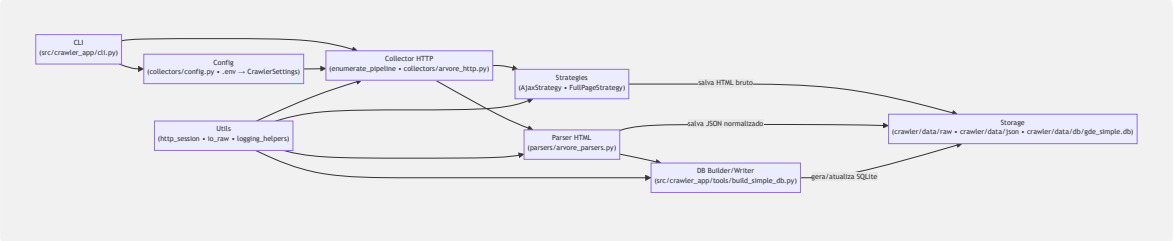
# C4 — Nível 2: Containers (Crawler)




## O que cada container faz (texto simples)

- **CLI (`src/crawler_app/cli.py`)** Roda os comandos: `collect`, `modalities`, `curriculum`, `build-db`. Recebe flags (ex.: `--strategy`, `--base-url`, filtros de curso/ano) e chama o pipeline com logs claros.
- **Config (`collectors/config.py` + `.env` → `CrawlerSettings`)** Lê as variáveis do `.env` (base URL, timeouts, cooldown, pastas) e permite sobrescrita via CLI/CI.
- **Collector HTTP (`enumerate_pipeline` + `collectors/arvore_http.py`)** Lista cursos, modalidades e currículos; monta URLs, aplica *polite sleep* e registra respostas.
- **Strategies (`AjaxStrategy` • `FullPageStrategy`)** Duas formas equivalentes de coletar o currículo (XHR vs. página completa). Usadas como *fallback* entre si. Salvam o **HTML bruto** para auditoria.
- **Parser HTML (`parsers/arvore_parsers.py`)** Converte HTML do GDE em **JSON normalizado** (listas/dicionários). Extrai selects, blocos de integração e atributos de disciplinas (tipo, semestre, catálogo).
- **DB Builder/Writer (`src/crawler_app/tools/build_simple_db.py`)** Lê os JSONs e constrói/atualiza o **SQLite** em `crawler/data/db/gde_simple.db` (schema + carga idempotente).
- **Utils (`http_session`, `io_raw`, `logging_helpers`)** Sessão HTTP com login/retries; escrita de arquivos RAW/JSON com nomes idempotentes; helpers de log.
- **Storage**
  - `crawler/data/raw/`: HTML bruto coletado.
  - `crawler/data/json/`: JSON já parseado/normalizado.
  - `crawler/data/db/gde_simple.db`: banco consolidado (consumo pelo backend em leitura).

---

## Refatorações e Code Smells (refactoring.guru)

> Referência: https://refactoring.guru/refactoring/smells

| Smell (refactoring.guru) | Sinal no Crawler | Refactor aplicado | Onde aparece |
|---|---|---|---|
| **Long Method** / **Complex Conditionals** | Lógica de "como coletar" crescendo e com `if/elif` por rota | **Replace Conditional with Polymorphism** + **Extract Method** | Estratégias `AjaxStrategy/FullPageStrategy` (separam variações de coleta); `cli` chama o ponto único |
| **Duplicated Code** | Repetição de logging/retry em diferentes pontos | **Extract Method/Class** para utilitários | `utils/logging_helpers.py`, `utils/http_session.py` concentram comportamento comum |
| **Feature Envy** | Coletor manipulando detalhes de escrita de arquivos | **Move Method** para o módulo de IO | `utils/io_raw.py` padroniza nomes/paths e escrita de RAW/JSON |
| **Primitive Obsession** | Muitos parâmetros soltos para configuração | **Introduce Parameter Object** | `CrawlerSettings` centraliza base URL, timeouts, cooldown, diretórios |
| **Shotgun Surgery** | Mudanças no schema impactavam vários pontos | **Encapsulate Construction** do DB | `tools/build_simple_db.py` vira o *único* lugar de schema/carga; demais só geram JSON |
| **Inappropriate Intimacy** | Parser "conhecia" regras do DB | **Separate Concerns** (Parser puro → DB Builder consome) | `parsers/arvore_parsers.py` só gera dados; `build_simple_db.py` cuida do SQLite |
| **Comments Smell / Magic Numbers** | Comentários explicando "por que" de números soltos | **Replace Magic Number with Named Const** + docstrings curtas | Constantes em `config` e small docstrings nos pontos críticos |
| **Dead Code** (quando | Helpers antigos não | **Remove Dead Code** | Limpezas pontuais durante a adoção das strategies e utils |

identificado) usados

**Resultado prático:** menos `ifs`, menos repetição, pontos de mudança centralizados (config/DB), e logs/artefatos RAW previsíveis. O pipeline fica mais **legível, testável e seguro** para evoluir.