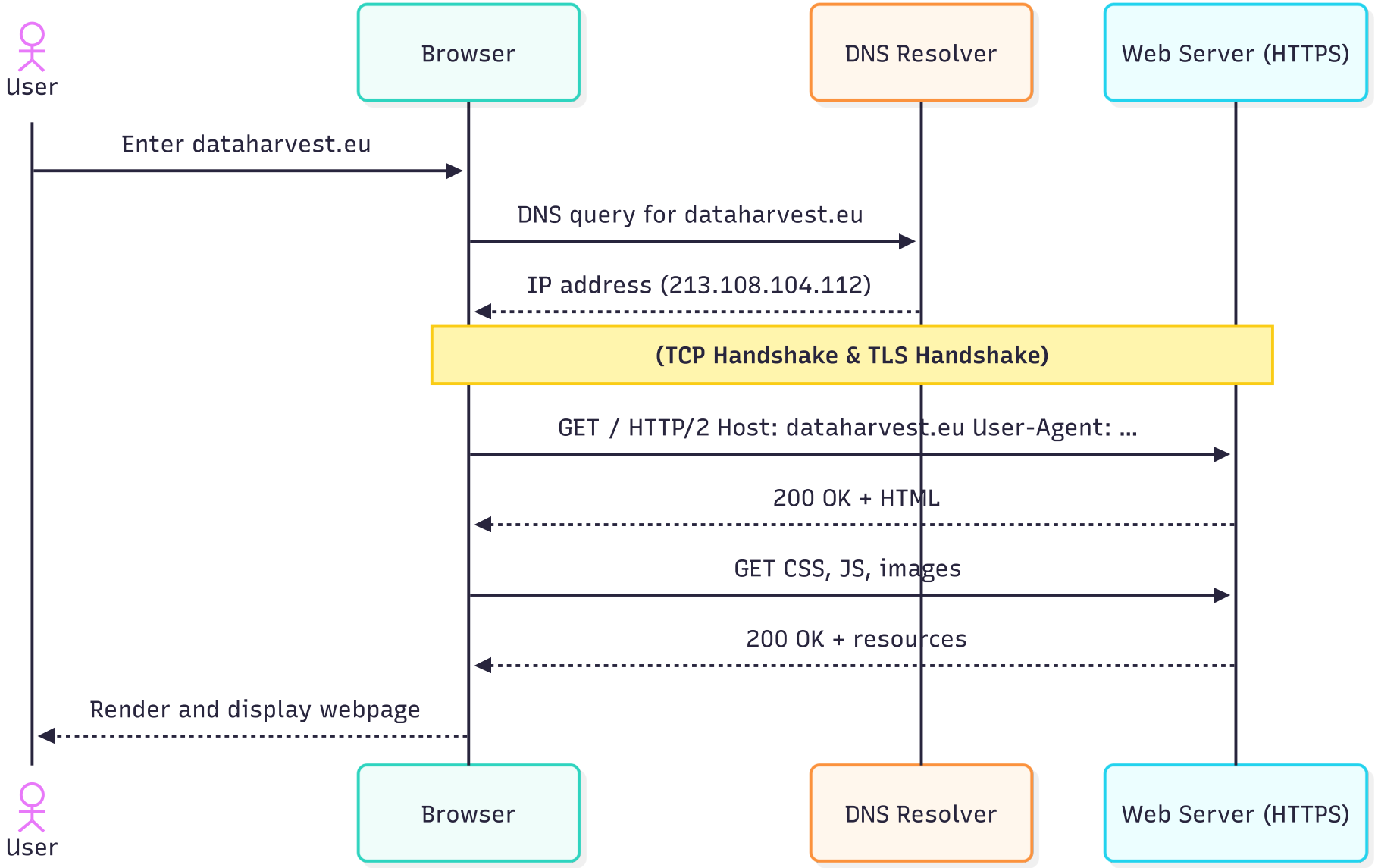
Choosing a web scraping strategy

by **Stephanie Jauss** (SWR Data Lab)
and **Verena Steinacher** (pub.tech)

Agenda

1. How the web works
2. How to prepare for scraping
3. How to decide on a strategy
4. Examples from real-life projects
5. How to run a scraper in production

1. How the web works


User
Browser
DNS Resolver
Web Server (HTTPS)
Enter dataharvest.eu
DNS query for dataharvest.eu
IP address (213.108.104.112)
(TCP Handshake & TLS Handshake)
GET / HTTP/2 Host: dataharvest.eu User-Agent: ...
200 OK + HTML
GET CSS, JS, images
200 OK + resources
Render and display webpage
User
Browser
DNS Resolver
Web Server (HTTPS)

HTTP Request

```
GET / HTTP/1.1
host: www.bahn.de
accept: text/html,application/xhtml+xml,application/xml;
        q=0.9,image/avif,image/webp,image/apng,*/*;
        q=0.8,application/signed-exchange;v=b3;q=0.7
accept-language: de-DE,de;q=0.9,en-US;q=0.8,en;q=0.7
user-agent: Mozilla/5.0 (Macintosh;
            Intel Mac OS X 10_15_7) AppleWebKit/537.36
            (KHTML, like Gecko) Chrome/137.0.0.0 Safari/537.36
cookie: ...
```

HTTP Request

1. **Request Line**: `GET /index.html HTTP/1.1`
   → HTTP method & version, URL
2. **Headers**: User-Agent, Accept, ...
   → meta information about request (check with dev tools or httpbin.org/headers)
3. **Body**: data (e.g. form data)

REST

- = Representational State Transfer
- Architecture style for web APIs
- Important HTTP methods (but not all developers stick to them):

Method	URL	Action
GET	`/users` / `/users/123`	Read all users / user with ID 123
POST	`/users`	Create new user
PUT	`/users/123`	Replace user 123
DELETE	`/users/123`	Delete user 123

HTTP Response

1. **Status Line**: `HTTP/1.1 200 OK`
   → HTTP version, status code and text
2. **Headers**: Content-Type, Content-Length etc. e.g. `Content-Type: text/html; charset=UTF-8`
   → meta information about the response
3. **Body**: the content: HTML, JSON, image etc.

Status Codes

- **1xx**: Informative (e.g. 100 Continue)
- **2xx**: Success (e.g. 200 OK, 201 Created)
- **3xx**: Redirect (e.g. 302 Found)
- **4xx**: Client error (e.g. 404 Not Found, 403 Forbidden, 429 Too Many Requests)
- **5xx**: Server error (e.g. 500 Internal Server Error, 503 Service Unavailable)

2. How to prepare for scraping

Tool

Use a browser and get familiar with the dev tools

- Different styles across browsers but typical structure
- Might need to be activated once
- **Network tab**: inspect HTTP requests and responses
  - Copy request as cURL
  - Disable cache, preserve log
- **Elements**: DOM inspector
- **Console**: Test JavaScript
- **Sources**: Inspect loaded JavaScript files

Tool

For complex cases, consider using an API client

- Alternative to using a CLI like curl or Python directly
- Helpful to prepare scraping
- Tools recommendations
  - Bruno (Open Source)
  - Insomnia
  - Postman

Things to look for

- Which requests do I need?
- How do they work?

