Gym Stock Bot

COMS E6998 Cloud Computing and Big Data Project

Andrew Brigante AB5184
Michael Makris MM3443

April 27, 2021

YouTube Link: https://youtu.be/SNbgaIE9LNk

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-progress):

Python Lambda Function:

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

DynamoDB:

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.

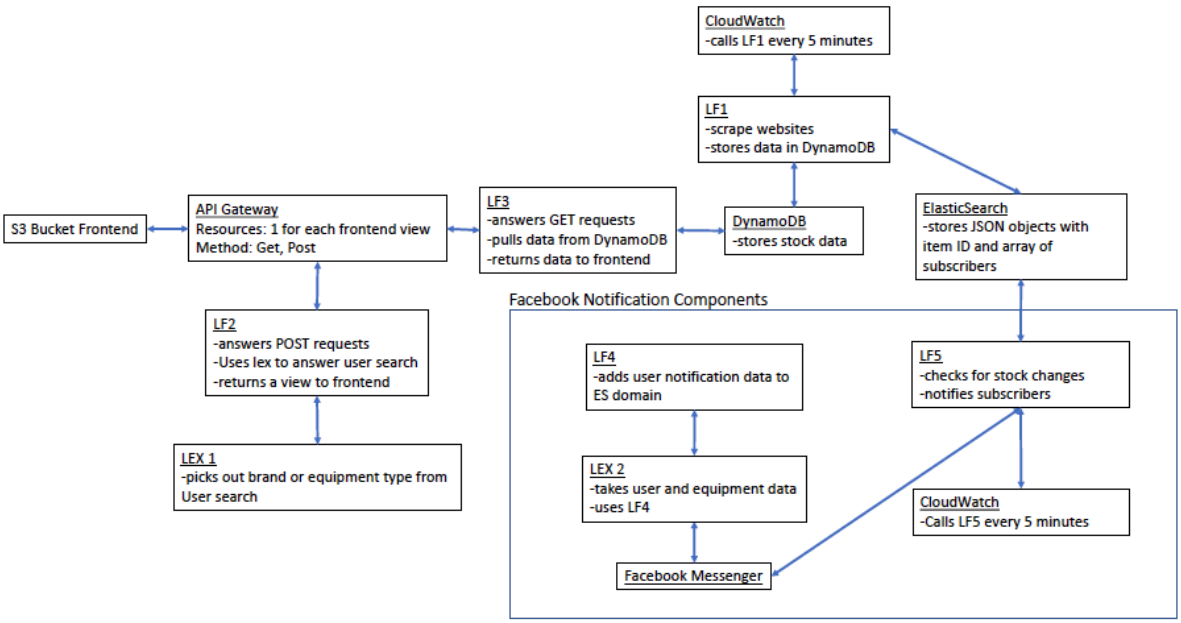
2/20/2012: Clickable prototype

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

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

Andrew Brigante

3/15/2021: Architecture Diagram



Gym Stock Bot

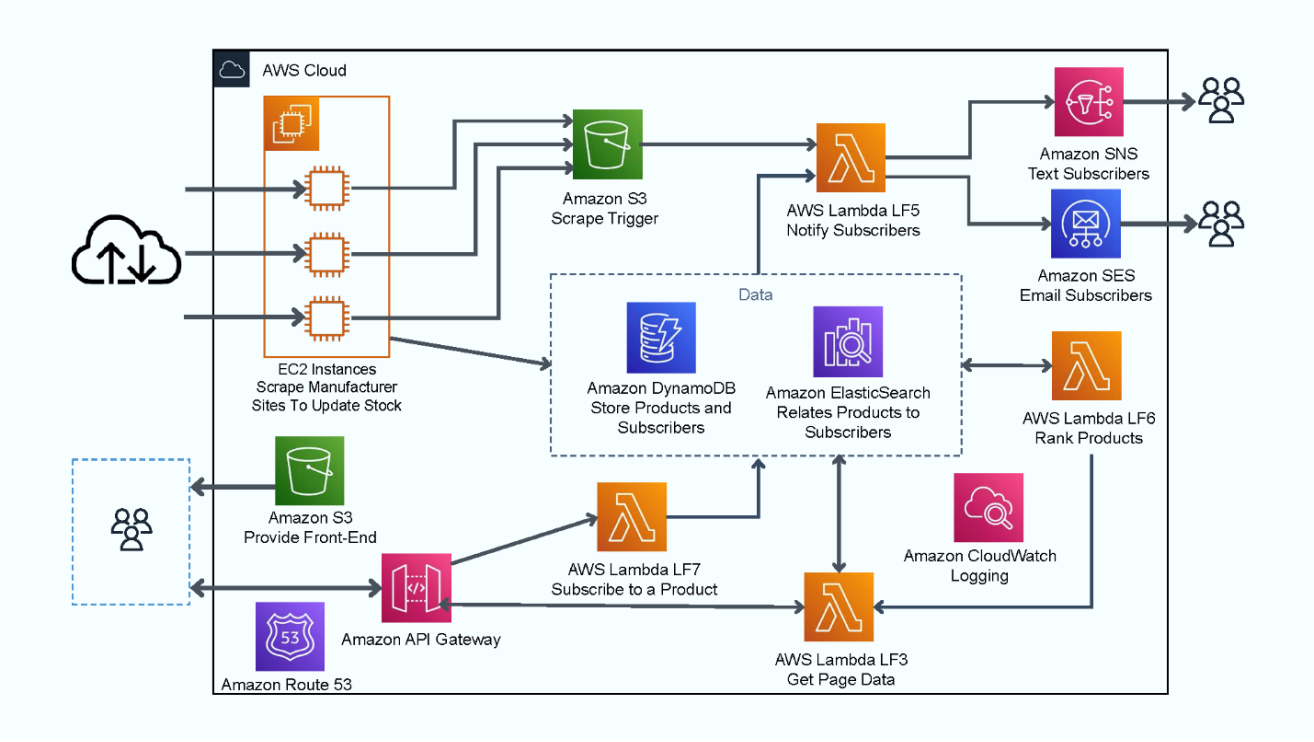
Need

During the COVID-19 pandemic and the foreseeable future, the demand for gym equipment has skyrocketed. Just trying to purchase a few pieces of gym equipment requires lots of time and effort to find in stock.

Solution

We came up with a simple site to let users discover gym items in-stock. Our engine scrapes known manufacturer sites regularly to update our inventory list. Users can visit our site to see what is available across manufacturers and categories of products. If items are not in stock, our users can subscribe to receive notifications when there are more quantities available. In addition, we present products ranked by popularity to show what other users are interested in.

Architecture



Workflows and supporting schema

1. Scrape manufacturers to find which products are in stock
    a. One EC2 instance process for each manufacturer
    b. Process crawls manufacturer website and identifies if each product is in stock.
    c. If in stock, store in list
    d. Update DynamoDB status for all the products of that manufacturer.
    e. At end of scrape, create a file in S3 bucket with list of products in stock.
    f. S3 bucket insert triggers lambda (LF5.py).
    g. Lambda pulls filename from event and processes list of products in stock.

h. For each product, look up in ES list of users to notify.
i. Build text message and email.
j. 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)
    a. ElasticSearch:
        i. Index on the manufacturer_product name of all products
        ii. Look to user table and create an array for the ES product index so you have an array of subscribers, store as JSON
    b. DynamoDB Products table
        i. One time manual insertion of all products with primary key Manufacturer + Product Name - TBD to automate
        ii. InStock flag to update with every scrape and provide status to lambda for front-end queries.
        iii. 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
    a. ElasticSearch
        i. Create domain of user subscriptions to products
        ii. For each manufacturer_product name add an array of users interested in product
        iii. Domain 'subscription' schema:
            1. Product (string, Manufacturer + Product Name)
            2. Users (string array, UserName)
    b. DynamoDB
        i. Create table of user information.
        ii. 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)
