Group 28: Findme Recipe

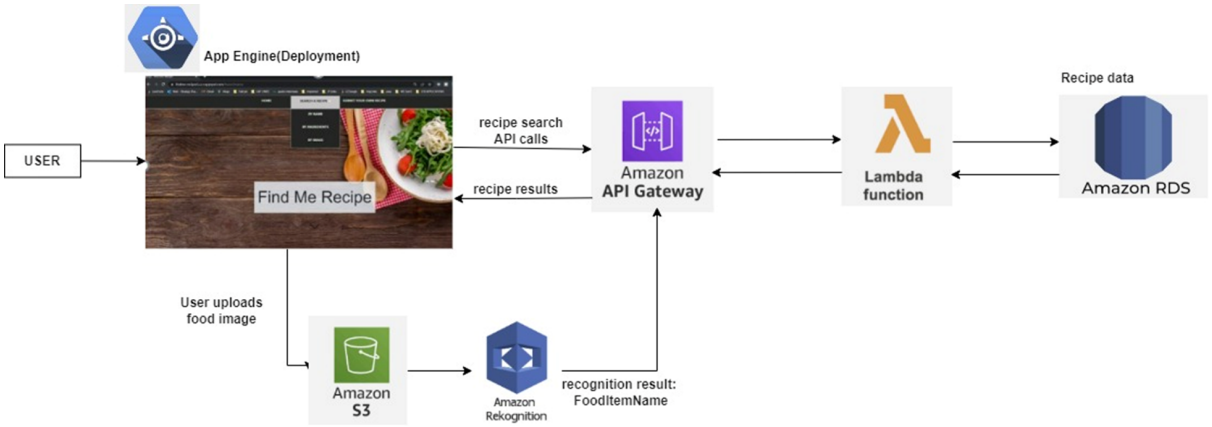
Application Overview:
Application addressed issues to find quick recipes based on available ingredients, or filter food items based on name. Example if a user wants to make Sandwich our app returns recipes for different types of sandwiches.
FindMe Recipe is a web application to find recipes based:

- Food item name
- Ingredients available

Application also detects food items using images and allows users to contribute recipes to the dataset.

Components used:
Compute service : App Engine (Platform as a service)
Data persistence layer: AWS RDS (relational database)
Serverless: Lambda Functions

Architecture:

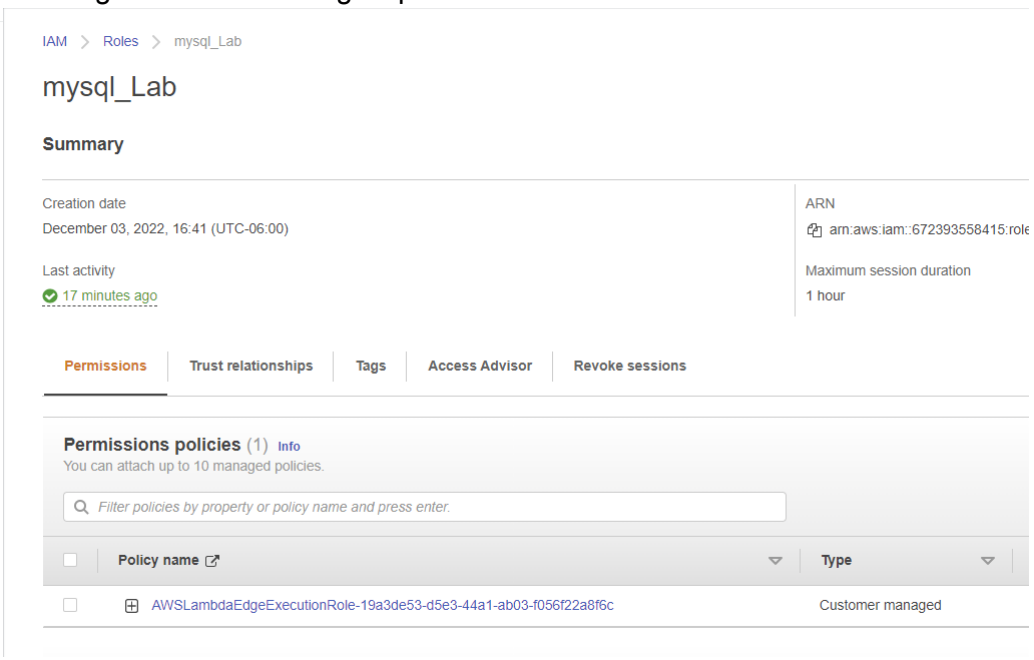


Description:
User interacts with application hosted on App Engine using external API.
When a user performs search based on foodItemName or Ingredients, it triggers a restful call to API Gateway, which inturn triggers a lambda function. Entire backend logic is in LambdaFunction which calls the RDS database. It fetches results as per their search query
When user uploads an image to know the recipe, the image gets uploaded to S3 bucket. Using Amazon rekognition to identify the image, json response is returned , our application fetches fooditemname from the response and forwards the call to API gateway by passing fooditemname as query search parameter. This calls the appropriate lambda function and fetches the food item recipe of the uploaded image and displays it on screen.

