# Frontend React — Exercício Guiado (fetch, `useEffect`, loading/erro)

Prof. Me. Tássio Auad

Laboratório de Programação FullStack

## Contexto

Dando continuidade ao VassCommerce, vamos consumir a API no **frontend** usando `fetch` com `async/await`, controlando **carregamento**, **erros** e usando `useEffect` (dependências e *cleanup*). Também veremos como **cancelar** requisições com `AbortController` para evitar efeitos colaterais.

## Objetivo

Ao final, você terá:

- Um **service** com `listarCategorias()` (GET) e `criarProduto()` (POST).
- Uma página `CatalogoPage` que busca dados com `useEffect`, exibe **loading/erro/dados**.
- Um exemplo de **cancelamento** de requisição com `AbortController`.

## Pré-requisitos

Projeto React criado (Vite/CRA), rodando em `http://localhost:{porta}` e API disponível em `/api`. Se sua rota real for diferente, ajuste a `BASE_URL`.

## Passo 1 — Crie o serviço HTTP (`services/api.js`)

### 1.1 Função de utilidade para ler JSON com segurança

```
// src/services/api.js
const BASE_URL = '/api' // ajuste se necessário

async function safeJson(res) {
  try { return await res.json() } catch { return null }
}
```

### 1.2 GET: listar categorias

```
export async function listarCategorias(signal) {
  const res = await fetch(`${BASE_URL}/categoria`, {
    method: 'GET',
    headers: { 'Accept': 'application/json' },
    signal // opcional: para cancelamento
  })
  if (!res.ok) {
    const err = await safeJson(res)
    const msg = err?.error || err?.message || `HTTP ${res.status}`
    throw new Error(msg)
  }
  return res.json()
}
```

### 1.3 POST: criar produto

```
export async function criarProduto(dto) {
  const res = await fetch(`${BASE_URL}/produto`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Accept': 'application/json'
    },
    body: JSON.stringify(dto)
  })
  if (res.status === 201) return res.json()
  if (res.status === 400) {
    const err = await safeJson(res)
    throw new Error(err?.error || 'Dados inválidos')
  }
  throw new Error(`HTTP ${res.status}`)
}
```

**Checkpoint:** código compila; nenhuma chamada ainda foi feita na UI.

## Passo 2 — Página que busca dados (`CatalogoPage.jsx`)

### 2.1 Estrutura inicial: estados, `useEffect` e renderização condicional

```
// src/pages/CatalogoPage.jsx
import { useEffect, useState } from 'react'
import { listarCategorias } from '@/services/api'
```

```
export default function CatalogoPage() {
  const [dados, setDados] = useState([])
  const [loading, setLoading] = useState(true)
  const [erro, setErro] = useState(null)

  useEffect(() => {
    let ativo = true
    listarCategorias()
      .then(json => ativo && setDados(json))
      .catch(e => ativo && setErro(e.message))
      .finally(() => ativo && setLoading(false))
    return () => { ativo = false } // evita setState após desmontagem
  }, [])

  if (loading) return <p>Carregando...</p>
  if (erro)    return <p style={{color:'crimson'}}>Erro: {erro}</p>

  return (
    <main>
      <h2>Categorias</h2>
      <pre>{JSON.stringify(dados, null, 2)}</pre>
    </main>
  )
}
```

**Checkpoint:** ao abrir a rota da página, você vê "Carregando…", depois JSON das categorias (ou uma mensagem de erro amigável).

## Passo 3 — Cancelamento com `AbortController` (refino do efeito)

Troque o `useEffect` para **evitar respostas tardias** quando o usuário sair da página rapidamente.

```
useEffect(() => {
  const ac = new AbortController()
  setLoading(true); setErro(null)

  listarCategorias(ac.signal)
    .then(json => setDados(json))
    .catch(e => {
      // ignorar abortos propositais
      if (e.name !== 'AbortError') setErro(e.message)
    })
    .finally(() => setLoading(false))
```

```
  return () => ac.abort() // cancela a fetch em andamento
}, [])
```

**O que mudou?** Passamos `signal` para a fetch e, no *cleanup*, chamamos `abort()`. Isso evita *race conditions*.

# Passo 4 — Formulário simples de criação (POST)

Inclua um pequeno formulário na mesma página (ou em outra) para exercitar `criarProduto()`.

