Fundação

Stack, bootstrap, estrutura de pastas e configuração de ambientes.

01 — Stack e decisões

01 — Stack e decisões

> Define a stack fixa do projeto web e o que cada peça resolve. Itens marcados **FIXO** não são escolha do projeto novo — mudá-los quebra compatibilidade com o time. Itens **NEGOCIÁVEL** podem variar se houver motivo declarado. Versões verificadas em 2026-08-19 contra o npm registry e testadas de verdade (`tsc --noEmit`, `lint`, `build`, tela no browser) — não é combinação só especulada.

Stack

Pe	Ve	St	Por quê
Noc	>= 24	FIXO	Última LTS ativa (Node 24, "Krypton"). `process.loadEnvFile` exige ≥ 20.6

Pe	Ve	St	Por quê
Typ	`^6.0.3`	FIXO	Bumpado de 5.9. **Teto real:** `typescript-eslint` (v8, latest) exige `typescript >=4.8.4 <6.1.0` — `6.0.3` é a versão mais alta possível hoje. `7.x` fica pra quando houver suporte
Vite	`^8.2`	FIXO	Bumpado de 4. `@vitejs/plugin-react` v6 exige `vite ^8` — os três (`vite`, `@vitejs/plugin-react`, `vite-plugin-checker`) sobem juntos

Pe	Ve	St	Por quê
Rea	`^19.1.1`	FIXO	Bumpado de 18. `@curio/client` só exige `react >=16.8.0` — sem bloqueio
`@cu`	`^1.1.1`	FIXO	Transporte proprietário para o backend. Não avaliado nesta rodada (decisão do projeto)
MUI (`@mui/material`)	`^9.0.0`	FIXO	Bumpado de 5 direto para 9 ver migração MUI 5→9
`@mu`	`^9.0.0`	FIXO	Acoplado ao major do `@mui/material`. Peer cobre React 19

Pe	Ve	St	Por quê
`@ta`	`^5.`	FIXO	Bumpado de 4.36. `cacheTime` → `gcTime` já ajustado nos hooks `react-query`
`rea`	`^7.`	FIXO	Bumpado de 6. `BrowserRouter` não aceita mais a prop `future` `router-dom`
`rea`	`^7.`	FIXO	Bumpado de 7.48 — exigido pelo `@hookform/resolvers` v5 (ver Zod abaixo) `hook-form`
`@ho`	`^5.`	FIXO	Bumpado de 3 — **obrigatório** para usar Zod v4, não é opcional `resolvers`

Pe	Ve	St	Por s quê
`zod`	`^4.#`	FIXO	Bumpado de 3. Mudança de tipos exigiu ajuste em `useValidatedForm` — ver abaixo
`dat`	`^4.#`	FIXO	Bumpado de 2. Exige `@mui/x-date-pickers/AdapterDateFns` (v9 já é o adapter para v3/v4 — ver nota)
ESLint 10 + Prettier 3	—	FIXO	Bumpado de 8. **Flat config** (`eslint.config.js` não mais `.eslintrc.json`) — ver

Pe	Ve	St	Por quê
`typ`	`^8.57.0`	FIXO	Bumpado de 7. Compatível com ESLint 8/9/10 e TS `<6.1.0` — é o teto que bloqueia o TypeScript acima
`esl`	`^7.11.0`	FIXO	Bumpado de 4. `recommended` ficou bem mais amplo (regras novas tipo `set-state-in-effect`) — ver `eslint-plugin-react-hooks` 14

Pe	Ve	St	Por s que
husky + lint-staged	—	FIXO	Gate de pré-commit. `lint-staged` bumpado de 15 para 17 (dev tool, sem mudança de config)
Em	`^11`	FIXO	Peer dependency do MUI

Sem suíte de testes por padrão. Este guia não inventa uma nem assume que ela existe. O gate de qualidade é type-check + lint + build — ver 14 e 19. Se o projeto novo quiser testes, essa é uma decisão a tomar explicitamente no início, não depois.

Por que nem tudo foi para o latest

Em 2026-08-19, depois de uma rodada completa de atualização testada de verdade, só sobrou um pacote fora do latest:

Pa	Ac	La ho	Ficou det fora porque

			`typescript-eslint@8.67.0` (latest estável) exige `typescript >=4.8.4 <6.1.0` — não há versão estável do `typescript-eslint` que aceite TS 7 ainda (só canary). `6.0.3` é o teto real: existe uma linha `6.x` entre o `5.9` antigo e o `7.0` novo, e ela cabe no range aceito — não pule direto de 5→7 sem checar se há uma minor intermediária. Bumpar
Typ	`6.0.3`	**7.0.2**	

Todo o resto (MUI, Vite, ESLint, date-fns, typescript-eslint, react-hooks) já está no latest — ver as notas de migração abaixo para quem for repetir isso num projeto mais antigo.

Migração MUI 5 → 9: o que mudou de verdade

O caminho **não é incremental por major** (5→6→7→8→9): `@mui/x-date-pickers@9` exige `@mui/material: "^7.3.0 || ^9.0.0"` — não aceita a v8 como peer. Suba os três pacotes (`@mui/material`, `@mui/icons-material`, `@mui/x-date-pickers`) juntos, direto pra v9.

