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

Hook	Camada	Depende de	Documento
`useConnectQuery`	Sessão	`api/session.connect`, `STORAGEKEY`	05
`useLoginMutation`	Sessão	`api/session.login`, `Session`	05
`useAuthQuery`	Sessão	os dois acima + `logoutSession`	05
`mutationMessages`	Caso de uso	`NotificationProvider`	06
`useCurioMutation`	Caso de uso	`@curio/client/react`, `mutationMessages`	06
`useUseCaseControls`	Caso de uso	idem + `isAuthError` + `useCaseTriggersLogger`	06
`useSearcher`	Caso de uso	`AuthProvider`	06
`useHookMutation`	Caso de uso	`mutationMessages`	09
`useTabCloseCallback`	Abas	`TabsContext`, `CurrentTabContext`	09
`useValidatedForm`	Formulário	`react-hook-form`, `zod`	11

Hook	Camada	Depende de	Documento
`useDebounce`	Utilitário	nada	—
`useLocalStorage`	Utilitário	nada	—

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`
`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:

Situação	Use
Tela só de filtro + lista, sem estado no servidor	`useSearcher`
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 apoio, 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`.

Revision #2
Created Thu, Aug 20, 2026 5:17 PM by Geraldo Barbosa
Updated Tue, Aug 25, 2026 4:57 PM by Geraldo Barbosa