Query Parameters

- Start after a `?`, multiple params are separated with `&`
- Simplified example: `https://www.bahn.de/buchung/fahrplan/suche?so=Hamburg%20Hbf&zo=Berlin%20Hbf` (`%20` = encoded space)
- Show headers in a nice way: dev tools or urlprettyprint.com

Pagination

- = Content split across multiple pages
- Different types
  - Page-based: `/products?page=2`
  - Offset pagination: `/api/items?offset=20&limit=10`
  - Cursor pagination: `/api/feed?after=abc123` → look for keywords like "next", "nextPage", "reference", "cursor"
- Lazy loading in frontend (on scroll / button click) usually corresponds to one of these versions in the requests, too

Cookies

- = Small data packets which the server stores in the browser
- Sent back to the server with subsequent requests (in header field `Cookie`)
- Used, for example, to maintain logged-in (session) or store privacy preferences
- Components: name + value: `sessionId=abc123`
  (+ optionally expiration date, domain, path, security flags)

CSRF tokens

- CSRF = Cross-Site Request Forgery
- Protection mechanism against unauthorized requests
- Token is generated during login or page load
- Must be included in every sensitive request (e.g. form submission) – often in `X-CSRF-Token` header field, sometimes in body
- Found in the HTML (`<meta>` tag / hidden `<input>` field) or in a JavaScript object

```
<head>
  <meta name="csrf-token" content="a1b2c3d4e5f6g7h8i9j0"/>
</head>
```

Demo: Prepare scraping

3. How to decide on a strategy

Increasing resistance from the target system requires **increasing sophistication by the scraper**

Questions to ask yourself

1. The basics

- Is it a one-time project or do I want to monitor changes continuously?
  → How do I want to run the scraper in production?
  → How to write my code so it's easy to maintain?
- Is there an easier way? (e.g. data dump, request the data, API, or RSS feed)
- Has someone else already done the same?
  → search GitHub repositories with `site:github.com` or ask colleagues

2. The website

- Does the browser load the **data in a structured format** (e.g. JSON)?
- If not, can I reliably extract the data from the **HTML structure**?
  - beautifulsoup
  - Multiple fields in a single HTML tag? **Regex** can help (regex101, AutoRegex)
- Can I successfully **reconstruct** the requests from the browser, e.g., in an API client?
  - Which **headers** do I need to include (User-Agent, cookies, CSRF token, etc.)?
  - How can I **adjust search parameters** if necessary? Do I need additional requests?
  - Is there **pagination**, and can I construct requests for subsequent pages?

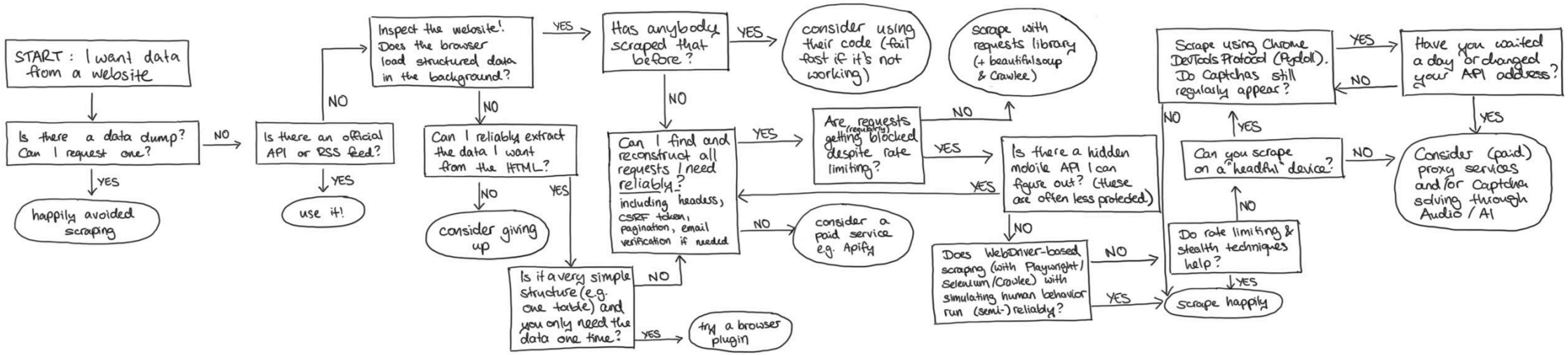
3. Ethics & Legal

- Does the page allow scraping (robots.txt, ToS)? Or is there sufficient public interest?
- Can I provide a user agent disclosing my identity and the purpose of my scraping?

```
import requests
url = f"https://..."
headers = {
    "User-Agent": "MyApp/1.0 (firstname.lastname@example.com)"
}
response = requests.get(url, headers=headers)
...
```

- How many requests can I reasonably expect the page to handle? → don't send unnecessary requests, spread out requests
- You might want to involve your legal department

'How to scrape this?' Flowchart

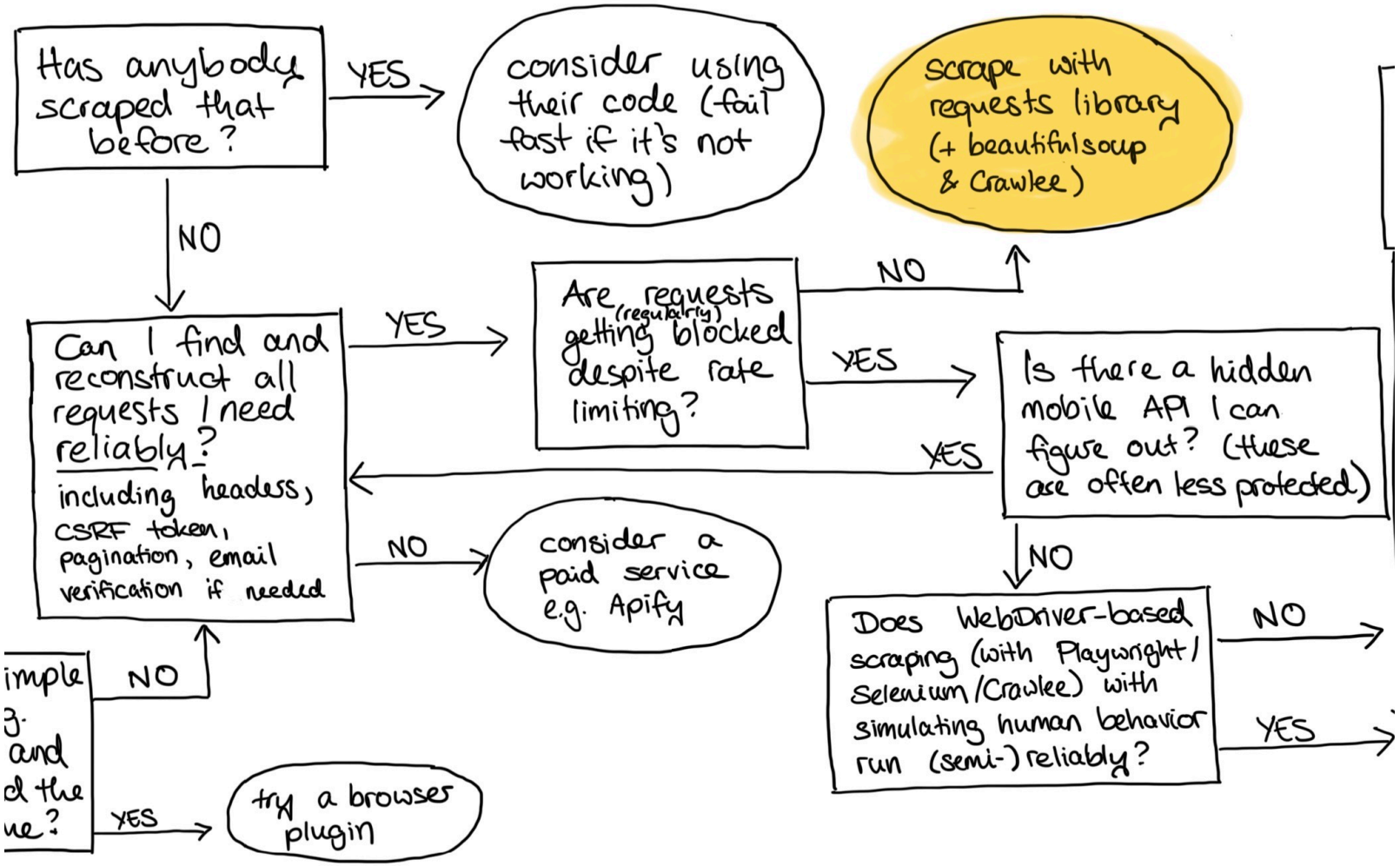


Genereal tips / truths

- Scraping is always a bit unpredictable
- The larger the website, the more protected is usually is (Cloudflare, Akamai)
- You can scrape pretty much everything – but at which cost
  → choose the easiest way possible
- When running a scraper in production in a cloud, blocking is even harder to prevent

