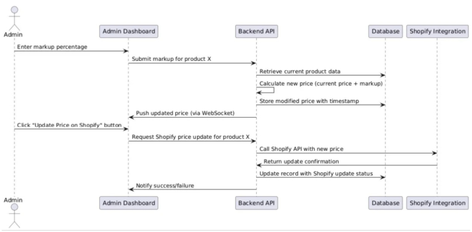
How to design a web scraper for amazon that updates your shopify store for price fluctuations in real time?

Leave a Comment / Uncategorized / By anandakrishnanabhijith



1. High-Level Overview

Our application consists of the following main modules:

- Data Ingestion Module
  - **Responsibility:** Poll Amazon for product details (price, image, description, stock, delivery date) via the Oxylabs API.
  - **Process:** Extracts and normalizes data, then writes it into the database along with timestamps.
  - **Scheduling:** Can be implemented using a scheduler (cron jobs or a task queue like Celery/RabbitMQ).
- Database
  - **Responsibility:** Persist raw product data and later store the modified price (with markup) along with timestamps.
  - **Suggestions:** PostgreSQL or MongoDB depending on our query and scalability requirements.
- Backend API Server
  - **Responsibility:**
    - Serves as the middleware between the frontend and the backend services.
    - Provides REST (or GraphQL) endpoints for fetching product data.
    - Processes markup calculations, saves updates, and triggers Shopify updates.
    - Pushes real-time updates (via WebSockets) to the Admin Dashboard.
  - **Tech Options:** Node.js with Express, Python (FastAPI or Django), or any other modern backend framework.
- Real-Time Communication Server
  - **Responsibility:**
    - Delivers live updates to the Admin Dashboard (e.g., using WebSockets with Socket.io).
  - **Integration:** Could be integrated within our Backend API Server or as a dedicated microservice.
- Admin Dashboard (Frontend)
  - **Responsibility:**
    - Display real-time product data.
    - Allow the admin to apply a percentage markup.
    - Provide a control (button) for each product to trigger the update on Shopify.
  - **Tech Options:** Modern JavaScript frameworks such as React, Angular, or Vue.js.
  - **Real-Time:** Use WebSocket clients (e.g., Socket.io client) to receive live updates.
- Shopify Integration Module
  - **Responsibility:**
    - When an admin approves a price update, this module calls the Shopify API to update the product's price in real time.
  - **Security:** Ensure that the API credentials and tokens for Shopify are securely stored and managed.
- Monitoring & Logging
  - **Responsibility:**
    - Monitor data ingestion jobs, API performance, error handling, and system alerts.
  - **Tools:** Prometheus, Grafana, and the ELK stack (Elasticsearch, Logstash, Kibana).

---

2. Data Flow & Component Interaction

1. **Data Ingestion:**
   - A scheduled job (or continuously running service) uses the **Oxylabs API** to fetch real-time product data from Amazon.
   - The **Data Ingestion Module** processes and stores the data in the **Database** with the current timestamp.
2. **Real-Time Dashboard:**
   - The **Backend API Server** queries the **Database** and pushes live updates to the **Admin Dashboard** via WebSockets.
   - The dashboard displays the current price along with product details.
3. **Price Markup & Update:**
   - An admin enters a markup percentage on the dashboard.
   - The **Backend API** calculates the new price and stores it (with a timestamp) in the **Database**.
   - The dashboard immediately reflects the modified price.
4. **Shopify Update:**
   - Upon clicking the "Update Shopify" button, the **Backend API** triggers the **Shopify Integration Module**.
   - The module calls the **Shopify API** to update the product's price in our Shopify store.
   - A confirmation is received and the database is updated accordingly.
5. **Feedback Loop:**
   - Any successful updates (or errors) are fed back into the system and displayed on the dashboard for transparency.

---

3. Suggested Tools & Tech Stack

Layer	Technology Options
Frontend	ReactJS / Angular / Vue.js, with real-time libraries such as Socket.io-client
Backend	Node.js with Express, Python (FastAPI/Django), or Java (Spring Boot)
Database	PostgreSQL (for relational integrity) or MongoDB (for flexibility and scalability)
Task Scheduling	Cron jobs, Celery (Python), Bull (Node.js), or RabbitMQ for asynchronous processing
Real-Time Updates	Socket.io (for WebSocket communication)
Shopify API	Official Shopify API libraries (available for Node.js, Python, etc.)
Containerization	Docker, with orchestration via Kubernetes if scaling is required
Monitoring	Prometheus, Grafana, ELK (Elasticsearch, Logstash, Kibana)
Authentication	JWT/OAuth for secure API access between modules and for admin authentication

---

4. Additional Considerations & Requirements

- Authentication & Security:
  - Use API keys and tokens for secure communication with Oxylabs and Shopify.
  - Implement admin authentication (e.g., using JWT or OAuth) on the dashboard and API endpoints.
- Error Handling & Retry Logic:
  - Implement robust error handling and retry mechanisms in the Data Ingestion and Shopify Integration modules.
  - Log errors for later analysis and use monitoring tools to alert on failures.
- Scalability & Performance:
  - Consider caching frequently accessed data.
  - Use load balancing for your backend services if the number of products or request rate is high.
  - For very high-frequency updates, a message queue (e.g., RabbitMQ or Kafka) can help decouple and process tasks asynchronously.
- Deployment & CI/CD:
  - Containerize your services using Docker.
  - Use CI/CD pipelines (e.g., GitHub Actions, Jenkins) for automated testing and deployment.
- Data Retention & Analytics:
  - Optionally, store historical data to analyze price trends over time.
  - Provide admin reports and dashboards for analytics.

. Outro

This design outlines a modular and scalable approach:

- **Data Ingestion** gathers real-time product data.
- **Backend API** processes data, calculates markups, and communicates with both the **Admin Dashboard** and **Shopify**.
- **Real-Time Communication** ensures the dashboard stays updated as Amazon prices fluctuate.
- **Shopify Integration** lets admins push approved price changes to their online store.

← Previous Post

Leave a Comment

Your email address will not be published. Required fields are marked *

Type here..

Name*

Email*

Website

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT »