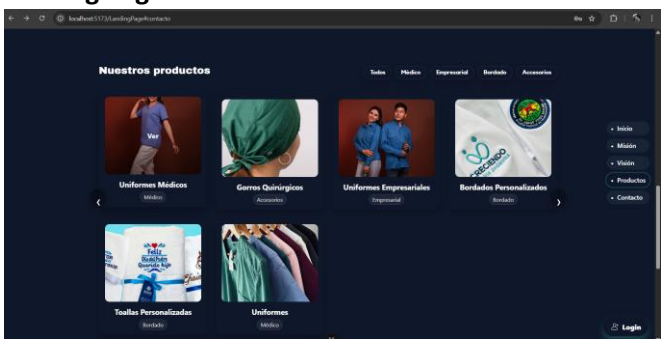
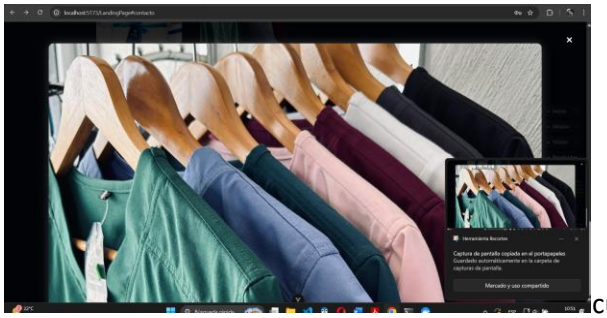
Investigación para Refactorización:

Vistas

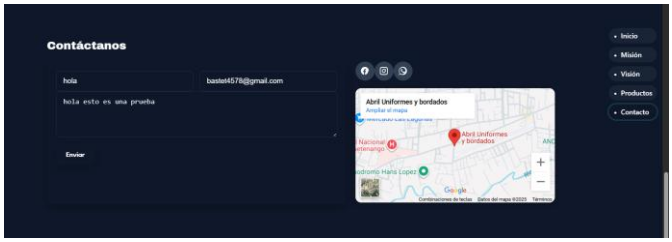
1. **Landing Page**



las imágenes no se cargan automáticamente, cuando se hace scroll hacia arriba o hacia abajo las imágenes vuelven a cargase.

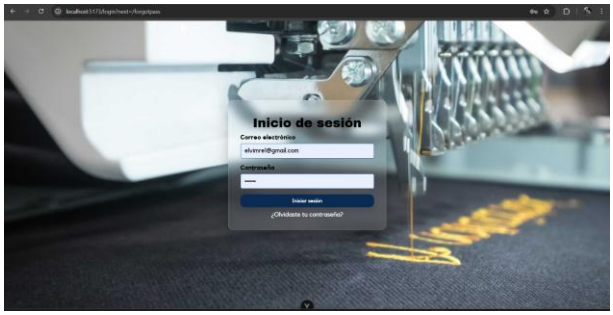


cuando se da clic para ampliar las imágenes en el catálogo de productos estas tienen un tamaño muy grande.



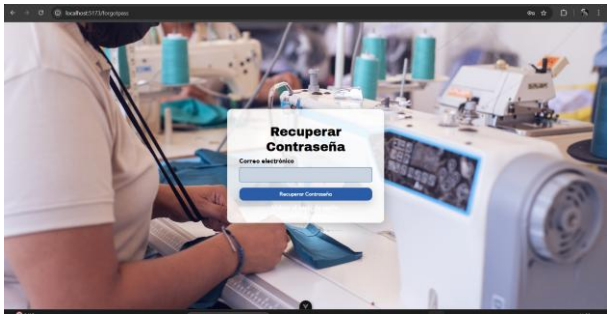
Este apartado no sirve de nada porque cuando se da en enviar redirige al WhatsApp de la empresa xd

2. **Login**



Todo bien, aunque algo hay que cambiar en cuanto a colores porque es lo que nos critican jsjs