Superfície de quebra real, testada com `tsc --noEmit`:

- **Sem uso de `Grid`** no projeto — a maior fonte de quebra entre majors do MUI não se aplicou aqui. Se o projeto novo usa `Grid`, verifique a API v1→v2 separadamente.
- **`InputProps`, `inputProps`, `InputLabelProps` foram removidos** de `TextField`/`Checkbox` (não são só deprecados — o tipo não aceita mais). Viram `slotProps`: `InputProps` → `slotProps.input`, `inputProps` → `slotProps.htmlInput`, `InputLabelProps` → `slotProps.inputLabel`. Isso afeta qualquer componente de `common/` que componha `TextField` (`FormField`, `CurrencyInput`, `PhoneInput`, `CnpjInput`, `DatePickerInput`, `DataTable`, `TransferListCard`).
- **Props de atalho de estilo direto em `Box`/`Stack`/`Typography`/`DialogTitle`** (`display`, `alignItems`, `justifyContent`, `fontWeight`, etc., sem passar por `sx`) pararam de tipar — MUI removeu o suporte a "system props" desses componentes. Mova pra `sx={{ ... }}`.
- **`disableEscapeKeyDown` foi removido** de `Dialog`/`Modal` sem substituto direto — se o `Dialog` não tem `onClose`, a prop já era redundante (nada fecha no Escape de qualquer forma); se tem, trate a `reason` dentro do próprio `onClose`.
- **`@mui/x-date-pickers` inverteu a convenção de adapter entre majors:** na v6/v7, `AdapterDateFns` era para date-fns v2 e `AdapterDateFnsV3` para v3/v4. Na v9, inverteu — `AdapterDateFns` (sem sufixo) já é o adapter para v3/v4, e `AdapterDateFnsV2` é o legado. **Confira o import de novo a cada major**, não assuma que o nome antigo continua correto.

date-fns e o adapter do x-date-pickers

Bumpar `date-fns` sem trocar o subpath do adapter (ver acima) quebra silenciosamente em runtime, não em tipo — o import resolve, mas o parsing de data se comporta diferente. Sempre os dois juntos.

Decisões que valem entender

O backend não é REST

`@curio/client` é um transporte RPC proprietário. Não existe `GET /api/fornecedores`. Existe: abrir um **caso de uso** por id numérico e enviar **requests nomeados** (`RM_OBTEM_LISTA`) dentro dele.

Consequência prática: não use `axios`, `fetch` direto (fora do carregamento do `config.json`), nem qualquer geração de cliente a partir de OpenAPI. Ver 06.

O `fetch` do config é a única exceção

`src/lib/curio/index.ts` faz `fetch("./config.json")` para descobrir a URL do serviço em runtime. Isso é intencional: permite trocar de ambiente sem rebuild. Ver 04.

React Query v5 — `gcTime`, não `cacheTime`

A opção `cacheTime` foi renomeada para `gcTime` na v5, em `defaultOptions.queries` e `defaultOptions.mutations` do `queryClient.ts`, e em qualquer `useQuery` que a declare (`useConnectQuery`). `isPending` já era o nome correto desde a v4.36 — nenhuma mudança ali.

Zod v4 exige `@hookform/resolvers` v5 e um ajuste de generics

Zod v4 mudou a arquitetura interna de tipos de `ZodType`. Duas consequências reais, encontradas rodando `tsc` de verdade, não hipotéticas:

1. `@hookform/resolvers` v3 só aceita Zod `^3` — subir o Zod **exige** subir os resolvers para v5 (que por sua vez exige `react-hook-form >= 7.55.0`). Não é uma escolha independente.
2. O hook genérico `useValidatedForm<T extends z.ZodType>` deixou de compilar: o `output` de `z.ZodType` em v4 não fecha automaticamente em `FieldValues`. A correção é dupla — apertar o bound do genérico para `z.ZodType<FieldValues>`, e no ponto de chamada do `zodResolver`, usar um cast de tipo (`as any` justificado + `as unknown as Resolver<...>`) porque o tipo interno `Zod4Type` do resolver não compõe com um schema genérico sem perder a inferência. Ver `src/hooks/useValidatedForm.ts` — o cast é só de tipo; `zodResolver` continua validando em runtime exatamente como antes.

React Router v7 — sem prop `future`

Os *future flags* (`v7_startTransition`, `v7_relativeSplatPath`) que o `BrowserRouter` aceitava na v6 viraram comportamento padrão na v7. A prop `future` não existe mais — remova-a de `main.tsx`.

Estado de servidor no React Query, estado de UI no componente

Não há Redux, Zustand ou store global. O que vem do backend vive no React Query; o resto é `useState` local ou Context (`AuthProvider`, `NotificationProvider`, `TabsContext`). Se você sentir falta de uma store global, provavelmente está guardando resposta de servidor no lugar errado.

Erro é global, não local

Nenhuma página trata erro de request com `try/catch` para exibir mensagem. O `QueryCache` / `MutationCache` do `queryClient` captura e empurra para o `NotificationProvider`. Ver 07.

Negociável

Item	Padrão sugerido	Quando mudar
Porta do dev server	`5000`	Conflito local; ajuste em `vite.config.ts`
Paleta do tema	Azul `#1976d2`	Identidade visual do produto novo
Navegação por abas	Opcional	Só adote se o produto realmente precisar de abas
Dark mode	Ausente	Se o produto exigir; ver 12

Ite	Pad sug	Quando mud ar
`sourcemap` em produção	`true`	Desligue se o bundle for público e sensível

02 — Bootstrap

02 — Bootstrap

> Do repositório vazio até `npm run dev` servindo a aplicação. Siga na ordem; cada passo assume o anterior. Ao final, o checklist "Fundação" de 00-INDICE.md deve estar todo marcado.