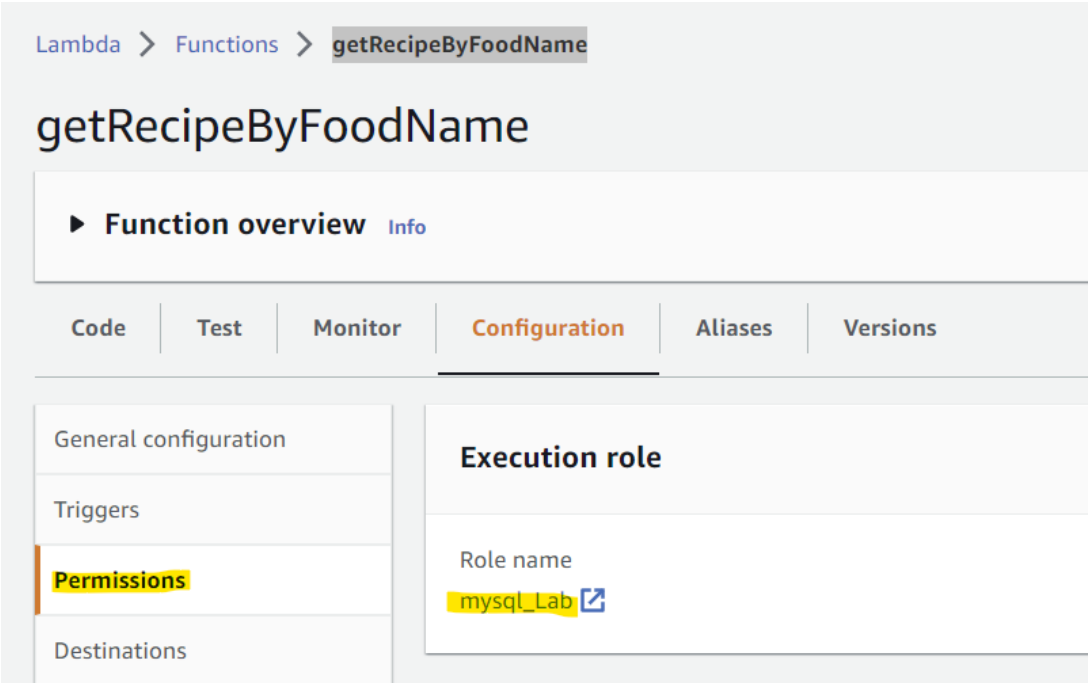
Flask application:
Created a flask application which uses API endpoints created using API Gateway which fetches data from aws RDS using lambda service.

Serverless: Lambda Functions

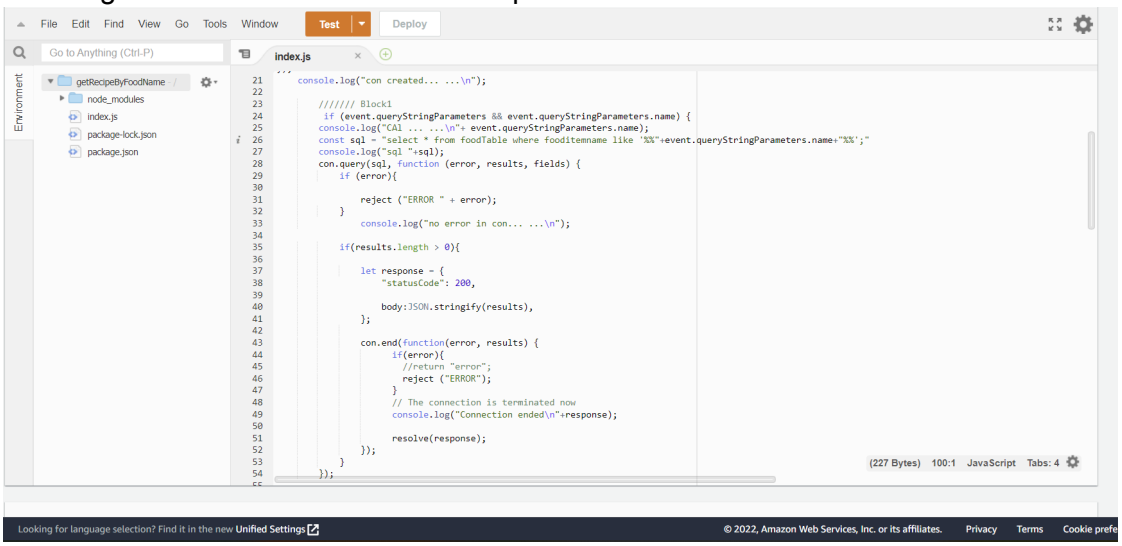
Lambda function named "**getRecipeByFoodName**"
Steps:
1. Configure IAM role and give permission for lambda execution



2.Under permission configure role:



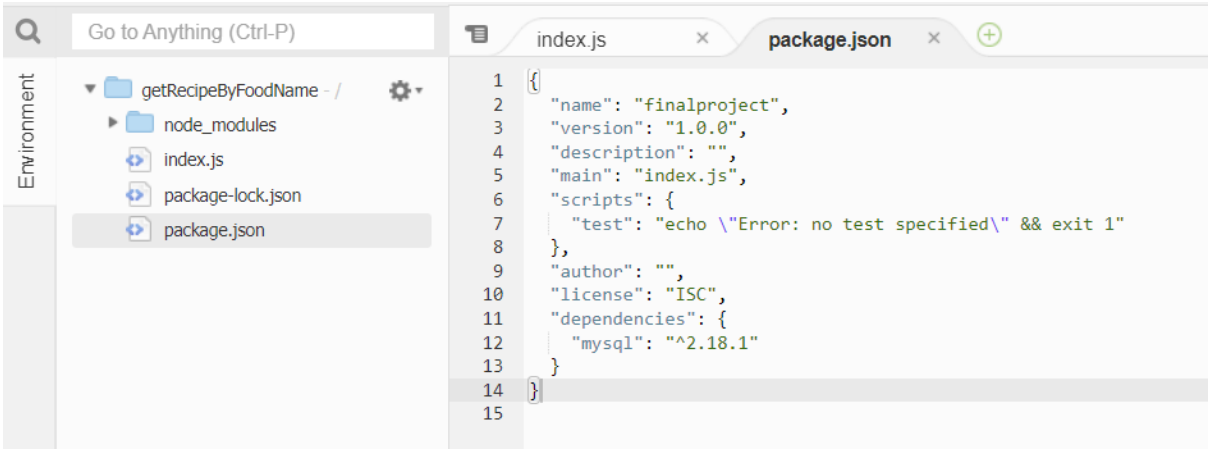
3. Configure source code for backend api



