AI Competitor Spy Agent — Step-by-step Project Roadmap

TL;DR: One single repo, iterative development. Start by building a robust data-collection core (scrapers + DB + scheduler). Then add analysis (NLP + LLMs), and finally a UI + automation (reports/alerts). This document is a *daily, hands-on* roadmap designed so you always work inside the same project and feel continuous progress.

How to use this roadmap

- Follow the *Day-by-day* tasks in order. If a day references a technology you don't know yet, **read that short topic** (links you find) and then immediately apply it in the same repo.
- Commit small, frequent changes. Use feature branches (feature/day-XX) and open PRs to `main` when a day's goal is complete.
- Treat every day as a mini-iteration: **Plan → Implement → Commit → Test → Write a one-paragraph changelog** in `docs/diary.md`.
- This roadmap assumes ~evening sessions (single focused session per day). If you have more time on weekends, use it for the larger integration tasks.

Project conventions & single-repo philosophy

Repo name: `competitor-spy-agent` (one repo covers scrapers, backend, workers, and frontend)

Branching: `main` (productionlike), `dev` (integration), `feature/<short-name>` for tasks.

Minimal folder layout (start here):

```
competitor-spy-agent/
├── README.md
├── .gitignore
├── pyproject.toml   # OR requirements.txt + requirements-dev.txt
├── docker-compose.yml
├── .env.sample
├── src/
│   ├── scraper/        # scraping logic, parser, playwright helpers
│   ├── db/             # models, migrations (alembic), session
│   ├── api/            # FastAPI app (simple endpoints to read data)
│   ├── workers/        # scheduled jobs/celery tasks (later)
│   ├── nlp/            # NLP helpers (keyword extraction, clustering)
│   └── utils/          # logging, http client, config
```

```
├── tests/
├── scripts/            # dev convenience scripts (seed, run-local)
└── docs/
```

Initial developer setup (very first steps)

1. Create the repo on GitHub. Add `README`, `LICENSE` and `CONTRIBUTING.md`.
2. Locally:

```
# create venv and activate (POSIX shown)
python -m venv .venv
source .venv/bin/activate
python -m pip install -U pip
```

1. Add `requirements.txt` (or `pyproject.toml`) and install minimal deps:

```
requests
beautifulsoup4
lxml
playwright
sqlalchemy
alembic
pydantic
fastapi
uvicorn[standard]
python-dotenv
pandas
pytest
loguru
```

1. Install Playwright browsers:

```
playwright install
```

1. Create `.env.sample` with keys and developer notes (no secrets in repo):

```
DATABASE_URL=sqlite:///./data/dev-data.db
USER_AGENT=competitor-spy/0.1
OPENAI_API_KEY=
PROXY_URL=
```

1. Add pre-commit hooks (formatters): `black`, `isort`, `ruff` or `flake8`.

Starter repo scaffold: minimal files to create today

- `src/scraper/simple_scraper.py` — single-file scraper that uses `requests + bs4` to fetch a page and extract: title, meta description, h1/h2 list, outbound links, canonical, and article text.
- `src/db/models.py` — SQLAlchemy models for `Site` and `Page` and helper to create `sqlite` DB.
- `src/utils/http_client.py` — wrapper for requests with headers, retries, and basic backoff.
- `README.md` — project purpose + how to run the simple scraper.

Day-by-day plan (Days 1–30 — extremely hands-on; everything happens inside the same repo)

> Each day's entry has: **Goal**, **What to implement**, **Commands / code pointers**, **Deliverable / commit message**. If you finish early, move to the next day.

Day 1 — Project init + very small scraper (foundation)

Goal: Create the repo, venv, install deps, and make a *working single-file scraper*.

Implement: - Repo + branch `feature/day-01`. - `src/scraper/simple_scraper.py` that: - Fetches a URL using `requests` with a `User-Agent` header. - Parses HTML with BeautifulSoup to extract: title, meta description, h1/h2, all links (absolute), canonical link, and article text (best-effort: `article`, `.content`, `.post`). - Prints JSON to stdout. - Add `README` instructions on how to run it.

Deliverable/commit: `feat(day-01): add simple requests+bs4 scraper; README run steps`

> *Starter code snippet (put in the file):* see the canvas doc for the full snippet.

Day 2 — Sitemap & seed-list crawling

Goal: Crawl a site's sitemap to build a seed URL list and test scraping multiple pages.

Implement: - Add `src/scraper/sitemap.py` with a `parse_sitemap(sitemap_url)` function. - Build a small `crawl_seed.py` script that reads sitemap URLs and runs `simple_scraper` on the first N pages. - Save scraped JSON files into `data/raw/<domain>/`.

Deliverable/commit: `feat(day-02): sitemap parser + seed crawl; save JSON snapshots`

Day 3 — CSV/JSON export & simple dedup

Goal: Persist raw outputs to CSV/JSON and avoid duplicates.

Implement: - Add `data/` write helpers. - Maintain a `seen_urls.txt` or use a `checksums.json` to deduplicate by normalized URL. - Add script `scripts/export_csv.py` to convert raw JSON snapshots into a single CSV for analysis.

