

Índice de la Documentación

1. Introducción

2. Glosario

3. Estructura del Proyecto

4. Configuración del Proyecto

5. Estado Global (Pinia) [ESTAMOS AQUI]

6. Configuración de Rutas

7. Main.js

8. Views

9. Componentes

10. Implementaciones

1. Introducción

El siguiente proyecto es una página web de ropa que usa la API FakeStore. Usamos dicha API, con la finalidad de obtener todos los productos, realizar búsquedas y ordenarlas por categoría.

Las tecnologías usadas son Vue 3, Pinia, Vite y Bootstrap.

2. Glosario

Conceptos clave utilizados en el proyecto:

Pinia (librería para comunicarse entre componentes): librería para manejar estados (datos que quieres guardar y compartir entre componentes). En conclusión: Sirve para guardar datos globales. Por ejemplo:

- Qué productos hay en el carrito.
- Si el usuario está logueado.
- El nombre del usuario.

Todo eso lo puedes guardar en Pinia y accederlo desde cualquier parte de la app.

Store: es una memoria reactiva, vive en el navegador mientras la app esté activa.

Gestor de estado: es una solución que permite que los componentes de tu aplicación compartan y accedan a datos de manera eficiente sin necesidad de pasar propiedades de un componente a otro de forma manual.

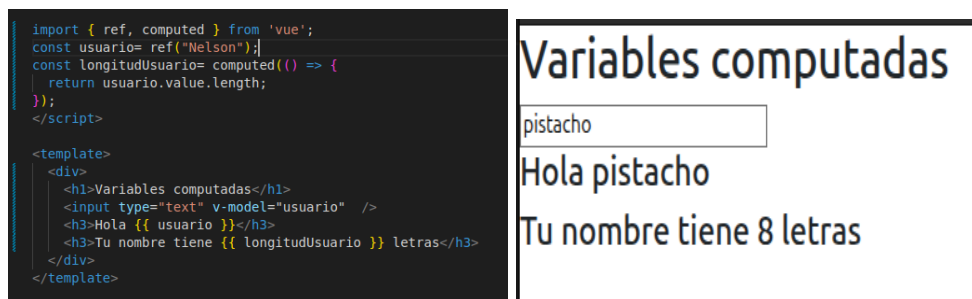
Router: son como enlaces que te conducen a otro contenido (cambia la estructura del DOM por la que quieres ver). Además, puedes pasar valores en las url como producto/:id para cambiar al detalle del producto específico. Sirve para cambiar entre páginas de la aplicación.

- Cuando haces clic en “Ver producto”, vas a /producto/5
- Cuando vas al carrito, vas a /carrito
- Cuando te logueas, vas a /login

Es como un GPS que te dice en qué página estás y te lleva a otra cuando haces clic.

Propiedades computadas: en palabras generales es una variable que depende de otra principal, usualmente esa principal tiene ref y la que depende de ella usa un computed.

Ejemplo, según el nombre que coloques en el input, la longitud va a cambiar, ésto es lo que se conoce como propiedades computadas:



Watch: se usa para ejecutar código en respuesta a esos cambios: puede ser actualizar datos, llamar APIs, validar, sincronizar con almacenamiento local, o incluso comunicar datos a otro lugar (como el componente padre).

SPA (Single Page Application): es una sola página que su contenido va cambiando de acuerdo a varias acciones.

Props (propiedad): es una propiedad que recibe un componente hijo del padre.

3. Estructura del Proyecto

A continuación, mostramos la estructura de la aplicación:

```
/public      # Archivos estáticos
/assets      # Recursos como imágenes y estilos
/src
├── components/ # Componentes reutilizables
├── views/      # Vistas principales de la aplicación
├── router/    # Configuración de rutas (index.js)
├── stores/    # Gestión del estado con Pinia
├── App.vue     # Componente raíz
└── main.js    # Punto de entrada de la aplicación
```

4. Configuración del Proyecto

Mostramos las dependencias usadas en nuestro proyecto:

```
{
  "name": "tienda",
```

```

"private": true,
"version": "0.0.0",
"type": "module",
"scripts": {
  "dev": "vite",
  "build": "vite build",
  "preview": "vite preview"
},
"dependencies": {
  "bootstrap": "^5.3.5",
  "pinia": "^3.0.1",
  "vue": "^3.5.13",
  "vue-router": "^4.5.0"
},
"devDependencies": {
  "@vitejs/plugin-vue": "^5.2.1",
  "vite": "^6.2.0",
  "vite-plugin-vue-devtools": "^7.7.2"
}
}

```

5. Estado Global (Pinia)

Explicación sobre la gestión del estado con Pinia (store).

