# Real-Time Product Information Pipeline Report

## Approach

* Asynchronous Scraping: The system employs httpx in async mode to fetch Google Shopping results with non-blocking calls, allowing rapid and concurrent data collection for each product query.
* HTML Parsing: lxml extracts product name, brand, price, and weight directly from Google Shopping HTML results in a structured, efficient manner.
* API Architecture: Uses Django’s async views (deployed under an ASGI server like Uvicorn) to expose a clean REST endpoint (/api/products/?query=...) for requesting structured product data.
* Caching: Integrates Django's caching system to store results of frequent searches, minimizing redundant scrapes and dramatically improving API latency for hot queries.

## Challenges

* Dynamic Website Structure: Google Shopping regularly changes its HTML layout, which complicates consistent and reliable data extraction.
* Bot Detection and Blocking: Repeated, automated scraping requests from a fixed IP are subject to rate limiting or blocking by Google.
* Async in Django: Properly configuring Django's async stack (e.g., using Uvicorn or Daphne, avoiding ORM blocking operations) can introduce deployment complexity.
* Data Extraction Ambiguity: Product attributes like “weight” may be formatted inconsistently or missing from the results, making robust parsing non-trivial.

## Solutions

* Adaptive XPath/CSS Selectors: Wrote scraping logic to be easily updatable by adjusting XPaths when Google’s DOM changes; extraction robustly defaults missing fields to empty strings.
* Asynchronous Pipeline: Employed httpx.AsyncClient and async def views in Django to maximize throughput and non-blocking I/O.
* Caching via Middleware: Applied cache\_page decorator to API endpoints and recommended Redis or memory-based caches to instantly serve repeated requests without repeated scraping.
* User-Agent Spoofing: Set User-Agent headers in outbound requests to mimic real browsers and reduce bot detection.
* Weight Extraction Logic: Implemented lightweight logic to extract weight strings from names and titles for basic coverage.

## Improvements

* Proxy Rotation: Integrate rotating proxies or third-party scraping APIs to avoid Google rate limits and increase reliability at scale.
* Headless Browser Automation: Switch to Playwright or Selenium for scraping if static parsing fails due to heavy client-side rendering on Google Shopping.
* Advanced Extraction Logic: Use NLP or regex post-processing to more accurately extract fuzzy details like product weight, volume, and packaging.
* Error Handling & Logging: Enhance the system with detailed logging, retry and backoff for failed fetches, and robust exception management for better observability.
* Scalable Deployment: Use Docker and orchestrate with Kubernetes for horizontal scaling across multiple async Django workers.
* Monitoring and Metrics: Integrate Prometheus/Grafana to track latency, error rates, and cache performance for operational excellence.

## Summary

The project delivers a performant, real-time, and scalable solution, with clear paths for increased robustness and scale through proxy usage, better extraction, and operational monitoring.