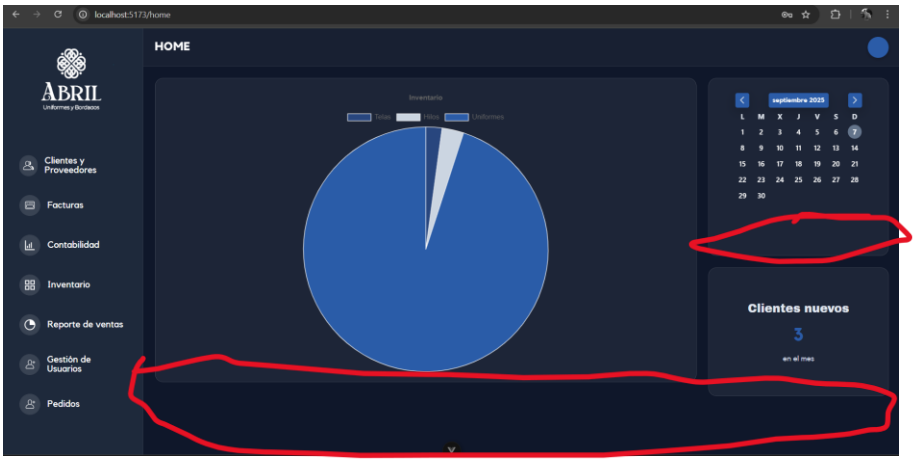
3. **Forgot pass**



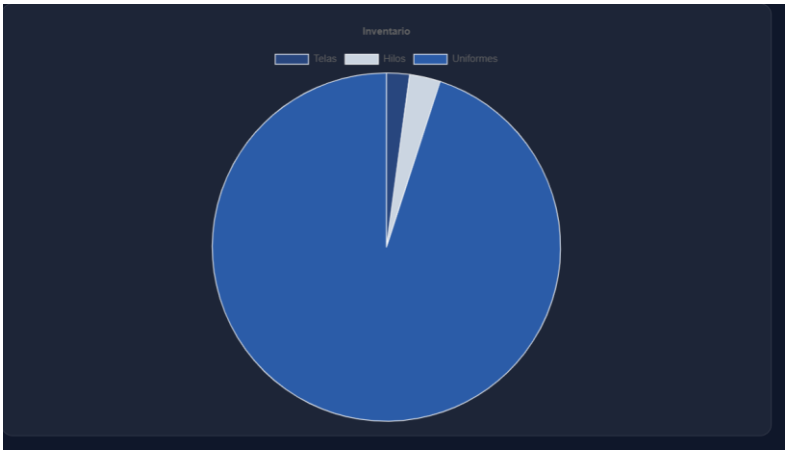
*en login “¿olividaste tu contraseña?” al hacer clic no te lleva a esta página.

Cambiar el color del texto “volver a iniciar sesión” y hacerlo funcional.