A continuación, detallamos nuestro primer store llamado [products.js](#):

Creamos la store useProductsStore que guardará en el state todos los productos, categorías, productos por categorías, entre otros.

```

export const useProductsStore = defineStore('allProduct', {
  state: () => ({
    allProduct: [], // Todos los productos
    categories: [], // Categorías únicas
    productsByCategory: {}, // Productos separados por categoría
    loading: false,
    error: null,
  })
})

```

En la función actions, es la encargada de hacer la petición a la API y guardar los datos en formato json en las variables declaradas en el state.

```
actions: {
  async fetchProducts() {
    this.loading = true;
    this.error = null;
    try {
      const response = await fetch('https://fakestoreapi.com/products');
      this.allProduct = await response.json();

      // Extraer categorías únicas
      this.categories = [...new Set(this.allProduct.map(product =>
product.category))];
    } catch (error) {
      this.error = error;
      console.error('Error al obtener los productos:', error);
    } finally {
      this.loading = false;
    }
  },
},
});
```

A continuación, detallamos la store [buscar.js](#):

6. Configuración de Rutas

Definición de rutas con Vue Router.

```
import { createRouter, createWebHistory } from 'vue-router';
import ProductsView from '../views/ProductsView.vue';
import HomeView from '../views/HomeView.vue';
import CartView from '../views/cartView.vue';
```

```
import ProductDetailView from '@/views/ProductDetailView.vue';
import ContactoView from '../views/ContactView.vue';

const routes = [
  {
    path: '/',
    name: 'Home',
    component: HomeView,
  },
  {
    path: '/products',
    name: 'Products',
    component: ProductsView
  },

  {
    path: '/contacto',
    name: 'Contacto',
    component: ContactoView
  },

  {
    path: '/cart',
    name: 'Cart',
    component: CartView, // Componente de la página de detalle de compra
  },

  {
    path: '/product/:id', // Ruta dinámica con el ID del producto
    name: 'ProductDetail',
    component: ProductDetailView,
  },
];

const router = createRouter({
  history: createWebHistory(),
```

```
    routes,
  });

export default router;
```

7. Main.js

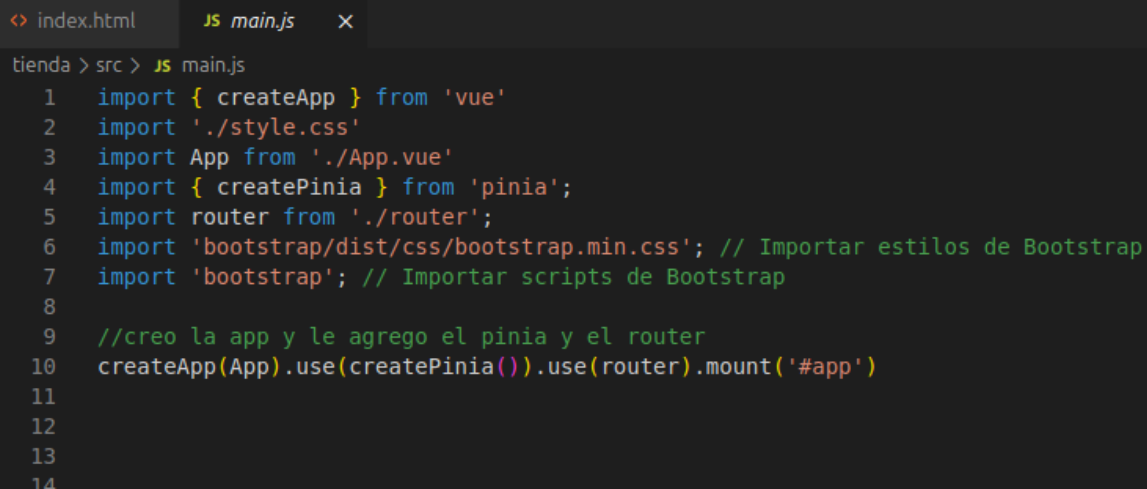
Configuración del archivo principal de la aplicación.

[Main.js](#)

En este fichero, montamos nuestra aplicación principal.

Creamos la App y usamos Pinia como un gestor de estado centralizado. Pinia organiza y mantiene el estado compartido de la aplicación en 'stores', que son objetos reactivos definidos en archivos separados. Aunque no es una base de datos real, puede verse como una memoria temporal reactiva (puede cambiar), que todos los componentes pueden leer y modificar de forma controlada.

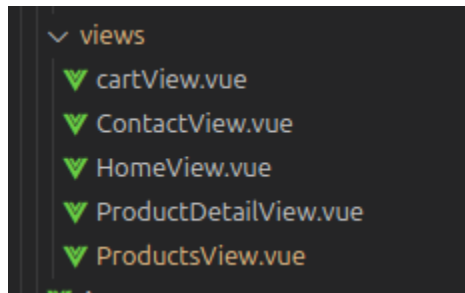
Asimismo, usamos router para poder cambiar estas vistas y componentes.

