**📁 PROJECT LAYOUT & GLOBAL REQUIREMENTS *for Forum Backup***

**1. Directory tree (root-level, no src/ folder)**

forum\_mirror/ # Git / workspace root

├─ config/

│ ├─ \_\_init\_\_.py

│ ├─ defaults.yaml

│ └─ settings.py

├─ core/

│ ├─ \_\_init\_\_.py

│ ├─ fetcher.py

│ ├─ throttle.py

│ ├─ state.py

│ ├─ pathutils.py

│ ├─ redirects.py

│ └─ adblock.py

├─ crawler/

│ ├─ \_\_init\_\_.py

│ ├─ discover.py

│ └─ scheduler.py

├─ downloader/

│ ├─ \_\_init\_\_.py

│ ├─ assets.py

│ └─ workers.py

├─ processor/

│ ├─ \_\_init\_\_.py

│ ├─ orchestrator.py

│ └─ rewrite/

│ ├─ \_\_init\_\_.py

│ ├─ links.py

│ └─ assets.py

├─ cli/

│ ├─ \_\_init\_\_.py

│ ├─ \_\_main\_\_.py # run via: python -m cli

│ └─ auth.py

├─ utils/

│ ├─ \_\_init\_\_.py

│ ├─ files.py

│ └─ logging.py

├─ requirements.txt

└─ README.md

**2. requirements.txt**

aiohttp

beautifulsoup4

PyYAML

python-slugify

tqdm

ijson

**3. Config package (config/)**

**3.1 config/defaults.yaml**

*Generated on first run then editable by user.*

yaml

folder\_mapping:

f: categorias

t: topicos

u: users

g: grupos

admin: admin

privmsg: privmsg

profile: profile

ignored\_prefixes: ["/admin", "/modcp", "/profile"]

blacklist\_params: ["vote","mode","friend","foe","profil\_tabs"]

max\_asset\_kb: null # null = unlimited

slug\_max\_len: 120

ad\_hosts: [] # merged with StevenBlack list

tracker\_patterns: []

ad\_sources:

- url: https://raw.githubusercontent.com/StevenBlack/hosts/master/hosts

cache\_days: 7

**3.2 config/settings.py**

| **Function / Const** | **Purpose** |
| --- | --- |
| `init(backup\_root: Path, user\_yaml: Path | None)` |
| Globals produced | FOLDER\_MAPPING, IGNORED\_PREFIXES, BLACKLIST\_PARAMS, AD\_HOSTS, TRACKER\_PATTERNS, AD\_SOURCES, MAX\_ASSET\_KB, SLUG\_MAX\_LEN, BACKUP\_ROOT |

**Imports**: Path (pathlib), yaml, os.

**4. Core package (core/)**

Below: **public functions / classes** each file must expose, plus the key imports.

**4.1 core/fetcher.py**

*Reusable aiohttp session with adaptive throttle.*

| **Class** | **Methods** | **Notes** |
| --- | --- | --- |
| Fetcher(cfg, throttle, cookies) | `fetch\_text(url, allow\_redirects=True) -> (status:int, text:str | None, final\_url:str)<br>fetch\_bytes(url) -> (status:int, bytes |

**Imports**: aiohttp, asyncio, ClientTimeout, TCPConnector.

**4.2 core/throttle.py**

*Adaptive delay & worker scaling.*

| **Class** | **Purpose** |
| --- | --- |
| ThrottleController(cfg) | Tracks RTT, 2xx streaks, 429/5xx; exposes delay, workers, and API: before\_request() (async sleep) after\_response(status) (update logic). |

**4.3 core/state.py**

*Two compact JSON databases + queue semantics.*

| **Part** | **Public API** | **Key points** |
| --- | --- | --- |
| **crawl\_state** | add\_url(path, rel\_path) get\_next(phase:str) (phase = "discover" or "download") mark\_discovered / mark\_downloaded / mark\_redirect\_source update\_after\_fetch(success, err="") pending\_count() save() (stream-write via ijson) load() | Record format = ["rel\_path", redirect\_flag, status, retries, error]. |
| **assets\_cache** | get\_asset(url) add\_asset(url, rel\_path) | Stored as "url": "assets/…/file.ext". |

Uses asyncio.Lock, temp-file + os.replace() for atomic writes.