4. **Home**



Hacer que todos los widgets estén alineados

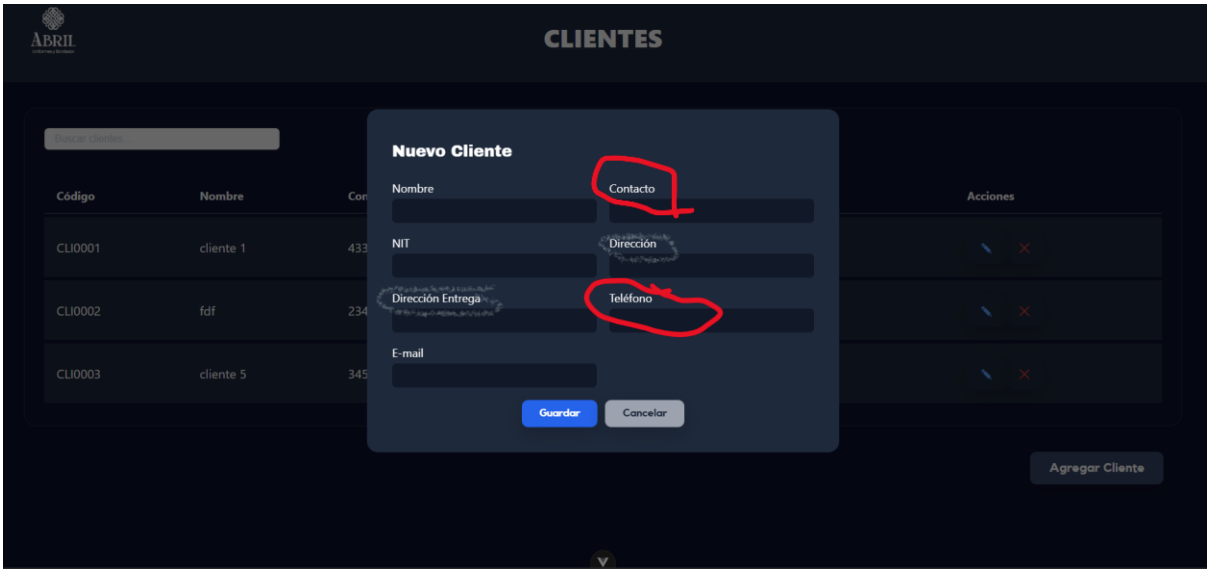


Cambiar paleta de colores en general.

5. **Clientes (si se pudiera cambiar el nombre vista porque es un poco confuso que sean clientes cuando también son proveedores)**

Centrar un poco mas las tablas

6. **Clientes registro**



*Quitar una de esas, verificar cual es la que guarda el dato. (desde el semestre pasado estamos con esto xd)
*¿Dirección de entrega y dirección?

acciones de clientes y acciones de proveedores no son iguales.

7. Proveedores

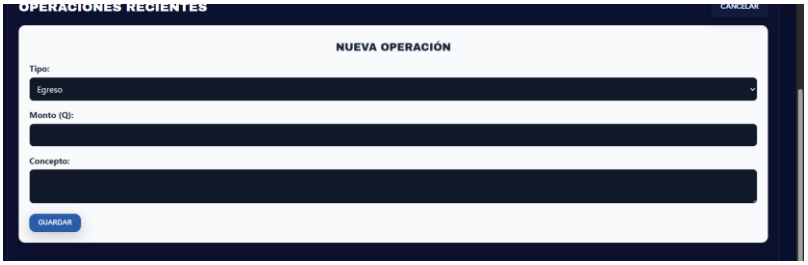
La tabla de proveedores no es igual a la de clientes.

8. Bill page

Todo bien, solo mucho azul (paleta de colores en general).