A screenshot of a code editor with two tabs: 'index.html' and 'JS main.js'. The 'JS main.js' tab is active, showing the following code:

```
tienda > src > JS main.js
1  import { createApp } from 'vue'
2  import './style.css'
3  import App from './App.vue'
4  import { createPinia } from 'pinia';
5  import router from './router';
6  import 'bootstrap/dist/css/bootstrap.min.css'; // Importar estilos de Bootstrap
7  import 'bootstrap'; // Importar scripts de Bootstrap
8
9  //creo la app y le agrego el pinia y el router
10 createApp(App).use(createPinia()).use(router).mount('#app')
11
12
13
14
```

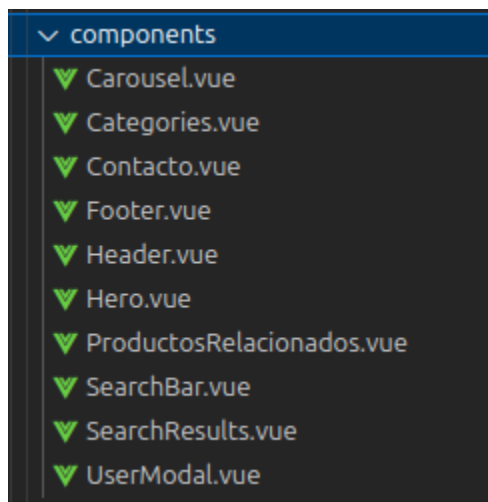
8. Views

Vistas .



9. Componentes

Componentes .

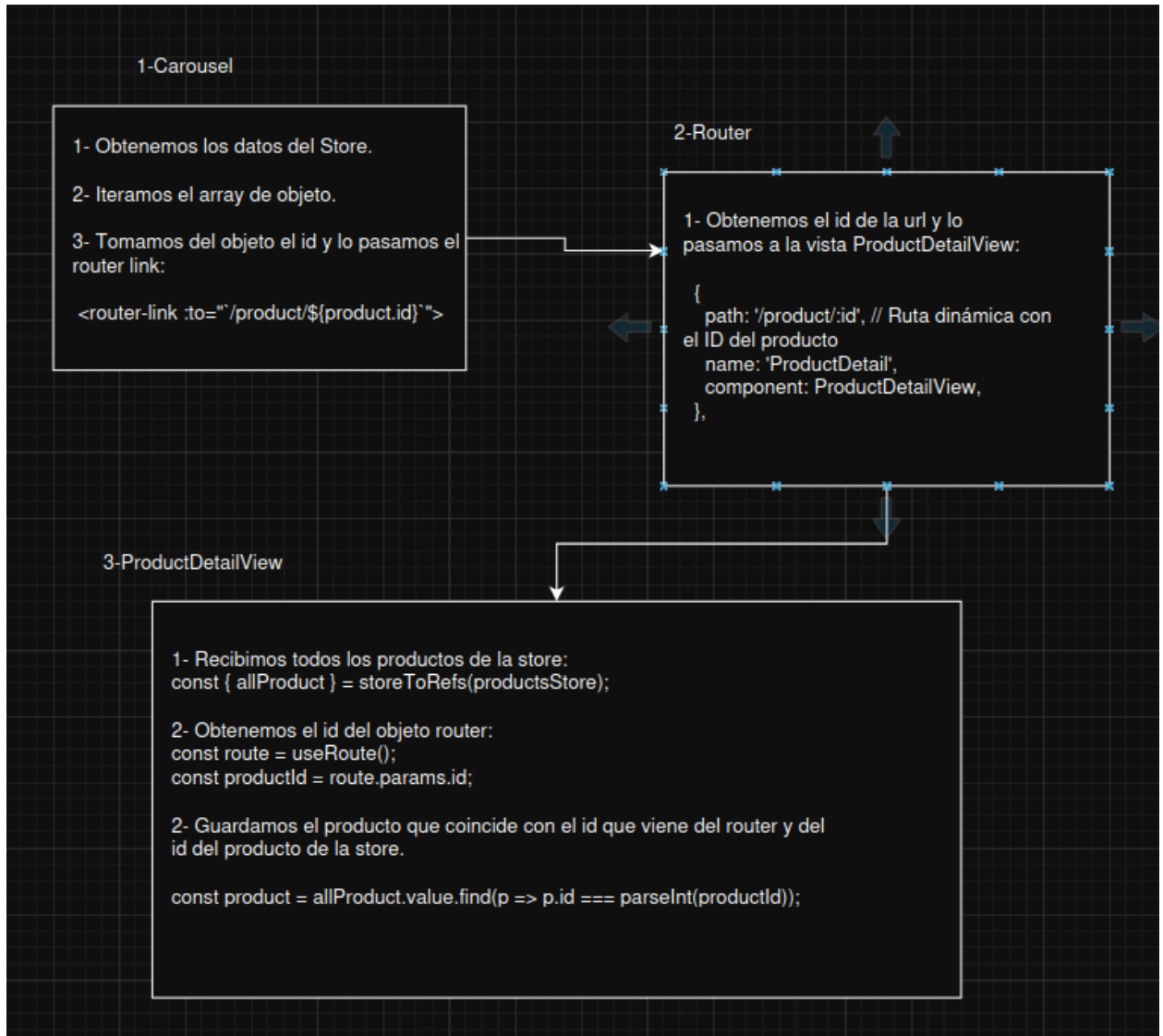


10. Implementación

Componentes .