1. Node

O projeto exige Node ≥ 24 (última LTS). Fixe a versão no repositório para que os git hooks consigam validá-la sem depender do shell do dev.

```
// .node-version
24.19.0
```

Com fnm:

```
fnm install 24 && fnm use
```

> **Armadilha conhecida.** Se o `fnm` não estiver avaliado no perfil do shell (`fnm env`), o Node ativo pode ser uma versão antiga e o `setEnvironment.js` falha em `process.loadEnvFile`. GUIs de Git tipicamente não herdam o perfil — por isso o `pre-commit` valida a versão explicitamente (ver 14).

2. Esqueleto

```
npm create vite@latest . -- --template react-ts
```

Depois **remova** o que o template traz e não usamos: `src/App.css`, `src/index.css`, `src/assets/`. **Mantenha** `eslint.config.js` — o próprio scaffold do Vite já gera flat config, e é isso que este guia usa (ver 14); você vai reescrever o conteúdo, não trocar de formato.

3. Dependências

Sem `@latest` em nada. `npm i` sem versão instala o mais novo do registro, o que pode não ter sido verificado ainda contra este guia — hoje só o TypeScript está deliberadamente atrás do latest (01). Pin explícito em tudo:

```
npm i @curio/client@^1.4.2 @emotion/react@^11.14.0 @emotion/styled@^11.14.1
@hookform/resolvers@^5.9.1 @mui/icons-material@^9.3.1 @mui/material@^9.3.1 @mui/x-date-
pickers@^9.11.0 @tanstack/react-query@^5.101.4 @tanstack/react-query-devtools@^5.101.4 date-
fns@^4.4.0 react@^19.2.0 react-dom@^19.2.0 react-hook-form@^7.85.0 react-router-dom@^7.18.2
zod@^4.4.3
```

```
npm i -D @eslint/js@^10.0.1 @types/node@^26.2.0 @types/react@^19.2.2 @types/react-dom@^19.2.4
@typescript-eslint/eslint-plugin@^8.67.0 @typescript-eslint/parser@^8.67.0 @vitejs/plugin-
react@^6.0.5 eslint@^10.8.1 eslint-config-prettier@^10.1.8 eslint-plugin-react-hooks@^7.1.1
```

```
eslint-plugin-react-refresh@^0.5.4 globals@^17.11.0 husky@^9.1.7 lint-staged@^17.3.0
prettier@^3.9.6 typescript@^6.0.3 vite@^8.2.1 vite-plugin-checker@^0.14.5
```

Versões exatas em 01-STACK-E-DECISOES.md — essa tabela é a fonte de verdade; se as duas divergirem, ela vence.

> “ `@types/node@^26` **com Node** `>=24` **em runtime é uma folga deliberada.** A linha 26.x dos tipos é a mais recente disponível hoje; a linha 24.x parou em `24.13.3` (sem novas patches). Não houve conflito de `tsc` /build ao testar essa combinação de verdade, mas se aparecer algum tipo de API que não existe no Node 24 real, prenda de volta em `^24` .

4. `package.json`

```
{
  "name": "nome-do-projeto",
  "version": "1.0.0",
  "private": true,
  "type": "module",
  "scripts": {
    "env:dev": "node ./setEnvironment.js dev",    "env:homolog": "node ./setEnvironment.js
homolog",
    "env:prod": "node ./setEnvironment.js prod",
    "dev": "npm run env:dev && vite",
    "start": "npm run env:dev && vite",
    "start:homolog": "npm run env:homolog && vite",    "start:prod": "npm run env:prod &&
vite",
    "build": "npm run lint && npm run env:prod && tsc && vite build",    "build:homolog": "npm
run env:homolog && tsc && vite build",
    "preview": "vite preview",    "lint": "eslint . --report-unused-disable-directives --max-
warnings 0",
```

```
    "format": "prettier --write \"src/**/*.{ts,tsx,css,json,md}\"",    "format:check":
"prettier --check \"src/**/*.{ts,tsx,css,json,md}\"",
    "prepare": "husky"
  },
  "lint-staged": {
    "src/**/*.{ts,tsx}": [      "eslint --fix --report-unused-disable-directives --max-warnings
0",
      "prettier --write"
    ],
    "**/*.{json,css,scss,md,html,yml,yaml}": ["prettier --write"]
  },
  "engines": { "node": ">=24.0.0" }
}
```

> `prepare` **depende do layout do repositório.** Acima está a forma para um repositório de módulo único (projeto web na raiz). Se o web for submódulo de um monorepo, o script muda de forma — algo como `"prepare": "cd ../.. && husky caminho/do/modulo/.husky"`, apontando para a raiz real do repositório git. Se o projeto novo for um módulo dentro de um monorepo, replique essa forma — ver 14.

5. `vite.config.ts`

```
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";
import { resolve } from "path";
import checker from "vite-plugin-checker";
export default defineConfig({
  plugins: [
    react(),
    checker({
```

```
      typescript: true,
      overlay: { initialIsOpen: false, position: "tl" },
      terminal: true
    })
  ],
  define: {
    // curio espera globals de node; browser nao tem
    global: "globalThis",
    "process.env": {}
  },
  resolve: {
    alias: {
      "@": resolve(import.meta.dirname, "./src"),      "@components":
resolve(import.meta.dirname, "./src/components"),      "@pages": resolve(import.meta.dirname,
"./src/pages"),
      "@hooks": resolve(import.meta.dirname, "./src/hooks"),      "@utils":
resolve(import.meta.dirname, "./src/utils"),      "@types": resolve(import.meta.dirname,
"./src/types"),
      "@api": resolve(import.meta.dirname, "./src/api"),      "@context":
resolve(import.meta.dirname, "./src/context"),      "@theme": resolve(import.meta.dirname,
"./src/theme")
    }
  },
  server: { port: 5000, open: true },
  build: { outDir: "dist", sourcemap: true }
});
```

O bloco `define` **não é opcional**: `@curio/client` referencia `global` e `process.env`, que não existem no browser. Sem ele a aplicação quebra em runtime no primeiro request.