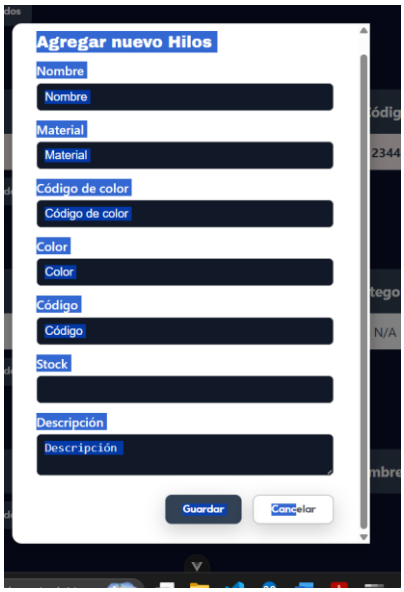
9. **Accounting**

Mucho azul

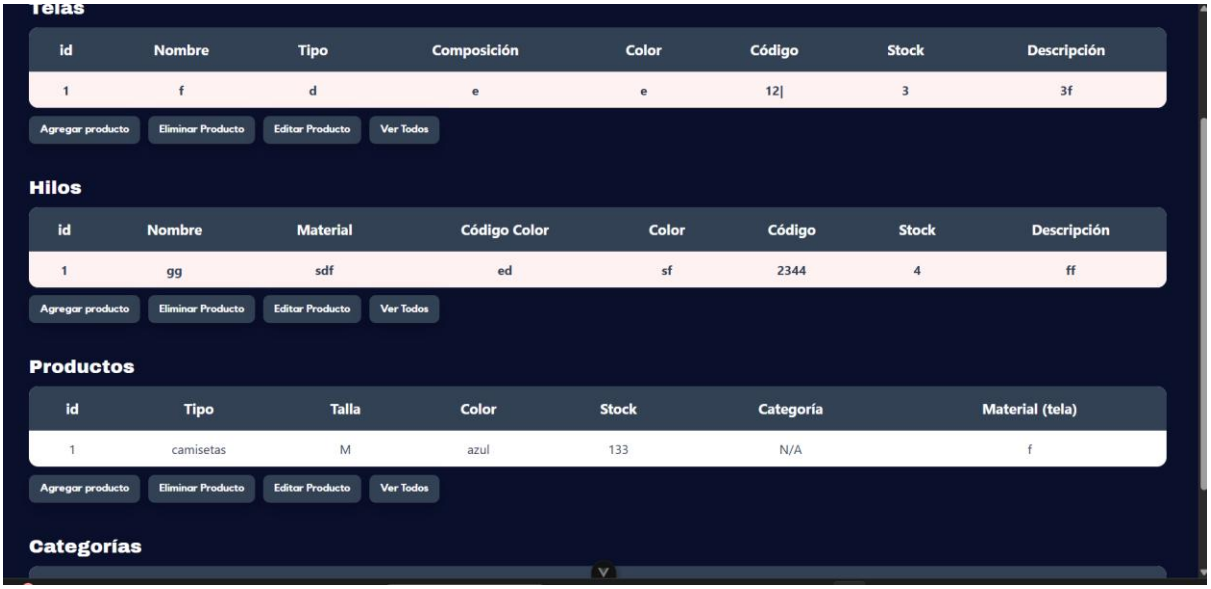


Todo bien

10. **Mi_inventario**



no se ven los textos



No tienen el mismo estilo (en telas e hilos no es blanco el fondo)

11. Reporte de ventas:

Agregar más colores porque predomina el azul

12. Register

Todo bien

13. Pedidos (cambiar el nombre del archivo de la vista “envíos”)

Cambios para realizar en base a deuda técnica y reglas de refactorización

---

- múltiples archivos CSS, variables duplicadas y conflictos entre estilos globales y locales
- lógica y estilos UI duplicados (tablas, modales y formularios)
- Hay 4 sistemas de diseño es decir 4 archivos .css mas los estilos locales que se definen en cada vista.

Plan 1: Refactorización de STYLES (Con la Nueva Paleta)

Objetivo: Crear una única fuente de verdad (styles.css) para todos los estilos, eliminar 4 sistemas de CSS en conflicto y aplicar una nueva paleta de colores profesional donde el azul predomine, pero no sea la *única* opción.

1. Ir a styles.css y remplazar el root por el que se va definir.
2. **Eliminar Duplicados:** Borrar los bloques :root y @font-face
3. **Limpiar Globales:** Borrar reglas de layout (.sidebar, .topbar, .chart-area, etc.) de styles.css se manejan desde el styles scoped del home.
4. **Centralizar Utilidades:** Mover la clase .nav-search (copiada en 4 vistas) a styles.css para que sea una utilidad global.

Plan 2: Refactorización de COMPONENTES UI (DRY)

Problema: Múltiples tablas, modales y formularios duplicados. **Acción:** Crear una carpeta /src/components/base/ y crear:

- <BaseTable.vue> (para reemplazar las 6 tablas).
- <BaseModal.vue> (para reemplazar los 4 modales).
- <BaseInput.vue>, <BaseSelect.vue>, <BaseButton.vue> (para reemplazar los 3 formularios de auth y estandarizar todos los demás).

(**Plan 3** (Lógica/API/Estado: Servicios + Pinia)

Plan 4 (SRP: Aislar PDF/Descomponer God Component)

Plan 5 (Infra: Arreglar Router/Tests)

1. Landing page:

Tarea de UI/UX:

- (Bug Imagen Modal): Arreglar el CSS del modal de imagen; la imagen debe tener max-width: 90vw y max-height: 90vh para que no se salga de pantalla.
- (Bug Carga Imagen): Asegurarse de que las imágenes del carrusel usen <img loading="lazy"> para optimizar la carga.

