🧠 STAGE-1 SCRAPER — ONE PAGE CANONICAL TEMPLATE

> **Purpose:** Static website → clean structured data → JSON **Survives failures. No silent bugs. No RAM stupidity.**

🔒 SYSTEM CONTRACT (READ THIS FIRST)

```
INPUT   : URL
OUTPUT  : list[dict] → JSON
FAILURE : handled, logged, never crashes silently
```

🧱 FILE: `scraper.py`

```python
import requests
import json
import time
import logging
from bs4 import BeautifulSoup
from urllib.parse import urljoin
from pathlib import Path
from typing import Optional

# -------------------- LOGGING (NON-NEGOTIABLE) ------------------
--
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s | %(levelname)s | %(message)s"
)
logger = logging.getLogger(__name__)


# -------------------- SCRAPER SYSTEM --------------------
class Stage1Scraper:
    """
    Stage-1 Static Website Scraper
    Sync • Production-Safe • Reusable
    """

    # -------------------- INIT = STATE ONLY --------------------
    def __init__(self, base_url: str):
        self.base_url = base_url
        self.headers = {
            "User-Agent": "Mozilla/5.0",
            "Accept-Language": "en-US,en;q=0.9"
        }
        self.data: list[dict] = []

    # -------------------- NETWORK LAYER --------------------
    def fetch_html(self, url: str) -> Optional[str]:
        """
        Talks to internet.
        Returns HTML or None.
        Never crashes the system.
        """
        try:
            response = requests.get(url, headers=self.headers,
timeout=10)

            if response.status_code == 200:
                response.encoding = "utf-8"
                return response.text

            logger.warning(f"{response.status_code} while fetching
{url}")
            return None

        except requests.RequestException as e:
            logger.error(f"Network error: {e}")
            return None

    # -------------------- PARSE LAYER --------------------
    def parse_html(self, html: str) -> BeautifulSoup:
        """
        HTML string → DOM tree
        No logic. No validation.
        """
        return BeautifulSoup(html, "lxml")

    # -------------------- VALIDATION GATE --------------------
    def is_valid_record(self, item: dict) -> bool:
        """
        Bad data never enters the system.
        """
        return all(item.values())

    # -------------------- EXTRACTION PIPELINE -------------------
-
    def extract_records(self, soup: BeautifulSoup) -> None:
        """
        DOM → structured records
        extract → validate → append
        """
        records = soup.find_all("article")  # CHANGE PER SITE

        for record in records:
            # ---- SELECTORS (MEANING-BASED, NOT CLASS-BASED) ----
            title = record.find("h3")
            price = record.find("p", class_="price_color")
            availability = record.find("p", class_="instock")

            item = {
                "title": title.get_text(strip=True) if title else None,
                "price": price.get_text(strip=True) if price else None,
                "availability": availability.get_text(strip=True) if availability else None,
            }

            if not self.is_valid_record(item):
                continue

            self.data.append(item)

    # -------------------- PAGINATION CONTROLLER --------------------
    def extract_all_pages(self) -> None:
        """
        Controls flow.
        Stops safely when pagination breaks.
        """
        current_url = self.base_url

        while True:
            html = self.fetch_html(current_url)
            if html is None:
                logger.info("Stopping: no HTML received")
                break

            soup = self.parse_html(html)
            self.extract_records(soup)

            logger.info(f"Scraped: {current_url}")
            time.sleep(1)  # Respect server

            next_btn = soup.find("li", class_="next")
            if not next_btn:
                break
            next_link = next_btn.find("a")["href"]
            current_url = urljoin(current_url, next_link)

    # -------------------- SAVE OUTPUT --------------------
    def save_to_json(self, filename: str = "data.json") -> None:
        path = Path(__file__).parent / filename
        with open(path, "w", encoding="utf-8") as f:
            json.dump(self.data, f, indent=4, ensure_ascii=False)


# -------------------- ENTRY POINT --------------------
if __name__ == "__main__":
    BASE_URL = "https://example.com"

    scraper = Stage1Scraper(BASE_URL)
    scraper.extract_all_pages()
    scraper.save_to_json()

    logger.info(f"Total records scraped: {len(scraper.data)}")
```

🧠 HOW YOU USE THIS (IMPORTANT)

Every new static website:

You **ONLY change**:

1. `BASE_URL`
2. Selector logic inside `extract_records()`

You **NEVER touch**:

- fetch logic
- pagination loop structure
- validation gate
- logging
- save logic

That's how systems stay stable.

🧪 DEBUG FLOW (WHEN STUCK)

Read this **top to bottom**, no skipping:

```
1. Is fetch_html returning HTML?
2. Is soup being created?
3. Are records found?
4. Are selectors returning None?
5. Is validation skipping everything?
6. Are logs telling the truth?
```

No guessing. No panic. Just trace.

🏁 FINAL MENTAL MODEL (LOCK THIS)

> **Scraping is not extraction. Scraping is controlled failure handling.**

This template:

- keeps you calm
- keeps code readable
- scales to async later
- makes you a **builder**, not a copier 🚀