Sample code to search food item by name:

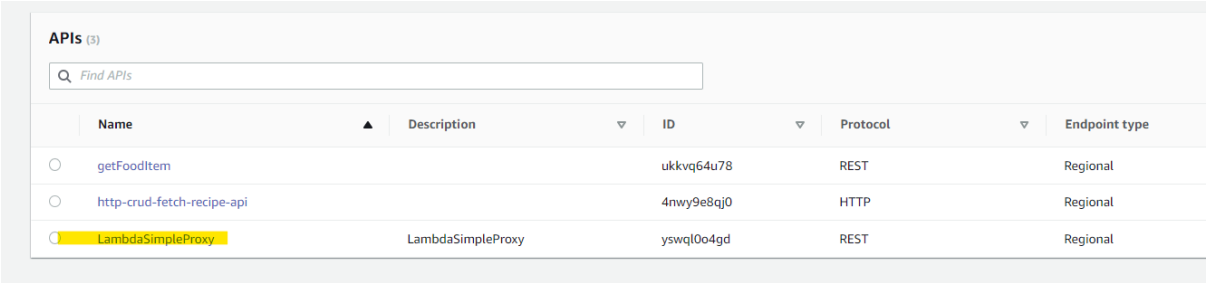
```
const sql = "select * from foodTable where fooditemname like '%%"+event.queryStringParameters.name+"%%';"
        console.log("sql "+sql);
        con.query(sql, function (error, results, fields) {
            if (error){
            reject ("ERROR " + error);
            }
            console.log("no error in con... ...\n");
            if(results.length > 0){
                let response = {
                  "statusCode": 200,
                  body:JSON.stringify(results),
                   };
                con.end(function(error, results) {
                        if(error){
                          //return "error";
                          reject ("ERROR");
                        }
                        console.log("Connection ended\n"+response);
                        resolve(response);
                });
            }
        });
}
```

Package.json: add project dependencies

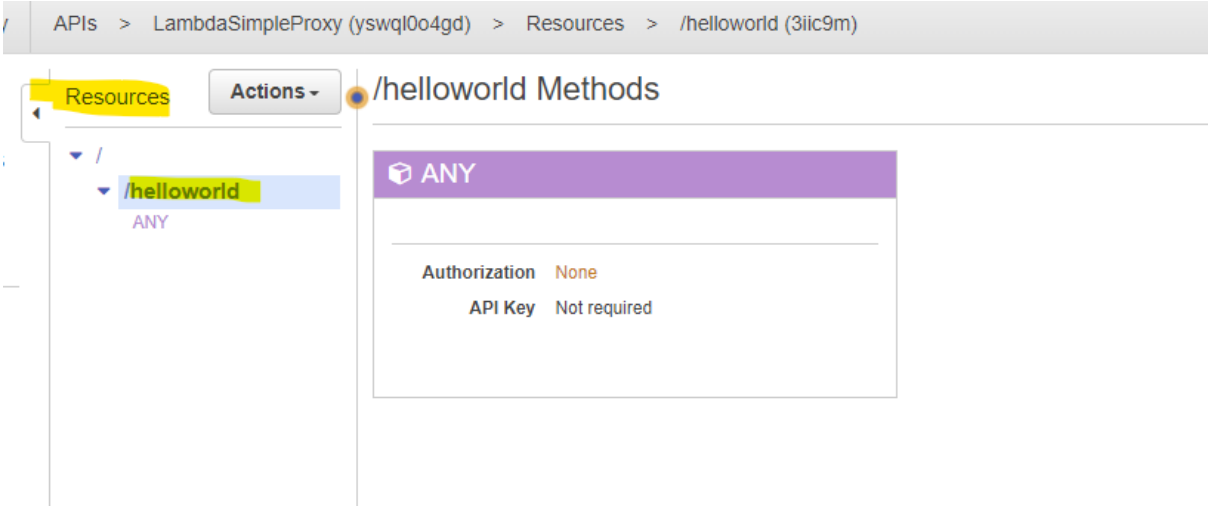


4. Setup API Gateway using RESI API

For this project we created a REST API:



Under API, created a resource named helloworld:



Path is configured as: /helloworld

Under resource create method as ANY: ANY allows single API method setup for GET POST etc.

Method is integrated with lambda function:

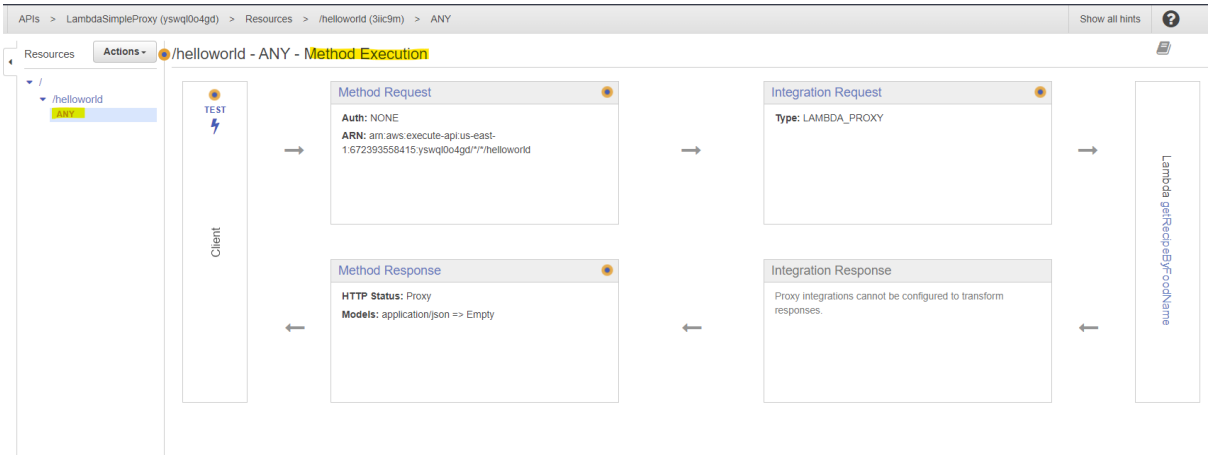
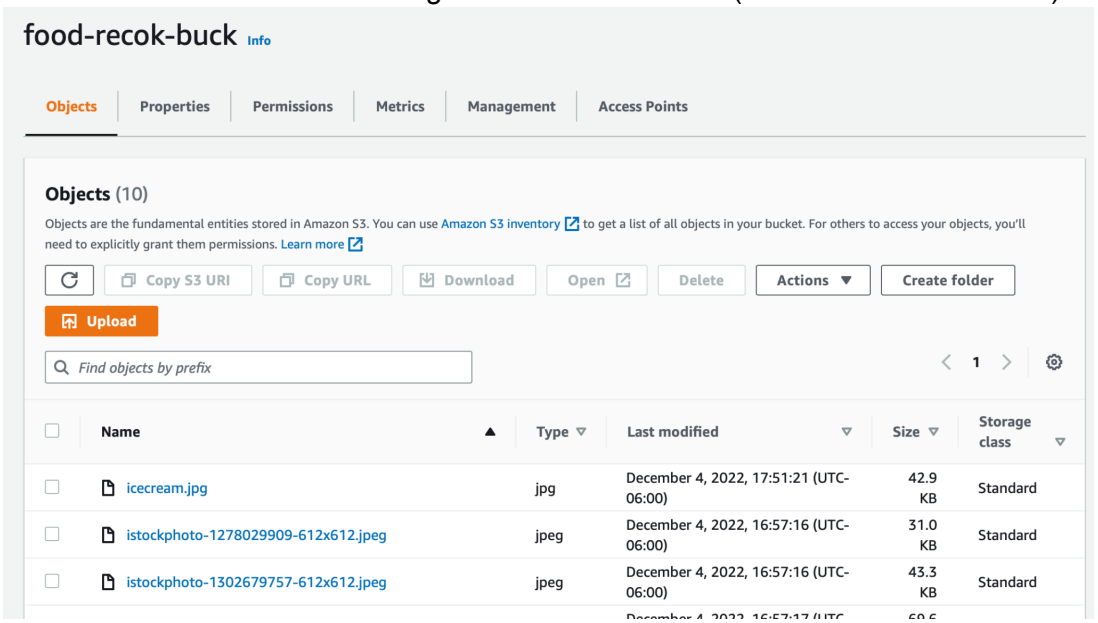


Image Recognition using AWS Rekognition:

It is a tool used for image analysis.
It detects objects, scenes, activities, landmarks, faces, dominant colours and image quality.
Here we are using the tool for analysing the image uploaded. The user is given a facility to upload an image and then the application analyses the food from the image and gives the search results for the recipe related to that image.
We are using the data and API provided by AWS.
The boto3 script is used to store the uploaded image in S3 bucket on amazon and to analyse the image using Rekognition.

Below screenshot shows the images stored in S3 bucket(name : food-recok-buck).