4. Examples from real-life projects

Example 1: Easy scraping with requests



Tool

requests library

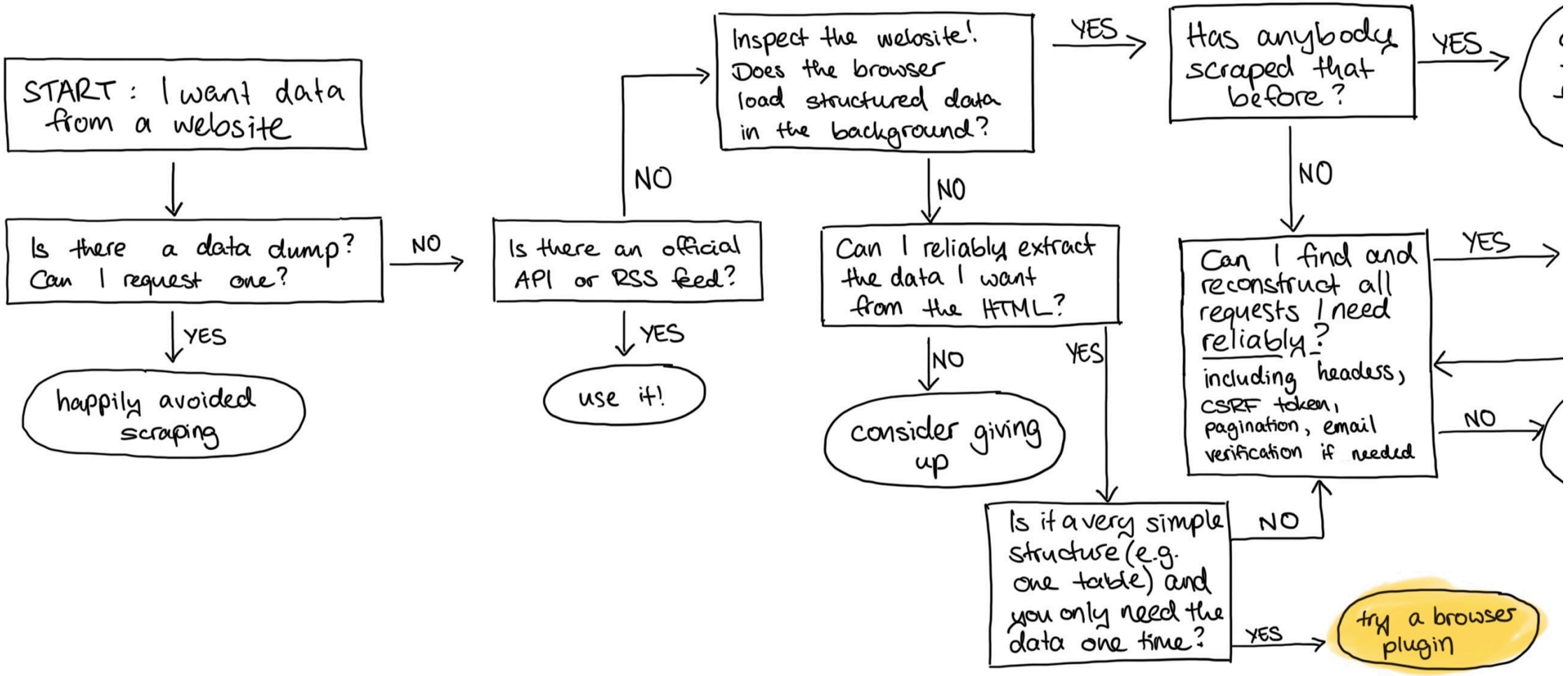
- Download the page/data: requests.get(url) (or post) to fetch what the server returns (HTML or JSON).
- Look at response.status_code (e.g. 200 = success, 404 = not found, 429 = too many requests).
- Parse and extract what you need:
  - JSON: use response.json()
  - HTML: parse response.text with a parser like BeautifulSoup
- Write to i.e. csv, database, etc.

Example: Immoscout scraping

```
BASE = "https://api.mobile.immobilienscout24.de/search/list"
headers = {
    "User-Agent": "ImmoScout_27.3_26.0_._",
    "Accept": "application/json",
    "Content-Type": "application/json",
    "Connection": "keep-alive",
}
body = {"supportedResultListType": [], "userData": {}}

def fetch_listings_for_city(geocode, city_name):
    params = {
        "realestatetype": "apartmentrent",
        "numberofrooms": "1-2",
        ...
    }
    for page_num in range(1, total_pages + 1):
      params["pagenumber"] = page_num
      try:
          resp = requests.post(BASE, params=params, json=body, headers=headers, timeout=10)
          resp.raise_for_status()
          data = resp.json()
          ...
```

