# 09 — Layout e menu lateral

# 09 — Layout e menu lateral

> O shell da aplicação: `AppBar`, `Drawer` com menu em árvore, área de conteúdo. `menuTree.ts` é a fonte única do menu — nenhum item é escrito em JSX. A navegação é **híbrida**: cada folha do menu abre por **rota** ou por **aba**, e o shell suporta as duas ao mesmo tempo.

## O shell

```
<Main>                        ← elemento da rota protegida
  <TabsProvider>
    <MainContent>
      <AppBar>                título + sair      <Drawer>                menu a partir de
menuTree
      <Box component="main">        <TabBar />            abas abertas (some se não
houver)        <Box>                 ← <Outlet/>: rota atual, visível quando activeTabId ===
null
        {tabs.map(...)}       ← painéis de aba, todos montados, só o ativo visível
```

`Main` é elemento da rota protegida em `AppRouter` (10). Por isso `TabsProvider` pode usar `useNavigate`: já está dentro do router.

## Os dois modos de navegação

| Modo | O que acontece ao clicar | Estado da tela | URL muda? |
|---|---|---|---|

| | | | |
|---|---|---|---|
| `"route"` | `navigate("/" + path)` e zera a aba ativa | Perdido ao sair | Sim |
| `"tab"` | `openTab(node)` — painel novo, fica montado | **Preservado** enquanto aberta | Não |

Escolha por tela:

- **`"route"`** para tela de entrada e telas que se quer linkáveis/favoritáveis. Dashboard sempre.
- **`"tab"`** para telas de trabalho: cadastro, consulta, movimentação — onde o usuário alterna entre várias sem perder o que preencheu.

# Por que abas preservam estado

Todos os painéis de aba ficam montados; só o ativo é visível (`display: none` nos demais). Um formulário meio preenchido continua lá ao voltar. Isso tem duas consequências que **não** são opcionais:

1. O `UseCaseManager` da página usa `autoClose={false}` — o caso de uso permanece aberto no servidor enquanto a aba existir (06).
2. Cleanup de `useEffect` **não** dispara ao fechar a aba, porque o componente não desmonta por conta disso. Fechar a aba precisa ser explícito — ver `useTabCloseCallback`.

## `menuTree.ts`

```
// src/routes/menuTree.ts
import React from "react";
import type { Session } from "@/lib/curio";
import { PATHS } from "./paths";
/** Como a folha do menu abre: navegando pela rota, ou numa aba. */export type MenuOpenMode =
"route" | "tab";
/**
 * Modo usado quando o no nao declara `mode`.
 * Troque para "route" se o produto nao usar abas.
 */
export const DEFAULT_OPEN_MODE: MenuOpenMode = "tab";
export interface MenuNode {
  label: string;
```

```
  icon?: React.ComponentType;  /** Só em folha (sem children) — caminho de rota sem barra
inicial */
  path?: string;
  /** Só em nó pai (sem path) — grupo expansível */
  children?: MenuNode[];
  /** Sobrescreve DEFAULT_OPEN_MODE nesta folha */
  mode?: MenuOpenMode;
  /**
   * Só em folha — executa em vez de abrir tela (relatório, disparo pontual).   * Tem
precedência sobre `mode`. Ver "Itens de menu que executam ação".
   */
  action?: (session: Session | undefined) => Promise<void>;
}
export const menuTree: MenuNode[] = [
  {
    label: "Dashboard",
    path: PATHS.DASHBOARD,
    mode: "route" // tela inicial vive na URL, nao numa aba
  },
  {
    label: "Cadastro",
    children: [      { label: "Fornecedor", path: PATHS.CADASTRO.FORNECEDOR }, // usa o
default
      { label: "Relatório mensal", path: PATHS.RELATORIOS.MENSAL, mode: "route" }
    ]
  }
];
```

**Um projeto que não quer abas** troca `DEFAULT_OPEN_MODE` para `"route"` e não declara `mode` em lugar nenhum. `TabsProvider`, `TabBar` e `TabPageRenderer` continuam no código, inertes — `TabBar` retorna `null` sem abas. Nada a remover.

