Intro: Side Project. Use AWS CDK in TypeScript, Lambda code and web scraping in Python, and UI tooling in Java to webscrape Amazon and allow caching and tracking of Amazon pricing information

Project Structure

Main Internal APIs

void updateProductPrices()

- Also might just be a lambda
- updates products tracked by DynamoDB#1 if last updated over an hour ago
- Python script that webscrapes Amazon with Selenium/BeautifulSoup

void sendScheduledMessage()

- Might be just a Lambda
- Scheduled to run weekly
- Checks DynamoDB#2 and sends out emails to all users attached to SQS

Main External APIs

void updateUserDB(username=, password=, *add_track=, *remove_track=, *add_email=, *remove_email=)

- Update DynamoDB#2 for added and removed tracking links and emails
    - If emails updated, update SQS subscriptions
- Add new username/password hash to DynamoDB#2 if needed

table getTrackedPrices(username=, password=)

- Pull info from DynamoDB#2 for what is being tracked
- Pull info for prices from DynamoDB#1
    - If anything tracked not refreshed recently enough within hour, calls updateProductPrices()

AWS Infrastructure

- *AWS/DynamoDB*
    - DynamoDB #1
        - Only need to update if last updated over hour ago
        - Master Product Tracking Database
            - Key isProduct Link
            - Product Name
            - Original Price
            - Current Price

        - Historic Low
        - Last Updated (Web Cache)
    - DynamoDB #2
        - User Product Tracking Database
            - Key = Hash(Hash(User) | Hash(Password))
            - Links User Is Tracking
            - Emails To Send Tracked Info To (Also keep encrypted in table)
- *AWS/SES*
    - Email Ability
- *AWS/IAM-API Gateway*
    - APIs
    - Role to call external APIs via HTTP Queries
    - Role to call all internal APIs, limited perms
- *AWS/Lambda*
    - APIs
    - Scheduled emails
- *AWS/S3-CloudFormation-CDK*
    - Setup/Bootstrap for all the AWS infrastructure in TypeScript
- *AWS/CloudWatch*
    - Logs for MonitoringStack

User Interface - Java

- Java JFrame for UI
    - Allow Selecting Products To Track via URL
    - Allow Deselecting Products via Entering Just A Number
    - Allow Email Selecting/Deselecting
    - Pricing Information On Selected Products
- Login Information input to cache what we're tracking into DynamoDB #2 Upon Program Start and Program End
    - Parameters On Program Start
        - Will Ask Users If They Want To Save Info To Password File
    - Password File If Available
        - Will Ask Users If They Want To Use Info From There
- Save DynamoDB #1 calls for app version in Local Cache
    - Only need to update with DynamoDB if last updated over hour ago

    - Local Cache
        - Product Link
        - Product Name
        - Original Price
        - Current Price
        - Historic Low
        - Last Updated
- Download DynamoDBs Internal Function callable with parameter for program

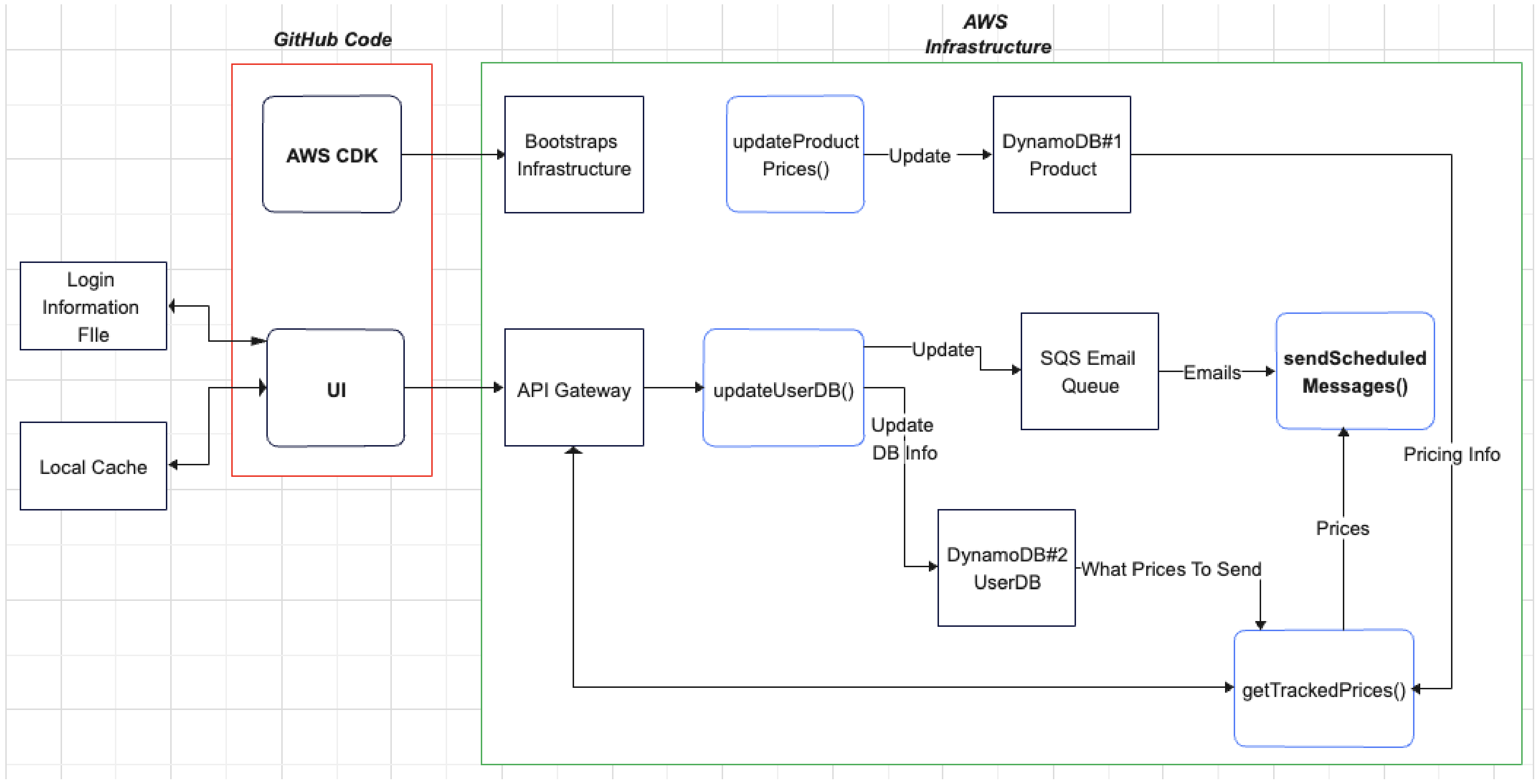
Testing

- TestNG and Spring Framework
- Proper Unit Tests for UI, Local Cache Functionality
- Proper Integration Tests for all APIs

Building

- Use Maven to build Java/CDK apps

Architecture Diagrams



Bolded are architecture starting points. Blue are APIs/Lambdas.

Timeline

1) Create API Plan ✅
2) Create Architecture Diagrams ✅
3) Web Scraping Script for Amazon Python **Oct 24**
   a) https://www.digitalocean.com/community/tutorials/scrape-amazon-product-information-beautiful-soup
      i) Good Resource For Reference
4) Create AWS Resources (IAM, S3, CloudFormation, CloudWatch, Lambda, DynamoDB, SQS, API Gateway) **Nov 10**
   a) Get APIs Running
5) Create GitHub Repo
6) Define AWS Resources with AWS CDK - Typescript **Dec 10**
7) Create UI to interact with APIs - Java **Dec 31**

a) Unit Tests and Integration Tests