Example 2: Brower Plugin



Tool

chrome web store

Erweiterungen und Designs suchen

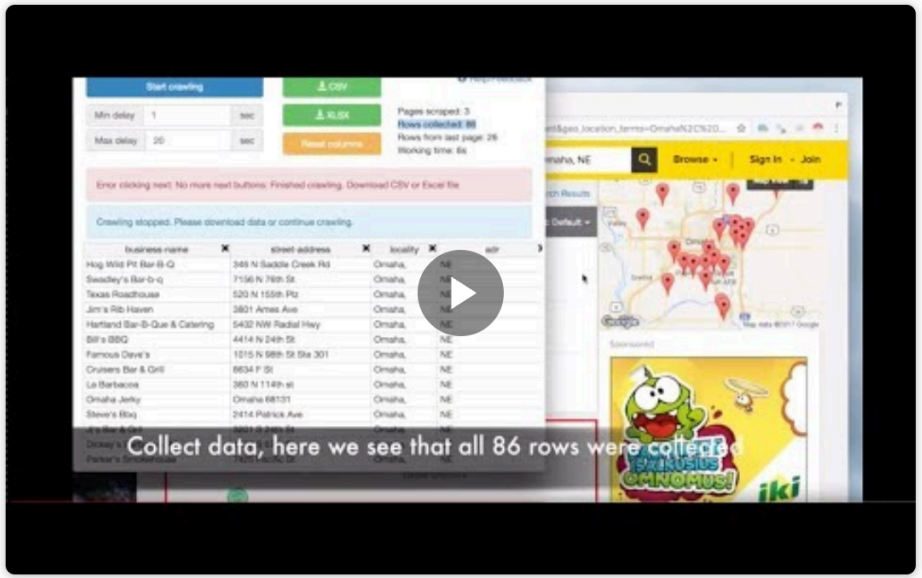
Entdecken Erweiterungen Designs

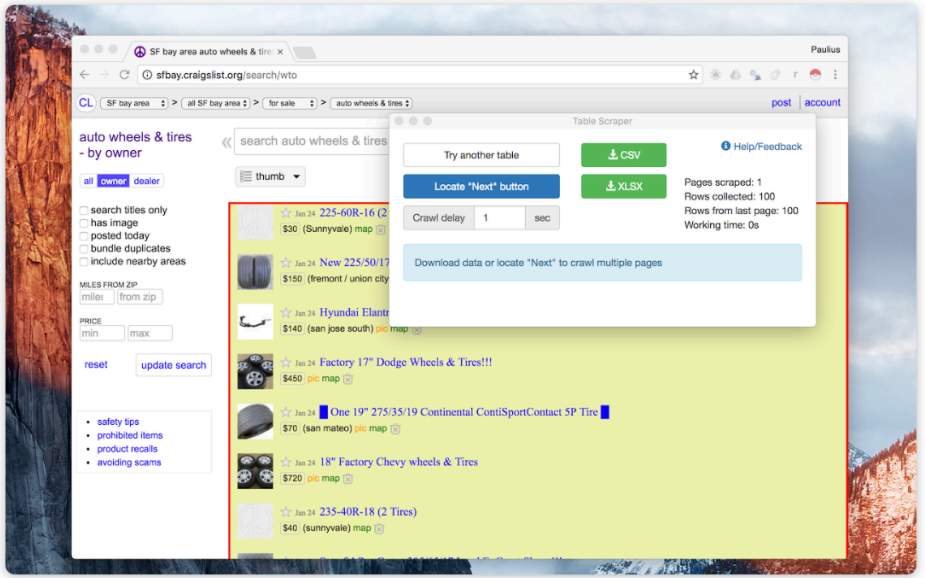
Instant Data Scraper

Hinzufügen

Flavr Technology, LP Empfohlen 4,9 ★ (7.330 Bewertungen) Teilen

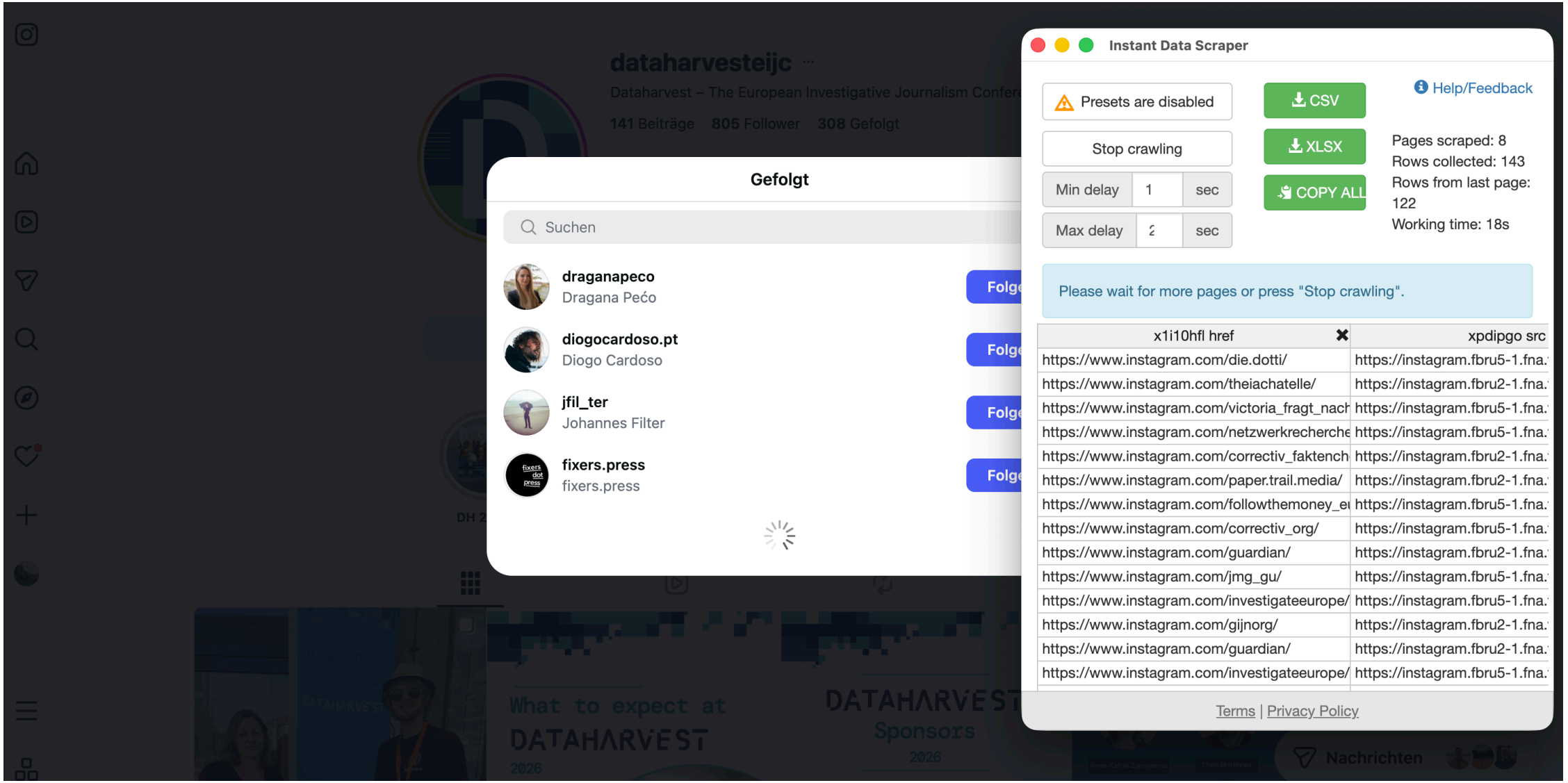
Erweiterung Workflow & Planung 1.000.000 Nutzer





Scraping Instagram followers/following lists

- Preparation for network analysis


dataharvesteijc
Dataharvest – The European Investigative Journalism Confer
141 Beiträge 805 Follower 308 Gefolgt
Gefolgt
Suchen
draganapeco
Dragana Pećo
diogocardoso.pt
Diogo Cardoso
jfil_ter
Johannes Filter
fixers.press
fixers.press
Folge
Instant Data Scraper
Presets are disabled
Stop crawling
Min delay 1 sec
Max delay 2 sec
CSV
XLSX
COPY ALL
Help/Feedback
Pages scraped: 8
Rows collected: 143
Rows from last page: 122
Working time: 18s
Please wait for more pages or press "Stop crawling".
x1i10hfl href
xpdipgo src
https://www.instagram.com/die.dotti/
https://www.instagram.com/theiachatelle/
https://www.instagram.com/victoria_fragt_nach
https://www.instagram.com/netzwerkrecherche
https://www.instagram.com/correctiv_faktench
https://www.instagram.com/paper.trail.media/
https://www.instagram.com/followthemoney_e
https://www.instagram.com/correctiv_org/
https://www.instagram.com/guardian/
https://www.instagram.com/jmg_gu/
https://www.instagram.com/investigateeurope/
https://www.instagram.com/gijnorg/
https://www.instagram.com/guardian/
https://www.instagram.com/investigateeurope/
Terms | Privacy Policy
What to expect at DATAHARVEST 2026
DATAHARVEST Sponsors 2026
Nachrichten

Learnings

- Instagram authentication is very sturdy
- Quick method for a one-time project
- Ideally use popsucket account because you could get restricted

