Diagrama de flujo de navegación y datos

Este diagrama muestra el recorrido de usuario y la interacción con los servicios API para los componentes de Libro Diario, Cierre Contable y Ejercicios Fiscales.

Endpoints utilizados

- **GET /api/asientosContables**

Recupera `asientosContables`, `centrosDeCosto` y `cuentasContables`.

- **POST /api/asientosContables**

Envía un nuevo **asientoContable** y retorna la lista actualizada.

- **GET /api/ejerciciosFiscales**

Obtiene todos los **periodos fiscales**.

Componentes

LibroDiario.tsx

Componente principal de la sección **Libro Diario**. Administra carga de datos, filtros y alterna entre lista y formulario de alta.

Funcionalidad principal

- Al montar, invoca `hacerPeticionApi()` para cargar datos.
- Controla estado de carga con `useTraerConfiguracion`.
- Alterna vista entre **tabla** y **formulario**.
- Permite exportar todo a Excel con `useDescargarExcel`.
- Implementa filtros locales con `useFiltrar`.

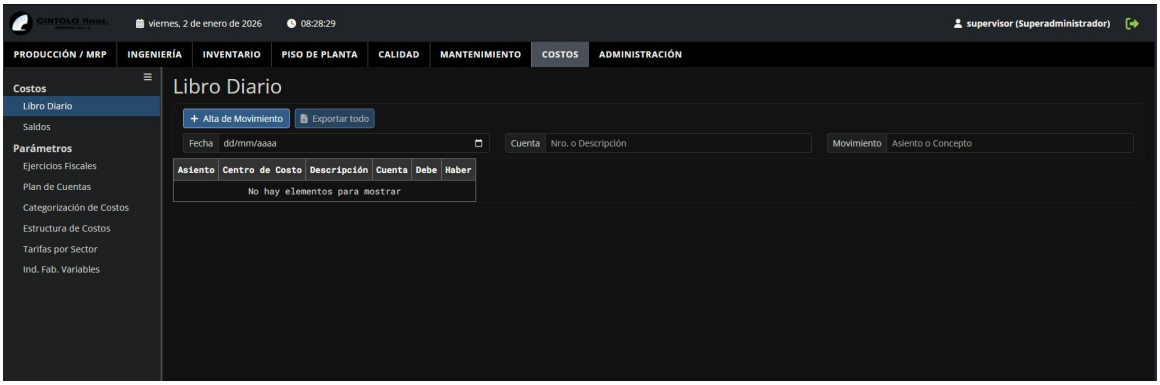

Figura 1 – Pantalla principal de Libro Diario con listado, filtros, acciones de alta y exportación.

Hooks y servicios

- `useTraerConfiguracion()` para estado global de carga.

- `useDescargarExcel()` dispara descarga de Excel.
- `useFiltrar()` guarda filtros de fecha, descripción1 y descripción2.
- `useMostrarError()` muestra alertas en caso de fallo.
- `useTraerUsuario()` gestiona permisos de usuario.

Flujo de datos

js

```
useEffect(() => {
  hacerPeticionApi().then(respuesta => {
    config.setCargando(false);
    const dld = respuesta.datos?.asientosContables;
    const ccs = respuesta.datos?.centrosDeCosto;
    const cuc = respuesta.datos?.cuentasContables;
    if (!dld || !ccs || !cuc) {
      mostrarError(respuesta);
      return;
    }
    setAsientosContables(dld);
    setCentrosDeCosto(ccs);
    setCuentasContables(cuc);
  });
}, []);
```

Subcomponentes

- **LibroDiarioTabla**: Muestra los asientos en tabla.
- **LibroDiarioAgregar**: Formulario para alta de nuevo asiento.

LibroDiarioTabla.tsx

Renderiza en tabla los registros de **asientoContable** con estilos y formato.

Props

- `datosLibroDiario: asientoContableType[] | null`

Renderizado

- Aplica estilo de encabezado con `generarEstilosDeCabezaDeTabla`.
- Si no hay datos, muestra `<NoHayElementos />`.
- Agrupa líneas de cada asiento usando `<Fragment>` y `rowSpan`.