# Invariantes

| Regra | Por quê |
| --- | --- |

| | |
|---|---|
| Nó tem `path` **ou** `children`, nunca ambos | `MenuTreeItem` decide por `if (node.children)`; pai nunca navega |
| `action` dispensa `path` | Nó de ação não navega — não precisa de rota |
| `mode` só em folha | Nó pai não abre nada, só expande |
| `path` sempre de `PATHS`, nunca literal | Renomear a rota em um lugar só |
| `label` único entre irmãos | É usado como `key` do React |
| Toda folha existe também em `routes.ts` | O `TabPageRenderer` resolve o `path` na mesma tabela de rotas |

A última é a que mais quebra: uma folha em `menuTree` sem entrada em `routes.ts` abre uma aba com "Rota não encontrada".

# MenuTreeItem

```
// src/components/layout/Main/MenuTreeItem.tsx
import React, { useState } from "react";import { useLocation, useNavigate } from "react-router-dom";import { Collapse, List, ListItemButton, ListItemIcon, ListItemText } from "@mui/material";
import { ExpandLess, ExpandMore } from "@mui/icons-material";import { DEFAULT_OPEN_MODE, type MenuNode } from "@/routes";
import type { Session } from "@/lib/curio";import { useAuth } from "@context/AuthProvider";
import { useTabs } from "@context/TabsContext";import { useHookMutation } from "@hooks/useHookMutation";
const noopAction = async () => undefined;
// renderizacao recursiva do menuTree.// folha: executa action, ou abre por rota (<Outlet/>) ou por aba — conforme node.mode.const MenuTreeItem: React.FC<{ node: MenuNode; depth: number }> = ({ node, depth }) => {
  const navigate = useNavigate();
  const location = useLocation();
  const { session } = useAuth();
  const { openTab, setActiveTab, activeTabId } = useTabs();  const [open, setOpen] = useState(false);
  // hook antes do early return de nó pai — ordem de hooks nao pode variar  const { mutateAsync: executeAction, isPending } = useHookMutation<void, Session | undefined>(    node.action ?? noopAction
  );
```

```
  const Icon = node.icon;
  const pl = (depth + 1) * 2; // indenta por nivel
  if (node.children) {
    return (
      <>
        <ListItemButton sx={{ pl }} onClick={() => setOpen((prev) => !prev)}>          {Icon &&
(
            <ListItemIcon>
              <Icon />
            </ListItemIcon>
          )}
          <ListItemText primary={node.label} />          {open ? <ExpandLess /> : <ExpandMore
/>}
        </ListItemButton>
        <Collapse in={open} unmountOnExit>
          <List disablePadding>
            {node.children.map((child) => (              <MenuTreeItem key={child.label}
node={child} depth={depth + 1} />
            ))}
          </List>
        </Collapse>
      </>
    );
  }
  const mode = node.mode ?? DEFAULT_OPEN_MODE;
  const handleClick = async () => {
    // action tem precedencia: executa e nao navega
    if (node.action) {
      await executeAction(session);
      return;
    }
    if (mode === "tab") {
      openTab(node);
      return;
    }    // rota: precisa zerar a aba ativa, senao o painel dela continua por cima do
<Outlet/>
    setActiveTab(null);
```

```
    navigate(`/${node.path}`);
  };
  // so destaca a rota atual quando nenhuma aba esta ativa  const isSelected = !node.action &&
mode === "route" && activeTabId === null && location.pathname === `/${node.path}`;
  return (    <ListItemButton sx={{ pl }} selected={isSelected} disabled={isPending}
onClick={handleClick}>
      {Icon && (
        <ListItemIcon>
          <Icon />
        </ListItemIcon>
      )}
      <ListItemText primary={node.label} />
    </ListItemButton>
  );
};
export default MenuTreeItem;
```

**O `setActiveTab(null)` ao navegar por rota não é detalhe.** Sem ele, o painel da aba ativa continua visível e o `<Outlet/>` fica escondido atrás — o clique parece não fazer nada.