Example 3: How to bypass email verification

- **The problem:** some websites require email verification to create accounts
  - Barrier for scraping
  - Your private email gets flooded with verification emails
  - Automated logins may trigger spam filters or security alerts leading to account lockouts or missed verification emails
- **Solution:** Temporary email services to receive verification codes without using personal email addresses (e.g. DropMail, TempMail, 10MinuteMail)

Tool

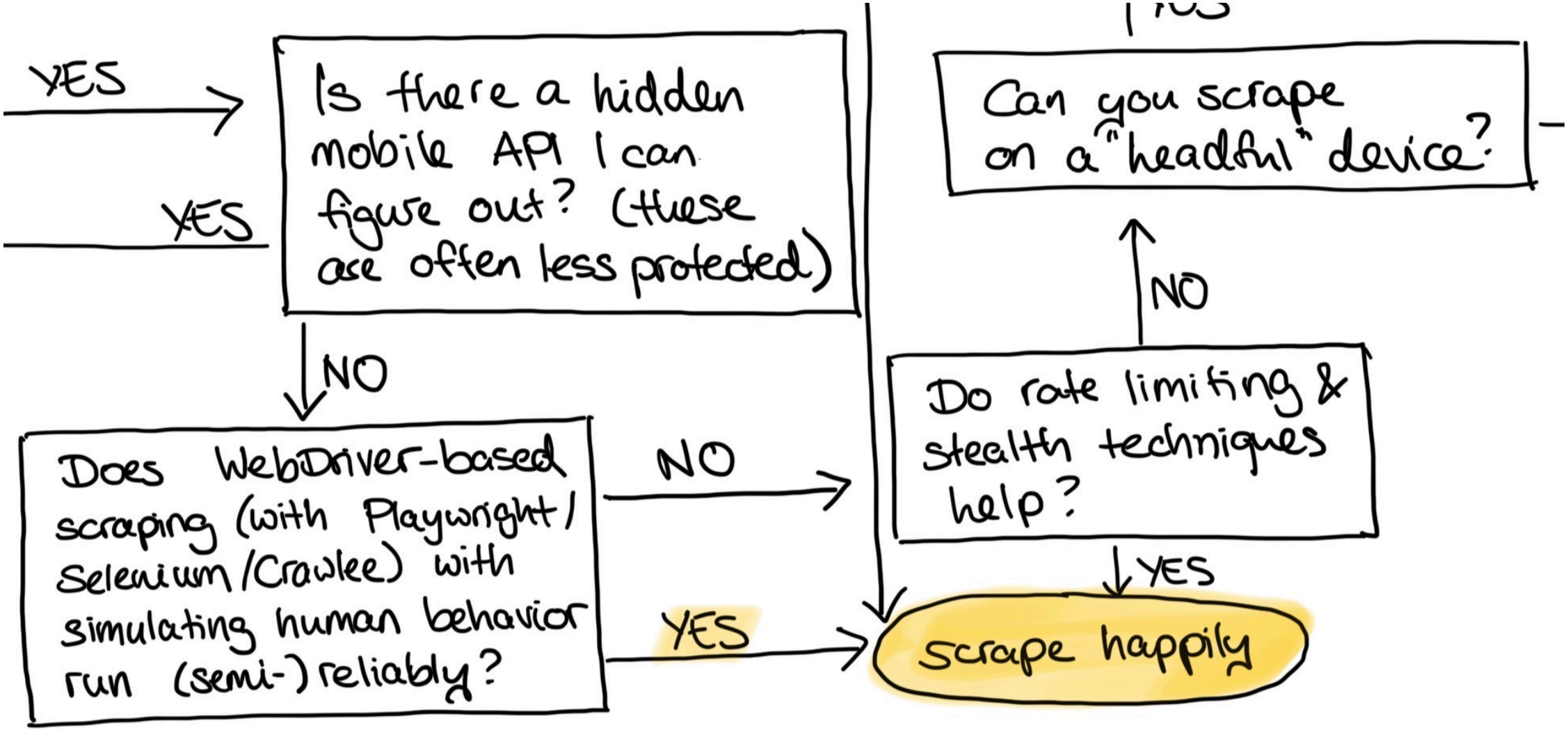
Dropmail

- Temporary inbox generation: DropMail creates a disposable email address like xyz789@dropmail.me
- The API is built around **sessions** = temporary mailboxes
- captures i.e. verification links in a temporary inbox
- The email address doesn’t require registration and only exists for a short time (10 min)
- Sessions expire automatically → everything is deleted permanently

Learnings

- At least 10 sec between registration and polling the inbox is needed to ensure the verification email has arrived before checking for it
- Some sites may block known temp email domains, so testing with different services recommended
- Convenient but less reliable than real inboxes: best for testing/scraping rather than critical communication

Example 4: Scraping X with Playwright & Crawlee



Tool

Crawlee

- Built for large-scale crawling, not just page scraping
- Unified API across HTTP scraping and browser automation
- Designed around reliability and cloud execution

```python
"""Logs into the website with a multi-step process."""
page: Page = context.page
browser_context: BrowserContext = page.context
request: Request = context.request

with open("credentials/x_account.json", "r") as f:
    credentials = json.load(f)

context.log.info("Starting login process...")
await page.goto("https://x.com/i/flow/login")

time.sleep(2)
# --- Step 1: Username ---
context.log.info("Entering username...")
username_field = page.locator("input[name='text']")
await username_field.fill(credentials["username"])
await username_field.press("Enter")

# Wait for the next step to load (adjust the selector as needed)
# this is a good indicator that the username was accepted
try:
    await page.wait_for_selector("input[name='password']", timeout=5000)
except:
    context.log.info("no password field found, maybe email verification is needed")

time.sleep(2)
# --- Step 2: Email Verification (Conditional) ---
try:
    email_field = page.locator("input[name='email']")
    await expect(email_field).to_be_visible(timeout=5000)
    context.log.info("Email verification required. Entering email...")
    await email_field.fill(credentials["email"])
    await email_field.press("Enter")
    # Wait for the password field to load
    await page.wait_for_selector("input[name='password']")
except:
    context.log.info("No email verification needed.")

time.sleep(2)
# --- Step 3: Password ---
context.log.info("Entering password...")
password_field = page.locator("input[name='password']")
await password_field.fill(credentials["password"])
await password_field.press("Enter")
```