Formatos y utilidades

- `formatearFecha(fecha, false, true)`
- `formatoMoneda(importe)`
- Advertencia cuando `idErp` es 0.

LibroDiarioAgregar.tsx

Formulario para crear un nuevo **asiento contable**. Gestiona líneas de Debe y Haber.

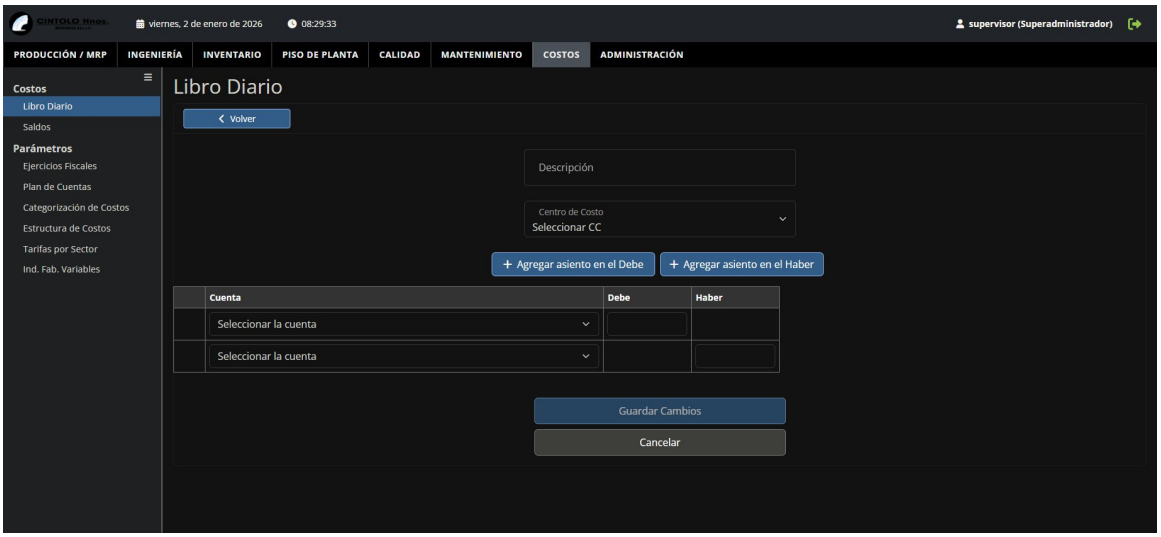

Figura 2 – Formulario de alta de un nuevo Asiento Contable con carga descripción, Centro de costos y cuenta; y gestión de líneas de Debe y Haber.

Props

- `centrosDeCosto: centroDeCostoType[] | null`
- `cuentasContables: cuentaContableType[] | null`
- `setAsientosContables: Dispatch<…>`
- `volver: () => void`

Estado interno

- `asientoContable`: inicia con 2 líneas (Debe + Haber).
- `guardando`: controla botón de envío.
- `esValido`: valida que suma Debe = suma Haber > 0 y campos completos.

Lógica de suma

js

```
const esValido = useMemo(() =>
  asientoContable.centroDeCostoId > 0 &&
  asientoContable.descripcion.length > 5 &&
  sumaDebe === sumaHaber &&
  sumaDebe > 0 &&
  !asientoContable.lineas.some(x => x.importe <= 0),
[asientoContable]);
```

Envío a API

js

```
const agregarAsientosContables = async () => {
  setGuardando(true);
  const respuesta = await hacerPeticionApi({
    method: 'POST',
    body: { asientoContable }
  });
  setGuardando(false);
  if (!respuesta.datos?.asientosContables) {
```

```
    mostrarError(respuesta);
    return;
  }
  setAsientosContables(respuesta.datos.asientosContables);
  volver();
};
```

Contabilizacion.tsx

Esqueleto de componente para **Cierre Contable**. Carga datos y muestra filtros, pero la tabla está comentada.

Estado y hooks

- Misma llamada `hacerPeticionApi()` que **Libro Diario**.
- Controla estado de carga, error y descarga de Excel.
- Define `filtro` local, pero no aplica filtrado.