Inicio:

Detalle producto carousel



Modal

1- UserModal

1- En el `onMounted`, se verifica si los datos (`userName` y `userAvatar`) están guardados en el `localStorage`. Si no están presentes, el modal se muestra automáticamente usando `bootstrap.Modal`.

```
onMounted() => {
  const storedName = localStorage.getItem('userName');
  const storedAvatar = localStorage.getItem('userAvatar');

  if (!storedName || !storedAvatar) {
    const modal = new bootstrap.Modal(document.getElementById('userModal'), {
      backdrop: 'static', // Evita que el usuario cierre el modal haciendo clic fuera
      keyboard: false, // Evita que el usuario cierre el modal con la tecla Esc
    });
    modal.show();
  }
};
```

2- Cuando el usuario hace clic en el botón "Guardar", se valida que haya ingresado un nombre y seleccionado un avatar. Los datos (`userName`, `userAvatar` y `isLoggedIn`) se guardan en el `localStorage`. El modal se cierra y la página se recarga para reflejar los cambios en el header.

```
const saveUserData = () => {
  if (userName.value.trim() === '' || selectedAvatar.value === '') {
    alert('Por favor, ingresa tu nombre y selecciona un avatar.');
```

```
    return;
  }
```

```
  localStorage.setItem('userName', userName.value);
  localStorage.setItem('userAvatar', selectedAvatar.value);
  localStorage.setItem('isLoggedIn', true); // Marca al usuario como logueado
```

```
  const modal = bootstrap.Modal.getInstance(document.getElementById('userModal'));
  modal.hide();
```

```
  window.location.reload(); // Recargar la página para reflejar los cambios
};
```

2- Header

1- En el `onMounted` del Header, se recuperan los datos (`userName`, `userAvatar` y `isLoggedIn`) desde el `localStorage`. Si el usuario está logueado (`isLoggedIn` es `true`), se muestran el nombre, el avatar y el botón de "Cerrar sesión". Si no está logueado, se muestran los botones de "Registrarse" e "Iniciar sesión".

```
onMounted() => {
  userName.value = localStorage.getItem('userName') || 'Invitado';
  userAvatar.value = localStorage.getItem('userAvatar') || '';
  isLoggedIn.value = !!localStorage.getItem('isLoggedIn'); // Verifica si el usuario está logueado
};
```