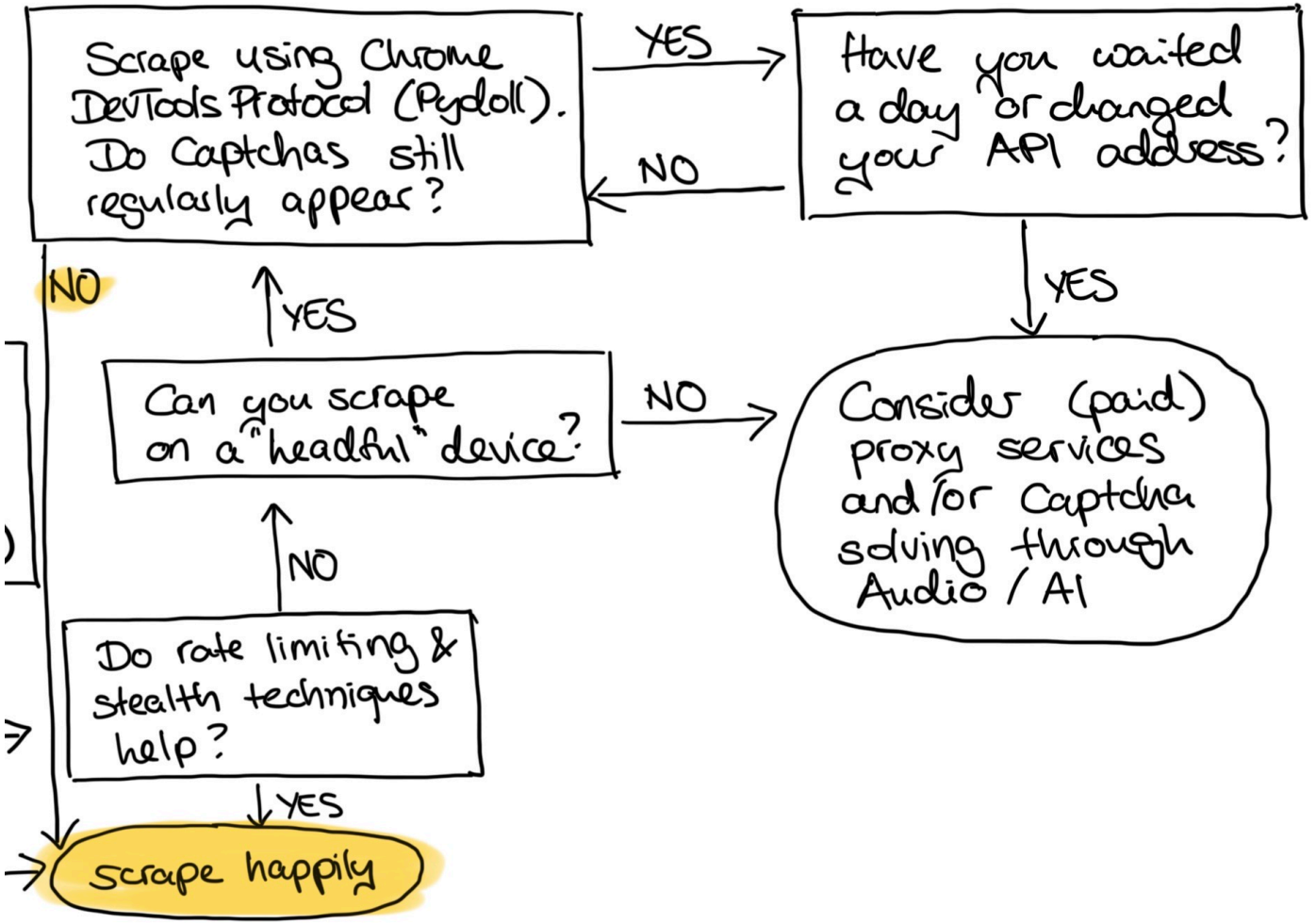

X Scraper

- Scraping X for specific right-wing accounts to reveal repost networks
- parts of X are JavaScript-rendered, login-gated, infinite-scroll, interaction-driven

Learnings

- Crawlee + Playwright was effective for scraping X
- Selector strategy matters
- Infinite-scroll scraping needs explicit stop logic

Example 5: Scraping Google shopping prices with Pydoll



Tool

Pydoll

- Simulates real human behavior patterns (mouse movements, keyboard inputs, etc.)
- Eliminates the need for webdriver: connects directly to browsers using their DevTools Protocol
- Improves CAPTCHA pass rates by emulating realistic browser behavior
- Built on native async/await for efficient concurrent browser automation.

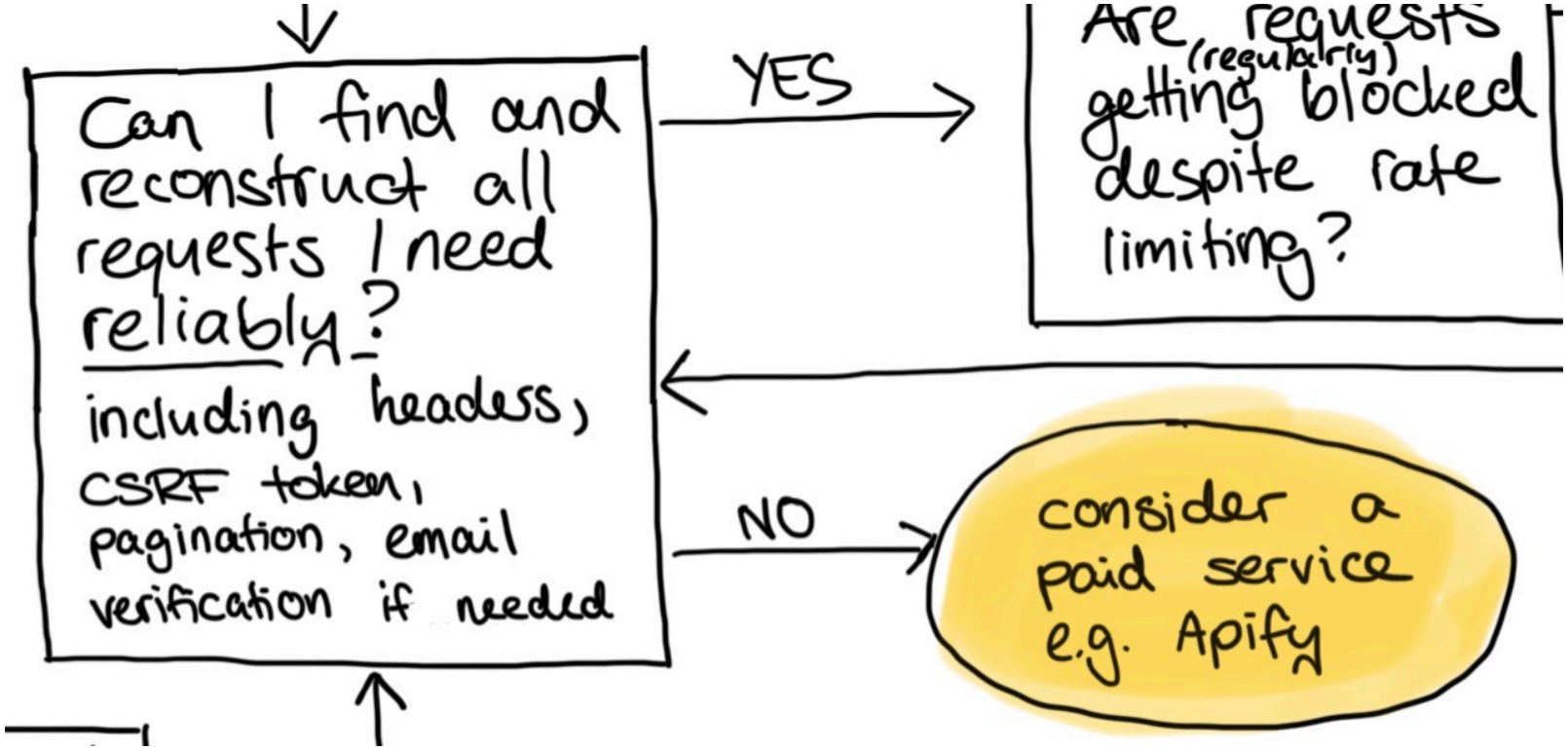
Google scraper

- Project: Automated scraping of Google Shopping search results for a defined list of products
- Google uses strong anti-scraping measures (CAPTCHAs, IP blocking, dynamic content) → therefore: Browser Automation with **pydoll** (Chromium)
- Workflow: Search a list of products multiple times → extract prices, positions and shop names from the product carousel → store the dataset for analysis
- Additional requirement: Screenshots

Learnings

- Headless mode is detectable
- Google's DOM is unstable
- CAPTCHA handling needs explicit detection
- Retry logic is essential
- Random sleep intervals mimic human behavior and reduce bot detection risk

Example 6: When to use paid tools



Scraping Twitter/X followers and followees

- Scraping X throught their API requires Pro access for 5k $ a month
- Otherwise you can only see the first X followers/following users

Tool

Apify

- apify.com
- Cloud-based web scraping and automation platform
- Run scrapers in the Apify cloud
- Scheduling and automation possible
- Marketplace of scrapers for many websites sold as SaaS by developers ("actors")
- Varying price models (per 1k entries / per month)
- Legal/ethical responsibility still on you as the user

Apify Store

Search Actors

Search

Social media | AI | Agents | Lead generation | E-commerce | SEO tools | ...

