

# 🕷️ Crawler — Documentação Viva (GDE Mobile · MC656-2025-s2)

**Escopo:** este documento lista **o que já foi feito** e **o que falta** no módulo *crawler*, no formato de **checklists hierárquicas** (listas, sublistas, sub-sublistas), para orientar o desenvolvimento contínuo do projeto (não é um artefato de A3, e sim do **projeto**).

## 1) Objetivo e Resultado Esperado

- **Objetivo do crawler**
  - Coletar dados do GDE (árvore/disciplinas/ofertas) com **login real** e **sessão autenticada**.
  - **Limpar e normalizar** dados para consumo pelo backend/app.
  - **Persistir em JSON** (snapshots) e permitir **seed do banco**.
- **Resultado final desejado**
  - Banco geral (seedável) cobrindo **~15 anos de catálogos**.
    - Estrutura inicial de DB simples (`simple_db.py`, `simple_schema.sql`).
    - Conteúdo completo com **semestre recomendado** por disciplina (após troca de getter).
    - Consolidação e deduplicação multi-ano.

## 2) Arquitetura Atual (visão de código)

- **Estrutura de diretórios (existente no zip)**
  - `crawler/readme.md` (orientações iniciais de ambiente)
  - `crawler/run_all_courses.sh` e `crawler/scripts/run_all.ps1` (execução)
  - `crawler/template.env` (exemplo de variáveis)
  - `crawler/src/crawler_app/`
    - `__init__.py` e `cli.py` (CLI base)
    - `collectors/`
      - `arvore_http.py` (requisições HTTP à árvore)
      - `config.py` (BASE + targets)
      - `enumerate_dimensions.py` (runner para coleta parametrizada)
      - `enumerate_pipeline.py` (pipeline de varredura e gravação RAW)
    - `parsers/`
      - `arvore_parsers.py` (**BeautifulSoup** para extrair selects/itens)
    - `db/`
      - `simple_db.py` (escrita/seed simples)
      - `simple_schema.sql` (schema inicial)
    - `tools/`
      - `build_simple_db.py` (ferramenta de construção)
    - `utils/`
      - `hashing.py` (hashes de conteúdo)
      - `http_session.py` (**create\_session**, **ensure\_csrf\_cookie**, **login\_via\_ajax**)
      - `io_raw.py` (estrutura de pastas RAW/WRITE)

- `logging_helpers.py` (`log_response_with_selects`, print de cookies)
  - **Próximas adições de arquitetura**
    - `clients/gde_api.py` (clientes tipados para endpoints JSON)
    - `parse/normalizers.py` (camada de normalização Pydantic)
    - `validate/` (regras de qualidade de dados e diffs)
    - `consolidate/` (merges multi-ano e deduplicação)
    - `export/` (artefatos para seed do backend)
- 

### 3) Setup, Autenticação e Sessão

- **Ambiente e variáveis**
    - `.env` (modelo em `template.env` / instruções em `readme.md`)
    - Leitura de credenciais (RA/senha)
  - **Sessão HTTP**
    - `create_session()` com headers consistentes (UA, accept, etc.)
    - `ensure_csrf_cookie()` (pré-requisitos do login)
    - `login_via_ajax()` (login real, sessão com token)
  - **Fortalecimentos pendentes**
    - Renovação de token (*refresh*) automática
      - Checagem de validade a cada  $N$  requisições
      - *Retry/backoff* em 401/403
    - Sanitização de logs
      - Nunca logar RA/senha/token/cookies
      - *Redactions* consistentes nos arquivos RAW/DEBUG
- 

### 4) Coleta (Collectors) — Estado Atual vs Pendências

#### 4.1 Coleta de Árvore/Seletores (HTML)

- **Fetch de páginas e fragments** (`arvore_http.py`)
  - `/arvore/` (GET com parâmetros)
  - Fragmento de “modalidades” (XHR/HTML)
  - *Polite sleep* para reduzir carga
- **Pipeline de varredura** (`enumerate_pipeline.py` → `enumerate_dimensions.py`)
  - Limpeza e recriação de pasta RAW por execução
  - *Selects* capturados e serializados
  - Iteração parametrizável por **curso/catalogo/periodo** via `config.py`
- **Pendências** (HTML → JSON endpoints)
  - Migrar para **text-scraper/endpoint-scraper** (capturar JSON do front)
    - Mapear endpoints e *query params* no `docs/gde_endpoints.md`
    - Reproduzir chamadas XHR com mesma sessão/autorização
    - Diminuir dependência de parsing HTML e *fragility* de seletores