# TabsContext

Guarda as abas, a ativa, e o registro de callbacks de fechamento.

| Membro | Para que serve |
|---|---|
| `tabs` / `activeTabId` | Estado das abas |
| `openTab(node)` | Abre o nó numa aba nova; ignora nó sem `path` |
| `closeTab(id)` | Dispara o callback registrado, remove a aba, escolhe a próxima ativa |
| `setActiveTab(id \| null)` | `null` devolve a tela ao `<Outlet/>` |
| `warningOpen` / `dismissWarning` | Aviso de limite de abas atingido |
| `registerCloseCallback` / `unregisterCloseCallback` | Base do `useTabCloseCallback` |

Decisões embutidas:

- `MAX_TABS = 8`**.** Passar disso vira aviso, não aba. Cada aba mantém um caso de uso aberto no servidor — o limite protege o backend, não só a interface.
- **Persistência em** `localStorage` (`"<projeto>:tabs"`). As abas sobrevivem ao refresh; o **estado interno das telas não** — elas remontam vazias. Só a lista de abas é restaurada.
- **Rótulo duplicado vira** `(2)`**.** Duas abas da mesma tela se distinguem.
- **Fechar a última aba navega para o dashboard**, evitando tela em branco.

# TabPageRenderer

Resolve o `path` da aba na **mesma tabela** `routes.ts` que o router usa — não há registro paralelo de telas.

```
// src/components/layout/TabPageRenderer/TabPageRenderer.tsxconst TabPageRenderer: React.FC<{
path: string }> = ({ path }) => {
  const route = routes.find((r) => r.path === path);  if (!route) return <Box sx={{ p: 3 }}>Rota
não encontrada: {path}</Box>;
  const PageComponent = route.element;  const Layout = route.layout as React.FC<{ children?:
React.ReactNode }> | undefined;
  const page = (
    <Suspense fallback={loadingFallback}>
      <PageComponent />
    </Suspense>
  );
  // layout de rota recebe a pagina como children (fora de aba ele usa <Outlet/>)  const content
= Layout ? <Layout>{page}</Layout> : page;
  return <TabErrorBoundary>{content}</TabErrorBoundary>;
};
```

Dois cuidados:

- `Suspense` **por aba.** As páginas são `lazy()`; sem isso, abrir uma aba suspenderia o shell inteiro.
- `ErrorBoundary` **por aba.** Erro numa aba não pode derrubar as outras. O boundary oferece "Tentar novamente" em vez de tela branca.

> **Layout em aba recebe `children`, não `<Outlet/>`.** Se você usa `layout` em `routes.ts` (10), o componente de layout precisa renderizar `children` **e** funcionar com `<Outlet/>` fora de aba. O mais simples é aceitar `children` opcional e cair para `<Outlet/>` quando ele não vier.

# Fechando o caso de uso da aba

Como o componente não desmonta ao fechar a aba, o `close()` do caso de uso precisa ser explícito:

```
// src/pages/Fornecedor/Cadastro/IncluirFornecedorPage.tsximport { useUseCaseControls,
useTabCloseCallback } from "@hooks/index";
const IncluirFornecedorContent: React.FC = () => {  const { open, close, status } =
useUseCaseControls();
  useTabCloseCallback(close); // encerra o caso de uso quando a aba fechar
  // ...
};
```

`useTabCloseCallback` é no-op quando a página veio pelo `<Outlet/>` (sem aba) — a mesma página serve aos dois modos sem `if`.

**Esquecer isso vaza caso de uso no servidor**: o usuário fecha a aba, o front esquece dela, e o backend segue com a sessão do caso de uso aberta até expirar.

Para a página se fechar sozinha (botão "Sair"):

```
const { closeTab } = useTabs();
const tabId = useCurrentTabId();
const handleSair = () => {
  if (tabId) closeTab(tabId);
};
```

# Registrar uma tela nova

Quatro passos, mesmo commit:

1. `paths.ts` — constante do caminho
2. `routes.ts` — rota com `lazy()` e `guard`
3. `menuTree.ts` — nó apontando para a constante, com `mode` se diferir do default
4. **Permissão** — se o projeto tiver controle de acesso

Pular o 3 dá rota acessível só por URL (às vezes é o que se quer). Pular o 2 dá item de menu que abre 404 no modo rota, ou "Rota não encontrada" no modo aba.

