05 — Curio: conexão e sessão

05 — Curio: conexão e sessão

> Como o front autentica, mantém e encerra a sessão com o backend Curio. Toda a integração com `@curio/client` está confinada em `src/lib/curio/` e `src/api/session.ts`. Nenhuma página importa `@curio/client` para configurar transporte.

Camadas

```
Página
  └─ useAuth()                 src/context/AuthProvider.tsx    ← o que a página usa
useAuthQuery()       src/hooks/useAuthQuery.ts       estado via React Query
login/connect   src/api/session.ts              operações de sessão
getSessionManager()  src/lib/curio/index.ts  transporte
                      └─ @curio/client
```

Componentes usam `useAuth()`. Nunca `useAuthQuery` diretamente, nunca `getSessionManager()`.

`src/lib/curio/`

Quatro arquivos.

`index.ts` — fábrica do transporte

```
```

```
// src/lib/curio/index.ts
import { V3RequestParser } from "@curio/client/parsers";import { Session } from "./Session";
import { SessionManager } from "./SessionManager";import { createV3Validator } from
"@curio/client/validators";
import { FetchLink } from "@curio/client/links";
export interface Config {  service: { url: string; server: string; system: string; port: string
};
  accessToken: string;
  resources: { get: string; put: string; viewer: string };
  logs: boolean;
}
const createSessionManager = (configuration: Config) => {  const link = new
FetchLink(configuration.service);  const requestParser = new
V3RequestParser(configuration.service);
  // pipeline: request -> parse -> json -> post -> valida  return new
SessionManager(configuration.service, (request) =>
    Promise.resolve(request)
      .then(requestParser.parse)
      .then(JSON.stringify)
      .then(link.parse)
      .then(link.post)
      .then((response) => response.json())
      .then(createV3Validator(request).validate)
  );
};
export const getSessionManager = async () => {
  const CONFIG_PATH = "./config.json";
  const cnfg = await (await fetch(CONFIG_PATH)).json();
  return createSessionManager(cnfg);
};
export type { Service as SV } from "./SessionManager";
export { Session, SessionManager };
export { isAuthError } from "./isAuthError";
export * from "./SessionManager";
```

> `getSessionManager()` **cria uma instância nova a cada chamada** — não é singleton, apesar do nome. Isso é aceitável porque só é chamado em `login`, `connect` e em requests anônimos. **Não chame dentro de componente ou hook de página.** Para falar com o backend a partir de uma tela, use a sessão do `useAuth()` — ver 06.

`Session.ts` — a sessão do app

```
// src/lib/curio/Session.ts
import { MainUseCase } from "@curio/client";
export class Session extends MainUseCase {
  module: number | undefined;
  storageToken: string | undefined;
}
```

Estende `MainUseCase` só para carregar `module` e `storageToken`. Se o projeto precisar de mais estado por sessão, é aqui.

`SessionManager.ts` — detecção de sessão morta

```
// src/lib/curio/SessionManager.tsimport { MainUseCase, RequestDriver, SecurityManager,
UseCaseMessageType } from "@curio/client";
import { Session } from "./Session";
import { isAuthError } from "./isAuthError";import { ABORT } from
"@curio/client/utils/constants";
export interface ConfigService {
  url: string;
  server: string;
  system: number;
  port: number;
  module: number;
```

```
  version: number;
}
// chave unica por app — nunca reaproveitar de outro projetoexport const STORAGEKEY =
"br.com.nomedoprojeto";
export class SessionManager extends SecurityManager {
  public session: MainUseCase | undefined;
  private _service: ConfigService;
  private _driver: RequestDriver;
  // eslint-disable-next-line @typescript-eslint/no-explicit-any -- @curio nao tipa o service
constructor(service: any, driver: RequestDriver) {
    super(service, driver);
    this._service = service;
    this._driver = driver;
  }
  public async openMainUseCase<U extends typeof MainUseCase>(username: string, password: string,
constructor?: U) {    this.session = await super.openMainUseCase(username, password,
constructor);
    this._attachListeners();
    return this.session! as InstanceType<U>;
  }
  public connectMainUseCase(mainUseCase: MainUseCase) {
    this.session = mainUseCase;
    this._attachListeners();
    return super.connectMainUseCase(mainUseCase);
  }
  // sessao anonima: pre-login (ex. obter versao)
  public anonymousSession() {
    return new Session(0, this._service, this._driver);
  }
  private _attachListeners() {    this.session!.addListener(ABORT, () =>
this._unauthenticate(false));    // eslint-disable-next-line @typescript-eslint/no-explicit-
any -- @curio nao tipa a mensagem    this.session!.addListener(UseCaseMessageType.RESPONSE,
(message: any) => {
      if (isAuthError(message.error)) this._unauthenticate(true);
    });
  }
  private _unauthenticate(expired: boolean) {
```

```
    this.session = undefined;
    if (expired) localStorage.removeItem(STORAGEKEY);
  }
}
```

`STORAGEKEY` identifica a sessão no storage. **Troque por um valor próprio do projeto novo.**

> **Cuidado com storage inconsistente.** Se o token é escrito em `sessionStorage` mas a limpeza remove de `localStorage` (ou vice-versa), a limpeza não limpa nada — são storages diferentes. **Escolha um e use o mesmo nos dois lugares.** O snippet acima e o de `session.ts` abaixo usam `sessionStorage` de forma consistente — token de sessão não deve sobreviver ao fechamento da aba.