Tarea de Arquitectura (Plan 1 y 4):

- **Eliminar** el bloque: root local y reemplazar los usos de CSS por las nuevas variables semánticas globales (ej. var(--color-action-primary)).
- Extraer la sección del catálogo de productos (carrusel, filtro, modal) a un nuevo componente <ProductShowcase.vue> (Plan 4).

2. Login

- **Tarea de UI/UX:**
    - (Paleta de Colores): La crítica de "mucho azul" se soluciona con el **Plan 1**. El botón usará <BaseButton> que usa var(--color-action-primary). Al definir esto como azul, mantenemos la predominancia de marca que pediste, pero de forma controlada.
- **Tarea de Arquitectura (Plan 2 y 3):**
    - Reemplazar el formulario (input/button) por <BaseInput> y <BaseButton> para eliminar CSS duplicado.
    - **Eliminar** import apiFetch y la escritura manual a sessionStorage.
    - Refactorizar el submit para que llame a useAuthStore.login(payload)

3. Forgot Pass

- **Tarea de UI/UX (BUGS):**
    - El link "volver a iniciar sesión" debe ser un <router-link to="/login"> para que funcione (actualmente es un <a href="#"> roto).
    - En login.vue, el link "¿Olvidaste tu contraseña?" debe ser un <router-link to="/forgotpass">

Tarea de Arquitectura (Plan 2, 3 y 5):

- En index.js, cambiar la ruta /forgotpass a meta: { requiresAuth: false } (Bug crítico de router).
- Reemplazar formulario por <BaseInput>/<BaseButton>.
- Eliminar apiFetch y llamar a useAuthStore.requestPasswordReset(email).

4. Home

- **Tarea de UI/UX:**
    - (Alineación/Color): se resuelven con el plan 1
    - **Acción: Eliminar** las reglas de layout (.sidebar, .chart-area, etc.) de styles.css global. Esto le dará control total al <style scoped> de home.vue, solucionando conflictos de alineación.
    - **Acción:** Usar las nuevas variables de en los KPIs (como "Clientes nuevos") para romper la monotonía del azul.
- **Tarea de Arquitectura (Plan 3):**
    - **Eliminar** import apiFetch, import bus (el listener bus.on) y la función pickDateField duplicada.
    - Importar los Stores (Clients, Inventory) y leer los KPIs directamente de los getters (ej. clientsStore.nuevosClientesDelMes).

5. Clientes (Vista)

- **Tarea de UI/UX:**
    - Centrar las tablas (cards). Renombrar la vista/ruta.
- **Tarea de Arquitectura (Plan 1 y 3):**
    - **CRÍTICO: Eliminar** los bloques :root y @font-face duplicados de este archivo.
    - Eliminar apiFetch. Leer los datos (.length) directamente de useClientsStore y useProvidersStore.

6. Clientes Registro

- **Tarea de UI/UX:**
    - (Duplicación): Eliminar la barra de búsqueda del cuerpo (.search-wrapper) y dejar solo la del slot del NavBar. (Esa barra usará el estilo global .nav-search que va estar en .css

Tarea de Arquitectura (Plan 2 y 3):

- Reemplazar la tabla y modal por <BaseTable> y <BaseModal ambos (Clientes y Proveedores) usarán el mismo componente base.

  Eliminar apiFetch, bus.emit y la lógica KPI duplicada (pickDateField). Refactorizar todo para usar useClientsStore.

7. Proveedores

- **Tarea de UI/UX:**
  - (Inconsistencia): hacer que la tabla sea igual a la de clientes
- **Tarea de Arquitectura (Plan 2 y 3):**
  - **Solución:** Reemplazar la tabla y modal por <BaseTable> y <BaseModal>. Esto *fuerza* a que sea idéntica a la vista de Clientes (que también será refactorizada).
  - Eliminar apiFetch y refactorizar para usar el nuevo useProvidersStore.

8. Bill Page

- **Tarea de UI/UX:**
  - (Paleta): "Mucho azul".
- **Tarea de Arquitectura (Plan 1, 3 y 4):**
  - Refactorizar los botones (ej. .btn-download) para usar las nuevas variables semánticas
  - **CRÍTICO (Plan 4):** Aislar la lógica de jsPDF. Mover todo el código de onDownloadPDF a un archivo util /utils/pdfGenerator.js.
  - Refactorizar apiFetch para usar useBillingStore.

