13 — Convenções

13 — Convenções

> Nomenclatura de arquivos, dados, hooks, handlers e constantes. A regra que governa tudo: **dado do backend mantém o nome do backend; lógica do front é em inglês.** Ao final, uma lista de anti-padrões reais e recorrentes que **não** devem ser replicados.

Idioma

Elemento	Idioma	Exemplo
Propriedade vinda do backend	Português	`_Nome`, `_DataExpiracao`, `Documentos`
Interface que espelha o backend	Inglês	`SupplierXML`, `SaveShipmentRequest`
Variável local, função, helper	Inglês	`isLoading`, `groupByLayout`, `handleClick`
Nome de componente	Inglês	`DataTable`, `LoadingButton`
Nome de página	Inglês	`IncludeSupplierPage`
Texto de interface	Português	`label="Nome do fornecedor"`
Mensagem de validação	Português	`"Informe o CNPJ"`
Comentário de código	Português	ver estilo
Segmento de URL	Inglês	`registration/supplier-type/add`

O critério: se o nome **atravessa a fronteira com o backend**, ele é do backend. Se vive só no front, é inglês.

Propriedades do backend: `_Prefixo`

```
interface FornecedorXML {
  _OID: string; // primitivo → underscore
  _Nome: string; // primitivo → underscore
  _Ativo: boolean; // primitivo → underscore
  _DataCadastro: string; // primitivo → underscore
  Endereco: EnderecoXML; // objeto → sem prefixo  Documentos: DocumentoXML[]; // coleção → sem
prefixo
}
```

Regra do Curio: **primitivo leva `_`, objeto complexo não leva.**

Isso não é decoração — é como o payload chega. Renomear para camelCase exigiria uma camada de mapeamento em toda request e resposta. O projeto opta por não ter essa camada: o custo é conviver com `_Nome` no front.

Consequências:

- Campos de formulário usam os mesmos nomes (`name="_Nome"`), evitando conversão no submit.
- Sufixo `XML` nas interfaces que espelham a estrutura do backend, distinguindo-as dos tipos do front.

Booleano do backend: `Flag`

O Curio não tem tipo booleano nativo trafegando na rede — o valor real que chega/sai é a string `"S"` ou `"N"`. Nunca converta para `boolean` na borda; mantenha `Flag` do início ao fim.

```
type Flag = "S" | "N";
interface FornecedorXML {
  _Ativo: Flag; // não `boolean`
}
```

Regra: **`true` → `"S"`, `false` → `"N"`**, em qualquer campo que atravesse a fronteira com o backend (request ou response). Se o front precisar de um `boolean` de verdade (ex.: `checked` de um `Checkbox`),

converta na borda da UI, nunca no tipo que representa o payload:

```
<Checkbox checked={fornecedor._Ativo === "S"} onChange={(e) => setValue("_Ativo",
e.target.checked ? "S" : "N")} />
```

Arquivos e pastas

Item	Padrão	Exemplo
Pasta de componente	`PascalCase`	`FormField/`
Componente	`PascalCase.tsx`	`FormField.tsx`
Estilos	`PascalCase.styles.ts`	`FormField.styles.ts`
Barrel	`index.ts`	—
Hook	`camelCase.ts`	`useCurioMutation.ts`
Utilitário	`camelCase.ts`	`formatters.ts`
Tipos de uma tela	`interfaces.ts`	`service/interfaces.ts`
Hooks de uma tela	`hooks.ts`	`service/hooks.ts`
Constantes de RM/UC	`constants.ts`	`service/constants.ts`
Schemas de uma tela	`schemas.ts`	—
Página	`PascalCasePage.tsx`	`IncluirFornecedorPage.tsx`

Páginas levam sufixo `Page`; componentes não. Deixa óbvio, no import, o que é rota e o que é peça.

Hooks

Tipo	Padrão	Exemplo
Query de coleção	`use{Entidade}s`	`useFornecedores`
Query de item	`use{Entidade}`	`useFornecedor`
Mutation de caso de uso	`use{Verbo}{Entidade}`	`useSalvaFornecedor`
Mutation genérica	`use{Verbo}{Entidade}Mutation`	`useCreateFornecedorMutation`
Utilitário	`use{Descritor}`	`useDebounce`, `useModal`

Hooks de caso de uso usam **verbo em português**, espelhando o request do backend:

```
RM_INCLUI_OBJETO    → useIncluiFornecedorRM_SALVA_OBJETO     → useSalvaFornecedor
RM_OBTEM_DOCUMENTOS → useObtemDocumentos
```

Isso torna rastreável qual hook corresponde a qual request sem abrir o arquivo.

Handlers e callbacks

```
interface Props {
  onSubmit: (data: FormData) => void; // prop → on{Ação}
  onCancel: () => void;
}
const Componente = ({ onSubmit }: Props) => {
  const handleFormSubmit = (data: FormData) => {
    // interno → handle{Ação}
    onSubmit(data);
  };
};
```

`on*` é o que o componente **recebe**. `handle*` é o que ele **define**. Nunca inverta — a distinção diz, na leitura, de onde vem o comportamento.