All Actors

View all →

Google Maps Scraper
compass/crawler-google-plac…
Extract data from thousands of Google Maps locations and businesses, including reviews, reviewer details, images, contact info,...
Compass ☆ 4.8 (1,403) 428K

TikTok Scraper
clockworks/tiktok-scraper
Extract data from TikTok videos, hashtags, and users. Use URLs or search queries to scrape TikTok profiles, hashtags, posts,...
Clockworks ☆ 4.7 (294) 186K

Website Content Crawler
apify/website-content-crawl…
Crawl websites and extract text content to feed AI models, LLM applications, vector databases, or RAG pipelines. The Actor...
Apify ☆ 4.5 (201) 128K

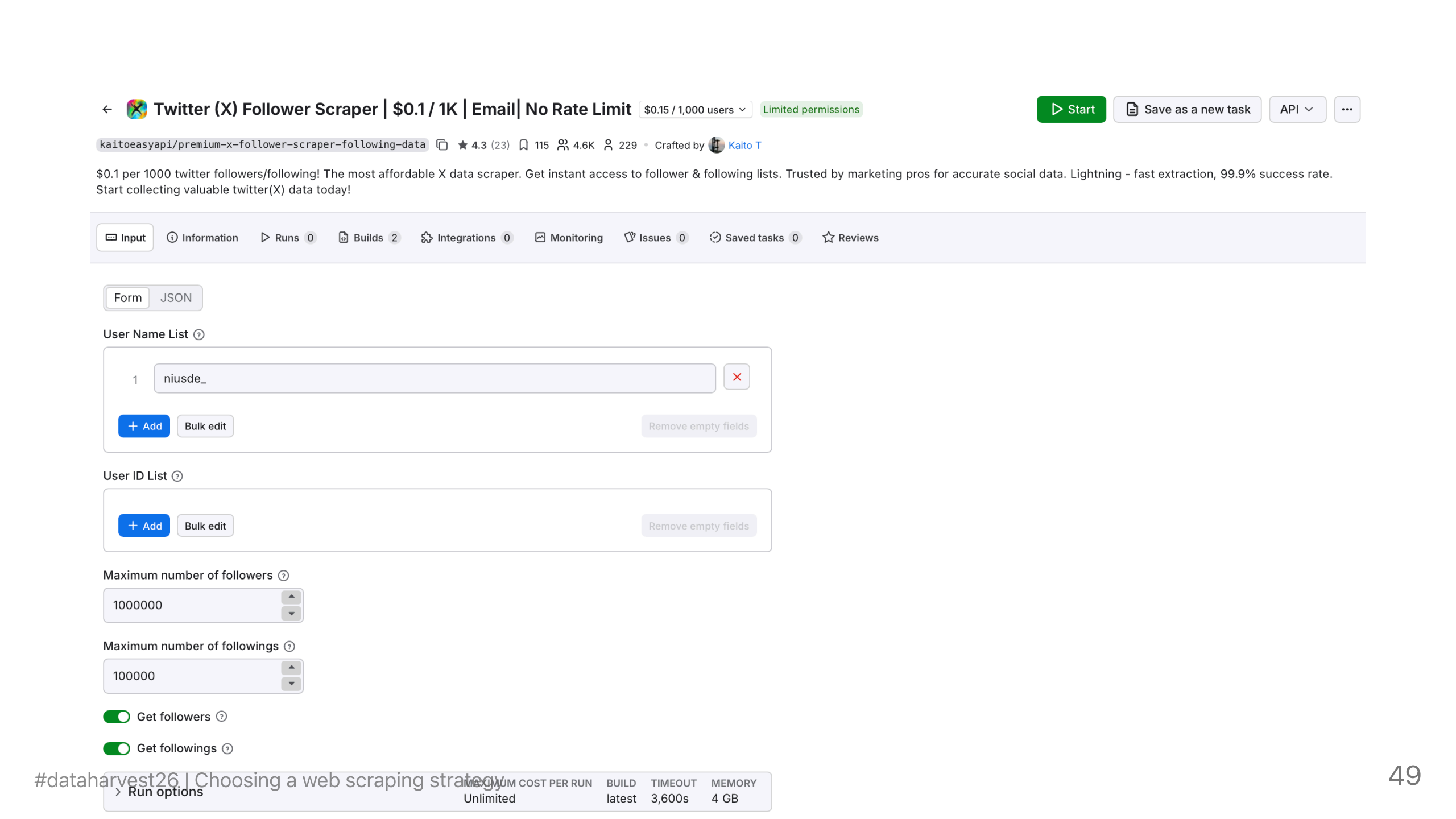
Instagram Scraper
apify/instagram-scraper
Scrape and download Instagram posts, profiles, places, hashtags, photos, and comments. Get data from Instagram using...
Apify ☆ 4.7 (422) 278K

Google Search Results Scra...
apify/google-search-scraper
Scrape Google Search Engine Results Pages (SERPs). Select the country or language and extract organic and paid results, AI Mode, ...
Apify ☆ 4.8 (137) 130K

E-commerce Scraping Tool
apify/e-commerce-scraping-t…
Scrape data from e-commerce websites with E-commerce Scraping Tool. Scrape almost any retail site in minutes, extract e-...
Apify ☆ 4.6 (43) 11K

Facebook Posts Scraper
apify/facebook-posts-scraper
Extract posts, videos, and engagement metrics from Facebook pages. Get text captions, reactions, video transcripts,...
Apify ☆ 4.5 (178) 75K

YouTube Scraper
streamers/youtube-scraper
YouTube crawler and video scraper. Alternative YouTube API with no limits or quotas. Extract and download channel...
Streamers ☆ 4.8 (164) 82K


Twitter (X) Follower Scraper | $0.1 / 1K | Email| No Rate Limit
$0.15 / 1,000 users
Limited permissions
Start
Save as a new task
API
kaitoeasyapi/premium-x-follower-scraper-following-data
4.3 (23)
115
4.6K
229
Crafted by Kaito T
$0.1 per 1000 twitter followers/following! The most affordable X data scraper. Get instant access to follower & following lists. Trusted by marketing pros for accurate social data. Lightning - fast extraction, 99.9% success rate. Start collecting valuable twitter(X) data today!
Input
Information
Runs 0
Builds 2
Integrations 0
Monitoring
Issues 0
Saved tasks 0
Reviews
Form
JSON
User Name List
1
niusde_
Add
Bulk edit
Remove empty fields
User ID List
Add
Bulk edit
Remove empty fields
Maximum number of followers
1000000
Maximum number of followings
100000
Get followers
Get followings
Run options
MAXIMUM COST PER RUN
Unlimited
BUILD
latest
TIMEOUT
3,600s
MEMORY
4 GB

Learnings

- Actors vary in quality
- Make use of free trials to test them
- Good for one-time scraping of very complex websites
- You have to focus what to scrape – many runs with a lot of data will be expensive