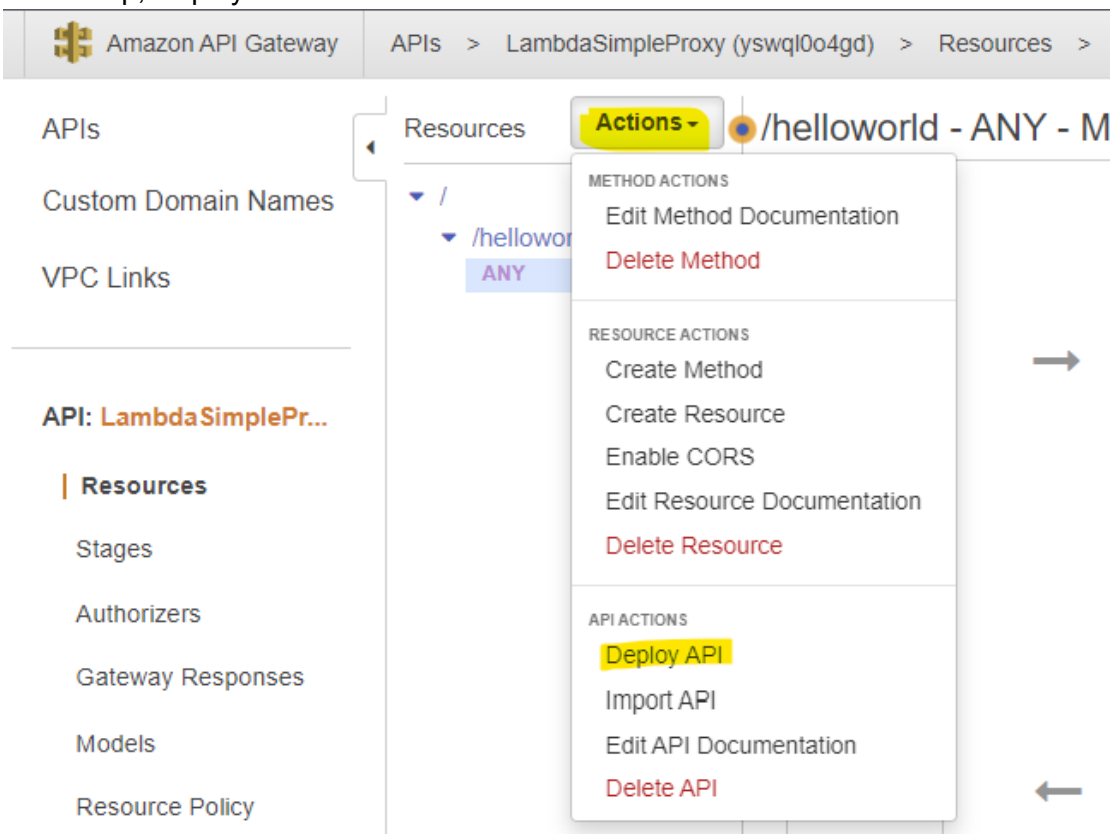



```
import csv
import boto3
import json
import pandas as pd

with open('new_user_credentials-2.csv','r') as input:
    next(input)
    reader=csv.reader(input)
    for line in reader:
        access_key_id=line[2]
        secret_access_key=line[3]


s3 = boto3.resource('s3',region_name='us-east-1',aws_access_key_id=access_key_id,aws_secret_access_key=secret_access_key
BUCKET = "food-recok-buck"


photo = 'pizza-rustic-italian-mozzarella-cheese-basil-leaves-35669930.jpg'
fp="foodimages/"+photo



client = boto3.client('rekognition',region_name='us-east-1',aws_access_key_id=access_key_id,aws_secret_access_key=secret

response= client.detect_labels(Image={
        'S3Object': {
            'Bucket': 'food-recok-buck',
            'Name': photo
        }},MaxLabels=10,MinConfidence=95)

jsonResp = json.dumps(response['Labels'],indent = 4)
print(jsonResp)
```

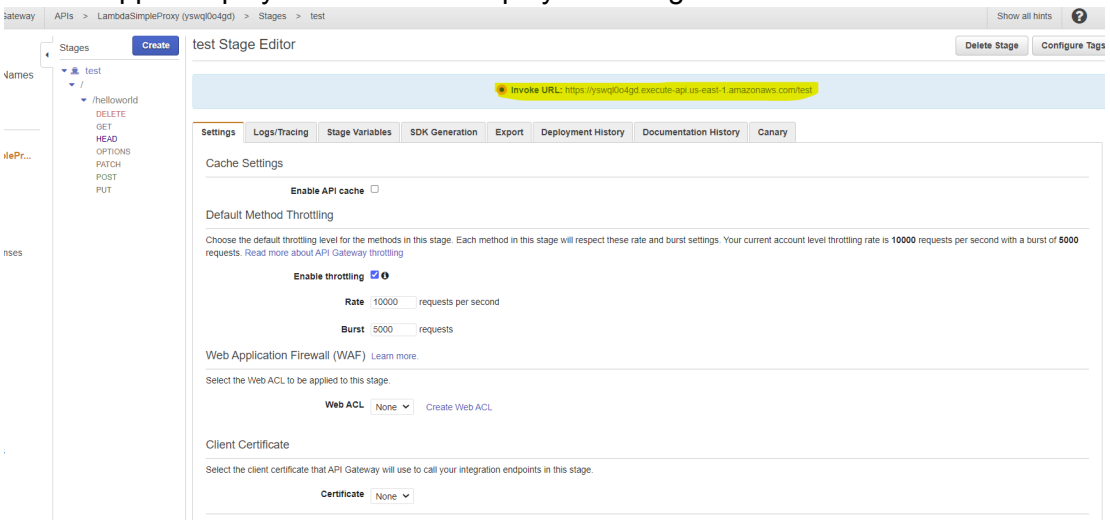
The result obtained from the above code is in the JSON format. We filter the results with confidence greater than 95 , parents with name food and categories with name Food and Beverage.


```
},
{
    "Name": "Pizza",
    "Confidence": 99.98148345947266,
    "Instances": [
        {
            "BoundingBox": {
                "Width": 0.8309263586997986,
                "Height": 0.9569106101989746,
                "Left": 0.07866659015417099,
                "Top": 0.009478035382926464
            },
            "Confidence": 99.79634094238281
        }
    ],
    "Parents": [
        {
            "Name": "Food"
        }
    ],
    "Aliases": [],
    "Categories": [
        {
            "Name": "Food and Beverage"
```


Post setup, Deploy API from actions menu:

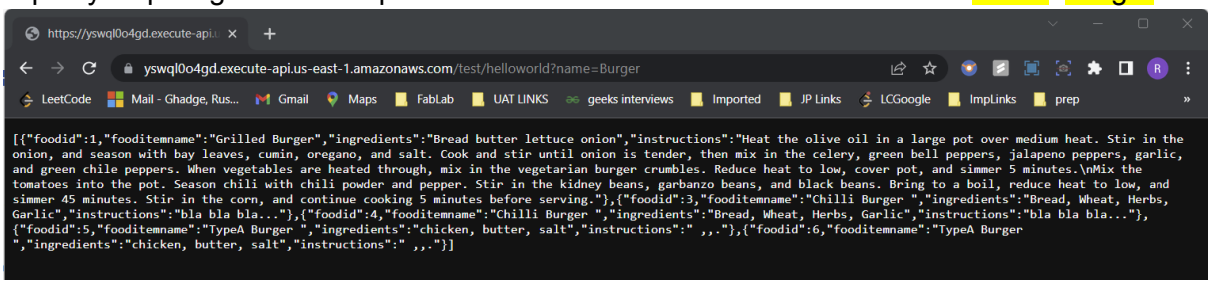


Once a app is deployed it created a deployment stage and returns a URL:



Testing URL Endpoint: Returns JSON

https://yswql0o4gd.execute-api.us-east-1.amazonaws.com/test/helloworld?**name**=Burger

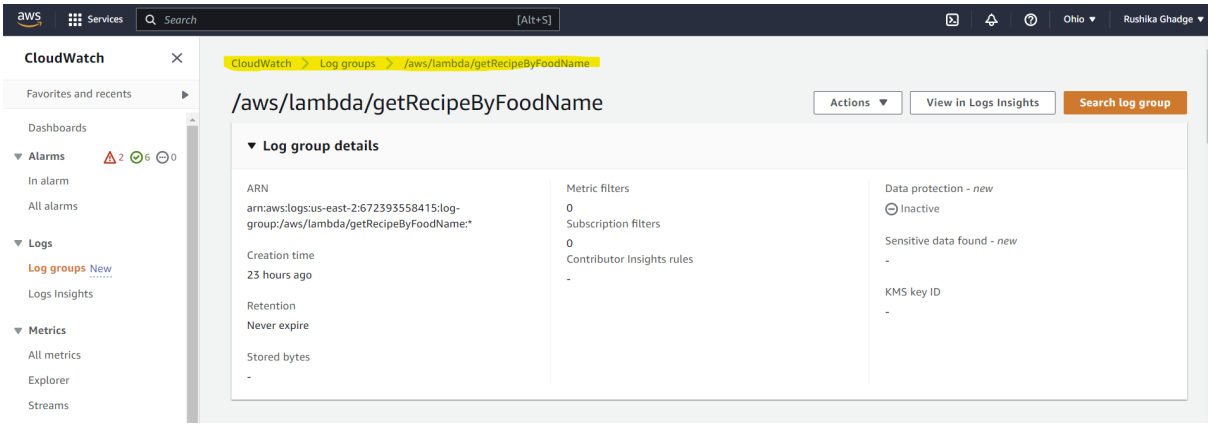


Same endpoint is used to search food items by ingredients, only query parameter changes

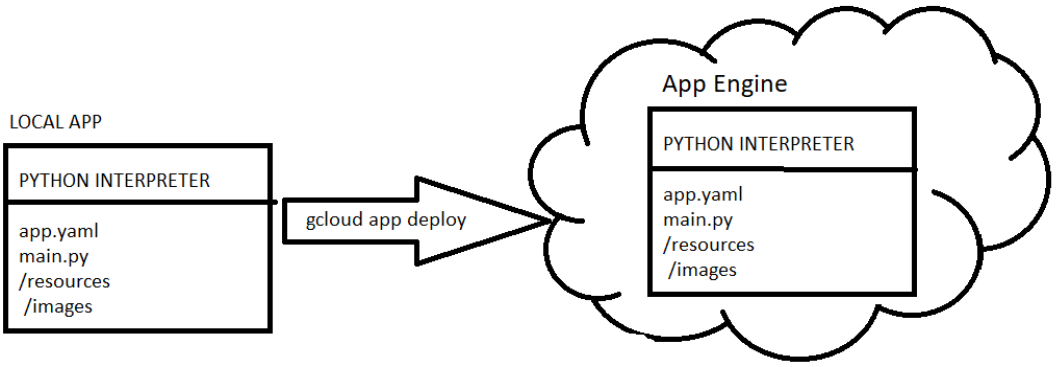
https://yswql0o4gd.execute-api.us-east-1.amazonaws.com/test/helloworld?ingredients=tomato%7Cchicken%7CPotato

Logs:

For logging all the logs are monitored using CloudWatch



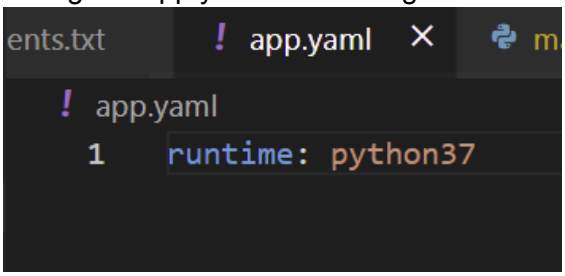
Deployment steps: App Engine



Setup steps:

- Install gcloud cli
- Run: `gcloud components install app-engine-python`

Configure app.yaml : this config file informs app engine about python runtime



Create project: in gcloud
gcloud projects create findme-recipe --set-as-default

Enable billing and cloud build API for created project:

```
C:\Windows\system32>gcloud beta billing projects link findme-recipe  --billing-account=0109DD-1104E7-37061D
billingAccountName: billingAccounts/0109DD-1104E7-37061D
billingEnabled: true
name: projects/findme-recipe/billingInfo
projectId: findme-recipe

C:\Windows\system32>gcloud services enable cloudbuild.googleapis.com
Operation "operations/acf.p2-64064644820-de223d30-08f4-4747-a5fb-0d10257ff25b" finished successfully.
```