#### 4.2 Coleta de Disciplinas/Ofertas/Professores

- **Disciplinas obrigatórias/eletivas** (estado atual)

- Extraídas da árvore/fragmentos
- **Pré/co-requisitos** (estrutura de integralização)
  - Grafo: nós (disciplinas) e arestas (relação)
- **Semestre recomendado**
  - Incluir campo `recommended_semester` (após troca de getter)
  - Validar coerência por `curso/catalogo`

#### 4.3 Coleta Multi-Ano (15 anos)

- **Loop parametrizado por catalogo**
    - Orquestrador `run_year(year)` usando **mesma sessão** (sem relogar)
    - `run_all_years(start=YYYY, count=15)` com:
      - Rate limit configurável (QPS)
      - Retry/backoff (falhas transitórias)
      - Cache local opcional (evitar requisições idênticas)
    - Relatório por ano (contagens de cursos/ofertas/arestas)
- 

### 5) Parsing e Normalização (Camada de Domínio)

- **Parsing atual (HTML)** (`parsers/arvore_parsers.py`)
    - Extração de `<select>` e `<option>` com **BeautifulSoup**
    - Tratamento de valores e rótulos (limpeza básica)
  - **Normalização Pydantic (pendente)**
    - `Course{ code, name, credits, type }`
      - `type` ∈ {obrigatoria,eletiva,livre} (normalizado)
      - `credits > 0`
    - `Offer{ course_code, term, class_id, schedule:[ScheduleSlot], teacher }`
      - `ScheduleSlot{ day ∈ {seg,...,sab}, start,end,room }`
      - Regras de interseção/confiança para conflitos de horário
    - `CurriculumNode{ course_code, category, recommended_semester? }`
      - `Edge{ from, to, type ∈ {prereq,coreq} }`
    - **Normalizers:** HTML/JSON → modelos tipados
      - `Coercion` de tipos (int/str/enum) e `defaults` consistentes
      - Funções puras com testes
- 

### 6) Persistência, Snapshots e Consolidação

- **RAW** (provas/referências)
  - Gravar respostas *in natura* (HTML/JSON) em `data/raw/{year}/...`
  - Nomear por endpoint/parâmetros (reprodutibilidade)
- **CLEAN** (dados normalizados)
  - `data/clean/{year}/{courses,offers,curriculum}.json`
  - `data/clean/all/{courses,offers,curriculum}.json` (consolidados)
  - **Consolidadores**
    - `consolidate_year(year)` (dedup por chave natural)
    - `consolidate_all()` (merge multi-ano, *conflict resolution*)

- **Hashes e integridade**
    - Calcular **sha256** por arquivo limpo
    - Gravar *manifest* com contagens e *hashes*
- 

## 7) Qualidade de Dados (DQ) e Diffs

- **Validações de regra de negócio**
    - Créditos positivos ( $\geq 1$ )
    - Horários válidos (intervalos e sobreposições)
    - *Edges* válidos (sem *loops* impossíveis; *from/to* existentes)
  - **Validações de preenchimento**
    - Campos obrigatórios não-nulos
    - Valores em domínios permitidos (enums)
  - **Diffs entre execuções**
    - Comparar **clean/{year}** com run anterior
      - Disciplinas adicionadas/removidas/alteradas
      - Ofertas com horário/docente alterado
    - Gerar **reports/diff-YYYYMMDD.md**
- 

## 8) Export para o Backend (Seeds/Fixtures)

- **DB simples** (**db/simple\_db.py**, **db/simple\_schema.sql**)
    - Ferramenta **tools/build\_simple\_db.py**
  - **Exportadores**
    - CSV/JSON em formato esperado pelo backend
    - Script **seed\_from\_json.py** (ou endpoint no backend para ingestão)
    - Verificação de chaves estrangeiras e ordinalidade
- 