9. Accounting

- **Tarea de UI/UX:** "Mucho azul".
- **Tarea de Arquitectura (Plan 1, 2 y 3):**
  - Usar la nueva paleta de acentos (Verde/Rojo) en los KPIs de Ingresos/Gastos.
  - Reemplazar la tabla (.operations-table) por <BaseTable>.
  - Usar la clase global .nav-search (el CSS copiado se elimina).
  - Refactorizar apiFetch y la lógica calculateSummary al nuevo useAccountingStore (el resumen será un getter).

10. Mi_Inventario

- **Tarea de UI/UX:**

    - (Inconsistencia): "No tienen el mismo estilo (telas e hilos no es blanco el fondo)". Esto es 100% correcto, valida la inconsistencia de tablas.
    - (Contraste): "no se ven los textos".

- **Tarea de Arquitectura (Plan 1, 2, 3 y 5):**
    - **Solución:** Reemplazar TODAS las tablas y el modal por <BaseTable> y <BaseModal>. El componente base tendrá el estilo correcto (fondo blanco, texto oscuro) solucionando AMBOS problemas de UI.
    - Usar la clase global .nav-search.
    - **Refactor Lógica (CRÍTICO):** Este es el "God Component". Eliminar apiFetch y event-bus (ambos on y emit).
    - **Acción Principal (Plan 5):** Descomponer esta vista. Crear sub-componentes (ej. <InventarioTelas>, <InventarioHilos>) y mover la lógica de API/Estado al nuevo useInventoryStore.

11. Reporte de Ventas

- **Tarea de UI/UX:**
    - (Paleta): "Agregar más colores porque predomina el azul".
- **Tarea de Arquitectura (Plan 1, 2 y 3):**
    - **Solución:** Modificar los métodos render...Chart. Crear un array con la nueva paleta (ej. [getCssVar('--color-action-primary'), getCssVar('--color-accent-info'), getCssVar('--color-accent-success'), ...]) y pasarlo al backgroundColor de los datasets de Chart.js.
    - Reemplazar la tabla (.dark-table) por <BaseTable>.
    - Refactorizar las 4 llamadas apiFetch para que sean una sola acción en useReportsStore (que por dentro hará un Promise.all).

12. Register

- **Tarea de UI/UX:** "Todo bien".
- **Tarea de Arquitectura (Plan 2 y 3):**
    - Refactorizar formulario a <BaseInput>, <BaseSelect>, <BaseButton>.
    - Refactorizar apiFetch para llamar a useAuthStore.register(payload).

13. Pedidos (Archivo: envios.vue)

- **Tarea de UI/UX:** Renombrar el archivo.
- **Diagnóstico de Arquitectura:** Re-implementa fetch/CSRF, usa localStorage manual, define URLs de API locales. Tiene tabla y modal únicos.
- **Plan de Acción:**
    1. **Acción Dev:** Renombrar el archivo a pedidos.vue y actualizar index.js.

2. **Lógica (Plan 3 - CRÍTICO): Eliminar** las funciones locales req y getCSRF. **Eliminar** la lógica de localStorage. **Eliminar** la lista de constantes de URLs. Refactorizar TODA la vista para que use el nuevo useOrdersStore.
3. **UI (Plan 2):** Reemplazar la tabla y el modal por <BaseTable> y <BaseModal>. Usar la clase global .nav-search.

Por qué usar pinia

Pinia es una librería de vue para la gestión de estado

usar Pinia nos permite guardar los datos (como la lista de clientes) en un "Store" central la primera vez que los pedimos. Así, evitamos que múltiples vistas (como home.vue, clientes.vue, y clientesregistro.vue) estén haciendo la misma llamada fetch cada vez que entramos a ellas. Esto también elimina el "código espagueti" del event-bus.js, porque ya no necesitamos "gritar" con bus.emit para forzar a otras vistas a recargar; como todas las vistas leen del mismo Store, se actualizan solas automáticamente cuando ese dato central cambia.