Cierre de sesión

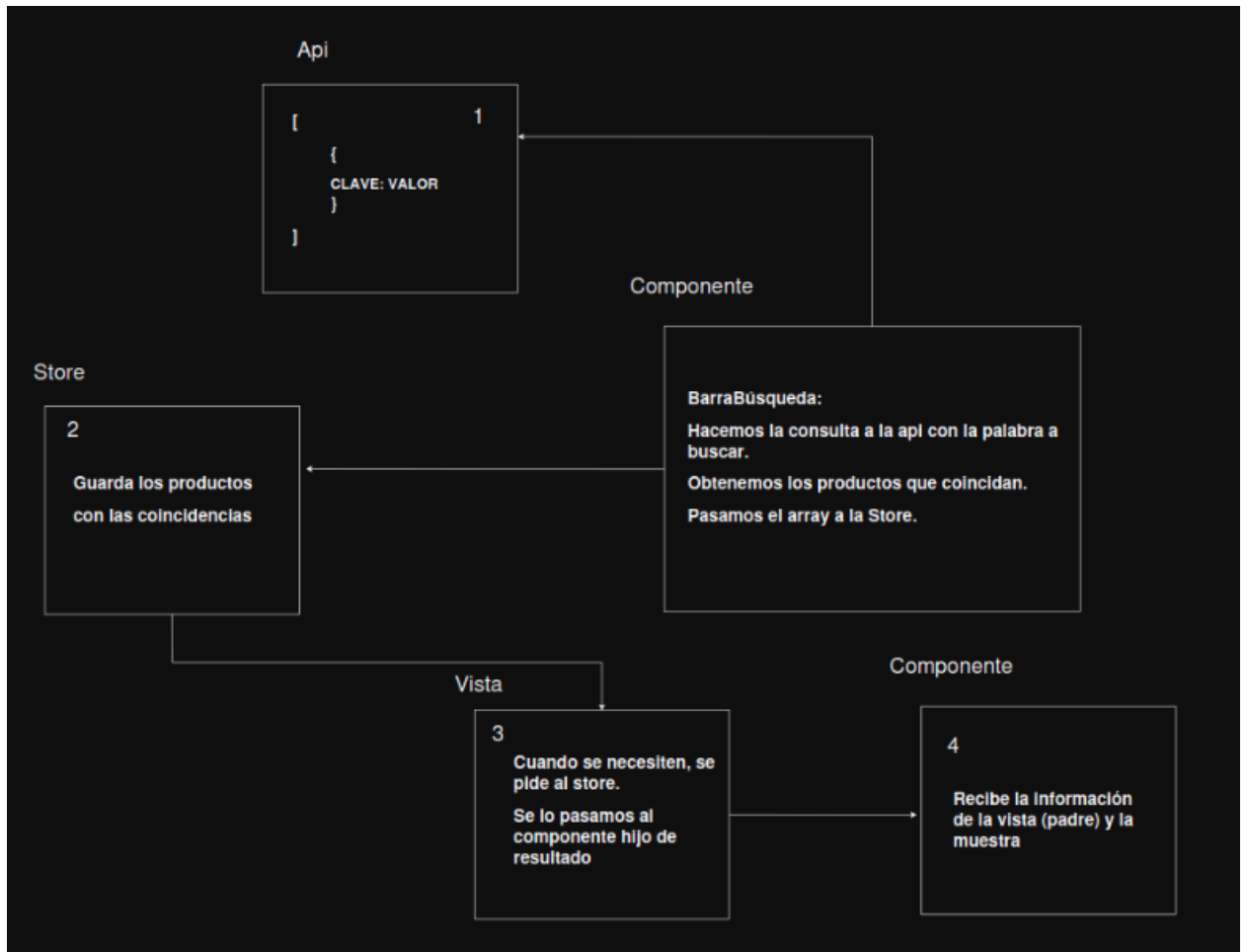
2- Cuando el usuario hace clic en "Cerrar sesión", se eliminan los datos del `localStorage`, se recarga la página y se actualizan las variables reactivas. La página se recarga para mostrar nuevamente el modal.

```
const logout = () => {
  localStorage.removeItem('isLoggedIn');
  localStorage.removeItem('userName');
  localStorage.removeItem('userAvatar');
  isLoggedIn.value = false;
  userName.value = 'Invitado';
  userAvatar.value = '';

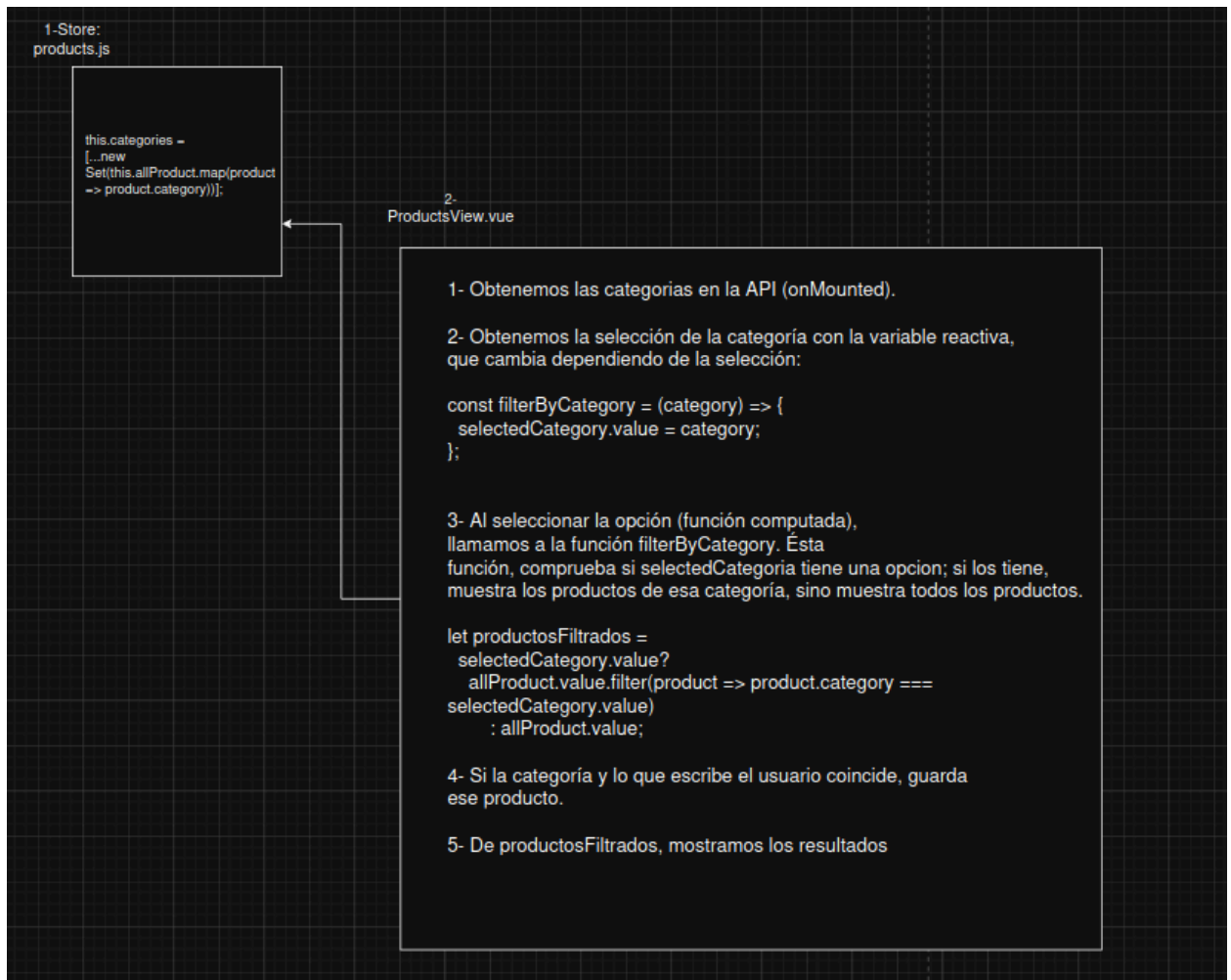
  window.location.reload(); // Recargar la página para mostrar el modal nuevamente
};
```

