Diagrama de Flujo de Navegación y Datos

Este diagrama muestra cómo fluyen la navegación y los datos entre componentes y servicios al cargar y usar la pantalla de Centros de Costo.

Ver diagrama completo de navegación y flujo de datos

Componente CentrosDeCosto

Descripción General

Componente principal que orquesta la carga, filtrado, paginación y acciones de exportación de los centros de costo. Controla el estado global y muestra el componente de detalle al consultar una fila.

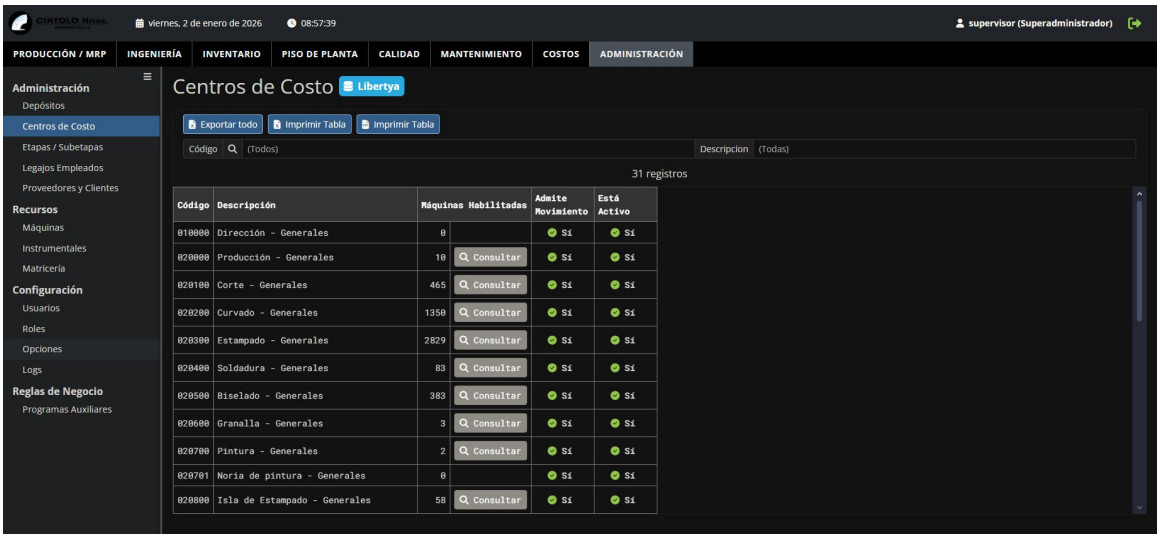

Figura 1 – Pantalla principal de Centros de Costo con listado, filtros y acciones de consulta, exportación e impresión.

Endpoints Utilizados

- **GET /centrosDeCosto**: Obtiene la lista completa.
- **GET /centrosDeCosto/exportar/excel** (vía `useDescargarExcel`): Descarga todos en Excel.
- No hay endpoint para impresión; se usa lógica cliente en `useImprimirTabla`.

Estados y Hooks

- `centrosDeCosto: centroDeCostoType[] | null` → almacena datos de API.

- `pagina: number` → página actual de tabla.
- `useTraerConfiguracion()` → controla `cargando` y mensajes.
- `useFiltrar()` → gestiona texto de búsqueda.
- `useDescargarExcel()` → prepara descarga.
- `useImprimirTabla(tablaRef.current, 'Centros-de-costo')` → genera PDF/Excel.
- `useMostrarError()` → muestra alertas de fallo.

Flujo de Datos

- On mount → `hacerPeticionApi()` llama GET `/centrosDeCosto`.
- Al recibir datos → `setCentrosDeCosto(ccs)`.
- Cada cambio en filtro → `useMemo` recalcula `muestra` y resetea `pagina`.
- Tabla recibe `muestra` y `pagina`.
- Clic en Consultar → abre modal con detalle de máquinas.

Posibles Fallas en la Lógica ❗

- Se comenta la línea de **`centroDeCosto`**, pero luego se usa, causa error de compilación.
- Llamar `setPagina(1)` dentro de `useMemo` causa re-render infinito.
- El filtro de código usa `st.id` en lugar de `st.cc`, no coincide con etiqueta “Código”.