**4.4 core/pathutils.py**

*Map URL → desktop filepath.*

| **Function** | **Description** |
| --- | --- |
| url\_to\_local\_path(path:str) -> str | *Lowercase* → slugify() tokens (max SLUG\_MAX\_LEN) → choose folder via FOLDER\_MAPPING (fallback *misc*) → detect collisions (-dupN). |

**4.5 core/redirects.py**

*Redirect graph manager, persisted to redirects.json.*

| **Class / Func** | **Purpose** |
| --- | --- |
| RedirectMap | add(src,dst) (avoid self-loops) • resolve(path) (visited-set + depth≤16) • stats helpers. |
| redirects | Singleton instance used project-wide. |

**4.6 core/adblock.py**

*Download StevenBlack hosts & expose quick blocker.*

| **Function** | **Purpose** |
| --- | --- |
| update\_hosts(backup\_root) (async) | Download each source in AD\_SOURCES if cache expired; merge hostnames into config.settings.AD\_HOSTS. |
| is\_blocked\_host(host:str) -> bool | Membership test. |

Requires aiohttp, re, time.

**5. Utils package (baseline)**

**5.1 utils/files.py *(already outlined)***

*safe\_file\_write, safe\_file\_read — atomic I/O.*

**5.2 utils/logging.py**

setup() → basic coloured / timestamped logging (optional).

### 📂 6 · Package crawler/

#### 6.1 crawler/discover.py

| **Element** | **Responsibility** | **Must import** |
| --- | --- | --- |
| IGNORED\_PREFIXES, BLACKLIST\_PARAMS, BASE\_URL, BASE\_DOMAIN | filter rules loaded from config.settings | from config.settings import … |
| \_strip\_fragment(url) | remove #anchor once and for all | str.split("#",1) |
| \_is\_valid\_link(href) | **internal** filter: same domain, not ignored prefix, allowed query params, no mailto: etc. | urllib.parse.urlparse, urljoin |
| \_path\_plus\_query(url) | return canonical key /path?query | … |
| handle\_redirect(worker\_id, src\_url, dst\_url, state) | log to redirects, mark source as redirect, enqueue dest | core.redirects.redirects, core.state.State |
| class LinkDiscoverer | one async worker |  |
| \_\_init\_\_(cfg,state,fetcher,worker\_id) | store refs + create AssetManager for prefetch (optional) |  |
| run() | pop **l** URLs via state.get\_next("discover"); terminate after 15×0.5 s empty loops |  |
| \_process(path) | 1) fetch HTML (allow\_redirects=False) 2) deal with 301/302 via handle\_redirect 3) save raw HTML (utils.files.safe\_file\_write) 4) call \_parse\_links() then state.mark\_discovered(path) | url\_to\_local\_path, safe\_file\_write |
| \_parse\_links(html) | use BeautifulSoup to find <a href>; for each valid link → state.add\_url(key, rel\_path); **returns count** | bs4.BeautifulSoup |

**Concurrency note**: you may spawn AssetManager.fetch() as fire-and-forget tasks to pre-download images/CSS in discovery. They use the same asset cache so there is no duplication.

#### 6.2 crawler/scheduler.py

| **API** | **Behaviour** |
| --- | --- |
| run\_discovery\_phase(cfg, state, fetcher) | • start **1** LinkDiscoverer. • poll state.pending\_count() every second; when pending ≥ 20 (or first worker finishes) spawn up to cfg.workers discoverers. • await gather(\*tasks); returns when all workers exit and **no 'l' left**. |
| run\_download\_phase(cfg, state, fetcher) | • create DownloadWorker instances = cfg.workers. • await gather(). • returns when every worker exhausts 'd' queue. |

Imports required: asyncio, LinkDiscoverer, DownloadWorker.

