Scout Backend Take Home Project Write-up

This project was fairly straightforward. This document will cover some of my design decisions. If you have any further questions about how I made specific decisions please let me know. Additionally, I would love any feedback on the quality of my code and anywhere you see an area for improvements.

Micro-Service:

This layer of the application utilizes Express and Redis. I broke the code into a few folders, including routes and services. In order to call our lambda function, I utilized the aws-sdk.

Redis Setup:

This was my first time working with Redis. This program assumes that Redis is installed in the environment in which the application is running. Redis client connection is established in db.js which is contained in the services folder. This utilizes a custom Redis library which is promise-based. Below you will find a table describing the three different types of key-value pairs in the cache.

Key	Datatype	Example
batch:id	Set	batch:1 = [item:1, item:2]
item:id	Hashmap	item:1 = { “status”:”SUCCESS”, ”Message”: “Item successfully listed on Stockx.” }
next_batch_id	Integer	Next_batch_id = 2

Each batch is associated with a set of item ids. This allows us to avoid listing an item with the same id multiple times, as we are able to call lambda conditionally based on the response of our attempt to add each item: id to the set.
I had considered structuring the Redis to instead only contain keys corresponding to batches, and having the value be a dictionary of dictionaries corresponding to each item. I decided to not go with this implementation as I wanted to be able to access each item based on its id, in case we later decided to add an endpoint to poll the status of an individual item.

Lambda Function:

This layer of the application utilizes an AWS lambda function to list passed-in items onto StockX. The code is fairly straightforward.

Deployment Script:

For the deployment script, I created a simple bash script. This script utilizes the AWS CLI. The role utilized by the script is hardcoded, and the script assumes that the AWS CLI has already been configured. On execution, the script deletes the current version of the lambda function, if it exists, and replaces it with a new version based on the current source code in the lambda directory.

Potential Improvements:

There are a number of places I feel I could have improved upon this project given more time. Below I have listed a few potential improvements.

- Addition of Docker. This would be especially useful for Redis.
- Further considerations of User Context:
    - **Further Error Handling and In-depth error messages:**
      Check for empty batches, check for negative or 0 price input for listings.
    - **Avoiding Duplicate Item Listings across batches:**
      Addition of checking for common item ids between batches to avoid duplicate listings. This could be achieved through the creation of another value-> key set which includes a set of all items contained in the cache. We can refer to this set when generating new batches in order to avoid duplication. If the item is still in progress for another batch, we can have the program wait for the status to change to success or failure, and act accordingly.
    - **Error Correction:**
      In the case that one of the listenings raises an error that has an actionable correction, we can prompt the correction of the specific fields which are raising errors based on our response from StockX.