🕸️ PHASE 1.3 — WEB SCRAPING (THE HUNT)

From Page Grabber → Data Engineer Mindset

> **Core Goal:**
> You don't "scrape pages".
> You **reverse-engineer data pipelines**.

🔴 STAGE 0 — THE ANATOMY (NON-NEGOTIABLE)

"The X-Ray"

🎯 Goal

Reverse-engineer a website **without writing a single line of Python**.

If you can't do this, **every scraper you write later will be fragile**.

🧠 What You're Really Learning

- Where data **actually** comes from
- Why HTML you see ≠ data you need
- How sites lie to scrapers

🔧 Skills to Learn (Visual + Thinking)

- HTML vs DOM vs JavaScript
- Server-side vs Client-side rendering
- Chrome DevTools:

    - **Elements tab** → DOM traversal
    - **Network tab** → Doc vs XHR vs Fetch
    - Finding the *real* data request
- Status codes **from a scraper's POV**

🎯 Mission — *The X-Ray*

Pick **one dynamic site** and answer:

1. Where is the data *actually* coming from?
2. Which request returns JSON?
3. What headers/cookies does it need?

🔥 Killer Move

- **Copy as cURL**
- Convert that cURL → Python (mentally or via tool)
- Realize: *"Oh… I don't even need a browser anymore."*

🧠 **Skill unlocked:**

> *Before coding, I know exactly what to hit.*

🟢 STAGE 1 — THE SNIPER (STATIC EXTRACTION)

"The Quote Hunter"

🎯 Goal

Build a **robust, production-quality static scraper**, not a toy.

🧰 Tools

- `requests`
- `BeautifulSoup4`

🔧 Skills to Learn

- HTTP GET vs POST (real use cases)
- Status codes:
  - 200 → OK
  - 403 → blocked
  - 429 → rate-limited
- Headers:
  - User-Agent
  - Accept / Accept-Language
- HTML DOM structure:
  - tags, attributes, nesting
- CSS Selectors:
  - class vs id
  - child (`>`) vs descendant ( )
  - selector fragility

🎯 Mission — *The Quote Hunter*

Scrape `quotes.toscrape.com` , but with:

- **OOP design** (Scraper class)
- **Type hints**
- **Pydantic models** for data
- Proper error handling

🔥 Killer Move

Unbreakable Selectors

- Avoid brittle class names
- Use semantic structure
- Predict what breaks when UI changes

🧠 **Skill unlocked:**

> *My scraper survives minor site changes.*

🟡 STAGE 2 — THE MACHINE GUN (ASYNC SCALE + EVASION)

“The Async News Aggregator”

🎯 Goal

Scrape **50–100 pages concurrently** without getting banned.

🧰 Tools

- `aiohttp`
- `asyncio`
- `ClientSession`

🔧 Skills to Learn

- Why `requests` fails at scale
- Async scraping architecture
- `ClientSession` lifecycle
- Connection pooling
- Timeouts & retries
- **Rate limiting with Semaphores**

🎯 Mission — *Async News Aggregator*

- Fetch many pages concurrently
- Respect limits
- Handle partial failures

🔥 Killer Moves

- **User-Agent rotation**
- Detecting **soft blocks**
  - 200 OK but empty / fake HTML
- Retry strategy (basic)

🧠 **Skill unlocked:**

> *Fast, polite, resilient scraping.*

🔵 STAGE 3 — ANTI-SCRAPING AWARENESS (HIDDEN KILLER)

(This is why most scrapers fail in real life)

🎯 Goal

Understand **how sites detect and stop you**.

🧠 What Sites Do

- IP-based blocking
- Rate-based bans
- Header fingerprinting
- Cookie/session checks
- JS challenges & honeypots

🔧 Skills to Learn

- Why UA rotation works (and when it doesn't)
- Session vs stateless scraping
- Cookie handling basics
- CAPTCHA *detection* (not solving)

🧠 **Skill unlocked:**

> *When a scraper dies, I know why.*

🟣 STAGE 4 — THE GHOST (DYNAMIC & HEAVY)

"The E-Commerce Monitor"

🎯 Goal

Scrape **JavaScript-heavy sites** that defeat normal scrapers.

🧰 Tools

- **Playwright (Async API)** ✅ (preferred)
- Selenium (secondary)

🔧 Skills to Learn

- Headless vs headed browsers
- Explicit waits (CRITICAL)
- Waiting for network idle
- DOM manipulation (click, scroll)
- Infinite scroll logic
- Shadow DOM basics

🎯 Mission — *E-Commerce Monitor*

Scrape prices from a JS-rendered site
(similar to Amazon / Myntra behavior).

🔥 Killer Move (ADVANCED)

Hybrid Scraping

1. Use Playwright to:
   - load page
   - get cookies / tokens
2. Pass them to `aiohttp`
3. Scrape at scale **without browser**

🧠 **Skill unlocked:**

> *Browser is a tool, not a crutch.*

🟤 STAGE 5 — THE FACTORY (SYSTEM ENGINEERING)

“The Harvester”

🎯 Goal

Build a **real data pipeline**, not a script.

🧰 Tools & Concepts

- Folder architecture
- Config-driven scraping
- Logging (`logging`)
- Retries (`tenacity`)

- Resume interrupted runs
- Data validation:
  - Pandas
  - Pydantic
- CSV / JSON export

🎯 Mission — *The Harvester*

A scraper that:

- can stop & resume
- logs failures
- retries safely
- validates data
- outputs clean datasets

🧠 **Skill unlocked:**

> *I don’t scrape pages. I run data systems.*

🧠 HOW YOU SHOULD LEARN (NON-NEGOTIABLE RULES)

❌ Don’t

- binge YouTube
- copy full scrapers
- memorize selectors
- start with code

✅ Do

- Inspect in Chrome first
- Predict failure cases
- Design flow in comments
- Code last

- Break things intentionally
- Debug like an engineer

🧭 FINAL LEARNING ORDER (LOCKED)

1️⃣ Stage 0 — Anatomy (DevTools mastery)
2️⃣ Stage 1 — Static Scraping
3️⃣ Stage 2 — Async Scraping
4️⃣ Stage 3 — Anti-Scraping Awareness
5️⃣ Stage 4 — Dynamic Scraping
6️⃣ Stage 5 — Scraping as a System

🔒 FINAL ARCHITECT STATEMENT

> **A beginner scrapes pages.**
> **A freelancer scrapes data.**
> **An engineer designs pipelines.**

This path takes you to **engineer territory**.