Productos:

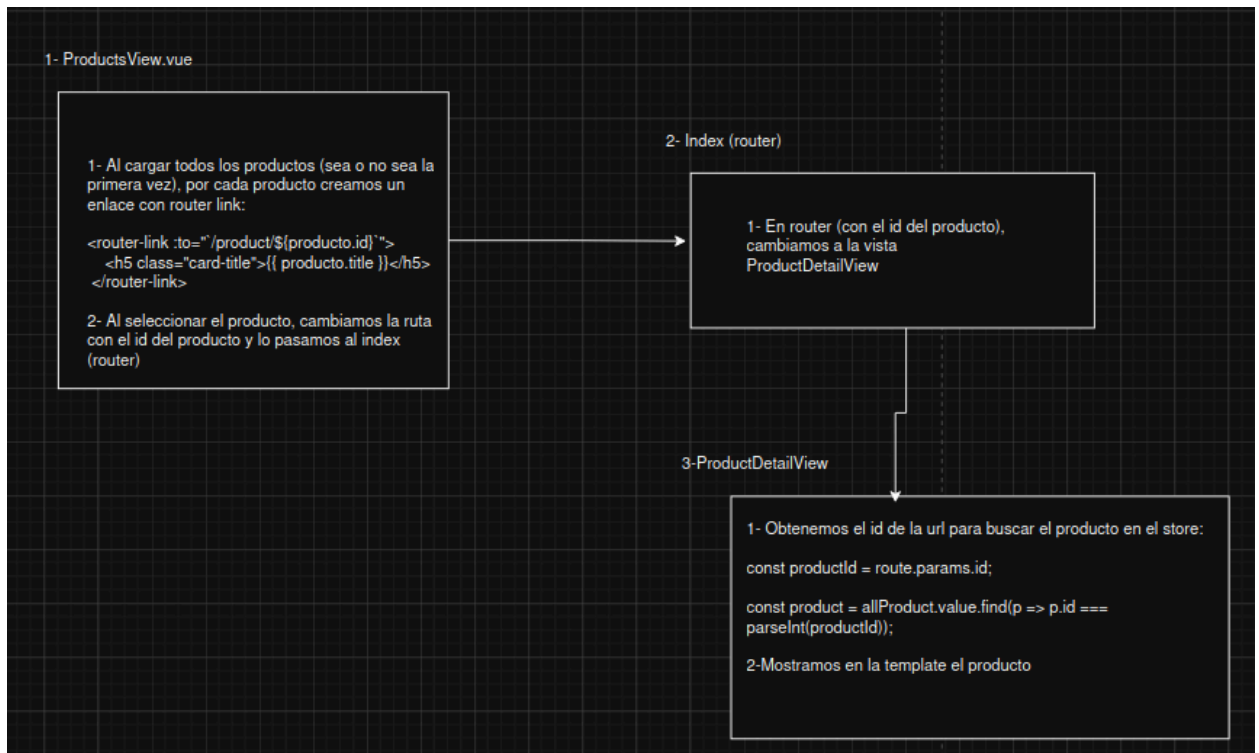
Barra de búsqueda



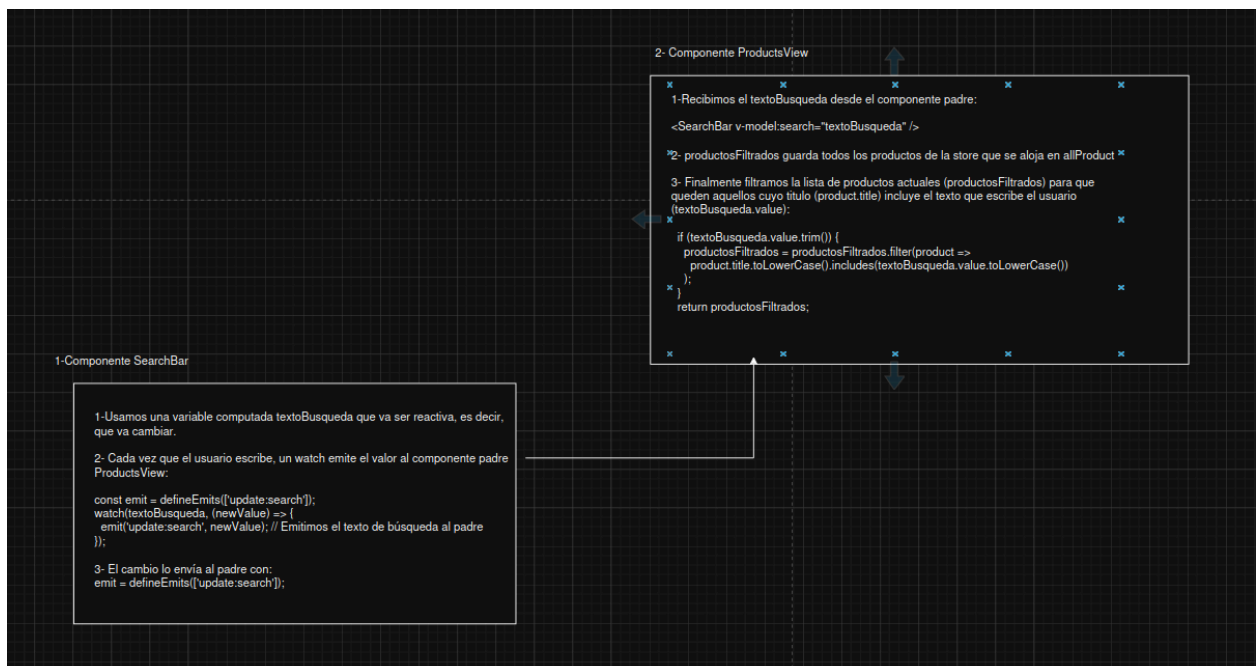
Filtros por categorías