Deliverable/commit: `chore(day-03): add persistence helpers + dedup logic`

Day 4 — DB & SQLAlchemy models (move from files to DB)

Goal: Persist pages to a database (start with SQLite locally).

Implement: - `src/db/models.py` with `Site` and `Page` models. Minimal fields: `id, site_url, url, title, meta, headings, raw_html, fetched_at, checksum`. - `src/db/session.py` to provide `SessionLocal` and `init_db()`. - Modify crawler to upsert pages into DB instead of writing raw files.

Deliverable/commit: `feat(day-04): add sqlalchemy models + DB persistence`

Day 5 — Respect robots.txt & polite crawling

Goal: Add robots.txt checks, rate limiting and simple retry/backoff.

Implement: - Use `urllib.robotparser` to check allowed paths for your `USER_AGENT`. - Add `time.sleep(random.uniform(1.0, 3.0))` between requests and exponential backoff on 5xx responses. - Log blocked URLs.

Deliverable/commit: `chore(day-05): robots.txt respect + polite throttling`

Day 6 — Handle dynamic sites with Playwright

Goal: Add Playwright scraper that can render JS pages.

Implement: - `src/scraper/play_scraper.py` with a `fetch_with_playwright(url)` helper. - Switch automatically to Playwright when robots indicate an index page requires JS or when HTML contains `__NEXT_DATA__`, heavy JS signals.

Deliverable/commit: `feat(day-06): add playwright fetcher for JS-heavy pages`

Day 7 — Parser hardening & unit tests

Goal: Make parsers robust and add tests.

Implement: - Parser refactor: `parser.py` with small functions: `extract_title`, `extract_meta`, `extract_headings`, `extract_links`, `extract_article_text`. - Add `tests/test_parser.py` with snapshot-like checks on saved HTML samples (5 different sites you scraped).

Deliverable/commit: `test(day-07): parser tests + parser refactor`

Day 8 — Snapshot versioning & diffs

Goal: Keep history of pages and compute diffs (what changed).

Implement: - Add `snapshots` table with `page_id, version, fetched_at, checksum, raw_html`. - When saving a page, insert new snapshot only if checksum changed. - Add a simple `diff` method using `difflib` to see what changed between last two versions.

Deliverable/commit: `feat(day-08): add snapshots + change detection`

Day 9 — CLI & small orchestration layer

Goal: Add a developer-friendly CLI to run common tasks.

Implement: - Use `typer` or `click` to add commands: `crawl-domain`, `crawl-url`, `list-sites`, `reindex`. - Commands should load `.env` config and use DB connection.

Deliverable/commit: `feat(day-09): add CLI (typer) for dev tasks`

Day 10 — Dockerize local stack

Goal: Make it easy to run locally with Docker (sqlite is fine to start, but add Postgres in compose for later).

Implement: - `Dockerfile` for the app. - `docker-compose.yml` with `app`, `postgres` (for future), `redis` (optional), and instructions in `README`.

Deliverable/commit: `chore(day-10): add Dockerfile + docker-compose (dev)`

Day 11 — Simple API to browse results (FastAPI)

Goal: Expose scraped pages via a tiny API to make it easy to integrate the frontend or tests.

Implement: - `src/api/main.py` with endpoints: `GET /sites`, `GET /pages?domain=`, `GET /pages/{id}`. - Add OpenAPI docs (auto-generated by FastAPI).

Deliverable/commit: `feat(day-11): add FastAPI read endpoints`

Day 12 — Add basic NLP: extract keywords & summarize (prototype)

Goal: Run a simple keyword extraction and produce one-line summaries.

Implement: - Integrate `spaCy` (or a simple TF/IDF using `scikit-learn`) for keyword extraction. - Add a `summarize_text(text)` wrapper using OpenAI (or local LLM if preferred) — keep this isolated behind `src/nlp/summarizer.py`. - Store summary + top keywords in the `pages` table.

Deliverable/commit: `feat(day-12): basic NLP (keywords + summary) stored on pages`

Day 13 — Batch processing & LLM rate management

Goal: Summarize many pages efficiently (batched & retried)

Implement: - Implement batch jobs to send multiple pages to the LLM with rate-limiting and retries. - Add a `llm_tasks` table to track LLM job status.

Deliverable/commit: `chore(day-13): add batched llm processing with retry`

Day 14 — Build a weekly report generator (proof-of-value)

Goal: Make one simple report: "Top 10 pages changed this week" with summaries.

Implement: - A script `reports/weekly_report.py` that queries snapshots, finds changed pages, and creates a markdown report in `reports/weekly-YYYYMMDD.md`. - Save a PNG chart if possible (use pandas to produce data; frontend later).

Deliverable/commit: `feat(day-14): weekly report generator (markdown)`

Day 15 — Notifications (Slack/email)

Goal: Push the weekly report by Slack webhook and/or email.

Implement: - Add a `NOTIFY` config with a Slack webhook; implement `notify.slack.send_markdown(report_md)`. - Add small tests for notification (dry-run mode).

