03 — Estrutura de pastas

03 — Estrutura de pastas

> Árvore canônica de `src/`, a responsabilidade de cada pasta e o que **não** pode morar nela. Estrutura é contrato: se um arquivo não cabe em nenhuma pasta, o problema é o arquivo.

Árvore

```
projeto/
├── .husky/                    hooks de git (versionado)├── config/                   um JSON
por ambiente (versionado)
│   ├── dev.json
│   ├── homolog.json
│   └── prod.json
├── docs/                      documentação do projeto├── harness/
disciplina de sessão — ver 18
├── public/
│   └── config.json            GERADO por setEnvironment.js — nao versionar
├── src/
│   ├── api/                   fronteira com o backend
│   ├── components/
│   │   ├── common/            reutilizáveis de domínio-neutro│   │   ├── layout/
shell da aplicação
│   │   └── <Especifico>/      componentes de uso restrito│   ├── context/              React
Context providers
│   ├── hooks/                 hooks reutilizáveis entre páginas
```

```
│   ├── lib/
│   │   └── curio/              wrapper do @curio/client│   ├── pages/                  uma
pasta por tela
│   ├── router/                 montagem do React Router│   ├── routes/
declaração de paths, rotas e menu
│   ├── theme/                  tema MUI
│   ├── types/                  tipos globais
│   ├── utils/                  funções puras
│   ├── App.tsx
│   └── main.tsx
├── .env                        local — nao versionar├── .env.example               versionado
├── .node-version
├── index.html
├── init.sh                     verificação de baseline — ver 19
├── setEnvironment.js
├── tsconfig.json
└── vite.config.ts
```

Responsabilidade por pasta

`src/api/`

A fronteira com o backend. Três arquivos, plano, sem subpastas:

Arquivo	Responsabilidade
`session.ts`	`login()`, `connect()`, `logoutSession()` — ver 05
`queryClient.ts`	Fábrica do `QueryClient` com tratamento global de erro — ver 07
`auth.ts`	Helpers de autenticação

Não pode morar aqui: requests de uma tela específica. Esses vivem em `src/pages/<Tela>/service/`, junto de quem os usa — ver 06.

`src/lib/curio/`

O único lugar que importa `@curio/client` diretamente para configurar transporte. Envolve o `SecurityManager` e expõe `getSessionManager()`.

Não pode morar aqui: regra de negócio, componente React, chamada de caso de uso específico.

`src/pages/`

Uma pasta por tela. Estrutura de uma tela completa:

```
src/pages/Fornecedor/Cadastro/
├── IncluirFornecedorPage.tsx          componente da tela├── IncluirFornecedorPage.styles.ts
objetos sx / styled
├── schemas.ts                         schemas zod do formulário├──
index.ts                           export { default } from "./IncluirFornecedorPage"└──
service/
    ├── hooks.ts                       hooks de caso de uso (useCurioMutation)    └──
interfaces.ts                  tipos de request/response do backend
```

Telas simples podem omitir `service/` e `schemas.ts`. `index.ts` é obrigatório — é o que permite `lazy(() => import("@pages/Fornecedor/Cadastro"))`.

Não pode morar aqui: componente usado por mais de uma tela (vai para `components/common/`), tipo usado por mais de uma tela (vai para `src/types/`).

`src/components/`

Subpasta	Critério
`common/`	Reutilizável e agnóstico de domínio (`DataTable`, `FormField`)
`layout/`	Shell: `Main`, `PageContainer`, `TabBar`
`<Especifico>/`	Reutilizado por 2+ telas mas preso ao domínio (`ActionModal`)

Regra: um componente só sai de `pages/` quando a **segunda** tela precisar dele. Não promova por antecipação.

Não pode morar aqui: chamada direta de caso de uso. Componentes recebem dados por props. Exceção: componentes de `layout/` podem consumir Context (`useAuth`, `useTabs`).

`src/hooks/`

Hooks reutilizáveis entre páginas: `useCurioMutation`, `useUseCaseControls`, `useValidatedForm`, `useSearcher`, `useDebounce`, `useLocalStorage`, `useModal`.

Não pode morar aqui: hook que serve a uma única tela — esse vai em `pages/<Tela>/service/hooks.ts`.

`src/context/`

Providers de estado global de aplicação: `AuthProvider`, `NotificationProvider`, `TabsContext`, `CurrentTabContext`.

Não pode morar aqui: estado de servidor. Isso é React Query.

`src/routes/` vs `src/router/`

Separação deliberada:

- `routes/` = **dados**. `paths.ts` (constantes de URL), `routes.ts` (tabela de rotas), `menuTree.ts` (árvore do menu). Nenhum JSX.
- `router/` = **comportamento**. `AppRouter.tsx` consome os dados e monta o React Router.

Ver 10.