### 📂 7 · Package downloader/

#### 7.1 downloader/assets.py → AssetManager

| **Member** | **Purpose** | **Implementation hints** |
| --- | --- | --- |
| \_\_init\_\_(fetcher, state) | store references; mkdir folders: BACKUP\_ROOT/assets/imagens/internal BACKUP\_ROOT/assets/files/internal BACKUP\_ROOT/external\_files | pathlib.Path.mkdir(parents=True, exist\_ok=True) |
| `fetch(url:str, kind\_hint:str)->str | None` | main async method: 1. **Block check**: if host in AD\_HOSTS → return None. 2. **Cache hit**: state.get\_asset(url). 3. **Download** via fetcher.fetch\_bytes. 4. **Size guard**: if MAX\_ASSET\_KB and len > limit → return None. 5. **Extension** via \_choose\_ext(). 6. Write file atomically with safe\_file\_write (mode "wb"). 7. Add to asset cache; return relative path. |
| \_choose\_ext(url, kind\_hint) | fallback logic when URL has no suffix; e.g. "fonts" → .woff | also consult Content-Type header if needed. |

Constants: IMAGE\_EXTS (png, jpg, …)

Imports: hashlib, mimetypes, os, pathlib.Path, urllib.parse.urlparse, config.settings, core.state, utils.files.

#### 7.2 downloader/workers.py → DownloadWorker

| **Stage** | **What it does** | **Notes** |
| --- | --- | --- |
| run() | infinite loop: path = await state.get\_next("download"); if None → break; else \_process(path) |  |
| \_process(path) | 1. Build absolute URL. 2. Fetch HTML **with redirects allowed**. 3. If final ≠ requested → handle\_redirect and return. 4. On 200: process\_html(final\_url, html, fetcher, state). 5. Save resulting HTML to url\_to\_local\_path(final\_url). 6. state.mark\_downloaded(path) and update tqdm progress (optional). | Catches exceptions → state.update\_after\_fetch(False,"download error"). |

Imports: urllib.parse.urljoin, traceback, processor.orchestrator, core.pathutils.

### 📂 8 · Package processor/

(Only interfaces here; full code in Part 3.)

| **File** | **Public function / class** |
| --- | --- |
| rewrite/links.py | rewrite\_links(soup, cur\_file:str, state) |
| rewrite/assets.py | \_rewrite\_head(), \_rewrite\_body\_assets() |
| orchestrator.py | process\_html(url, html, fetcher, state) -> str |

### 📂 9 · Package cli/

| **File** | **Functions (new)** | **Description** |
| --- | --- | --- |
| auth.py | handle\_authentication() | loads cookies → is\_logged\_in() → prompt wizard if needed. |
| \_\_main\_\_.py | See Part 3: interactive prompt + banners. |  |

### 📂 10 · Package utils/

| **File** | **Functions** |
| --- | --- |
| files.py | safe\_file\_write, safe\_file\_read (atomic) |
| logging.py | setup() (optional colourful logging) |

## How everything connects (textual walk-through)

1. **CLI** asks for URL / backup folder; calls config.settings.init().
2. Downloads StevenBlack hosts → merges into AD\_HOSTS.
3. Builds State, ThrottleController, Fetcher.
4. **Phase 1** – run\_discovery\_phase()  
   LinkDiscoverers: fetch raw HTML → save → extract links → queue them; prefetch assets.  
   Ends when no "l" left.
5. **Phase 2** – run\_download\_phase()  
   DownloadWorkers: pull "d" → fetch final HTML → processor.orchestrator.process\_html() → rewrite assets & links → save final file → mark "p".  
   Ends when no "d" left.
6. Save state, copy to crawl\_state\_final.json, exit.

**7 · Package processor∕**

**7.1 processor/rewrite/assets.py**

python

"""

Download & rewrite <head> and <body>‐level external resources.

Public coroutines

-----------------

\* \_rewrite\_head(soup, page\_url, asset\_mgr)

\* \_rewrite\_body\_assets(soup, page\_url, asset\_mgr)

Both modify the BeautifulSoup object \*\*in-place\*\* and never return HTML text.

