04 — Ambientes e configuração

04 — Ambientes e configuração

> Como o projeto descobre em runtime a qual backend falar. Dois mecanismos distintos: `config/{env}.json` (por ambiente, versionado) e `.env` (por máquina, local). Não os confunda. Regra de ouro: **`public/config.json` é gerado, nunca editado à mão, nunca versionado.**

O fluxo

```
config/{env}.json ──[setEnvironment.js]──▶ public/config.json ──[fetch em runtime]──▶ SessionManager
                                                ▲
                                                |
                                           .env (override)
```

1. `npm run dev` executa `npm run env:dev` antes do Vite.
2. `setEnvironment.js dev` lê `config/dev.json`, aplica overrides do `.env` e escreve `public/config.json`.
3. Em runtime, `getSessionManager()` faz `fetch("./config.json")` e constrói o transporte.

Por que em runtime e não em build: permite apontar o mesmo bundle para outro servidor

trocando um arquivo, sem recompilar. É o que viabiliza o deploy ISAPI — ver 15.

`config/{env}.json`

Um arquivo por ambiente. Versionados.

```
// config/dev.json
{
  "service": {    "url":
"https://srvd1.dev.exemplo.com.br/cxClient/cxIsapiClient.dll/gatewayJSONBalanced?version=4",
"server": "192.168.0.00",
    "system": "61",
    "port": "5369"
  },
  "accessToken": "SUBSTITUA",
  "resources": {    "get":
"https://srvd1.dev.exemplo.com.br/cxClient/cxIsapiClient.dll/getpr",    "put":
"https://srvd1.dev.exemplo.com.br/cxClient/cxIsapiClient.dll/putpr",    "viewer":
"https://srvd1.dev.exemplo.com.br/relatorios/viewer/rel.html?id="
  },
  "logs": true
}
```

Chaves

Chave	Tipo	O que é
`service.url`	`string`	Endpoint do gateway Curio. Inclui `?version=4`
`service.server`	`string`	IP/host do servidor de aplicação de destino
`service.system`	`string`	Identificador numérico do sistema no Curio
`service.port`	`string`	Porta do servidor de aplicação
`accessToken`	`string`	Token de acesso ao gateway

Chave	Tipo	O que é
`resources.get`	`string`	Endpoint de download de recurso/arquivo
`resources.put`	`string`	Endpoint de upload
`resources.viewer`	`string`	URL base do visualizador de relatórios; recebe o id concatenado
`logs`	`boolean`	Liga logs verbosos do transporte

O tipo é declarado em `src/lib/curio/index.ts`:

```
// src/lib/curio/index.ts
export interface Config {  service: { url: string; server: string; system: number; port: number
};
  accessToken: string;
  resources: { get: string; put: string; viewer: string };
  logs: boolean;
}
```

> **Cuidado com mentira de tipo.** Se o JSON trouxer `system` e `port` como **string** mas a interface `Config` declarar **number**, o TypeScript não acusa nada — o JSON é lido via `fetch`, sem checagem de tipo em runtime, e o Curio aceita ambos os formatos. Ainda assim é uma mentira de tipo. **Declare a interface fiel ao que o JSON realmente contém** (`system: string; port: string`, se for o caso) em vez de forçar um tipo que não corresponde ao dado real.

`setEnvironment.js`

```
#!/bin/node
// setEnvironment.js — copia config/{env}.json para public/config.json
import fs from "fs";
import path from "path";
const environment = process.argv[2];
if (!environment) {
```

```
  console.error("Por favor, especifique um ambiente: dev, homolog ou prod");
  process.exit(1);
}
// .env so eh lido se Node >= 20.6 e arquivo existeif (typeof process.loadEnvFile === "function"
&& fs.existsSync(".env")) {
  process.loadEnvFile(".env");
}
try {  const envFileContent = JSON.parse(fs.readFileSync(`./config/${environment}.json`,
"utf8"));
  // override so vale em dev — prod nunca le .env
  if (environment === "dev") {
    if (process.env.LOCAL_SERVER) {      envFileContent.service.server =
process.env.LOCAL_SERVER;      console.log(`service.server sobrescrito via LOCAL_SERVER:
${process.env.LOCAL_SERVER}`);
    }
    if (process.env.ACCESS_TOKEN) {      envFileContent.accessToken =
process.env.ACCESS_TOKEN;
      console.log("accessToken sobrescrito via ACCESS_TOKEN");
    }
  }
  console.log(`Configurando ambiente: ${environment}`);
  const publicDir = path.join(process.cwd(), "public");  if (!fs.existsSync(publicDir))
fs.mkdirSync(publicDir, { recursive: true });
  fs.writeFileSync(path.join(publicDir, "config.json"), JSON.stringify(envFileContent,
undefined, 2));
  console.log(`Ambiente ${environment} configurado com sucesso!`);
} catch (error) {
  console.error(`Erro ao configurar ambiente ${environment}:`, error.message);
  process.exit(1);
}
```

> **Nunca logue o valor do `ACCESS_TOKEN` no console.** Token não vai para stdout. O snippet acima já evita isso.

`.env` — variáveis por máquina

Serve para o dev apontar o ambiente `dev` ao **seu** servidor local sem sujar o `config/dev.json` compartilhado.

```
# .env.example — copie para .env e ajuste. .env nao eh versionado.
# Sobrescreve service.server no config gerado (so em dev)
LOCAL_SERVER="ENDERECO_DO_SERVIDOR_LOCAL"# Sobrescreve accessToken (so em dev)
ACCESS_TOKEN="TOKEN_DE_ACESSO"
```

Regras:

- `.env.example` é **versionado** e contém apenas placeholders.
- `.env` é **gitignored**.
- Overrides só se aplicam ao ambiente `dev`. `homolog` e `prod` ignoram o `.env` por construção — isso é proposital, para que um `.env` esquecido não vaze para um build de produção.
- Não use `import.meta.env` / prefixo `VITE_` para configuração de backend. Isso embute o valor no bundle em tempo de build e derrota o propósito do `config.json`. `import.meta.env.DEV` (flag de modo) é aceitável.

Adicionar um ambiente novo

1. Crie `config/{nome}.json` com todas as chaves.
2. Adicione ao `package.json`:

   ```
   "env:{nome}": "node ./setEnvironment.js {nome}","start:{nome}": "npm run env:{nome} &&
   vite"
   ```

3. Se o ambiente tiver build próprio, adicione `"build:{nome}"`.

Segurança

- **`public/config.json` no `.gitignore`.** É gerado; versioná-lo cria conflito toda vez que alguém troca de ambiente.

- `config/*.json` **é servido ao browser.** Tudo ali é público para quem abrir o DevTools. Não coloque segredo que não possa ser lido pelo usuário final.
- `accessToken` **em** `config/dev.json` **fica no histórico do Git para sempre.** Nunca commite um token real nesse arquivo. Commite `"accessToken": "SUBSTITUA"` e distribua o valor real via `.env` (dev) ou via substituição no pipeline (homolog/prod).

Revision #2

Created Thu, Aug 20, 2026 4:47 PM by Geraldo Barbosa

Updated Tue, Aug 25, 2026 4:51 PM by Geraldo Barbosa