5. How to run a scraper in production

Best Practices

- **Scrape, then parse** (don't do it in one step)
- **Error handling**: don't loose all data because one request fails
  → save continuously, logging, enable starting from where it failed
- **Monitor** your progress (e.g. with tqdm progress bar)
- Use version control (Git) for your code
- **Structure** your output
  - e.g. subfolder with date/timestamp for each run
  - database with timestamps (created, updated)
- Be aware that what's running stable on your laptop might not run stable in production

Tool

Airflow

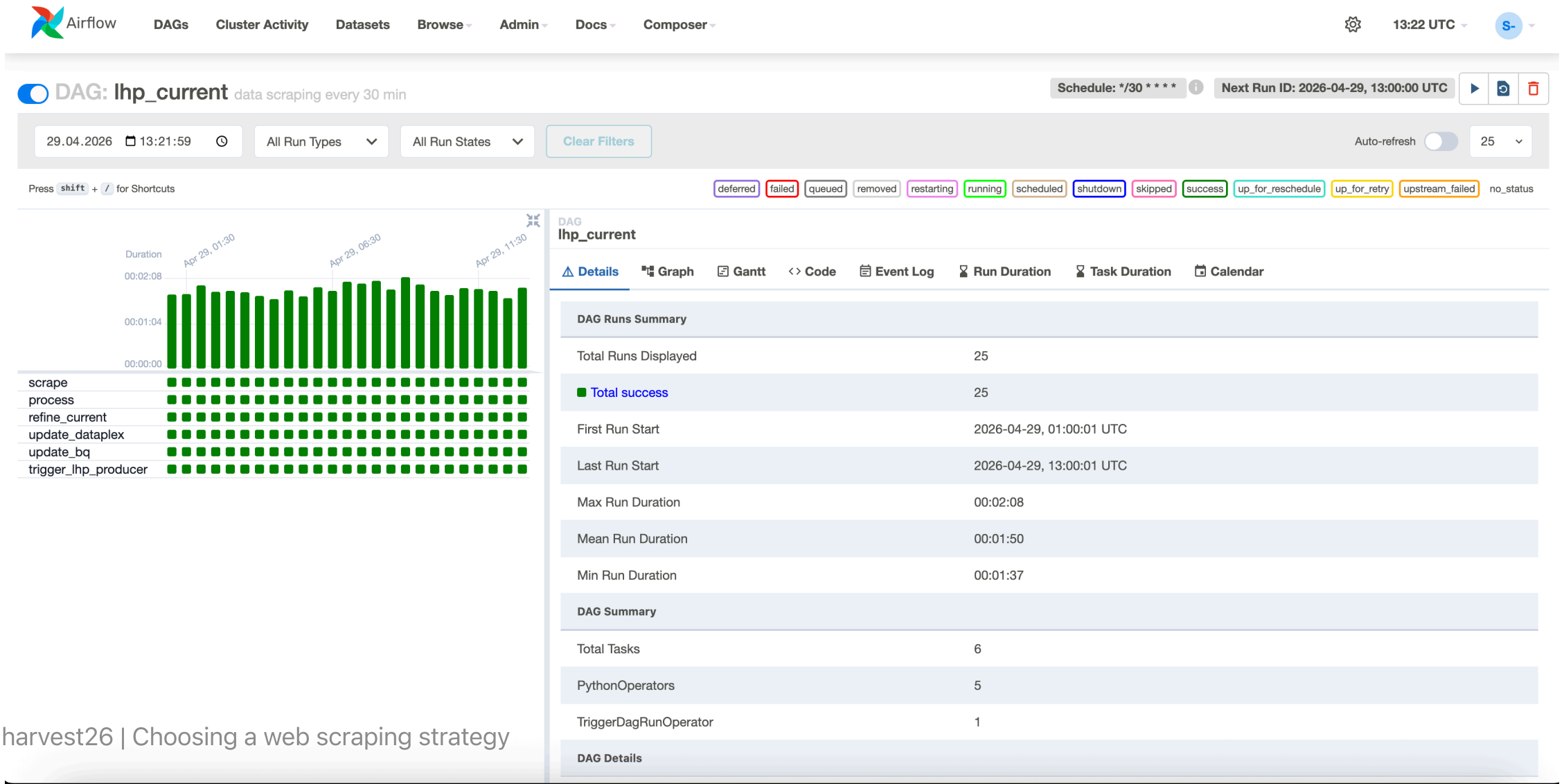
- Open-source tool to define, schedule and monitor workflows (called DAGs)
- Commonly used to automate data pipelines, including web scraping tasks
- Runs schedules without manual effort, handels retries, keeps logs to debug easily

Tool

DAG

- DAG = Directed Acyclic Graph
  - Defines a workflow as a set of tasks with clear dependencies between them
  - Directed → task flows in a specific direction (from start to end)
  - Acyclic → no cycles or loops back to previous tasks
- A DAG ensures tasks run in the correct sequence (e.g., scrape → clean → store data) and only start when their upstream dependencies are completed.

Example DAG


Airflow
DAGs
Cluster Activity
Datasets
Browse
Admin
Docs
Composer
13:22 UTC
DAG: lhp_current data scraping every 30 min
Schedule: */30 * * * *
Next Run ID: 2026-04-29, 13:00:00 UTC
29.04.2026 13:21:59
All Run Types
All Run States
Clear Filters
Auto-refresh
25
Press shift + / for Shortcuts
deferred
failed
queued
removed
restarting
running
scheduled
shutdown
skipped
success
up_for_reschedule
up_for_retry
upstream_failed
no_status
Duration
Apr 29, 01:30
Apr 29, 06:30
Apr 29, 11:30
00:02:08
00:01:04
00:00:00
scrape
process
refine_current
update_dataplex
update_bq
trigger_lhp_producer
DAG
lhp_current
Details
Graph
Gantt
Code
Event Log
Run Duration
Task Duration
Calendar
DAG Runs Summary
Total Runs Displayed 25
Total success 25
First Run Start 2026-04-29, 01:00:01 UTC
Last Run Start 2026-04-29, 13:00:01 UTC
Max Run Duration 00:02:08
Mean Run Duration 00:01:50
Min Run Duration 00:01:37
DAG Summary
Total Tasks 6
PythonOperators 5
TriggerDagRunOperator 1
DAG Details

Advantages of using Airflow for scraping

- **Scheduling & automation:** DAGs run scraping jobs on a fixed schedule (e.g., hourly, daily), eliminating manual triggers.
- **Task dependency management:** steps (fetch → parse → store) run in a certain order, and only proceed when upstream tasks succeed.
- **Retry & failure handling:** automatically retries failed scraping tasks, alerts when something breaks —> useful for unstable target sites.
- **Scalability & parallelism:** DAGs can run multiple scraping jobs in parallel across workers -> scrape many sources efficiently without bottlenecks.
- **Monitoring & logging:** built-in UI provides detailed logs, run history, and task status -> easier debugging

Resources

Tools

- browser dev tools
- API clients e.g. Bruno, Insomnia, Postman
- Crawlee, Playwright, Selenium
- DropMail
- Pydoll
- Airflow
- Apify scraper marketplace and cloud platform
- Browser plugins like Instant Data Scraper, Web Scraper

Useful Little Helpers

- httpbin.org/headers
- urlprettyprint.com/
- regex101.com
- autoregex.xyz
- tqdm progress bar
- beautifulsoup HTML parsing

Practice

- web-scraping.dev/
- www.scrapethissite.com/pages/

Thank you

Stephanie Jauss
stephanie.jauss@SWR.de

Verena Steinacher
steinacher.pub@SWR.de