## 4.1 Adicionar um mini-formulário

```
// dentro do return da CatalogoPage (abaixo do <h2/> por exemplo)
<section style={{marginTop:16}}>
  <h3>Novo Produto (exemplo)</h3>
  <FormNovoProduto onCriado={() => {
    // após criar, recarregar a lista de categorias se fizer sentido
    // (depende do design da API; aqui vamos só alertar)
    alert('Produto criado! (exemplo)')
  }}/>
</section>
```

## 4.2 Componente do formulário

```
// no mesmo arquivo (ou em src/components/FormNovoProduto.jsx)
import { useState } from 'react'
import { criarProduto } from '@/services/api'

function FormNovoProduto({ onCriado }) {
  const [nome, setNome] = useState('')
  const [valor, setValor] = useState('')
  const [categoriaId, setCategoriaId] = useState('')
  const [loading, setLoading] = useState(false)
  const [erro, setErro] = useState(null)

  async function onSubmit(e) {
    e.preventDefault()
    try {
      setLoading(true); setErro(null)
      await criarProduto({
        nome,
        valor: Number(valor),
        categoriaId: Number(categoriaId)
      })
```

```
      setNome(''); setValor(''); setCategoriaId('')
      onCriado && onCriado()
    } catch (ex) {
      setErro(ex.message)
    } finally {
      setLoading(false)
    }
  }

  return (
    <form onSubmit={onSubmit}>
      <input placeholder="Nome" value={nome}
             onChange={e => setNome(e.target.value)} />
      <input placeholder="Valor" value={valor} type="number" step="0.01"
             onChange={e => setValor(e.target.value)} />
      <input placeholder="Categoria ID" value={categoriaId} type="number"
             onChange={e => setCategoriaId(e.target.value)} />
      <button disabled={loading} type="submit">Criar</button>
      {erro && <p style={{color:'crimson'}}>{erro}</p>}
    </form>
  )
}
```

**Checkpoint:** ao enviar dados válidos, a API retorna `201` e o formulário limpa. Em erro `400`, a mensagem amigável aparece.

## Passo 5 — Experimentos guiados

- **Troque o endpoint** `/categoria` por uma URL inválida e veja o **tratamento de erro**.
- **Simule lentidão** (dev tools) e navegue para fora da página: confirme no console que o **abort** funcionou (sem "update on unmounted").
- **Provoque 400** no POST (por exemplo, valor vazio) e observe a mensagem vinda do backend.

## Erros comuns (e como resolver)

- **Esquecer** `Accept/Content-Type`: APIs podem recusar sem os cabeçalhos corretos.
- **Não checar** `res.ok`: você renderiza erro como se fosse sucesso. Sempre trate 4xx/5xx.
- **Falta de** `return` **no *cleanup*** do `useEffect`: pode gerar *warnings* e comportamentos estranhos.

- **Misturar responsabilidade de UI e HTTP**: mantenha *services* para a rede; páginas cuidam do estado visual.

## Checklist de entrega

- ( ) `services/api.js` com `listarCategorias()` e `criarProduto()`.
- ( ) `CatalogoPage` usa `useEffect` com **deps** e **cleanup** (`AbortController`).
- ( ) Renderização condicional de **loading**, **erro** e **dados**.
- ( ) Formulário POST funcional com mensagens de erro amigáveis.
- ( ) Código organizado (páginas vs. services) e comentários breves explicativos.

## Desafios bônus (opcional)

- Implementar **retry** no erro com um botão "Tentar novamente".
- Paginação simples: suportar `/categoria?page=1&size=10`.
- Mostrar **Skeleton** no lugar do texto "Carregando…".
- Extrair um **hook** `useFetchCategorias` que encapsule loading/erro/dados.

## Entrega

- Código-fonte (projeto React) com a página e os services.
- **README.md** explicando como rodar e a rota da página.
- (Opcional) Prints mostrando os três estados: *loading*, *erro*, *sucesso*.