6. `tsconfig.json`

```
{
  "compilerOptions": {
```

```
    "types": ["vite/client"],
    "target": "ES2020",
    "useDefineForClassFields": true,
    "lib": ["ES2020", "DOM", "DOM.Iterable"],
    "module": "ESNext",
    "skipLibCheck": true,
    "moduleResolution": "bundler",
    "allowImportingTsExtensions": true,
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react-jsx",
    "strict": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "noFallthroughCasesInSwitch": true,
    "paths": {
      "@/*": ["./src/*"],
      "@components/*": ["./src/components/*"],
      "@pages/*": ["./src/pages/*"],
      "@hooks/*": ["./src/hooks/*"],
      "@utils/*": ["./src/utils/*"],
      "@types/*": ["./src/types/*"],
      "@api/*": ["./src/api/*"],
      "@context/*": ["./src/context/*"],
      "@theme/*": ["./src/theme/*"]
    }
  },
  "include": ["src"],
  "references": [{ "path": "./tsconfig.node.json" }]
}
```

> “

> **Regra:** todo alias novo entra nos **dois** arquivos. Só no `vite.config.ts` → o build passa e o editor reclama. Só no `tsconfig.json` → o editor aceita e o build quebra.

7. Configuração de ambiente

Crie `setEnvironment.js`, `config/*.json` e `.env.example` conforme 04-AMBIENTES-E-CONFIG.md. Sem isso, `npm run dev` falha no primeiro passo.

8. `.gitignore`

```
# .gitignore
node_modules
dist
dist-ssr
*.local
.env
# gerado por setEnvironment.js — nao versionar
public/config.json
.vscode/*
!.vscode/extensions.json
.idea
.DS_Store
*.suo
*.ntvs*
*.njsproj
*.sln
*.sw?
# Harness (Claude Code) — estado local, nao versionado
harness/state/*.json
harness/state/*.md
```

```
!harness/state/_examples/
harness/handoffs/*.md
!harness/handoffs/_examples/
harness/user_preferences.md
```

9. Ponto de entrada

`index.html`, `src/main.tsx` e `src/App.tsx` conforme 03-ESTRUTURA-DE-PASTAS.md.

10. Hooks reutilizáveis

Copie os doze hooks/módulos de 21-CATALOGO-DE-HOOKS.md para `src/hooks/`, mais `src/utils/useCaseTriggersLogger.ts`, e crie o `src/hooks/index.ts`.

Faça isso **agora**, não na primeira tela: `useCurioMutation` e `useUseCaseControls` são a única forma suportada de chamar um caso de uso (06), e `useAuthQuery` é exigido pelo `AuthProvider` (05).

11. Tela de exemplo (placeholder)

Antes de existir qualquer contrato real com o backend, crie uma tela `Exemplo` seguindo exatamente a estrutura de 17-PRIMEIRA-TELA.md, mas com **todo id de caso de uso e nome de RM como placeholder explícito** (`"SUBSTITUA_USE_CASE_ID"`, `"RM_SUBSTITUA_SALVAR"`, ...) — nunca invente um valor plausível que pareça real.

Isso é obrigatório, não opcional, por dois motivos:

- É a única forma de provar, com `tsc` + `lint` + `build` reais, que a cadeia completa (`service/constants.ts` → `service/interfaces.ts` → `service/hooks.ts` → `Page.tsx` → `UseCaseManager` → `paths.ts`/`routes.ts`/`menuTree.ts`) compila e roteia antes de qualquer feature de negócio

existir.
- Dá ao time um exemplo executável para copiar, em vez de reconstruir o padrão de memória a cada tela nova.

Nomeie a entidade de forma que **não possa ser confundida com domínio real** — `Exemplo`, não uma entidade do produto. Comente no topo do arquivo que é placeholder e que deve ser removido/substituído quando a primeira tela de negócio for criada. Registre a rota e o item de menu normalmente (09, 10) — a tela precisa ser navegável e verificável no browser, não só compilar.

12. Qualidade e harness

- ESLint, Prettier, husky → 14
- `harness/` e `init.sh` → 18 e 19

13. Verificar

```
npm run lint && npx tsc --noEmit && npm run dev
```

Os três precisam passar. Se `npm run dev` abrir a página mas o console mostrar erro de `global is not defined`, o bloco `define` do passo 5 está faltando.

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:

Arqui	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/

Subp	Critério
`common/`	Reutilizável e agnóstico de domínio (`DataTable`, `FormField`)
`layout/`	Shell: `Main`, `PageContainer`, `TabBar`

Subp	Critério
`<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:

Arqui	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

Arqui	Conteúdo
handlers.js	`handleOpenReport` — abre o visualizador de relatório (`resources.viewer` do `config.json`, ver 04) numa aba nova
useCase	Log de debug de `useUseCaseControls` — ver 21
validat + validat	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;
```

04 — Ambientes e configuração

04 — Ambientes e configuração

> Como o projeto descobre em runtime a qual backend falar. Dois mecanismos distintos: `config/{env}.json` (por ambiente, versionado) e `.env` (por máquina, local). Não os confunda. Regra de ouro: **`public/config.json` é gerado, nunca editado à mão, nunca versionado.**

O fluxo

```
config/{env}.json ──[setEnvironment.js]──▶ public/config.json ──[fetch em runtime]──▶ SessionManager
                                                  ▲
                                                  |
```

```
                          .env (override)
```

1. `npm run dev` executa `npm run env:dev` antes do Vite.
2. `setEnvironment.js dev` lê `config/dev.json`, aplica overrides do `.env` e escreve `public/config.json`.
3. Em runtime, `getSessionManager()` faz `fetch("./config.json")` e constrói o transporte.

Por que em runtime e não em build: permite apontar o mesmo bundle para outro servidor trocando um arquivo, sem recompilar. É o que viabiliza o deploy ISAPI — ver 15.

config/{env}.json

Um arquivo por ambiente. Versionados.

