06 — Caso de uso

06 — Caso de uso

> Como uma tela fala com o backend Curio. Padrão vigente: `UseCaseManager` envolve a página, `useUseCaseControls` abre/fecha, `useCurioMutation` envia requests. **Não use `session.openUseCase()` direto numa página.** Esse é o caminho de baixo nível.

O modelo mental

O Curio não é REST. Não há endpoint por recurso. Há:

1. Um **caso de uso**, identificado por um id numérico (`"2544"`). Ele tem estado no servidor: você o abre, interage, e fecha.
2. **Requests nomeados** dentro dele (`"RM_INCLUI_OBJETO"`), que recebem e devolvem objetos.

Um caso de uso é aproximadamente "uma tela do sistema" do lado do servidor. Por isso o casamento natural é **um caso de uso por página**.

```
Página (UseCaseManager useCaseId="2544")
  ├─ useUseCaseControls()   → open() / close() / status  └─
useCurioMutation("RM_INCLUI_OBJETO")  → mutate() / mutateAsync()
```

Cadeia de chamada

Toda chamada ao backend segue a mesma cadeia de três camadas, sempre nesta ordem:

```
Componente (.tsx)  →  hook específico da feature (service/hooks.ts)  →  useCurioMutation
```

- **Componente** — chama o hook da feature, nunca `useCurioMutation` direto.
- **Hook específico da feature** — uma função de uma linha por request, tipada, nomeada pelo verbo do RM. É a única coisa que sabe qual string vai para o backend.
- **`useCurioMutation`** — genérico, não sabe nada sobre Fornecedor. Ver 21.

```
// service/hooks.ts — o hook específico é a ÚNICA camada que conhece o nome do RMexport const useSalvaFornecedor = () => useCurioMutation<void, SalvaFornecedorRequest>(FORNECEDOR_RMS.SALVAR);
```

```
// Page.tsx — o componente so conhece o hook, nunca a string do RMconst { mutateAsync: salvar, isPending } = useSalvaFornecedor();
```

Nunca pule uma camada. Chamar `useCurioMutation("RM_SALVA_OBJETO")` direto no componente espalha o nome do request por várias telas — renomear o RM no backend exige grep no projeto inteiro em vez de editar uma linha.

Convenção `_RMS`: use case e requests num só objeto

Cada feature declara **um objeto de constantes** com o id do caso de uso e o nome de todos os seus requests. Um arquivo, uma fonte da verdade.

```
// src/pages/Cadastro/Fornecedor/service/constants.ts
export const FORNECEDOR_RMS = {
  USE_CASE: "4821",
  OBTEM_DADOS: "RM_OBTEM_DADOS_FORNECEDOR",
  SALVAR: "RM_SALVAR_DADOS_FORNECEDOR",
  REMOVER: "RM_REMOVER_FORNECEDOR"
};
```

Regras:

- **Nome do objeto**: `{ENTIDADE}_RMS`, maiúsculo, no plural de "requests do módulo" — não é o

plural da entidade.

- `USE_CASE` é sempre a primeira chave. É o único valor que o `UseCaseManager` da página consome.
- Os demais valores são os nomes exatos dos requests, em `SCREAMING_SNAKE_CASE` com prefixo `RM_` no **valor** (a chave do objeto pode ser mais curta e legível — `SALVAR`, não `SALVAR_DADOS_FORNECEDOR`).
- **Nenhuma string de RM ou de id de caso de uso literal fora deste arquivo.** Nem em `hooks.ts`, nem na página.

`hooks.ts` importa a constante em vez de repetir a string:

```
// service/hooks.ts
import { useCurioMutation } from "@/hooks";
import { FORNECEDOR_RMS } from "./constants";
import type {
  SalvaFornecedorRequest,
  BuscaFornecedoresRequest,
  BuscaFornecedoresResponse,
  FornecedorInicialResponse
} from "./interfaces";
export const useIncluiFornecedor = () => useCurioMutation<FornecedorInicialResponse,
void>(FORNECEDOR_RMS.OBTEM_DADOS);
export const useBuscaFornecedores = () =>  useCurioMutation<BuscaFornecedoresResponse,
BuscaFornecedoresRequest>(FORNECEDOR_RMS.SALVAR);
export const useRemoveFornecedor = () => useCurioMutation<unknown, { Fornecedor: string
}>(FORNECEDOR_RMS.REMOVER);
```