        iii. Need to authenticate and identify users in the future with cognito
    c. Process

i. (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)
    }
ii. Also, ES is updated for product to include user in array.

4. Users unsubscribe to get notifications for products
    a. Users see on front-end list of products they are subscribed to. They can select to be removed from those notifications.
    b. API Gateway to Lambda to remove user from ES user array for product and to remove record from DynamoDB.
5. Update front-end with new inventory state constantly
    a. (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)
}

i. User can view products by Type within two lists
    1. In-stock products by popularity
    2. All products ranked by popularity
        a. Action to enter user info
        b. If user info has been entered then display Subscribe/Unsubscribe button next to each product
ii. User can view products by Manufacturer
    1. In-stock products by popularity
    2. All products ranked by popularity
        a. Action to enter user info
        b. If user info has been entered then display Subscribe/Unsubscribe button next to each product
iii. For each list, show a button to the right of each item to Subscribe to be notified when is stock or Unsubscribe

iv. 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

b. (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
    i. Body format: {“list”: “Type/Manufacturer”}
    ii. Type schema
        1. Type (string)
        2. Active (string)
        3. URL (string, link to image)
        4. LastUpdate (Date/Time)

c. (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
    i. Manufacturer schema
        1. Manufacturer (string, PK)
        2. Active (string)
        3. URL (string, link to image logo)
        4. LastUpdate (Date/Time)

Results

- We assigned one EC2 worker per manufacturer to speed up the scraping process and provide updates faster to users.
- The front-end is simple and intuitive, to provide users what they need. We provide two lists:
    - In-stock products so that users can jump and grab the items that are available.
    - All products for users to identify their options and look for items as they come in stock.
    - For unavailable items, users can subscribe to get notified when they become available.

Future enhancements

- In the future we plan to work with manufacturers to use APIs instead and have them push the update when they update their stock. It will also allow for the addition of new products instead of us semi-manually processing their catalogs.
- As we add more manufacturers, we will need to introduce a messaging process to handle the bigger inbound data pipeline.
- We currently do not formally maintain user accounts because there was no need to annoy users with a login process. If users want to receive notifications, then we capture their email or phone within the session. In the future we could a Cognito well-known source authentication process, if we decide more involved services like a shopping cart.