```
// config/dev.json
{
  "service": {    "url":
"https://srvd1.dev.exemplo.com.br/cxClient/cxIsapiClient.dll/gatewayJSONBalanced?version=4",
"server": "192.168.0.00",
    "system": "61",
    "port": "5369"
  },
  "accessToken": "SUBSTITUA",
  "resources": {    "get":
"https://srvd1.dev.exemplo.com.br/cxClient/cxIsapiClient.dll/getpr",    "put":
"https://srvd1.dev.exemplo.com.br/cxClient/cxIsapiClient.dll/putpr",    "viewer":
"https://srvd1.dev.exemplo.com.br/relatorios/viewer/rel.html?id="
  },
  "logs": true
}
```

Chaves

Cha	Tip	O que é
`serv`	`string`	Endpoint do gateway Curio. Inclui `?version=4`
`serv`	`string`	IP/host do servidor de aplicação de destino
`serv`	`string`	Identificador numérico do sistema no Curio
`serv`	`string`	Porta do servidor de aplicação
`acce`	`string`	Token de acesso ao gateway
`reso`	`string`	Endpoint de download de recurso/arquivo
`reso`	`string`	Endpoint de upload

Cha	Tip	O que é
`reso`	`string`	`iewer` URL base do visualizador de relatórios; recebe o id concatenado
`logs`	`boolean`	Liga logs verbosos do transporte

O tipo é declarado em `src/lib/curio/index.ts`:

```
// src/lib/curio/index.ts
export interface Config {  service: { url: string; server: string; system: number; port: number
};
  accessToken: string;
  resources: { get: string; put: string; viewer: string };
  logs: boolean;
}
```

> **Cuidado com mentira de tipo.** Se o JSON trouxer `system` e `port` como **string** mas a interface `Config` declarar **number**, o TypeScript não acusa nada — o JSON é lido via `fetch`, sem checagem de tipo em runtime, e o Curio aceita ambos os formatos. Ainda assim é uma mentira de tipo. **Declare a interface fiel ao que o JSON realmente contém** (`system: string; port: string`, se for o caso) em vez de forçar um tipo que não corresponde ao dado real.

setEnvironment.js

```
#!/bin/node
// setEnvironment.js — copia config/{env}.json para public/config.json
import fs from "fs";
import path from "path";
const environment = process.argv[2];
if (!environment) {
  console.error("Por favor, especifique um ambiente: dev, homolog ou prod");
  process.exit(1);
}
// .env so eh lido se Node >= 20.6 e arquivo existeif (typeof process.loadEnvFile === "function"
&& fs.existsSync(".env")) {
  process.loadEnvFile(".env");
}
try {  const envFileContent = JSON.parse(fs.readFileSync(`./config/${environment}.json`,
"utf8"));
  // override so vale em dev — prod nunca le .env
  if (environment === "dev") {
    if (process.env.LOCAL_SERVER) {      envFileContent.service.server =
process.env.LOCAL_SERVER;      console.log(`service.server sobrescrito via LOCAL_SERVER:
${process.env.LOCAL_SERVER}`);
    }
    if (process.env.ACCESS_TOKEN) {      envFileContent.accessToken =
process.env.ACCESS_TOKEN;
      console.log("accessToken sobrescrito via ACCESS_TOKEN");
    }
  }
  console.log(`Configurando ambiente: ${environment}`);
  const publicDir = path.join(process.cwd(), "public");  if (!fs.existsSync(publicDir))
fs.mkdirSync(publicDir, { recursive: true });
  fs.writeFileSync(path.join(publicDir, "config.json"), JSON.stringify(envFileContent,
undefined, 2));
  console.log(`Ambiente ${environment} configurado com sucesso!`);
```

```
} catch (error) {
  console.error(`Erro ao configurar ambiente ${environment}:`, error.message);
  process.exit(1);
}
```

> **Nunca logue o valor do `ACCESS_TOKEN` no console.** Token não vai para stdout. O snippet acima já evita isso.

`.env` — variáveis por máquina

Serve para o dev apontar o ambiente `dev` ao **seu** servidor local sem sujar o `config/dev.json` compartilhado.