# Ícones

```
import { Business } from "@mui/icons-material";
{ label: "Fornecedor", path: PATHS.CADASTRO.FORNECEDOR, icon: Business }
```

Passe o **componente**, não o elemento (`Business`, não `<Business />`). Se adotar ícones, use em todos os itens de primeiro nível — meio caminho fica pior que nenhum.

# Menu, rota e permissão

| Registro | Arquivo | Responde a |
|---|---|---|
| Caminho | `paths.ts` | Qual é a URL |
| Rota | `routes.ts` | Que componente, sob qual guard |
| Menu | `menuTree.ts` | Onde aparece e como abre |
| Permissão | backend | Quem pode ver/usar |

**O menu não filtra por permissão.** Todos os nós aparecem para qualquer usuário autenticado e o backend recusa a operação. Para esconder itens, acrescente `permissao?: string` ao `MenuNode` e filtre na renderização — é trabalho novo, não vem pronto.

# Itens de menu que executam ação

Uma folha pode **executar uma função** em vez de abrir tela — relatório que só gera um PDF, disparo

pontual sem interface própria.

```
// src/routes/menuTree.ts
{
  label: "Relatório de caixas abertas",
  action: handleAbrirRelatorioCaixas
}
```

```
// src/pages/Caixa/service/handlers.tsconst CAIXAS_ABERTAS = { USE_CASE_ID: "2702", RM:
"RM_OBTEM_CAIXAS_ABERTAS" };
export const handleAbrirRelatorioCaixas = async (session: Session | undefined) => {  if
(!session) return;
  const uc = await session.openUseCase(CAIXAS_ABERTAS.USE_CASE_ID);
  try {
    const result: ObtemRelatorioResponse = await uc.sendRequest(CAIXAS_ABERTAS.RM);    await
handleOpenReport(result.URIRelatorio._);
  } finally {
    uc.abort(); // sempre fecha — nao ha tela dona deste caso de uso
  }
};
```

O `MenuTreeItem` executa via `useHookMutation` (21):

```
const noopAction = async () => undefined;
// hook antes do early return de no pai — ordem de hooks nao pode variarconst { mutateAsync:
executeAction, isPending } = useHookMutation<void, Session | undefined>(  node.action ??
noopAction
);
const handleClick = async () => {
  // action tem precedencia: executa e nao navega
  if (node.action) {
    await executeAction(session);
    return;
  }
  // ... modo tab / route
};
```

Quatro pontos que não são opcionais:

- **O hook fica antes do `if (node.children)`.** Hook depois de early return muda a ordem entre renders e o React quebra. Daí o `noopAction` para nós que não têm `action`.
- **`action` tem precedência sobre `mode`.** Um nó com `action` não navega nem abre aba.
- **`isPending` desabilita o item** enquanto executa, evitando disparo duplo.
- **Erro vira notificação** pelo `mutationCache` global (07) — o handler não precisa de `try/catch` para exibir mensagem, só do `finally` que fecha o caso de uso.

> **Tipagem.** `action` é `(session: Session | undefined) => Promise<void>`. Declarar como `(params: unknown) => Promise<void>` exige um cast em cada nó do menu — não replique esse padrão.

# Verificação

- [ ] Item `mode: "route"` navega e a URL muda
- [ ] Item `mode: "tab"` abre aba e a URL **não** muda
- [ ] Com aba ativa, clicar num item de rota esconde o painel e mostra o `<Outlet/>`
- [ ] Preencher um campo numa aba, trocar de aba e voltar — o valor continua lá
- [ ] Fechar aba dispara o `close()` do caso de uso (verifique no Network)

- [ ] Fechar a última aba volta ao dashboard
- [ ] Abrir 9 abas mostra o aviso de limite na nona
- [ ] F5 restaura a lista de abas
- [ ] Abrir duas vezes a mesma tela gera "Título" e "Título (2)"

---

Uma implementação que só usa abas (com o dashboard tratado à parte por um `if` sobre string literal) é um ponto de partida comum; o modo híbrido com `MenuOpenMode` é a generalização adotada aqui.

---

Revision #2
Created Thu, Aug 20, 2026 4:55 PM by Geraldo Barbosa
Updated Tue, Aug 25, 2026 4:53 PM by Geraldo Barbosa