Diagrama de flujo de navegación y datos

Este diagrama ilustra cómo fluye la navegación entre componentes y cómo se intercambian datos con la API.

✏ DetalleMantenimiento.tsx

Este componente muestra y edita un **mantenimiento** existente. Recibe datos y estado de edición desde el padre.

- **Props principales**
- `mantenimientoMaquina`: Objeto con datos del mantenimiento.
- `modoEdicion`: Habilita o deshabilita edición de campos.
- `guardarCambios`: Callback para persistir los cambios.
- **UI**
- Campos de texto en solo lectura para descripción.
- Listado de tareas e insumos con botón *Agregar*.
- Select para responsable y checkbox de detención de producción.
- DatePicker para fecha de inicio y fin.

Endpoints utilizados

- Este componente **no** invoca directamente la API.
- La función `guardarCambios` suele mapearse a un **PUT /v1/mantenimientos/{id}** en el padre.

Observaciones y posibles fallas

- El select de responsable usa `onChange` para actualizar `responsable` en vez de `responsableId`, provocando que no cambie el ID correcto.
- Los modales de tareas e insumos están como *placeholders* (`<>Mostrar agregar...</>`), no importan ni muestran `ModalTareas` ni `ModalInsumos`.

estado.tsx

Renderiza un **badge** con ícono y color según el estado de un mantenimiento.

- **Función**: `generarEstado(estadoId, estadoDescripcion)`
- **Estados**
- 1: Fecha programada (azul).
- 2: En curso (spinner verde).
- 3: Completado (check verde).
- 4: Fallido (times rojo).
- otros: Desconocido (dark).

tsx

```
export const generarEstado = (estadoId: number, estadoDescripcion: string) => {
  if (estadoId === 1) {
    return (<span className='text-info'><i className='fas fa-calendar-alt me-
2'/>{estadoDescripcion}</span>);
  } else if (estadoId === 2) {
    return (<span className='text-success'><i className='fas fa-spin fa-spinner me-
2'/>{estadoDescripcion}</span>);
  } else if (estadoId === 3) {
    return (<span className='text-success'><i className='fas fa-check me-
2'/>{estadoDescripcion}</span>);
  } else if (estadoId === 4) {
    return (<span className='text-danger'><i className='fas fa-times-circle me-
2'/>{estadoDescripcion}</span>);
  } else {
    return (<span className='text-dark'><i className='fas fa-circle-xmark me-
2'/>{estadoDescripcion}</span>);
  }
}
```

Endpoints utilizados

- No realiza llamadas a la API.

Observaciones y posibles fallas

- Lógica clara y simple. No se detectan errores.

🏷 ModalTareas.tsx

Modal para **agregar tareas** a un mantenimiento.

- **Props**
- `tareas`: Lista completa de tareas disponibles.
- `mantenimientoMaquina`: Estado actual del mantenimiento.
- `setMantenimientoMaquina`: Actualiza el mantenimiento padre.
- `cerrarModal`: Cierra la ventana modal.
- **Flujo**
- Filtra tareas ya agregadas.
- Permite seleccionar una tarea y añadirla al array.
- Deshabilita botón si no hay selección.

tsx

```
<button
  onClick={() => {
    const tarea = tareas?.find(a => a.id === tareaId);
    if (!tarea) return;
    setMantenimientoMaquina(x => ({ ...x, tareas: [...x.tareas, tarea] }));
    cerrarModal();
  }}
  disabled={!(tareaId > 0)}
>
  <i className='fas fa-plus'/> Agregar
</button>
```

Endpoints utilizados

- No invoca la API.

Observaciones y posibles fallas

- Funcionamiento coherente. No se observan errores lógicos.

ModalInsumos.tsx

Modal para **agregar insumos** al mantenimiento.

- **Props**
- `insumos`: Catálogo de artículos.
- `mantenimientoMaquina`: Estado de mantenimiento.
- `setMantenimientoMaquina`: Setter del padre.
- `cerrarModal`: Cierra la ventana.
- **Flujo**
- Excluye los ya seleccionados.
- Añade el insumo elegido al array `insumos`.

tsx

```
<button
  onClick={() => {
    const insumo = insumos?.find(a => a.id === insumoId);
    if (!insumo) return;
    setMantenimientoMaquina(x => ({ ...x, insumos: [...x.insumos, insumo] }));
    cerrarModal();
  }}
  disabled={!(insumoId > 0)}
>
  <i className='fas fa-plus'/> Agregar
</button>
```

Endpoints utilizados

- No realiza llamadas a la API.

Observaciones y posibles fallas

- Lógica correcta y consistente.

🆕 MantenimientoAgregar.tsx

Formulario para **crear** un mantenimiento (preventivo o correctivo).

- **Props**
- `esPreventivo`: Define tipo de mantenimiento.
- Listas de `maquinasHerramientas`, `tareas`, `insumos`, `usuarios`.
- **Estado interno**
- `mantenimientoMaquina`: Datos iniciales via `mantenimientoMaquinaInicial`.

- Flags para mostrar modales.
- **Validaciones clave**
- Causa > 6 caracteres.
- Fecha fin > fecha inicio.
- Responsable seleccionado.
- Al menos una tarea.
- **Flujo de guardado**
- Convierte fechas JS a SQL.
- Llama a `hacerPeticionApi` con método `POST`.
- Actualiza estado con respuesta del servidor.
- Muestra error en caso de fallo.

Endpoint utilizado

Crear Mantenimiento (POST)

Observaciones y posibles fallas

- Gestión de errores adecuada gracias a `useMostrarError`.
- Validaciones cubren casos básicos.

_ModuloMantenimiento.tsx

Punto de entrada al **módulo de mantenimiento**. Monta el enrutador secundario según permisos.

- **Función principal**
- Obtiene rutas disponibles del usuario.
- Busca la ruta de mantenimiento en `rutasMantenimiento`.
- Renderiza `EnrutadorSecundario` si la ruta existe.

tsx

```
const ruta = usuario.rutas.find(r => r.path === rutasMantenimiento.path);
return ruta && <EnrutadorSecundario ruta={ruta}/>;
```

Endpoints utilizados

- No realiza peticiones a la API.

Observaciones y posibles fallas

- Lógica clara. No presenta errores detectados.