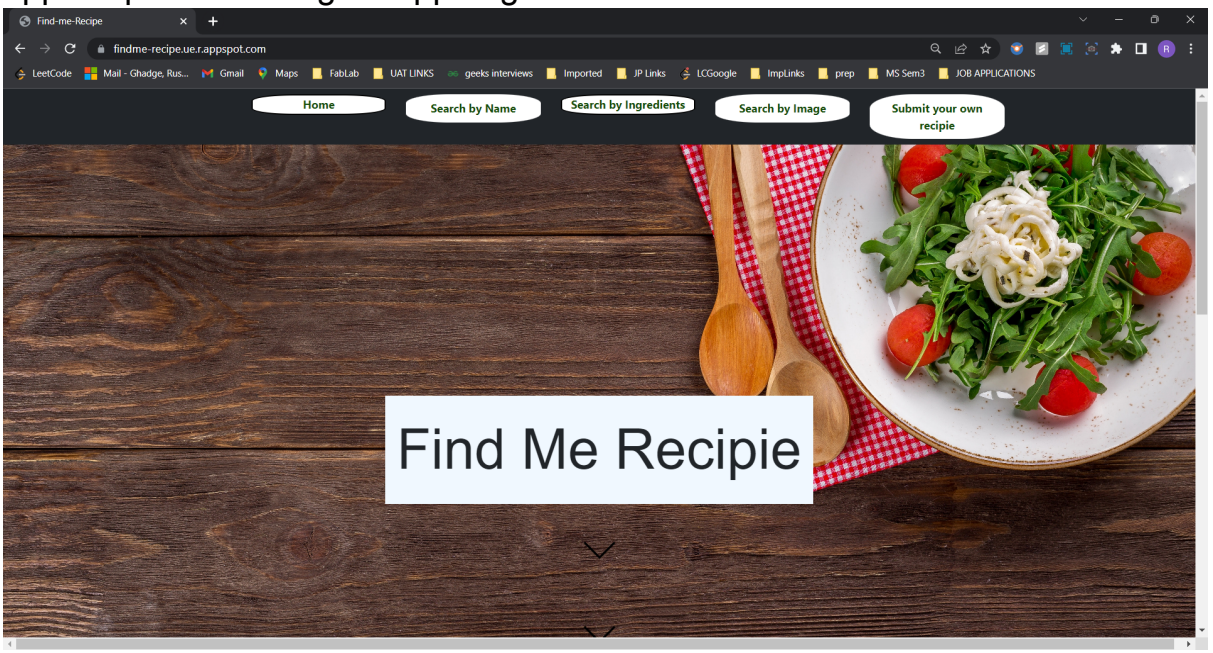
Create and setup app engine for project:

```
C:\Windows\system32>gcloud app create --project=findme-recipe
You are creating an app for project [findme-recipe].
WARNING: Creating an App Engine application for a project is irreversible and the region
cannot be changed. More information about regions is at
<https://cloud.google.com/appengine/docs/locations>.

Please choose the region where you want your App Engine application located:

 [1] asia-east1    (supports standard and flexible)
 [2] asia-east2    (supports standard and flexible and search_api)
 [3] asia-northeast1 (supports standard and flexible and search_api)
 [4] asia-northeast2 (supports standard and flexible and search_api)
 [5] asia-northeast3 (supports standard and flexible and search_api)
 [6] asia-south1   (supports standard and flexible and search_api)
 [7] asia-southeast1 (supports standard and flexible)
 [8] asia-southeast2 (supports standard and flexible and search_api)
 [9] australia-southeast1 (supports standard and flexible and search_api)
 [10] europe-central2 (supports standard and flexible)
 [11] europe-west   (supports standard and flexible and search_api)
 [12] europe-west2  (supports standard and flexible and search_api)
 [13] europe-west3  (supports standard and flexible and search_api)
 [14] europe-west6  (supports standard and flexible and search_api)
 [15] northamerica-northeast1 (supports standard and flexible and search_api)
 [16] southamerica-east1 (supports standard and flexible and search_api)
 [17] us-central    (supports standard and flexible and search_api)
 [18] us-east1      (supports standard and flexible and search_api)
 [19] us-east4      (supports standard and flexible and search_api)
 [20] us-west1      (supports standard and flexible)
 [21] us-west2      (supports standard and flexible and search_api)
 [22] us-west3      (supports standard and flexible and search_api)
 [23] us-west4      (supports standard and flexible and search_api)
 [24] cancel
Please enter your numeric choice:  18

Creating App Engine application in project [findme-recipe] and region [us-east1]....done.
Success! The app is now created. Please use `gcloud app deploy` to deploy your first app.
```

Deploy app:Go to path where we have yaml file in project and run below command to deploy app