"""

from \_\_future\_\_ import annotations

import re, asyncio

from urllib.parse import urljoin, urlparse

from bs4 import BeautifulSoup

from downloader.assets import AssetManager

from core.adblock import is\_blocked\_host

from config.settings import BASE\_URL, BASE\_DOMAIN

# ───────────────────────── helpers ──────────────────────────

CSS\_URL\_RE = re.compile(r"url\\(['\\\"]?(.\*?)['\\\"]?\\)")

IMG\_TAGS = ("img", "input") # tags that normally carry 'src'

async def \_download\_and\_replace(tag, attr, mgr: AssetManager):

url = urljoin(BASE\_URL, tag[attr])

rep = await mgr.fetch(url, "images")

if rep:

tag[attr] = rep

# ───────────────────────── <head> ───────────────────────────

async def \_rewrite\_head(soup: BeautifulSoup, page\_url: str, mgr: AssetManager):

head = soup.head

if not head:

return

# stylesheet / preload / icon

for link in head.find\_all("link", href=True):

rels = {r.lower() for r in link.get("rel", [])}

href = urljoin(BASE\_URL, link["href"])

host = urlparse(href).netloc.lower()

if is\_blocked\_host(host):

link.decompose()

continue

if "stylesheet" in rels:

repl = await mgr.fetch(href, "css")

if repl:

link["href"] = repl

elif rels & {"preload", "prefetch"}:

repl = await mgr.fetch(href, "misc")

if repl:

link["href"] = repl

elif "icon" in rels:

repl = await mgr.fetch(href, "images")

if repl:

link["href"] = repl

# external <script src="…">

for script in head.find\_all("script", src=True):

src\_abs = urljoin(BASE\_URL, script["src"])

if is\_blocked\_host(urlparse(src\_abs).netloc.lower()):

script.decompose()

continue

repl = await mgr.fetch(src\_abs, "js")

if repl:

script["src"] = repl

# inline <style> with font URLs

for style in head.find\_all("style"):

if not style.string:

continue

css = style.string

for orig in CSS\_URL\_RE.findall(css):

abs\_u = urljoin(page\_url, orig)

repl = await mgr.fetch(abs\_u, "fonts")

if repl:

css = css.replace(orig, repl)

style.string.replace\_with(css)

# ────────────────────── <body> & inline ─────────────────────

async def \_rewrite\_body\_assets(soup: BeautifulSoup, page\_url: str, mgr: AssetManager):

# <img>, <input type="image">

await asyncio.gather(\*[

\_download\_and\_replace(tag, "src", mgr)

for tag in soup.find\_all(IMG\_TAGS, src=True)

])

# <script src> inside body

await asyncio.gather(\*[

\_download\_and\_replace(script, "src", mgr)

for script in soup.find\_all("script", src=True)

])

# <source srcset="a.jpg 1x, b.jpg 2x">

for src in soup.find\_all("source", srcset=True):

newset = []

for part in src["srcset"].split(","):

url = part.split()[0]

repl = await mgr.fetch(urljoin(BASE\_URL, url), "images")

if repl:

newset.append(part.replace(url, repl))

if newset:

src["srcset"] = ",".join(newset)

# inline style="background:url(...)"

for tag in soup.find\_all(style=True):

style = tag["style"]

for orig in CSS\_URL\_RE.findall(style):

repl = await mgr.fetch(urljoin(page\_url, orig), "images")

if repl:

style = style.replace(orig, repl)

tag["style"] = style

**7.2 processor/rewrite/links.py**

python

"""

Rewrite every internal <a href> so it points to the correct local file.