Deliverable/commit: `feat(day-15): slack notifications for weekly reports`

Days 16–20 — Harden & extend

Goals & tasks: - Add tests for scheduling & API. - Add logging (loguru/structlog) and error capture (Sentry skeleton). - Add simple observability: request traces, metrics exported via `/metrics` (prometheus later). - Improve parser resilience for multiple markup patterns.

Deliverable milestone: `milestone: data+api+nlp pipeline complete; weekly report + notification working`

Days 21–25 — Scheduling & workers

Goal: Move from ad-hoc scripts to scheduled jobs.

Implement options: - Simple: `APScheduler` (in-process) for cron-like jobs. - Production: `Celery` + `Redis` for distributed workers.

Deliverable/commit: `feat(day-21-25): add scheduler + basic worker to run periodic crawls`

Days 26–30 — CI/CD, Docker production tweaks, docs & hand-off

Goal: Solidify delivery pipeline and docs.

Implement: - Add GitHub Actions: run tests, lint, build Docker image, optionally push to registry. - Add `docs/architecture.md` and `docs/runbook.md` for how the system collects and handles competitor data (GDPR/compliance notes). - Prepare `seed-data` and demo instructions so others can run it locally or on a small cloud instance.

Deliverable/commit: `chore(day-26-30): ci/cd + docs + runbook`

What you'll have after 30 days

- A single repo with: a **reliable crawler** (requests + Playwright), a **DB** with snapshots, a **basic NLP + LLM summary** pipeline, a **weekly report generator**, and **Slack/email** notifications. You'll also have tests, Docker compose, and CI.

This is a huge, functional MVP that gives real value and is ready to be iterated upon (scale, proxies, SEO APIs, vector DB, LangChain, etc.).

Proficiency checklist (tools → level → proof-of-proficiency)

Tool / Tech	Proficiency target	When to learn in roadmap	Proof-of-proficiency (do this)
Python (core)	Intermediate	Day 1	Write scripts that parse and store pages in DB
Git / GitHub	Intermediate	Day 1	Create branches & PRs; CI runs on PRs
requests / httpx	Intermediate	Day 1–3	Fetch pages and handle retries/ backoff
BeautifulSoup / lxml	Intermediate	Day 1–3	Extract title/meta/h1/h2 and links reliably
Playwright	Intermediate	Day 6	Render JS pages and extract content programmatically
SQLAlchemy / Postgres	Intermediate	Day 4	Persist pages, query snapshots efficiently
pandas / numpy	Beginner→Intermediate	Day 3, 14	Convert raw data to CSV & produce dataframes for reports
FastAPI	Intermediate	Day 11	Expose endpoints returning pages & summaries
spaCy / scikit-learn / KeyBERT	Intermediate	Day 12	Extract top keywords & cluster titles into topics
OpenAI API (or chosen LLM)	Intermediate	Day 12–13	Summarize 20 pages programmatically with prompt batching
Docker / docker-compose	Intermediate	Day 10	Run local stack with one command
Celery / Redis or APScheduler	Beginner→Intermediate	Day 21	Schedule recurring crawls & retry failed jobs
Playwright testing / pytest	Intermediate	Day 7	Unit tests & parser snapshot tests pass in CI
Slack / Email APIs	Beginner	Day 15	Post report to Slack via webhook
Observability (log + Sentry)	Beginner	Day 16	Capture at least one error with context in Sentry (dev DSN)

Quick notes on ethics, legality & scale

- Respect robots.txt and site terms. If you need to scrape protected data (login walls, paywalls), get permission.
- Start with low volume and sample the competitor sites. Add proxies and rotated UAs only when legitimately necessary and allowed.
- Keep PII out of your datasets and redact if you accidentally capture personal data.

Next-phase roadmap (after Day 30)

- **Scale scraping**: proxy pools, headless farms, prioritized frontier.
- **SEO integrations**: SerpAPI, Google Search Console, Ahrefs (paid APIs) for keyword & backlink signals.
- **Vector DB**: Pinecone/Weaviate/FAISS for semantic search across competitor content.
- **LangChain/LlamaIndex**: structured prompts, chain-of-thought summaries, question-answering over competitor content.
- **Advanced analytics**: content cadence detection, topic drift, SERP position tracking, creative ad copy extraction.

Final checklist before you start Day 1 (copy/paste into a new issue called `Day 1`)

- [ ] Create GitHub repo + add README
- [ ] Create `.venv` and install minimal deps
- [ ] Add `.env.sample` and `.gitignore`
- [ ] Add `src/scraper/simple_scraper.py` (basic requests + bs4)
- [ ] Commit & open PR to `dev`

If you want, I can: - Paste the exact **Day 1 starter code** into the chat now (copy-paste ready), OR - Walk you through each Day interactively: I give exact commands, code, and test steps for that day.

Choose one: **(A)** "Paste Day 1 code now" — I'll paste the ready-to-run scraper script; or **(B)** "Walk me through Day 1" — I'll give commands + git commit messages step-by-step.

(You don't need to reply with anything fancy — just say `A` or `B`.)

Good luck — you're building the highest-leverage part first (data). When you have the scraping+DB solid, the LLM layer will make the product magical. 🚀