Shopify Product Sync Algorithm

Overview

This project implements a **state-aware synchronization script** that connects a local Excel dataset with a Shopify store via the Admin GraphQL API.

The core challenge this script solves is transforming **flat, tabular data** (Excel) into **hierarchical, relational objects** (Shopify Products & Variants) while ensuring data integrity and respecting API rate limits.

Algorithmic Logic

1. Data Normalization (Flat to Hierarchical)

Raw Excel data is "flat"—meaning a T-shirt with 3 sizes (S, M, L) occupies 3 separate rows. If we processed this row-by-row, we would attempt to create the same T-shirt product 3 times.

The Algorithm:

1. The script utilizes `pandas` to read the entire dataset.
2. It performs a **Group By** operation on the `handle` column.
3. This transforms the N rows into 1 Product Object containing a list of N variants.

2. The "Idempotent" Sync Strategy

A critical requirement was ensuring that running the script multiple times does not corrupt the store (e.g., creating duplicate "Small" variants every time). The algorithm follows a **Check-Then-Act** pattern:

- **Step 1: Existence Check:** The script queries Shopify for the product `handle`.
- **Step 2: Branching Logic:**
    - **Path A (New Product):** If the handle returns `null`, the script constructs a large Mutation payload to create the Product, Options, and Variants in a single API call.
    - **Path B (Existing Product):** If the handle exists, the script enters "Update Mode." It pulls the *current* data from Shopify to compare against the *local* Excel data.

3. Smart Variant Matching (SKU Mapping)

When updating a product, simply pushing variant data would overwrite existing IDs or create duplicates. The script uses an **O(n) matching algorithm**:

1. **Fetch:** Downloads all existing variants for the product from Shopify.
2. **Map:** Creates a hash map (dictionary) where `Key = SKU` and `Value = Variant ID`.

3. **Compare:** Iterates through the Excel rows:
    - **If SKU exists in Map:** It retrieves the Shopify Variant ID and performs an `update` (modifying price/inventory).
    - **If SKU is missing:** It identifies this as a new variation (e.g., a new color added to the collection) and performs a `create` mutation.

4. Leaky Bucket Rate Limiting

Shopify's GraphQL API uses a calculated "Cost" system (Leaky Bucket algorithm) rather than a simple request count.

The Protection Mechanism:

- Every GraphQL response contains a `throttleStatus` object indicating the remaining "budget."
- The script inspects this budget after *every* transaction.
- **Threshold Check:** If `availablePoints < 200`, the script triggers a **Backoff Strategy**, pausing execution (`sleep(5)`) to allow the bucket to refill. This prevents `429 Too Many Requests` errors and ensures stability during bulk uploads.

5. Asynchronous Media Pipeline

Images cannot be uploaded directly with product data. The script implements a multi-stage pipeline:

1. **Encoding:** Converts local binary image files into `Base64` strings.
2. **Staged Upload:** Requests a temporary upload URL from Shopify.
3. **Association:** Once uploaded, the resulting resource ID is linked to the Product or Variant.

Technical Stack

- **Language:** Python 3
- **Data Processing:** Pandas (for efficient grouping and cleaning)
- **API Transport:** Requests (HTTP)
- **Interface:** Shopify Admin GraphQL API

File Structure

- `main.py`: Contains the core logic and GraphQL definitions.
- `credentials.py`: Stores sensitive API keys (excluded from version control).
- `products.xlsx`: The source of truth for product data.

NAME:AMAN GUPTA

ADMISSON NO.:21JE0083