"""

import os

from urllib.parse import urljoin, urlparse

from bs4 import BeautifulSoup

from config.settings import BASE\_URL, BASE\_DOMAIN

from core.pathutils import url\_to\_local\_path

from core.redirects import redirects

from core.state import State, REL # index 0 in the compact record

def rewrite\_links(soup: BeautifulSoup, cur\_file: str, state: State):

cur\_dir = os.path.dirname(cur\_file)

for a in soup.find\_all("a", href=True):

href = a["href"]

if href.startswith(("mailto:", "javascript:", "#")):

continue

abs\_url = urljoin(BASE\_URL, href)

base, \*frag = abs\_url.split("#", 1)

netloc = urlparse(base).netloc

if netloc and netloc != BASE\_DOMAIN:

# external link: leave unchanged

continue

key = urlparse(base).path + (f"?{urlparse(base).query}" if urlparse(base).query else "")

key = redirects.resolve(key)

rec = state.urls.get(key)

if not rec:

continue

target\_file = rec[REL]

rel\_link = os.path.relpath(target\_file, cur\_dir).replace(os.sep, "/")

if rel\_link.endswith("/index.html"):

rel\_link = rel\_link[:-len("index.html")] or "./"

if frag:

rel\_link += "#" + frag[0]

a["href"] = rel\_link

**7.3 processor/orchestrator.py**

python

"""

High-level coordinator that converts raw HTML into fully localised HTML.

Exported coroutine: process\_html(url, html, fetcher, state) -> str

"""

import asyncio, pathlib

from bs4 import BeautifulSoup

from downloader.assets import AssetManager

from processor.rewrite.assets import \_rewrite\_head, \_rewrite\_body\_assets

from processor.rewrite.links import rewrite\_links

from core.pathutils import url\_to\_local\_path

from core.state import State

async def process\_html(page\_url: str, html: str, fetcher, state: State) -> str:

"""

• Parse HTML with BeautifulSoup

• Localise head assets, body assets

• Rewrite internal anchors

• (Future) Pass through optimiser hook

"""

soup = BeautifulSoup(html, "html.parser")

mgr = AssetManager(fetcher, state)

save\_path = pathlib.Path(url\_to\_local\_path(page\_url))

await \_rewrite\_head(soup, page\_url, mgr)

await \_rewrite\_body\_assets(soup, page\_url, mgr)

rewrite\_links(soup, str(save\_path), state)

return await maybe\_optimize(soup)

# ─────────────────────── future hook ───────────────────────

async def maybe\_optimize(soup: BeautifulSoup) -> str:

"""

Stub for a future HTML minifier / template injector.

For now, returns the str(soup) unchanged.

"""

return str(soup)

**8 · Package cli∕**

**8.1 cli/auth.py**

python

"""

Cookie-based authentication flow.

"""

import json, aiohttp

from pathlib import Path

from urllib.parse import urljoin

from bs4 import BeautifulSoup

from utils.files import safe\_file\_write

from config.settings import BACKUP\_ROOT

COOKIES\_FILE = BACKUP\_ROOT / "cookies.json" # overwritten later at runtime

async def load\_cookies() -> dict:

if COOKIES\_FILE.exists():

return json.loads(COOKIES\_FILE.read\_text("utf-8"))

return {}

async def is\_logged\_in(session: aiohttp.ClientSession, forum\_url: str) -> bool:

r = await session.get(forum\_url)

html = await r.text()

soup = BeautifulSoup(html, "html.parser")

return any(urljoin(forum\_url, a["href"]).split("/",3)[3].startswith("profile")

for a in soup.find\_all("a", href=True))

async def handle\_authentication() -> tuple[dict,bool]:

"""

Returns (cookies, logged\_in\_flag).

Prompts user to re-enter cookies if not authenticated.

