ChaiVision - Coding Exercise (AI)

🧪 AI Engineer Take-Home — “Amazon Top-10 Intelligence (Keyword Edition)”

Stack: Python + FastAPI (required), Qwen API (required), any local vector store (FAISS/Chroma)

Goal: Enter a keyword (e.g., “massage gun”, “walking pad”, “monitor light bar”), scrape top 10 Amazon results, build a comparison table, and expose a Q&A API using only the scraped data.

✅ What You’ll Build

- **Scrape service:** Accepts a keyword and collects top-10 Amazon products + details.
- **Feature comparison artifact:** CSV derived from scraped records.
- **Q&A API:** FastAPI app backed by Qwen API and a local vector index over the scraped corpus.

⚠️ Practical Notes

- Respect Amazon anti-bot measures: minimal requests, polite backoff, local HTML fallback.
- Qwen API is required; provide your key via .env.
- Keep it lightweight — local run only (no cloud deploy needed).

📦 Suggested Repo Layout

```
amazon-top10-intel/
  README.md
  requirements.txt
  .env.example              # QWEN_API_KEY=
  data/
    products.jsonl
    products.csv
    feature_matrix.csv
  html_snapshots/           # 2–3 example HTML pages for fallback
  scrape/
    fetcher.py              # HTTP fetch + polite backoff + fallback loader
    parsers.py              # selectors/normalizers
    scrape.py               # CLI entrypoint
  bot/
    indexer.py              # builds vector store from products.jsonl
    app.py                  # FastAPI app with /scrape and /ask
  analysis/
    compare.py              # builds feature_matrix.csv
```

🔧 Tasks

1) Scrape Top-10 by Keyword

Inputs:

- keyword (string, required)

- n (int, default 10)
- use_local_html (bool, optional, parse from html_snapshots/)

Data to Capture:

- ASIN, title, price, rating, review_count, bullet_features, brand, dimensions/weight, category/breadcrumb, image_url

Outputs:

- data/products.jsonl
- data/products.csv

CLI Example:

```
# Live scrape
python scrape/scrape.py --q "massage gun" --n 10

# Fallback mode
python scrape/scrape.py --q "massage gun" --n 10 --use-local-html
```

FastAPI Endpoint (Optional):

```
POST /scrape
{
  "keyword": "massage gun",
  "n": 10,
  "use_local_html": false
}
Response: { "ok": true, "count": 10 }
```

---

2) Build Feature Comparison

- Script: analysis/compare.py
- Load products.jsonl, normalize types, extract features (battery_life, noise_level, attachments_count, weight, warranty, etc.)
- Output: data/feature_matrix.csv

Run:

```
python analysis/compare.py
```

---

3) Q&A API (FastAPI + Qwen API)

Indexer (bot/indexer.py):

- Convert products to text documents, build FAISS/Chroma vector index, persist locally.

FastAPI App (bot/app.py):

- POST /scrape → trigger scraping
- POST /index → rebuild vector index
- POST /ask → ask question based on scraped data

Example /ask Request:

```
{
  "question": "Which model under $100 has the best rating?"
}
```

Example Response:

```
{
  "answer": "Model X (ASIN: B0...) is highest-rated under $100 with 4.6★ from 8k+ reviews.",
  "sources": [
    {"asin": "B0XXXX", "snippet": "..."},
    {"asin": "B0YYYY", "snippet": "..."}
  ]
}
```

Run API:

```
uvicorn bot.app:app --reload
```

🧪 Test Keywords

- “massage gun”
- “walking pad”
- “monitor light bar”

Your system should handle **any keyword**, produce fresh products.jsonl/csv, rebuild the index, and answer grounded questions with cited ASINs/snippets.

🧰 Requirements

- Python + FastAPI
- Qwen API (load via .env)
- Vector store: FAISS or Chroma
- Local HTML fallback support

📦 Submission Checklist

- Host solution on **GitHub** and share access; submit within **1 week**.
- README.md with setup, env, commands, API examples, assumptions
- .env.example with QWEN_API_KEY=
- requirements.txt with pinned versions
- FastAPI app exposing /scrape, /index, /ask
- Generated files: products.jsonl, products.csv, feature_matrix.csv
- html_snapshots/ with a few example pages

🧮 Evaluation Rubric

- **Scrape robustness (35%)** — keyword param, resilient selectors, polite fetching, fallback working
- **Data quality (20%)** — clean normalization, useful feature extraction, tidy CSVs
- **Bot grounding (30%)** — retrieval works, Qwen prompts constrained, ASIN/snippet citations
- **Code quality & DX (15%)** — readable structure, simple logs, easy to run, clear README
