuetify (por el uso de prefijos v-).

```

1  <template>
2    <v-card
3      class="mx-auto"
4      width="300"
5    >
6      <v-list v-model:opened="open">
7        <v-list-item prepend-icon="mdi-home" title="Home"></v-list-item>
8
9        <v-list-group value="Users">
10          <template v-slot:activator="{ props }">
11            <v-list-item
12              v-bind="props"
13              prepend-icon="mdi-account-circle"
14              title="Users"
15            ></v-list-item>
16          </template>
17
18          <v-list-group value="Admin">
19            <template v-slot:activator="{ props }">
20              <v-list-item
21                v-bind="props"
22                title="Admin"
23              ></v-list-item>
24            </template>
25
26            <v-list-item
27              v-for="(title, icon), i in admins"
28              :key="i"
29              prepend-icon="icon"
30              title="title"

```

EXPLICACION DEL CÓDIGO

- Este componente proporciona una barra de navegación personalizada con:
- Un título estático.
- Un icono de navegación.
- Un espacio para personalizar contenido mediante slots.
- La integración del componente Enlaces para manejar o mostrar enlaces de navegación adicionales.

ESTRUCTURA

Template

<div>: Contenedor principal del componente.

<v-app-bar :elevation="2">: Barra de aplicación con elevación visual de nivel 2.

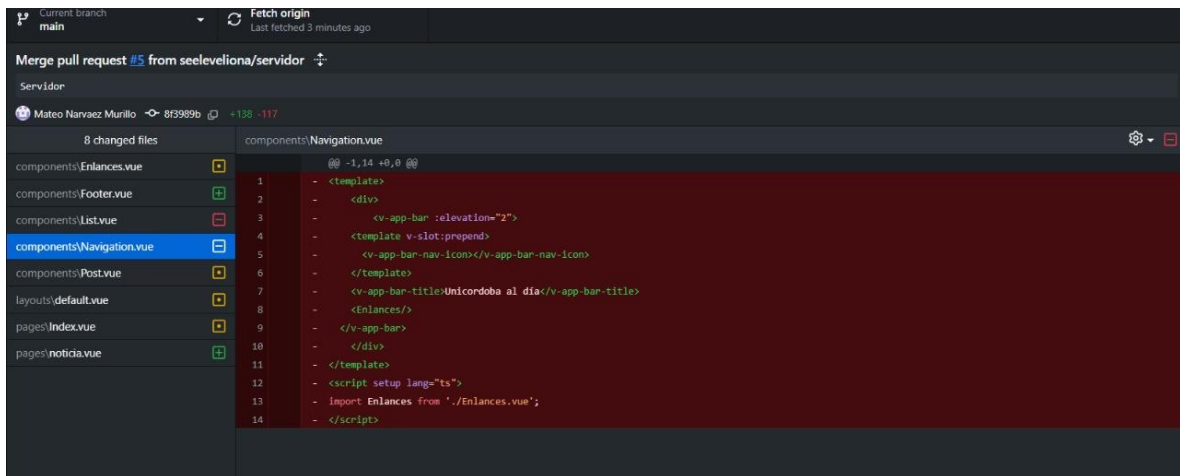
<template v-slot:prepend>: Slot para insertar contenido antes del contenido principal.

<v-app-bar-nav-icon>: Icono de navegación (generalmente usado para menús laterales).

<v-app-bar-title>: Título de la barra, en este caso "Unicordoba al día".

<Enlaces />: Componente personalizado "Enlaces", que se importa y renderiza dentro de la barra.

Script



```
@@ -1,14 +0,0 @@
1 - <template>
2 -   <div>
3 -     <v-app-bar :elevation="2">
4 -       <template v-slot:prepend>
5 -         <v-app-bar-nav-icon></v-app-bar-nav-icon>
6 -       </template>
7 -       <v-app-bar-title>Unicordoba al día</v-app-bar-title>
8 -       <Enlaces/>
9 -     </v-app-bar>
10 -   </div>
11 - </template>
12 - <script setup lang="ts">
13 -   import Enlaces from './Enlaces.vue';
14 - </script>
```