Observación

La lógica de renderizado de la tabla o formulario está comentada, por lo que actualmente no muestra datos.

EjerciciosFiscales.tsx 📊

Administra la lista paginada de **ejercicios fiscales** y exportación a Excel.

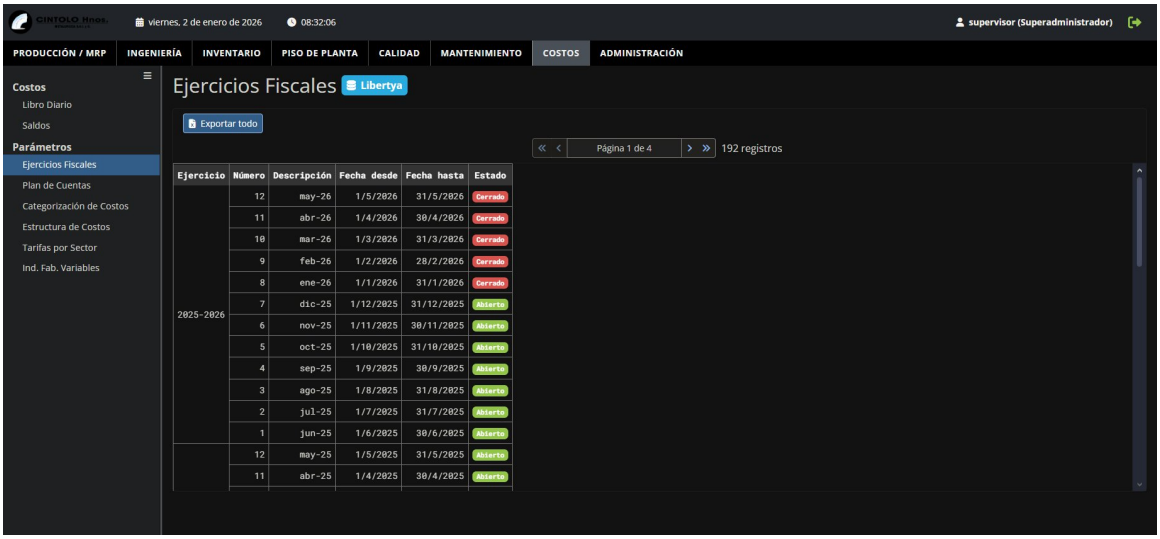

Figura 1 – Pantalla principal de Ejercicios Fiscales con listado, acción de exportación.

Flujo de datos

`js`

```
useEffect(() => {
```

```
  hacerPeticionApi().then(respuesta => {
    config.setCargando(false);
    const def = respuesta.datos?.ejerciciosFiscales;
    if (!def) {
      mostrarError(respuesta);
      return;
    }
    setperiodosFiscales(def);
  });
}, []);
```

Paginación

- Usa `useState(página)` y `ControlPaginas`.
- `useMemo` reinicia la página al cambiar `periodosFiscales` (side effect en useMemo).

Componentes internos

- **BadgeLibertya**: ícono de marca.
- **ControlPaginas**: navegación entre páginas.
- **EjerciciosFiscalesTabla**: tabla con 12 registros por página.

EjerciciosFiscalesTabla.tsx

Muestra un bloque de 12 periodos fiscales agrupados por año.

Props

- `pagina: number`
- `periodosFiscales: periodoFiscalType[] | null`

Paginación y agrupación

- `paginado(array, página)` extrae 12 items.
- Rowspan en columna de año cada 12 filas.
- Badges para estado: **Cerrado** (rojo) o **Abierto** (verde).

⚠ Fallas de lógica

- Filtros definidos en **LibroDiario** y **Contabilizacion** no se aplican a la lista mostrada.
- **Contabilizacion.tsx** no renderiza la tabla; el código está comentado.
- Uso de `useMemo` con `setPagina(1)` en **EjerciciosFiscales** produce side effect dentro de memoización.
- Generación de `id` con `Date.now()` en **LibroDiarioAgregar** puede colisionar si se crea rápidamente.
- En **LibroDiarioTabla**, los títulos de `titulosDeTabla` no coinciden con las columnas efectivas en `<thead>`.