```
# .env.example — copie para .env e ajuste. .env nao eh versionado.
# Sobrescreve service.server no config gerado (so em dev)
LOCAL_SERVER="ENDERECO_DO_SERVIDOR_LOCAL"# Sobrescreve accessToken (so em dev)
ACCESS_TOKEN="TOKEN_DE_ACESSO"
```

Regras:

- `.env.example` é **versionado** e contém apenas placeholders.
- `.env` é **gitignored**.
- Overrides só se aplicam ao ambiente `dev`. `homolog` e `prod` ignoram o `.env` por construção — isso é proposital, para que um `.env` esquecido não vaze para um build de produção.
- Não use `import.meta.env` / prefixo `VITE_` para configuração de backend. Isso embute o valor no bundle em tempo de build e derrota o propósito do `config.json`. `import.meta.env.DEV` (flag de modo) é aceitável.

Adicionar um ambiente novo

1. Crie `config/{nome}.json` com todas as chaves.
2. Adicione ao `package.json`:

   ```
   "env:{nome}": "node ./setEnvironment.js {nome}","start:{nome}": "npm run env:{nome} && vite"
   ```

3. Se o ambiente tiver build próprio, adicione `"build:{nome}"`.

Segurança

- **`public/config.json` no `.gitignore`.** É gerado; versioná-lo cria conflito toda vez que alguém troca de ambiente.
- **`config/*.json` é servido ao browser.** Tudo ali é público para quem abrir o DevTools. Não coloque segredo que não possa ser lido pelo usuário final.
- **`accessToken` em `config/dev.json` fica no histórico do Git para sempre.** Nunca commite um token real nesse arquivo. Commite `"accessToken": "SUBSTITUA"` e distribua o valor real via `.env` (dev) ou via substituição no pipeline (homolog/prod).

21 — Catálogo de hooks

21 — Catálogo de hooks

> Os hooks reutilizáveis que todo projeto web deste padrão deve ter em `src/hooks/`. São agnósticos de domínio: nenhum conhece Fornecedor, Documento ou qualquer entidade. **Copie os doze arquivos deste documento no bootstrap** — antes da primeira tela, não depois.

Inventário

Ho	Ca	De de	nde Documento
`useSessionQuery`	Sessão	os `STORAGEKEY`	`api/session.connect`
`useSessionMutation`	Sessão	os `Session`	`api/session.login`
`useSessionMutation`	Sessão	os dois + `logoutSession`	os dois

Ho	Ca	De de	nde Documento
`mutation`	Caso de uso	`NotificationProvider`	06
`useCurioMutation`	Caso de uso	`@curio/client/react`, `mutationMessages`	06
`useCurioMutation`	Caso de uso +	idem + `isAuthError`	06
`useHookMutation`	Caso de uso +	`useCaseTriggersLogger`	06
`useClearAuthProvider`	Caso de uso	`AuthProvider`	06
`useHookMutation`	Caso de uso	`mutationMessages`	09
`useAbasCallback`	Abas,	`TabsContext`, `CurrentTabContext`	09
`useFormulário`	Formulário	`react-hook-form`, `zod`	11
`useDebounce`	Utilitário	—	—
`useLocalStorage`	Utilitário	—	—

Mais um arquivo de apoio: `src/utils/useCaseTriggersLogger.ts`, exigido por `useUseCaseControls`.

`mutationMessages.ts` não é um hook de tela — é o núcleo compartilhado entre `useCurioMutation` e `useHookMutation` (ver abaixo). Copie-o junto dos dois.

Grafo de dependências

```
useAuthQuery ─┬─ useConnectQuery ── api/session.connect
              └─ useLoginMutation ── api/session.login
```

```
                └─ (logout) ── api/session.logoutSession
mutationMessages ── NotificationProvider (useNotifyMutationSuccess + tipo MutationMessages)
useCurioMutation ─┬─ UseCaseManager (@curio/client/react)                  └─ mutationMessages
useUseCaseControls ─┬─ UseCaseManager                    ├─
NotificationProvider
                    └─ utils/useCaseTriggersLogger
useSearcher ──── AuthProvider (session direta, sem UseCaseManager)useHookMutation ────
mutationMessages (funcao async qualquer)
useTabCloseCallback ─┬─ TabsContext
                     └─ CurrentTabContext
useValidatedForm ── react-hook-form + zod
useDebounce, useLocalStorage ── nada
```

Ordem de cópia: `utils/useCaseTriggersLogger.ts` e `mutationMessages.ts` primeiro (nada depende deles e várias coisas dependem deles), depois os demais hooks. `useAuthQuery` por último — ele importa os outros dois de sessão.

Barrel

```
// src/hooks/index.ts
// Sessao e autenticacao
export { useAuthQuery } from "./useAuthQuery";export { useConnectQuery } from
"./useConnectQuery";
export { useLoginMutation } from "./useLoginMutation";
// Caso de uso Curio
export { useCurioMutation } from "./useCurioMutation";export { useUseCaseControls } from
"./useUseCaseControls";
export { useSearcher } from "./useSearcher";export { useHookMutation } from "./useHookMutation";
export { useNotifyMutationSuccess, type MutationMessages } from "./mutationMessages";
// Abas
export { useTabCloseCallback } from "./useTabCloseCallback";
// Formularios
export { useValidatedForm } from "./useValidatedForm";
```

```
// Utilitarios
export { default as useDebounce } from "./useDebounce";export { default as useLocalStorage }
from "./useLocalStorage";
```

Note a mistura de export nomeado e default: `useDebounce` e `useLocalStorage` usam `export default`; os demais, export nomeado. É herança da referência. **Num projeto novo, padronize em export nomeado** — o barrel fica uniforme e o rename fica rastreável.

Camada de sessão

Os três hooks de sessão formam uma composição. `AuthProvider` consome **apenas** `useAuthQuery`; página nenhuma toca nos outros dois.

`useConnectQuery`

Reconecta a partir do token guardado, no boot e no refresh de página.