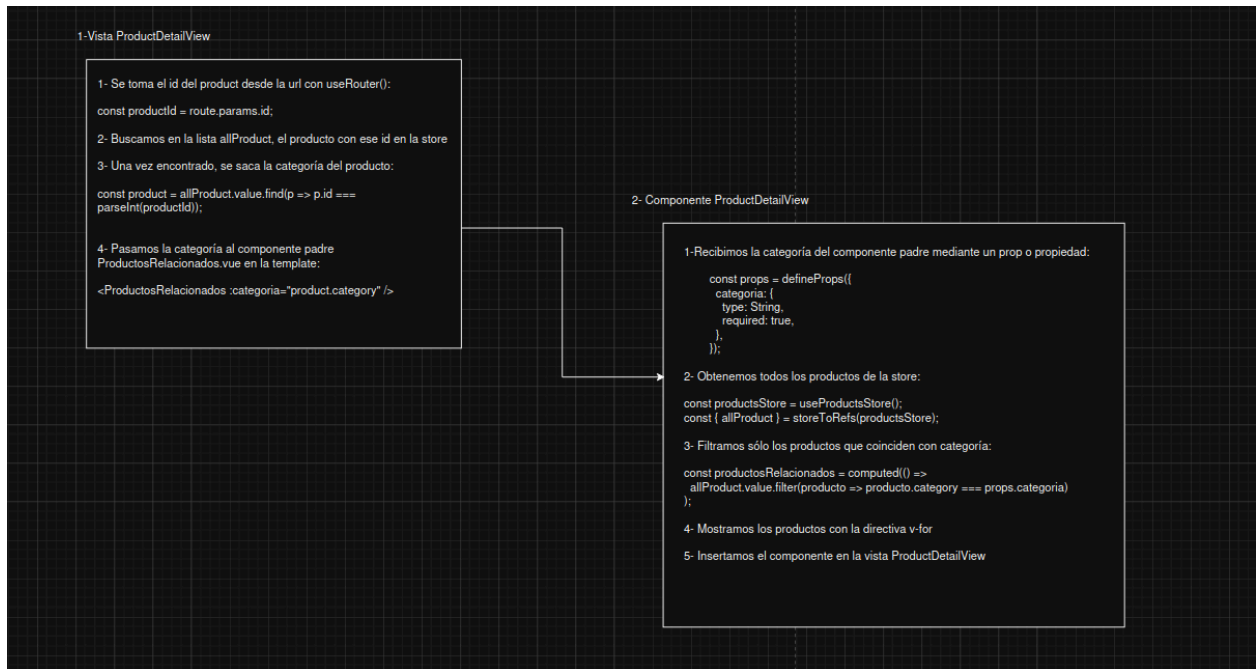
Detalle producto



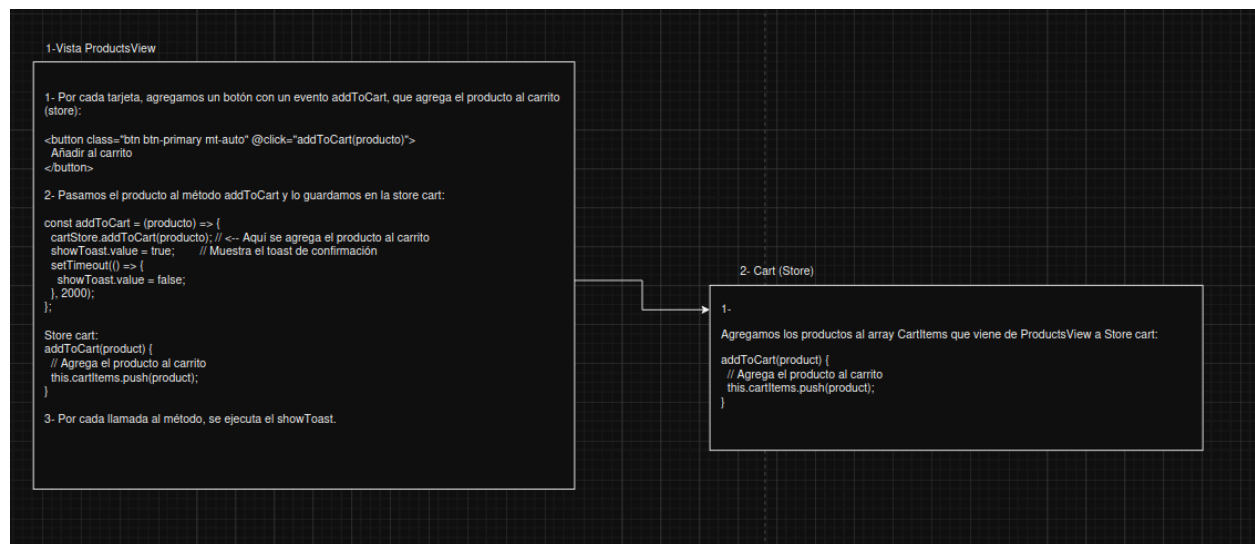
Búsqueda en tiempo real



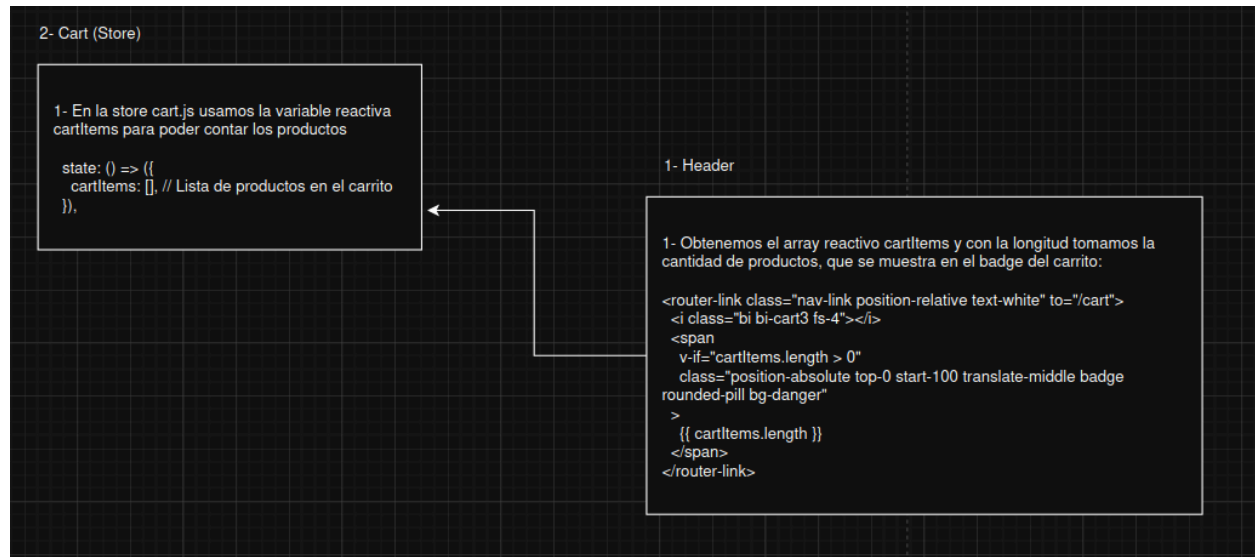
Productos relacionados



Carrito: Productos que se agrega



Carrito de compras contador reactivo



Mostrar los productos de finalizar compra