"""

import aiohttp, asyncio

cookies = await load\_cookies()

async with aiohttp.ClientSession(cookies=cookies) as s:

ok = await is\_logged\_in(s, BACKUP\_ROOT.meta["forum\_url"])

if ok:

print("✅ [Cookies] Logged in")

return cookies, True

ans = input("Reconfigure cookies? (y/N): ").lower()

if ans != "y":

return cookies, False

# wizard

domain = BACKUP\_ROOT.meta["domain"]

ck1 = f"fa\_{domain.replace('.','\_')}\_data"

ck2 = f"fa\_{domain.replace('.','\_')}\_sid"

print("Paste cookie values:")

cookies = {ck1: input(f"{ck1}: ").strip(),

ck2: input(f"{ck2}: ").strip()}

await safe\_file\_write(COOKIES\_FILE, json.dumps(cookies, indent=2), "w")

return cookies, True

*(CLI banner+prompt code was delivered in previous messages and should remain unchanged.)*

**9 · Package utils∕**

**9.1 utils/logging.py**

python

import logging, sys

from datetime import datetime

def setup(level="INFO"):

logging.basicConfig(

level=getattr(logging, level),

format="%(asctime)s | %(levelname)7s | %(message)s",

datefmt="%H:%M:%S",

handlers=[logging.StreamHandler(sys.stdout)]

)

Call utils.logging.setup("INFO") inside cli/\_\_main\_\_.py if you want colour-free logging.

**🔖 10 · README.md (starter)**

# Forum-Mirror

Offline spider that clones phpBB / Forumotion boards with assets, link rewriting,

ad-blocking and resume support.

## Quick start

```bash

git clone https://github.com/you/forum\_mirror.git

cd forum\_mirror

python -m pip install -r requirements.txt

python forum\_mirror.py # interactive wizard

By default the mirror lands in ~/Desktop/<forum-slug>/.

**Features**

* **Two-phase crawl** – discovery + download with adaptive throttling.
* **Asset deduplication** – every image/font/CSS saved once.
* **Ad & tracker blocking** – integrates StevenBlack hosts list.
* **Resume safely** – state is saved every N pages.
* **Per-forum YAML** – customise folder mapping & filters.

# 11 · Testing package tests/

tests/

├─ \_\_init\_\_.py

├─ test\_pathutils.py

├─ test\_redirects.py

└─ conftest.py

### 11.1 tests/conftest.py

python

import pytest, tempfile, shutil, os

from pathlib import Path

@pytest.fixture

def tmp\_root(tmp\_path):

"""

Create an isolated BACKUP\_ROOT folder for each test.

Auto-cleaned by pytest.

"""

return tmp\_path

### 11.2 tests/test\_pathutils.py

python

from core.pathutils import url\_to\_local\_path

def test\_basic\_mapping(tmp\_root, monkeypatch):

monkeypatch.setattr("config.settings.BACKUP\_ROOT", tmp\_root)

out = url\_to\_local\_path("/f17-something")

assert out.endswith("categorias/f17-something.html")

### 11.3 tests/test\_redirects.py

python

from core.redirects import RedirectMap

def test\_chain\_resolution():

rm = RedirectMap(":memory:") # in-RAM

rm.map.update({"/a":"/b", "/b":"/c"})

assert rm.resolve("/a") == "/c"

Run with pytest -q.

# 12 · Lint & Format

### 12.1 pyproject.toml

toml

[tool.black]

line-length = 88

target-version = ["py310"]

[tool.ruff]

line-length = 88

select = ["E", "F", "I"] # errors, flakes, isort

ignore = ["E501"] # already handled by black

### 12.2 .editorconfig

ini

[\*]

charset = utf-8

indent\_style = space

indent\_size = 4

end\_of\_line = lf

trim\_trailing\_whitespace = true

insert\_final\_newline = true

# 14 · Future HTML optimizer module

processor/

└─ optimize/

├─ \_\_init\_\_.py

└─ htmlmin.py

### 14.1 processor/optimize/htmlmin.py

python

"""

Pluggable HTML minifier – currently no-op.

Swap `run()` with any real compressor (htmlmin, bs4 tweaks, etc.)

and orchestrator.maybe\_optimize() will import & use it automatically

if the module exists.

"""

def run(html: str) -> str:

# placeholder – return unchanged

return html

Modify orchestrator hook:

python

async def maybe\_optimize(soup):

try:

from processor.optimize.htmlmin import run

return run(str(soup))

except ModuleNotFoundError:

return str(soup)

CI pipeline runs the same checks on every push.