Estado comentado

La variable centroDeCosto está comentada y provoca error en tiempo de compilación.

Filtrado inseguro

Mover setPagina fuera de useMemo evita bucles de render innecesarios.

Fragmento de Código

tsx

```tsx
useEffect(() => {
  hacerPeticionApi().then(respuesta => {
    config.setCargando(false);
    const ccs = respuesta.datos?.centrosDeCosto as centroDeCostoType[] | undefined;
    if (!ccs) {
      mostrarError(respuesta);
      return;
    }
    setCentrosDeCosto(ccs);
  });
}, []);
```

Componente CentrosDeCostoTabla

Descripción General

Tabla que muestra paginación, estado y botón de consulta de cada centro. Incorpora estilos dinámicos y valida presencia de datos.

Propiedades

- `centrosDeCosto: centroDeCostoType[] \| null` → datos filtrados.
- `pagina: number` → página actual.
- `setCentroDeCosto` → función para mostrar detalle.
- `tablaRef: RefObject<HTMLTableElement>` → referencia para impresión.

Render y Estilos

- Genera estilos con `generarEstilosDeCabezaDeTabla`.
- Usa `<style>` interno para cabecera fija.
- Define columnas: Código, Descripción, Máquinas, Movimiento, Activo.

Lógica de Paginación

- Aplica `paginado(centrosDeCosto, pagina)`.
- Si no hay elementos, muestra `<NoHayElementos />`.

Validaciones y Componentes Auxiliares

- Botón **Consultar** solo si hay máquinas activas.
- Muestra estado con `<SiNo formato={1} />`.
- Usa `NoHayElementos` para lista vacía.

Fragmento de Código

tsx

```tsx
<tbody>
  {!!centrosDeCosto?.length
    ? paginado(centrosDeCosto, pagina).map(cc => (
        <tr key={cc.id}>
          <td className='text-center-d'>{cc.cc}</td>
          <td>{cc.descripcion}</td>
          <td className='text-end-d'>
            {cc.maquinas?.filter(m => m.estaActiva).length || 0}
          </td>
          <td className='text-center-d' style={{ width: '120px' }}>
            {(cc.maquinas?.filter(m => m.estaActiva).length || 0) > 0 && (
              <button
                className='btn btn-sm btn-secondary py-1'
                onClick={() => setCentroDeCosto(cc)}
              >
                <i className='fas fa-search' /> Consultar
              </button>
            )}
          </td>
          <td className='text-center-d'>
            <SiNo formato={1} valor={cc.admiteMovimiento} />
          </td>
          <td className='text-center-d'>
            <SiNo formato={1} valor={cc.estaActivo} />
          </td>
        </tr>
```

```
      ))
    : <NoHayElementos />
  }
</tbody>
```

Componente CentrosDeCostoMaquinas

Descripción General

Modal que lista las máquinas activas de un centro de costo. Permite consultar detalle y cerrar el diálogo.

Figura 2 - Vista de detalles de Máquinas activas de un Centro de Costo.

Propiedades

- `centroDeCosto: centroDeCostoType` → centro seleccionado.
- `setCentroDeCosto` → cierra modal al pasar `null`.

Render del Modal

- Usa clases Bootstrap para modal grande.
- Incluye estilo interno para límite de altura y scroll.

Filtrado de Máquinas

- Filtra con `m.estaActiva`.
- Ordena por `id`.
- Muestra código con `codMaquina(m)` y detalles.

Cierre del Modal

- Botón **Cerrar** llama `setCentroDeCosto(null)`.
- Fondo semitransparente detiene interacción con fondo.

Fragmento de Código

tsx

```
{!!centroDeCosto.maquinas?.length && centroDeCosto.maquinas
  .filter(m => m.estaActiva)
  .sort((m1, m2) => m1.id - m2.id)
  .map(m => (
    <div key={m.id} className='form-control mb-3'>
      <div className='form-control' style={{ border: 'none' }}>
        <label>{codMaquina(m)} - {m.descripcion} ({m.marca})</label>
      </div>
    </div>
))}
```

Con esta documentación tienes una visión clara de la estructura, flujo de datos y potenciales mejoras en la lógica de los componentes de Centros de Costo.