## 9) CLI de Operação (unificada)

- **CLI base** (**cli.py** presente)
  - **Comandos finais**
    - **healthcheck** (login + chamada simples autenticada)
    - **run\_year --year YYYY**
    - **run\_all\_years --start YYYY --count 15**
    - **validate** (DQ + integridade)
    - **consolidate\_all**
    - **export --format csv|json**
- 

## 10) Observabilidade, Resiliência e Ética de Coleta

- **Logs básicos** (**utils/logging\_helpers.py**)
- **Melhorias de observabilidade**
  - Log estruturado (nível, ctx, request\_id)
  - Sumário por execução (contagens/tempos/erros)

- *Tracing* por ano/endpoint
  - **Resiliência**
    - *Retry* com *backoff exponencial*
    - *Circuit breaker* para timeouts seguidos
    - Ajuste de *polite\_sleep* por endpoint
  - **Ética e conformidade**
    - *Rate limit* configurável
    - Respeito a *robots/ToS* quando aplicável
    - Execuções fora do horário de pico (se necessário)
- 

## 11) Migração para Text-Scraper / Endpoints JSON (DETALHADO)

- **Descoberta de endpoints (DevTools → Network)**
    - Identificar rotas de:
      - Listagem de cursos
      - Ofertas por período
      - Estrutura de árvore/pré/co-req
      - Metadados de **semestre recomendado**
    - Anotar:
      - Método (GET/POST), *headers*, *cookies*, *CSRF*
      - Parâmetros de *query* e *payload*
      - Exemplos de resposta (salvar *fixtures*)
  - **Clientes tipados (clients/gde\_api.py)**
    - `list_courses(year)`
    - `list_offers(year, term)`
    - `get_curriculum(course_id, year)`
    - `get_prereqs(course_id, year)`
    - `get_semester_map(course_id, year)`
  - **Integração com sessão existente**
    - Reaproveitar token/cookies do login atual
    - Replicar *headers* do front (UA, XHR, CSRF quando houver)
  - **Paridade com versão HTML**
    - *Parity tests*: JSON vs HTML para o mesmo **curso/ano**
    - Definir *fallback* para campos ausentes no JSON
- 

## 12) Backlog Técnico (prioridade por valor)

- **(P1)** Adicionar **semestre recomendado** na normalização (destrava E2 no app).
  - **(P1)** Implementar **loop multi-ano (15 anos)** com relatório por ano.
  - **(P1)** Migrar coleta principal para **endpoints JSON** (robustez/velocidade).
  - **(P2)** Validadores e **diffs** (qualidade e rastreabilidade).
  - **(P2)** Exportadores e **seed** plug-and-play no backend.
  - **(P3)** Observabilidade avançada (tracing, métricas, circuit breaker).
- 

## 13) Comandos Recomendados (ao final desta fase)

```
# Sanidade de login/sessão
python -m crawler_app.cli healthcheck

# Coleta por catálogo (reaproveita token)
python -m crawler_app.cli run_year --year 2025

# Coleta de ~15 anos (com rate limit e retry)
python -m crawler_app.cli run_all_years --start 2025 --count 15

# Consolidação + validações
python -m crawler_app.cli consolidate_all
python -m crawler_app.cli validate

# Export para backend
python -m crawler_app.cli export --format json
```

---

## 14) Anexos e Referências (no repositório)

- [crawler/readme.md](#) (setup de ambiente)
- [crawler/template.env](#) (variáveis)
- [crawler/src/crawler\\_app/utils/http\\_session.py](#) (sessão/login/csrf)
- [crawler/src/crawler\\_app/collectors/\\*.py](#) (coleta atual HTML)
- [crawler/src/crawler\\_app/parsers/arvore\\_parsers.py](#) (parsing HTML)
- [crawler/src/crawler\\_app/db/\\*](#) (DB simples, schema e builder)
- [crawler/src/crawler\\_app/tools/build\\_simple\\_db.py](#) (ferramenta)