`isAuthError.ts` — o que conta como falha de autenticação

```
// src/lib/curio/isAuthError.ts
import { DestinataryNotFoundError } from "@curio/client/errors";
// code -1 = sessao inexistente no servidorexport const isAuthError = (error: unknown): boolean
=>  error instanceof DestinataryNotFoundError || String((error as { code?: unknown } | null |
undefined)?.code) === "-1";
```

Ponto único de decisão. Se o backend introduzir outro código de sessão inválida, muda só aqui.

`src/api/session.ts`

```
// src/api/session.tsimport { ConnectionError, DestinataryNotFoundError } from
"@curio/client/errors";
import { Session, getSessionManager } from "../lib/curio";import { STORAGEKEY } from
"../lib/curio/SessionManager";
export interface LoginParams {
```

```
  email: string;
  password?: string;
}
export async function login(params: LoginParams) {
  const { email, password } = params;
  try {
    const sessionManager = await getSessionManager();    const session = await
sessionManager.openMainUseCase(email, password ?? "", Session);
    session.module = 0;
    session.storageToken = STORAGEKEY;    sessionStorage.setItem(STORAGEKEY,
session.token.toString());
    return session;
  } catch (error) {
    return error as Error;
  }
}
// reconecta a partir do token guardado (refresh de pagina)export async function connect(token:
string) {
  try {
    const sessionManager = await getSessionManager();    sessionManager.connectMainUseCase(new
Session(token, sessionManager.service, sessionManager.driver));    await
sessionManager.session!.timeoutCheck();    const session = sessionManager.session! as
Session;
    session.module = 0;
    session.storageToken = STORAGEKEY;    sessionStorage.setItem(STORAGEKEY,
session.token.toString());
    return session;
  } catch (error) {
    if (error instanceof DestinataryNotFoundError) return error;    if (error instanceof
ConnectionError) return error;
    return error as Error;
  }
}
export async function logoutSession(session?: Session) {
  if (session) session.abort();
  sessionStorage.removeItem(STORAGEKEY);
```

```
}
```

> `login` e `connect` **retornam o erro em vez de lançar.** Quem chama precisa testar `result instanceof Error`. É contraintuitivo e fácil de errar — se o projeto novo puder, prefira deixar lançar e tratar no hook.
>
> **Cuidado com `localStorage.clear()` no logout** — ele apaga preferências de UI não relacionadas à sessão. O snippet acima remove só a chave da sessão.

Requests anônimos (pré-login)

Para chamar o backend antes de autenticar — por exemplo, exibir a versão do servidor na tela de login:

```
// src/api/session.ts
export const VERSION = { useCase: "3916", getVersion: "RM_OBTER_VERSAO" };
export async function obterVersaoRequest() {
  const sessionManager = await getSessionManager();  const anonymousSession =
sessionManager.anonymousSession();  const uc = await
anonymousSession?.openUseCase(VERSION.useCase);  const response = await
uc?.sendRequest(VERSION.getVersion);  await uc?.abort(); // sempre fechar — sessao anonima nao
tem dono
  return response.Versao._ as string;
}
```

`AuthProvider`

```
// src/context/AuthProvider.tsx (trecho essencial)export const AuthProvider: React.FC<{
children: React.ReactNode }> = ({ children }) => {
  const { notification } = useNotification();
  const navigate = useNavigate();  const { session, isLoading, isAuth, login: authLogin, logout:
```

```
authLogout, refetchConnection } = useAuthQuery();
  // reconecta no boot se ha token guardado
  useEffect(() => {
    if (sessionStorage.getItem(STORAGEKEY)) refetchConnection();
  }, []);
  const logout = useCallback(() => {
    authLogout();
    navigate("/", { replace: true });
  }, [authLogout, navigate]);
  // fonte unica de logout: reage a notificacao de erro de auth.  // ref evita disparar 2x — logout muda de identidade a cada render.  const handledNotificationRef = useRef<NotificationType | null>(null);
  useEffect(() => {
    if (!notification) return;    const expiredSession = notification.message === "Sessão expirada!";
    if (!notification.isAuthError && !expiredSession) return;    if (handledNotificationRef.current === notification) return;    handledNotificationRef.current = notification;
    logout();
  }, [notification, logout]);
  const login = async (params: LoginCredentials) => {
    await authLogin(params);
    navigate("/dashboard");
  };
  return (    <AuthContext.Provider value={{ isAuth, session, isLoading, login, logout }}>{children}</AuthContext.Provider>
  );
};
```

Ciclo de vida

Evento	O que acontece
Boot com token	`refetchConnection()` → `connect(token)` → sessão restaurada
Login	`login()` → token no `sessionStorage` → navega para `/dashboard`

Evento	O que acontece
Request com sessão morta	`isAuthError` → notificação com `isAuthError: true` → `AuthProvider` desloga
`ABORT` do servidor	Listener no `SessionManager` limpa a sessão
Logout manual	`session.abort()` → limpa storage → navega para `/`

Detecção de expiração por string

```
const expiredSession = notification.message === "Sessão expirada!";
```

Comparar mensagem literal é frágil: muda a tradução no backend, o auto-logout para de funcionar sem erro visível. **Preferível** é o backend sinalizar com um código que `isAuthError` reconheça. Se precisar manter, isole a string numa constante e documente a dependência.

Revision #2
Created Thu, Aug 20, 2026 4:48 PM by Geraldo Barbosa
Updated Tue, Aug 25, 2026 4:51 PM by Geraldo Barbosa