ElasticSearch Domain Master User

Username: ccbdp

Password: MichaelIsTheBest1!

Running Notes:

1. Uniform primary key (ex. UPC) in NoSQL
2. Move LF1 to a lambda, possibly ditch selenium
3. Scraping on a timer, is manufacturer based (scraping is hard-coded for each manufacturer’s individual site)
4. (Excuse): Initially we will be scraping from the manufacturer. Later when established, we will ask manufacturers to provide an API to their inventory DB.
5. Scrape Rep & Titan old stock bot
6. Needs crawling functionality to scrape sites to get all of the URLs of the products
7. Add specs for the products
8. Add user pool, possibly cognito
9. Add frontend feature to let users sign up for product updated (either email or text)
   * User: account name, email address and/or phone number, product\_name, notification\_frequency (number of hours) and Last\_Notification\_Sent (Date/Time)
   * if current\_time > Last\_Notification\_Sent + notification\_frequency then send notification
   * Contact user if item is in stock (if stock is True)
   * Ask user how many times they would like to be notified, and the frequency (1 per day, 1 per 3 day, 1 per week)
10. Trigger when stock is true that sends a notification to a user after every scrape. Method to send only once, possibly remove record once notified
11. Scrape every morning to update our inventory.
12. Way to display items they have currently requested to be notified for, and ability to remove those items
13. ElasticSearch:
    * Just stores the product name of the items in stock
    * Wipe the index when scraping and just add product names that are in stock
    * Look to user table and create an array in the ES index so you have an array of subscribers, store as JSON
14. EC2 to Lambda:
    * File in s3
    * S3 trigger lambda on insert
    * When ec2 function finishes add file to the s3 bucket, thus triggering the lambda
    * The lambda will then use SES and SNS to text/email
15. Efficiency of Scraping
    * Split it into three EC2 instances where each instance handles one manufacturer
    * “We are scraping constantly with 3 instances for efficiency and response time”

Workflows:

1. Scrape manufacturers to find which products are in stock
   1. One EC2 instance process for each manufacturer
   2. Process crawls manufacturer website and identifies if each product is in stock.
   3. If in stock, store in list
   4. Update DynamoDB status for all the products of that manufacturer.
   5. At end of scrape, create a file in S3 bucket with list of products in stock.
   6. S3 bucket insert triggers lambda (LF5.py).
   7. Lambda pulls filename from event and processes list of products in stock.
   8. For each product, look up in ES list of users to notify.
   9. Build text message and email.
   10. For each user, if current\_time > user.Last\_Notification\_Sent + user.notification\_frequency then send notification(s) depending on if user.email and user.phone are populated.
2. Create list of products in dynamodb and ES (one time manual? or separate workflow)
   1. ElasticSearch:
      1. Index on the manufacturer\_product name of all products
      2. Look to user table and create an array for the ES product index so you have an array of subscribers, store as JSON
   2. DynamoDB Products table
      1. One time manual insertion of all products with primary key Manufacturer + Product Name - TBD to automate
      2. InStock flag to update with every scrape and provide status to lambda for front-end queries.
      3. Product Schema:
         1. ID (string, Manufacturer + Product Name, PK)
         2. Manufacturer (string)
         3. Product (string)
         4. Type (string)
         5. URL (string)
         6. InStock (bool)
         7. LastUpdate (Date/Time)
3. Users subscribe to get notifications for products
   1. ElasticSearch
      1. Create domain of user subscriptions to products
      2. For each manufacturer\_product name add an array of users interested in product
      3. Domain 'subscription' schema:
         1. Product (string, Manufacturer + Product Name)
         2. Users (string array, UserName)
   2. DynamoDB
      1. Create table of user information.
      2. User schema:
         1. UserName (string, PK)
         2. Email (string)
         3. Phone (string)
         4. Product (string, Manufacturer + Product Name, PK of Product table)
         5. NotificationFrequency (int, number of hours)
         6. LastNotificationSent (Date/Time)
      3. Need to authenticate and identify users in the future with cognito
   3. Process
      1. (LF7)When user subscribes to be notified for a product, user record is created through API Gateway by Lambda into DynamoDB User table. One entry per subscription.
         1. Method subscribe()/POST Command format:

{

\*“UserName”: ,

\*“ProductID”: ,

“Email”: ,

“Phone”: ,

“Frequency”(in hours): ,

“Subscribe”: (True = insert, False = remove\*)

}

* + 1. Also, ES is updated for product to include user in array.

1. Users unsubscribe to get notifications for products
   1. Users see on front-end list of products they are subscribed to. They can select to be removed from those notifications.
   2. API Gateway to Lambda to remove user from ES user array for product and to remove record from DynamoDB.
2. Update front-end with new inventory state constantly
   1. (LF3) API Gateway to Lambda to query which items are in stock, build page response to display to user.