```
const connectQuery = useConnectQuery();
```

Três decisões embutidas:

- **`enabled: hasToken`** — sem token, a query nem roda. Evita request inútil no primeiro acesso.
- **Timeout de 8s via `Promise.race`** — `connect()` não tem timeout próprio; sem isso a tela fica em loading indefinido quando o servidor não responde.
- **Os `throw` explícitos** — `connect()` **retorna** o erro em vez de lançar (05). Este hook é a fronteira que converte isso no que o React Query espera.

`useLoginMutation`

```
const loginMutation = useLoginMutation();
await loginMutation.mutateAsync({ email, password });
```

Grava o token no `sessionStorage` e invalida `["auth", "connect"]`, mantendo um único estado de sessão.

`useAuthQuery`

Compõe os dois e expõe a API que o `AuthProvider` usa:

```
const { session, isAuth, isLoading, login, logout, refetchConnection } = useAuthQuery();
```

Campo	O que é
`session`	`Session` ativa, ou `undefined`
`isAuth`	`!!session`
`isLoading`	Reconectando **ou** autenticando
`login`	`(params) => Promise<Session>`
`logout`	Aborta no servidor, reseta e limpa o cache
`refetchConnection`	Força a reconexão — usado no boot pelo `AuthProvider`

Campo	O que é
`isConnecting` / `isLoggingIn` / `connectError` / `loginError`	Estado granular, para telas de login

A sessão vem de `loginMutation.data ?? connectQuery.data`: um login recém-feito tem precedência sobre a reconexão.

> **O `logout` precisa chamar `logoutSession(session)`.** Limpar `sessionStorage` e o cache **não** encerra a sessão no servidor — só `session.abort()` faz isso. É comum uma implementação de `useAuthQuery` omitir essa chamada e deixar sessão órfã no backend; a versão deste catálogo corrige.

Camada de caso de uso

Núcleo compartilhado: `mutationMessages.ts`

`useCurioMutation` e `useHookMutation` são, na maior parte, o mesmo hook: um `useMutation` que aceita `msgSucesso` / `msgErro` / `msgErroFallback` e dispara a notificação de sucesso do mesmo jeito. A diferença real entre os dois está só em **como** `msgErro` / `msgErroFallback` chegam ao `mutationCache.onError` global — porque a forma de `TParams` é diferente:

- `useCurioMutation`: `TParams` é sempre um objeto (o payload do backend), então as mensagens cabem dentro dele — viajam por `variables`.
- `useHookMutation`: `TParams` é o valor que a função recebe (uma `Session`, `void`, o que for), não necessariamente um objeto — as mensagens não cabem ali. Viajam pelo `meta` da mutation,

que o React Query aceita justamente para isto.

O que é idêntico foi extraído para um arquivo à parte, e os dois hooks reutilizam:

```
// src/hooks/mutationMessages.ts
import { useCallback } from "react";import { useNotification } from
"@context/NotificationProvider";
export interface MutationMessages {
  msgSucesso?: string;  /** Sobrescreve qualquer mensagem de erro do backend — sempre prevalece.
*/
  msgErro?: string;  /** Usada só quando o backend não devolve mensagem e msgErro não foi
definido. */
  msgErroFallback?: string;
}
// unico ponto que dispara a notificacao de sucesso — compartilhado por todo hook de mutation
export function useNotifyMutationSuccess() {
  const { setNotification } = useNotification();
  return useCallback(
    (msgSucesso?: string) => {      if (msgSucesso) setNotification({ type: "success", message:
msgSucesso });
    },
    [setNotification]
  );
}
```

`queryClient.ts` também importa só o **tipo** `MutationMessages` (`import type`, sem custo em runtime) para ler `msgErro`/`msgErroFallback` de `variables` **ou** de `mutation.meta` no mesmo lugar — ver 07.

Não force os dois hooks a usar o mesmo mecanismo de transporte (`variables` vs `meta`). A diferença não é acidental — é consequência de `TParams` ter formas diferentes. Force-fit aqui pioraria os dois lados: `useCurioMutation` perderia a possibilidade de variar `msgErro` por chamada, ou `useHookMutation` teria que envolver todo `TParams` num objeto só para caber uma mensagem.

`useCurioMutation`

O mais importante do conjunto. Envia um request nomeado dentro do caso de uso aberto pelo `UseCaseManager`.

```
export const useSalvaFornecedor = () => useCurioMutation<void,
SalvaFornecedorRequest>("RM_SALVA_OBJETO");
```

```
const { mutateAsync: salvar, isPending } = useSalvaFornecedor();
await salvar({
  Fornecedor: { _OID: "123", _Nome: "ACME" },
  msgSucesso: "Fornecedor salvo com sucesso!",
  onSuccess: () => setModalAberto(true)
});
```

Quatro coisas que ele faz e o `useMutation` cru não faria:

1. **`msgSucesso`**, **`msgErro`**, **`msgErroFallback`** viram notificação e são **removidos do payload** antes de ir ao backend (`delete params.msgSucesso`, etc.).
2. **A mensagem de erro segue prioridade fixa**, resolvida no `mutationCache.onError` global (07), não dentro deste hook: `msgErro` (sempre prevalece) → mensagem do backend → `msgErroFallback` → fallback genérico interno. `useCurioMutation` **não** tem `onError` próprio — se tivesse, a notificação apareceria duas vezes.
3. **Callbacks no mesmo objeto dos parâmetros** — `onSuccess`, `onError`, `onSettled`, `onMutate` convivem com os params num argumento só. Difere do React Query puro, onde seriam um segundo argumento.
4. **Tipagem condicional em `TParams`** — quando é `void`, o argumento inteiro fica opcional (`incluirFornecedor()`); quando não é, fica obrigatório.

Use `isPending`, não `isLoading` — em mutations da v4.36 o `isLoading` está deprecado.

`useUseCaseControls`

Abre e fecha o caso de uso, e expõe o `status`.