Constantes

```
// Módulo, imutável, conhecido em tempo de escrita → SCREAMING_SNAKE_CASEconst FORNECEDOR_RMS =
{ USE_CASE: "4821", SALVAR: "RM_SALVAR_DADOS_FORNECEDOR" };export const STORAGEKEY =
"br.com.nomedoprojeto";
// Objeto de configuração / default → camelCaseconst defaultFormValues = { _Nome: "", _Ativo:
true };
const fornecedorKeys = { all: ["fornecedores"] as const };
```

Ids de caso de uso e nomes de request **sempre** dentro de um objeto `_RMS` — ver 06 — nunca literal no JSX.

Tipos e interfaces

- `interface` para formato de objeto; `type` para união, interseção e utilitário.
- Interface de props do componente **exportada** e nomeada `{Componente}Props`.
- Tipo de formulário **inferido** do zod (`z.infer`), nunca escrito à mão — ver 11.
- Sufixo `XML` para interfaces que espelham o backend.
- Sufixos `Request` / `Response` para payloads de caso de uso.

Comentários

Comentário em código segue estilo ultra-comprimido (telegráfico), não prosa:

```
// ERRADO
// This function is responsible for grouping the routes by their layout so that// we can render one parent Route per layout.
// CERTO
// agrupa por layout pra montar uma <Route> pai por layout
```

Comente **por quê**, não **o quê**. O código já diz o quê.

Quando um `eslint-disable` for inevitável, justifique na mesma linha:

```
// eslint-disable-next-line @typescript-eslint/no-explicit-any -- @curio nao tipa o service
constructor(service: any, driver: RequestDriver) {
```

`--` seguido do motivo é obrigatório. `eslint-disable` sem justificativa não passa em revisão.

Imports

Ordem, com linha em branco entre grupos:

```
// 1. React e libs externas
import React, { useEffect, useState } from "react";
import { Box, Button } from "@mui/material";
// 2. Aliases internos
import { useAuth } from "@context/AuthProvider";
import { FormField } from "@components/common";
// 3. Relativos da própria feature
import { useSalvaFornecedor } from "./service/hooks";import { containerStyles } from
"./IncluirFornecedorPage.styles";
```

Use aliases para cruzar pastas (`@components/...`), relativos dentro da própria feature (`./service/...`). Nunca `../../../hooks/useX` — para isso existe `@hooks`.

Anti-padrões recorrentes

Implementações reais tendem a acumular as mesmas inconsistências. A coluna "Padrão correto" é o que o projeto novo adota sempre — nunca replique a coluna da esquerda.

Anti-padrão	**Padrão correto**
Duas pastas para o mesmo domínio em idiomas diferentes (`Documentos/` **e** `Documents/`)	Um idioma só para nomes de pasta de página. Adote **português**
Constantes de rota misturando idiomas (`PATHS.DOCUMENTS.*` ao lado de `PATHS.DOCUMENTOS.*`)	Um idioma só nas URLs. Adote **português**
`CLAUDE.md` /guia descreve uma pasta de requests que não existe mais no código	Requests por tela em `pages/<Tela>/service/` — 03
Documentação diz uma porta; `vite.config.ts` usa outra	Documente a porta real, confira contra o código
Documentação cita uma `STORAGEKEY` de exemplo; o código usa outra	Chave própria do projeto, documentada corretamente
Documentação diz `sessionStorage`; o código de logout limpa `localStorage` (ou vice-versa)	Um storage só, consistente — 05
Logout faz `localStorage.clear()` (apaga tudo, não só a sessão)	Remover só a chave da sessão
Barrel (`index.ts`) desatualizado, não exporta componente/módulo já existente	Barrel atualizado no mesmo commit do componente

Anti-padrão	Padrão correto
Campo de config tipado como `number`, mas o JSON de runtime entrega `string`	Tipar como `string` — 04
Rota como string literal (`"dashboard"`) em vez da constante (`PATHS.DASHBOARD`)	Sempre a constante
Código de debug (`<Profiler>`, `console.log`) atrás de uma env var que nunca é definida	Remover código morto; sem `eslint-disable` decorativo
Script de setup de ambiente loga o valor de um token/segredo	Nunca logar segredo
Arquivo de config de ambiente com token real commitado	Placeholder no repo; valor via `.env` — 04
Detecção de expiração de sessão comparando string de mensagem de erro	Sinalização por código de erro em `isAuthError`
`login()` / `connect()` retornam o erro em vez de lançar	Deixar lançar e tratar no hook
Estilos num objeto `styles = { root, drawer, ... }` por arquivo	Constantes nomeadas exportadas individualmente — ver 12

Se você encontrar mais algum ao portar código, **acrescente aqui** em vez de resolver em silêncio.

Revision #4
Created Thu, Aug 20, 2026 5:37 PM by Geraldo Barbosa
Updated Tue, Aug 25, 2026 4:54 PM by Geraldo Barbosa