E a página importa a mesma constante para o `useCaseId` do `UseCaseManager` — ver "A página" abaixo. **Um único arquivo muda quando o backend renumerar o caso de uso ou renomear um request.**

Anatomia de uma tela

Uma tela que fala com o backend tem quatro arquivos:

```
src/pages/Fornecedor/Cadastro/
├── IncluirFornecedorPage.tsx    componente + UseCaseManager├── schemas.ts
```

```
validação zod
└── service/    ├── constants.ts              FORNECEDOR_RMS — useCaseId + nomes dos
requests
    ├── interfaces.ts             tipos de request/response    └── hooks.ts
um hook por request
```

service/interfaces.ts

Tipa o que entra e sai do backend. Convenção `_Prefixo` — ver 13.

```
// src/pages/Fornecedor/Cadastro/service/interfaces.ts
export interface FornecedorXML {
  _OID: string;
  _Nome: string;
  _CNPJ: string;
  _Ativo: boolean;
  Endereco?: EnderecoXML;
}
export interface EnderecoXML {
  _OID: string;
  _Logradouro: string;
  _Cidade: string;
}
// resposta de abertura: backend devolve o objeto recem-criadoexport interface
FornecedorInicialResponse {
  Fornecedor: FornecedorXML;
}
export interface SalvaFornecedorRequest {
  Fornecedor: {
    _OID: string;
    _Nome: string;
    _CNPJ: string;
    Endereco?: { _OID: string };
  };
}
```

```
export interface BuscaFornecedoresRequest {
  OBJECTID: { _Nome: string; _Ativo: boolean };
}
export interface BuscaFornecedoresResponse {
  Response: FornecedorXML[];
}
```

`service/hooks.ts`

Um hook por request. Uma linha cada.

```
// src/pages/Fornecedor/Cadastro/service/hooks.tsimport { useCurioMutation } from
"@/hooks/useCurioMutation";
import { FORNECEDOR_RMS } from "./constants";
import {
  BuscaFornecedoresRequest,
  BuscaFornecedoresResponse,
  FornecedorInicialResponse,
  SalvaFornecedorRequest
} from "./interfaces";
export const useIncluiFornecedor = () => useCurioMutation<FornecedorInicialResponse,
void>(FORNECEDOR_RMS.OBTEM_DADOS);
export const useBuscaFornecedores = () =>  useCurioMutation<BuscaFornecedoresResponse,
BuscaFornecedoresRequest>(FORNECEDOR_RMS.SALVAR);
export const useSalvaFornecedor = () => useCurioMutation<void,
SalvaFornecedorRequest>(FORNECEDOR_RMS.SALVAR);
```

Assinatura: `useCurioMutation<TResposta, TParametros>(nomeDoRequest)`. Use `void` quando não há parâmetros ou não há resposta útil.