```
```

```
const { open, close, status, error } = useUseCaseControls();
```

`status` vai de `"idle"` a `"open"`. O `open` devolvido é a versão segura: o `open()` do Curio lança, e este converte em notificação.

Descobrir os requests disponíveis:

```
const { open, status } = useUseCaseControls({ enableLogs: true });
```

Loga no console, só em desenvolvimento, os RMs chamáveis no estado atual. **Remova antes de commitar** — o hook `check-debug-flags.sh` do pré-commit barra `enableLogs: true` (14). Exige `src/utils/useCaseTriggersLogger.ts`.

useSearcher

Busca genérica pelas operações `120` (buscar) e `134` (obter contexto), **sem** `UseCaseManager`.

```
const { getContext, search } = useSearcher<FiltrosBusca, ResultadoBusca>("465");const resultado
= await search.mutateAsync({ _Nome: "ACME", _Ativo: true });
```

Usa a sessão do `useAuth()` diretamente. Escolha entre os dois caminhos:

Situa	Use
Tela só de filtro + lista, sem estado no servidor	`useSearcher`

Situa	Use
Incluir, alterar, salvar — há estado no caso de uso	`UseCaseManager` + `useCurioMutation`

useHookMutation

Roda uma **função async qualquer** como mutation — para ação que não passa por `UseCaseManager`. O caso canônico é o `action` de um item de menu (09).

```
// src/hooks/useHookMutation.ts
import { useMutation } from "@tanstack/react-query";import { MutationMessages, useNotifyMutationSuccess } from "./mutationMessages";
export const useHookMutation = <TData = unknown, TParams = void>(  func: (params: TParams) => Promise<TData>,
  messages: MutationMessages = {}
) => {
  const notifySuccess = useNotifyMutationSuccess();  const { msgSucesso, msgErro, msgErroFallback } = messages;
  return useMutation<TData, Error, TParams>({
    mutationFn: (params) => func(params),    meta: { msgErro, msgErroFallback }, // TParams nao e' objeto — nao cabe em variables
    onSuccess: () => notifySuccess(msgSucesso)
  });
};
```

```
const { mutateAsync: executar, isPending } = useHookMutation<void, Session | undefined>(handleAbrirRelatorio, {
  msgSucesso: "Relatório gerado.",
  msgErroFallback: "Não foi possível gerar o relatório."
```

```
});
await executar(session);
```

Diferença para `useCurioMutation`: aquele envia um request **dentro** de um caso de uso já aberto; este executa uma função que abre e fecha o próprio caso de uso. Ambos reutilizam `MutationMessages` / `useNotifyMutationSuccess` de `mutationMessages.ts` — ver "Núcleo compartilhado" acima.

> **Assinatura divergente de um padrão comum.** É comum ver `msgSucesso` / `msgErro` dentro dos parâmetros (`{ options: { ... } }`) com o parâmetro tipado como `unknown`. Isso quebra quando o parâmetro é uma instância de classe como `Session`, e obriga a um cast em cada nó do menu. Aqui as mensagens são **argumento de criação** do hook e `TParams` é genérico de verdade.

Abas

`useTabCloseCallback`

Registra o que rodar quando a aba da página for fechada.

```
const { close } = useUseCaseControls();
useTabCloseCallback(close);
```

Existe porque abas ficam **montadas** o tempo todo — só o painel ativo é visível. Logo, cleanup de `useEffect` **não** dispara ao fechar a aba. Sem este hook, o caso de uso fica aberto no servidor depois que o usuário fecha a aba.

É no-op quando a página veio pelo `<Outlet/>` da rota, então a mesma página serve aos dois modos de navegação sem `if`.

Formulário e utilitários

`useValidatedForm`

```
const form = useValidatedForm({
  schema: fornecedorSchema,
  defaultValues: { _Nome: "", _Ativo: true }
});
```

Garante `zodResolver` e `mode: "onChange"` uniformes. **Nunca use `useForm` direto** — ver 11.

`useDebounce`

```
const termoDebounced = useDebounce(termo, 400);
```

Atrasa a propagação de um valor. Use em filtro que dispara busca a cada tecla.

`useLocalStorage`

```
const [colunas, setColunas, removeColunas] = useLocalStorage("tabela.colunas", colunasPadrao);
```

Valor JSON no `localStorage`, sincronizado entre abas — dispara um `CustomEvent` próprio porque o evento `storage` nativo não chega à aba que escreveu.

Não use para token de sessão. Isso é responsabilidade de `lib/curio` + `api/session`, que usam `sessionStorage` (05).

O que não copiar de uma

implementação existente

É comum achar hooks em `src/hooks/` que **parecem** genéricos mas não pertencem a este catálogo:

Sinal	Por quê
Nome ligado a uma tela específica (ex.: `useDashboard`)	Casos de uso do domínio daquele projeto. Específico
Envelopa `useState` booleano só para renomear (ex.: `useModal`)	Abstração fina demais para justificar existir

Verificação

Depois de copiar:

```
npx tsc --noEmit && npm run lint
```

Erros que aparecem quando falta uma peça:

Erro	Falta

`Cannot find module '@utils/useCaseTriggersLogger'`	O arquivo de apojo, ou o alias `@utils`
`Cannot find module '@curio/client/react'`	Versão do `@curio/client` sem entrada `react`
`Property 'isPending' does not exist`	React Query v4 antigo — exige 4.36+
`useNotification precisa estar dentro de...`	`NotificationProvider` fora do lugar em `main.tsx` (03)

Os hooks de sessão só se provam com backend real: faça login, dê F5 (deve manter a sessão) e faça logout (deve voltar ao login). Os de caso de uso só se provam na primeira tela (17).

Confira sempre contra este catálogo antes de copiar um hook de outra implementação — o defeito de logout descrito acima é um erro recorrente em versões de `useAuthQuery`.