Format:

{

“SearchType”: Type/Manufacturer,

“SearchKey”: (BARBELL, DUMBELL, ROGUE, TITAN, etc.)

“SearchScope”: InStock/All,

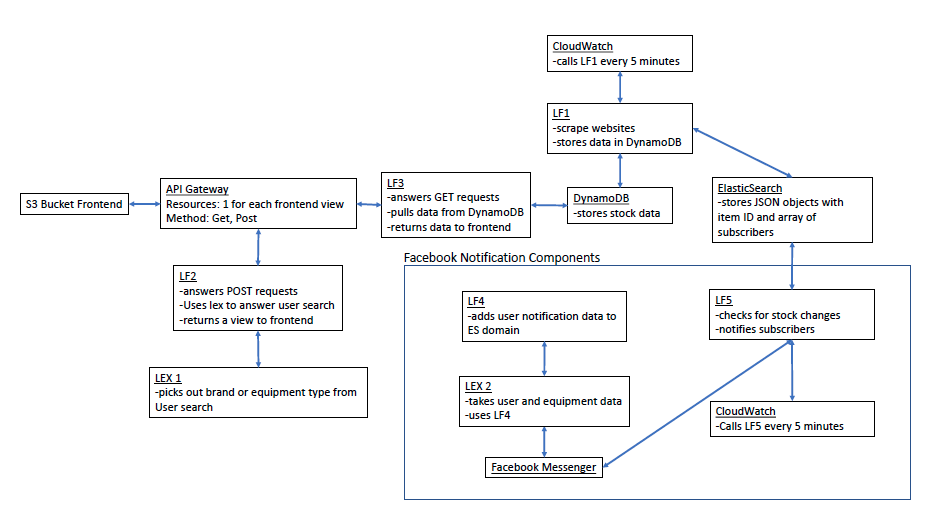
“UserName”: (“Empty” if null)

}

* + 1. User can view products by Type within two lists
       1. In-stock products by popularity
       2. All products ranked by popularity
          1. Action to enter user info
          2. If user info has been entered then display Subscribe/Unsubscribe button next to each product
    2. User can view products by Manufacturer
       1. In-stock products by popularity
       2. All products ranked by popularity
          1. Action to enter user info
          2. If user info has been entered then display Subscribe/Unsubscribe button next to each product
    3. For each list, show a button to the right of each item to Subscribe to be notified when is stock or Unsubscribe
    4. API Gateway should return list of products for that type or manufacturer, in-stock or all, with flag to identify if user is already subscribed to the product. Method getStock() should filterBy Manufacturer/Type, filterBy InStock/All, OrderBy NumberOfSubscribers Desc and return product record with subscribed flag
  1. **(LF6)API Gateway** to Lambda to query for list of Product Types to display on home page for Product search: method getList("Type") returns JSON sorted list of Type and URL for Active=True
     1. Body format: {“list”: “Type/Manufacturer”}
     2. Type schema
        1. Type (string)
        2. Active (string)
        3. URL (string, link to image)
        4. LastUpdate (Date/Time)
  2. **(LF6)API Gateway** to Lambda to query for list of Manufacturers to display on home page for Product search: method getList("Manufacturer") returns JSON sorted list of Manufacturers and URL for Active=True
     1. Manufacturer schema
        1. Manufacturer (string, PK)
        2. Active (string)
        3. URL (string, link to image logo)
        4. LastUpdate (Date/Time)

Us-west-2 Oregon

3/15/2021: Architecture Diagram



2/20/2012: Clickable prototype

<https://projects.invisionapp.com/prototype/ckldyfyui00j51l019t8ur744/play>

Currently only the Barbell categorie, and the Rogue Fitness company buttons are clickable. They show my proposed layout of each of the different page types.

Andrew Brigante

Columbia COMS: E6998 Project Proposal

**Gym Equipment In-stock Application**

**Overview:**

During Covid-19 and the foreseeable future, the demand for at-home gym equipment has skyrocketed. Just trying to order a few pieces of equipment so an individual can workout at home can be an impossible task. I would like to make an application/bot that would actively scrape popular fitness equipment websites and maintain a database of what is current in-stock. Users would access this application from a website, or Facebook chat, where they could search for in-stock/available equipment. The user could then create an account and setup notifications for when an item they are looking for goes in stock. Also, the user could search for the stock history of the products, thus being able to see the frequency of when the items are available.

**Backend Components (Dynamic/In-progresss):**

Python Lambda Function:

A python function that would scrape popular fitness websites to collect the data.

DyanamoDB:

A NoSQL database would be used to store all of the information collected from screen scraping.

Trigger:

This would call the python function and update the database with the current stock.

Cognito/User Database:

Since the users can sign up for stock alerts, the application would need to use cognito to save user information.

Notification system:

A notification system would need to be created that would monitor stock updates and notify users when something comes in-stock.