`src/utils/`

Funções puras, sem hook/Context/sessão:

Arquivo	Conteúdo
`formatters.ts`	Formatação para exibição: CPF, CNPJ, telefone, CEP, moeda, data/hora, truncar texto
`validators.ts`	Validação booleana: CPF, CNPJ, e-mail, telefone, CEP, senha forte, URL
`handlers.ts`	`handleOpenReport` — abre o visualizador de relatório (`resources.viewer` do `config.json`, ver 04) numa aba nova

Arquivo	Conteúdo
`useCaseTriggersLogger.ts`	Log de debug de `useUseCaseControls` — ver 21
`validators/Cnpj.ts` + `validators/index.ts`	Validador de CNPJ no formato `{ isValid, errorMessage }` que `MaskedInput.customValidate` espera — ver 08

`validators/` é uma subpasta, não um arquivo — existe porque `MaskedInput` (em `common/`) importa `validateCnpj` com uma assinatura específica (`ValidationResult`), diferente do `isValidCNPJ` booleano de `validators.ts`. Os dois convivem: use `isValidCNPJ` para checagem simples, `validateCnpj` quando o consumidor for um `MaskedInput`.

Não pode morar aqui: qualquer coisa que use hook, Context ou toque a sessão.

`src/types/`

Tipos usados por mais de uma tela. Tipos de request/response de um caso de uso específico ficam em `pages/<Tela>/service/interfaces.ts`.

Barrel exports (`index.ts`)

Cada pasta de componente e a raiz de `common/`, `layout/` e `hooks/` expõem um `index.ts`.

```
// src/components/common/index.ts
export { default as DataTable } from "./DataTable";export { default as FormField } from
"./FormField";
export type { Column } from "./DataTable";
```

> **Armadilha comum.** O barrel é mantido à mão e sai de sincronia com facilidade: um componente novo é criado mas não é exportado no `index.ts`. Resultado: importes inconsistentes, uns pelo barrel e outros pelo caminho completo.
> **Ao criar um componente, atualize o barrel no mesmo commit.**

Pontos de entrada

```
// src/main.tsx
import React, { useState } from "react";
import ReactDOM from "react-dom/client";
import { BrowserRouter } from "react-router-dom";import { QueryClientProvider } from
"@tanstack/react-query";import { ReactQueryDevtools } from "@tanstack/react-query-devtools";
import { ThemeProvider } from "@mui/material/styles";import CssBaseline from
"@mui/material/CssBaseline";import { LocalizationProvider } from "@mui/x-date-
pickers/LocalizationProvider";import { AdapterDateFns } from "@mui/x-date-
pickers/AdapterDateFns";
import { ptBR } from "date-fns/locale";
import App from "./App";
import theme from "@/theme";
import { AuthProvider } from "@/context/AuthProvider";import { NotificationProvider,
useNotification } from "@/context/NotificationProvider";import { createQueryClient } from
"@/api/queryClient";
// queryClient precisa do setNotification -> so pode nascer dentro do NotificationProviderconst
AppWithQueryClient: React.FC = () => {
  const { setNotification } = useNotification();  const [queryClient] = useState(() =>
createQueryClient(setNotification));
  return (
    <QueryClientProvider client={queryClient}>
      <AuthProvider>
        <App />
        <ReactQueryDevtools initialIsOpen={false} />
      </AuthProvider>
    </QueryClientProvider>
  );
};
ReactDOM.createRoot(document.getElementById("root")!).render(
  <React.StrictMode>    <BrowserRouter future={{ v7_startTransition: true, v7_relativeSplatPath:
true }}>
      <ThemeProvider theme={theme}>
        <CssBaseline />        <LocalizationProvider dateAdapter={AdapterDateFns}
adapterLocale={ptBR}>
          <NotificationProvider>
            <AppWithQueryClient />
          </NotificationProvider>
```

```
        </LocalizationProvider>
      </ThemeProvider>
    </BrowserRouter>
  </React.StrictMode>
);
```

A ordem dos providers é obrigatória. `NotificationProvider` precisa envolver o `QueryClientProvider` porque o `queryClient` recebe `setNotification` na construção. `AuthProvider` precisa estar dentro do `QueryClientProvider` (usa React Query) e dentro do `BrowserRouter` (usa `useNavigate`).

```
// src/App.tsx
import React from "react";
import { useAuth } from "@context/AuthProvider";import LoadingScreen from
"@components/LoadingScreen";
import AppRouter from "@/router";
const App: React.FC = () => {
  const { isLoading } = useAuth();
  if (isLoading) return <LoadingScreen />;
  return <AppRouter />;
};
export default App;
```

Revision #4
Created Thu, Aug 20, 2026 4:46 PM by Geraldo Barbosa
Updated Tue, Aug 25, 2026 4:51 PM by Geraldo Barbosa