A página

```
// src/pages/Fornecedor/Cadastro/IncluirFornecedorPage.tsximport React, { useEffect, useRef }
from "react";
```

```
import { Box, Button } from "@mui/material";
import { UseCaseManager } from "@curio/client/react";
import { useAuth } from "@/context/AuthProvider";import { useUseCaseControls, useValidatedForm }
from "@/hooks";
import { fornecedorSchema, type FornecedorFormData } from "./schemas";import { FORNECEDOR_RMS }
from "./service/constants";import { useIncluiFornecedor, useSalvaFornecedor } from
"./service/hooks";
const IncluirFornecedorContent: React.FC = () => {  const { open, status } =
useUseCaseControls();
  const { data: fornecedorData, mutate: incluirFornecedor, isPending: isIncluindo } =
useIncluiFornecedor();  const { mutateAsync: salvarAsync, isPending: isSalvando } =
useSalvaFornecedor();
  // refs evitam reabrir/reinicializar em re-render
  const hasAttemptedOpenRef = useRef(false);
  const hasInitializedRef = useRef(false);
  useEffect(() => {
    if (status === "idle" && !hasAttemptedOpenRef.current) {      hasAttemptedOpenRef.current =
true;
      open();
    } else if (status === "open" && !hasInitializedRef.current) {      hasInitializedRef.current
= true;
      incluirFornecedor();
    }
  }, [status, open, incluirFornecedor]);
  const form = useValidatedForm({
    schema: fornecedorSchema,
    defaultValues: { _Nome: "", _CNPJ: "" }
  });
  const handleSalvar = async (data: FornecedorFormData) => {
    await salvarAsync({
      Fornecedor: {
        _OID: String(fornecedorData?.Fornecedor._OID ?? ""),
        _Nome: data._Nome,
        _CNPJ: data._CNPJ
      },
      msgSucesso: "Fornecedor salvo com sucesso!"
```

```
    });
  };
  return (
    <Box>      <Button onClick={form.handleSubmit(handleSalvar)} disabled={isIncluindo ||
isSalvando}>
        Salvar
      </Button>
      {/* campos — ver 11 */}
    </Box>
  );
};
const IncluirFornecedorPage: React.FC = () => {
  const { session } = useAuth();
  return (    <UseCaseManager session={session} useCaseId={FORNECEDOR_RMS.USE_CASE}
autoClose={false} openOnMount={false}>
      <IncluirFornecedorContent />
    </UseCaseManager>
  );
};
export default IncluirFornecedorPage;
```

Por que dois componentes

`UseCaseManager` é um provider. Os hooks `useUseCaseControls` e `useCurioMutation` consomem o contexto dele, então **precisam estar em um componente filho**. O componente externo só faz o wiring; o conteúdo real vive no `Content`.

Isso não é opcional. Chamar `useCurioMutation` no mesmo componente que renderiza `UseCaseManager` falha em runtime.

Props do `UseCaseManager`

Prop	Valor usual	Efeito
`session`	`useAuth()`	Sessão autenticada
`useCaseId`	`FORNECEDOR_RMS.USE_CASE`	Id do caso de uso no servidor

Prop	Valor usual	Efeito
`autoClose`	`false`	`true` fecha ao desmontar. Use `false` com navegação por abas
`openOnMount`	`false`	`false` dá controle explícito da abertura via `open()`

Com `openOnMount={false}`, **a página é responsável por chamar `open()`** — daí o `useEffect` com o `hasAttemptedOpenRef`.

useUseCaseControls

```
// src/hooks/useUseCaseControls.tsexport const useUseCaseControls = (options:
UseUseCaseControlsOptions = {}) => {
  const { enableLogs = false } = options;  const { open, close, status, error,
triggersFromCurrentState } = useUseCaseManager();  const { setNotification } =
useNotification();
  // open() do curio lanca; aqui vira notificacao
  const openSafe = useCallback(
    async (openOptions?: OpenOptions) => {
      try {
        await open(openOptions);
      } catch (err) {        const message = err instanceof Error ? err.message : "Erro ao abrir
o caso de uso";        setNotification({ message, type: "error", isAuthError: isAuthError(err)
});
      }
    },
    [open, setNotification]
  );
  // ...  return { open: openSafe, close, status, error, logCurrentStateTriggers:
logCurrentState };
};
```

`status` assume `"idle"` → `"open"`. A máquina de estados típica da página é exatamente o `useEffect` mostrado acima: abrir quando `idle`, inicializar quando `open`.

Descobrir os requests disponíveis

Passe `enableLogs: true` para logar, em desenvolvimento, os triggers que o caso de uso expõe no estado atual:

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

Útil quando você não sabe o nome exato do request. **Remova antes de commitar.**

`useCurioMutation`

Envolve `useMutation` do React Query sobre o `sendRequest` do caso de uso.

```
const { data, mutate, mutateAsync, isPending, isError } = useCurioMutation<TData,
TParams>("RM_NOME");
```

Mensagens de sucesso e erro

`msgSucesso`, `msgErro` e `msgErroFallback` são passados junto dos parâmetros e **removidos antes de chegar ao backend**:

```
salvar({
  Fornecedor: { _OID: "123", _Nome: "ACME" },
  msgSucesso: "Fornecedor salvo com sucesso!",
  msgErro: "Não foi possível salvar."
});
```

