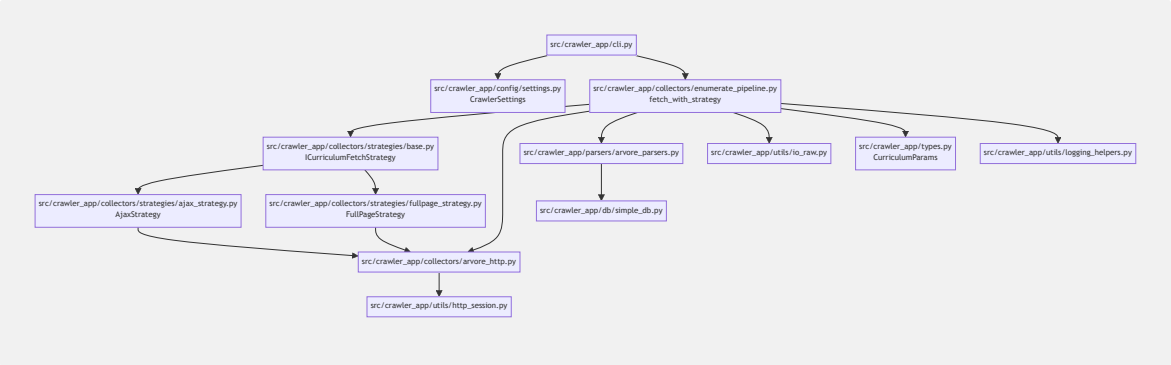
C4 — Nível 3: Componentes (Crawler)



ASCII: CLI → pipeline → strategies/arvore/parser; utils apoiam; saída segue para JSON/DB.

Componentes (linguagem simples)

`src/crawler_app/cli.py`

- **Função:** ponto de entrada. Lê flags (`--strategy`, filtros de curso/ano) e chama o pipeline.
- **I/O:** argumentos da CLI + `.env` (via `CrawlerSettings`); imprime logs e aciona coleta/parsing/DB.

`src/crawler_app/config/settings.py` (`CrawlerSettings`)

- **Função:** centraliza configuração (base URL, timeouts, cooldown, pastas de saída).
- **I/O:** carrega do `.env` e permite sobrescrita por flags; expõe um objeto único de settings.

`src/crawler_app/types.py` (`CurriculumParams`)

- **Função:** empacota parâmetros de currículo (curso, catálogo, modalidade, período, cp).
- **I/O:** evita passar “parâmetros soltos” entre funções; torna assinaturas mais claras.

`src/crawler_app/collectors/enumerate_pipeline.py`

- **Função:** orquestra a coleta: escolhe estratégia, chama HTTP, salva RAW, chama parser e organiza JSON.
- **Pontos-chave:** `fetch_with_strategy()`, `enumerate_dimensions()` e utilitários de leitura/conversão.

`src/crawler_app/collectors/arvore_http.py`

- **Função:** encapsula chamadas HTTP ao GDE (URLs, headers, espera cortês, captura de resposta).

- **Pontos-chave:** `polite_sleep()`, `fetch_arvore_page()`, `fetch_modalidades_fragment()`, `fetch_curriculum_ajax_response()`, `fetch_full_arvore_page()`, `curriculum_label()`.

`src/crawler_app/collectors/strategies/base.py` (`ICurriculumFetchStrategy`)

- **Função:** define o contrato para variações de coleta de currículo.
- **Observação:** permite alternar entre rotas AJAX e página completa sem `if/elif` espalhados.

`src/crawler_app/collectors/strategies/ajax_strategy.py` (`AjaxStrategy`)

- **Função:** coleta via AJAX autenticado; registra e salva HTML/fragmento como RAW.
- **Uso:** caminho "rápido" quando endpoints XHR estão estáveis.

`src/crawler_app/collectors/strategies/fullpage_strategy.py` (`FullPageStrategy`)

- **Função:** coleta a página completa como **fallback** quando AJAX falha ou está fora do ar.
- **Uso:** garante resiliência mantendo a trilha de auditoria (HTML bruto).

`src/crawler_app/parsers/arvore_parsers.py`

- **Função:** transforma HTML da árvore em JSON normalizado (cursos, catálogos, modalidades, disciplinas).
- **Saída:** listas/dicionários prontos para armazenar em `data/json` e alimentar o DB.

`src/crawler_app/db/simple_db.py`

- **Função:** schema e upserts no SQLite; centraliza criação/atualização de tabelas.
- **Pontos-chave:** `get_conn()`, `ensure_schema()`, `upsert_*()` e `commit()`.

`src/crawler_app/utils/http_session.py`

- **Função:** cria `requests.Session` com retries, cookies/CSRF e login.
- **Efeito:** reduz falhas intermitentes e padroniza autenticação.

`src/crawler_app/utils/io_raw.py`

- **Função:** nomes idempotentes e escrita de arquivos RAW/JSON; garante diretórios.
- **Efeito:** organização previsível dos artefatos em `crawler/data/raw` e `crawler/data/json`.

`src/crawler_app/utils/logging_helpers.py`

- **Função:** logs úteis (status, selects, amostras de HTML) para inspecionar rapidamente a coleta.
- **Efeito:** facilita depuração sem abrir o arquivo inteiro.

Refatorações e Code Smells (refactoring.guru)

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

Smell (guru)	Sinal no código	Refactor aplicado	Onde
Long Method / Complex Conditionals	Muitos `if/elif` para caminhos de coleta	**Replace Conditional with Polymorphism** + **Extract Method**	`strategies/base.py`, `ajax_strategy.py`, `fullpage_strategy.py`; `pipeline` chama 1 método polimórfico
Duplicated Code	Repetição de login/retry/log em pontos distintos	**Extract Class/Method** (utilitários)	`utils/http_session.py`, `utils/logging_helpers.py`, `utils/io_raw.py`
Feature Envy	Coletor manipulando escrita de arquivos diretamente	**Move Method** para IO dedicado	`utils/io_raw.py` concentra salvar RAW/JSON; pipeline só chama a API
Primitive Obsession	Funções com muitos parâmetros primitivos	**Introduce Parameter Object**	`types.py` → `CurriculumParams`; `settings.py` → `CrawlerSettings`
Shotgun Surgery	Mudanças no schema afetavam vários módulos	**Encapsulate Construction / Centralize Schema**	`db/simple_db.py` concentra `ensure_schema()` e `upsert_*()`
Inappropriate Intimacy	Parser "sabia" do formato do DB	**Separate Concerns** (parser puro → DB cuida do schema)	`parsers/arvore_parsers.py` só normaliza dados; `db/simple_db.py` grava
Comments Smell / Magic Numbers	Números "mágicos" de timeout/cooldown	**Replace Magic Number with Named Const** + settings	`settings.py` e constantes documentadas; menos comentários explicando números

Dead Code (quando encontrado)	Helpers não utilizados após refactor	**Remove Dead Code**	Limpezas pontuais em utils/collectors durante adoção das strategies

Efeito geral: menos condicionais espalhadas, utilitários reaproveitáveis, assinaturas curtas (objetos de parâmetro), schema/gravação centralizados e logs padronizados. O pipeline fica mais simples de entender, testar e manter.