`msgSucesso` dispara notificação verde no sucesso. Para erro, a mensagem exibida segue uma ordem de prioridade, decidida uma única vez no `mutationCache.onError` global (07) — não no hook:

1. **`msgErro`**, se informado — **sempre prevalece**, mesmo que o backend tenha devolvido mensagem própria. É o dev pedindo explicitamente para sobrescrever.
2. Mensagem do backend (`error.message`, ou `error.detail` se for `ValidationError`), se houver.
3. **`msgErroFallback`**, se informado — só é usado quando o backend não devolveu mensagem

alguma.
4. Fallback genérico interno do hook ("Ocorreu um erro ao processar a requisição.").

```
salvar({
  Fornecedor: { _OID: "123", _Nome: "ACME" },  msgErroFallback: "Não foi possível salvar o fornecedor."  // se o backend devolver mensagem, ela aparece; msgErroFallback só entra se ele nao devolver nada
});
```

Omitir os três não silencia o erro — o `queryClient` global sempre notifica, no mínimo com o fallback genérico.

> **Não trate erro dentro de um hook específico da feature.** A prioridade acima é resolvida **uma vez**, no `mutationCache.onError` global. Um `onError` local em `useCurioMutation` duplicaria a notificação — foi exatamente esse bug que motivou consolidar a lógica num só lugar.

Callbacks

`onSuccess`, `onError`, `onSettled` e `onMutate` vão no **mesmo objeto** dos parâmetros, não num segundo argumento:

```
salvar({
  Fornecedor: { _OID: "123", _Nome: "ACME" },
  onSuccess: () => setModalAberto(true),
  onError: (error) => console.error(error)
});
```

Difere do React Query puro. `useCurioMutation` separa os callbacks dos parâmetros internamente.

Buscas genéricas: `useSearcher`

Para telas de busca sobre entidades que o backend expõe via operações genéricas (`120` = buscar, `134` = obter contexto), sem caso de uso dedicado:

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

Usa a sessão do `useAuth()` diretamente, **sem** `UseCaseManager`. Use quando a tela é só filtro + lista. Se houver estado no servidor (incluir, alterar, salvar), use `UseCaseManager`.

Contrato com o backend

O que o front precisa saber — o resto é responsabilidade de quem escreve o caso de uso no servidor.

Aspecto	**Contrato**
Identificação	Id numérico como string (`"4821"`)
Request	Nome em `SCREAMING_SNAKE_CASE`, prefixo `RM_` (`RM_SALVA_OBJETO`)
Parâmetros	Objeto aninhado espelhando a entidade (`{ Fornecedor: { _OID, _Nome } }`)
Filtros	Frequentemente sob a chave `OBJECTID`
Resposta	Objeto com a entidade na raiz (`{ Fornecedor: {...} }`) ou `{ Response: [...] }`
Primitivos	Prefixo `_` (`_Nome`, `_OID`)
Objetos	Sem prefixo (`Endereco`, `Documentos`)
Datas	String ISO com tempo: `"2026-08-18T00:00:00.0"`
Erro	`Error` lançado pelo transporte; `ValidationError` traz `detail: string[]`
Sessão morta	`DestinataryNotFoundError` ou `code === "-1"` — ver 05

Não invente nomes de request. Eles vêm do caso de uso do servidor. Confirme com quem o implementou ou use `enableLogs`.

Erros comuns

Sintoma	Causa
Hook lança "fora do contexto"	`useCurioMutation` no mesmo componente que renderiza o `UseCaseManager`
Request roda antes do caso de uso abrir	Faltou aguardar `status === "open"`
Caso de uso reabre a cada render	Faltou o `useRef` de guarda
`msgSucesso` chega ao backend	Versão do `useCurioMutation` sem o `delete params.msgSucesso`
Data rejeitada pelo backend	Faltou o sufixo `T00:00:00.0`

Revision #4
Created Thu, Aug 20, 2026 4:49 PM by Geraldo Barbosa
Updated Tue, Aug 25, 2026 4:52 PM by Geraldo Barbosa