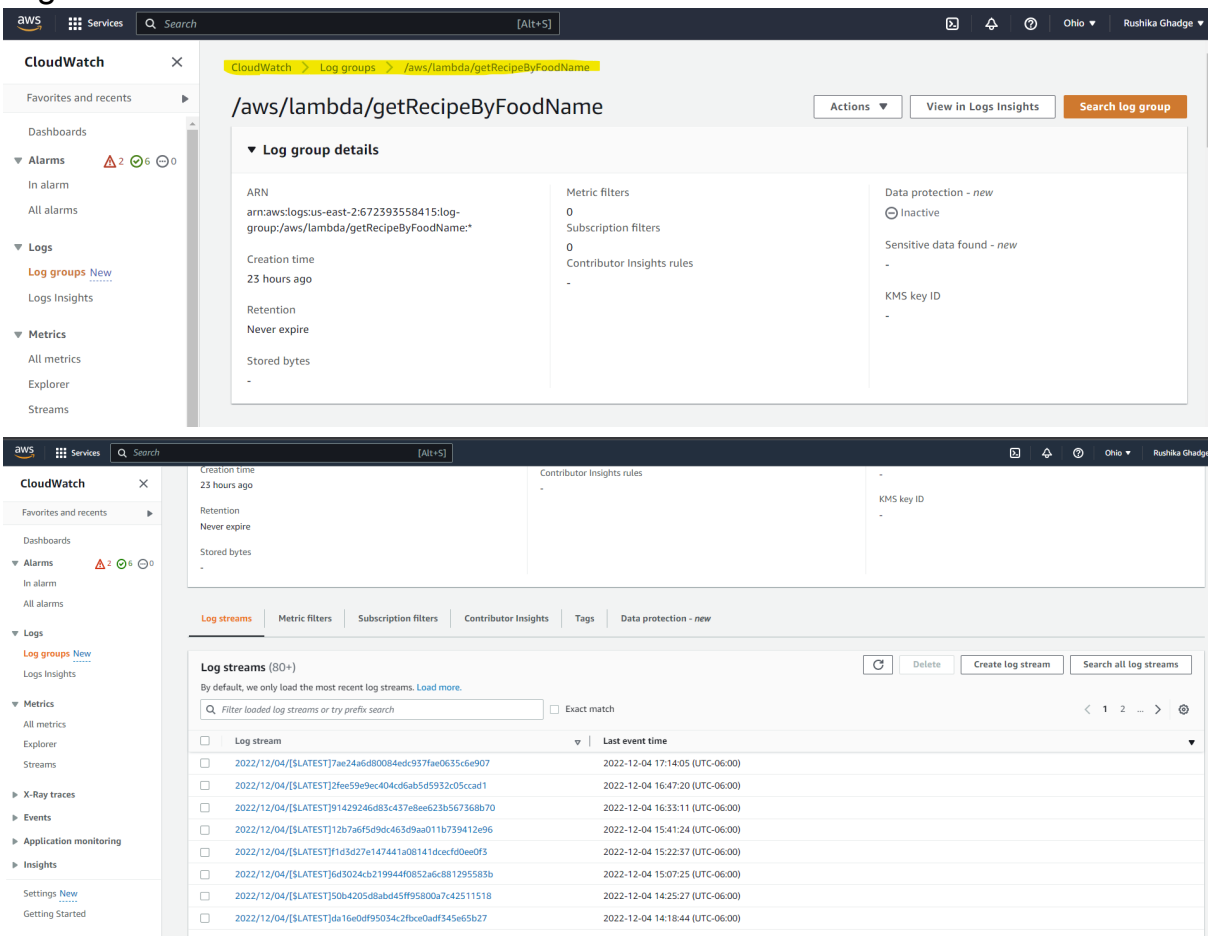
```
C:\Users\rushi\Downloads\Flsk-app\Flsk-app>gcloud app deploy
Services to deploy:

descriptor:                  [C:\Users\rushi\Downloads\Flsk-app\Flsk-app\app.yaml]
source:                      [C:\Users\rushi\Downloads\Flsk-app\Flsk-app]
target project:              [findme-recipe]
target service:              [default]
target version:              [20221204t132802]
target url:                  [https://findme-recipe.ue.r.appspot.com]
target service account:      [App Engine default service account]


Do you want to continue (Y/n)?  Y

Beginning deployment of service [default]...
Created .gcloudignore file. See `gcloud topic gcloudignore` for details.
#============================================================#
#= Uploading 17 files to Google Cloud Storage               =#
#============================================================#
File upload done.
Updating service [default]...done.
Setting traffic split for service [default]...done.
Deployed service [default] to [https://findme-recipe.ue.r.appspot.com]
```

App is up and running on app engine



Thus this app is a container which is executing on Google cloud infrastructure.

Logs can be traced from Cloudwatch:

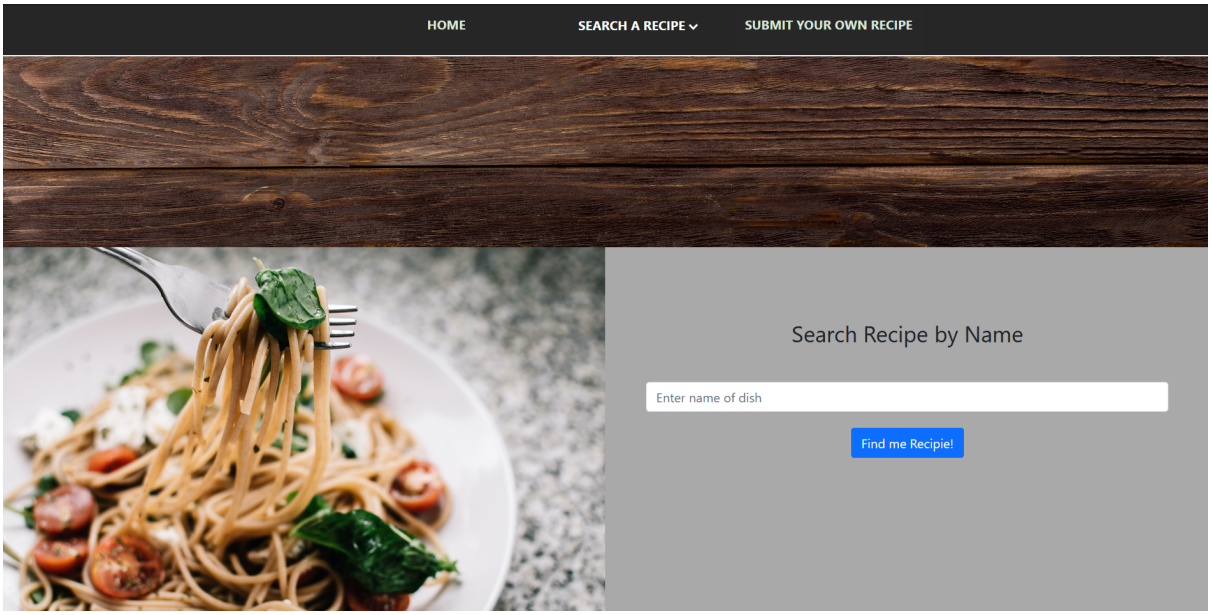


Application UI Screen Shots:

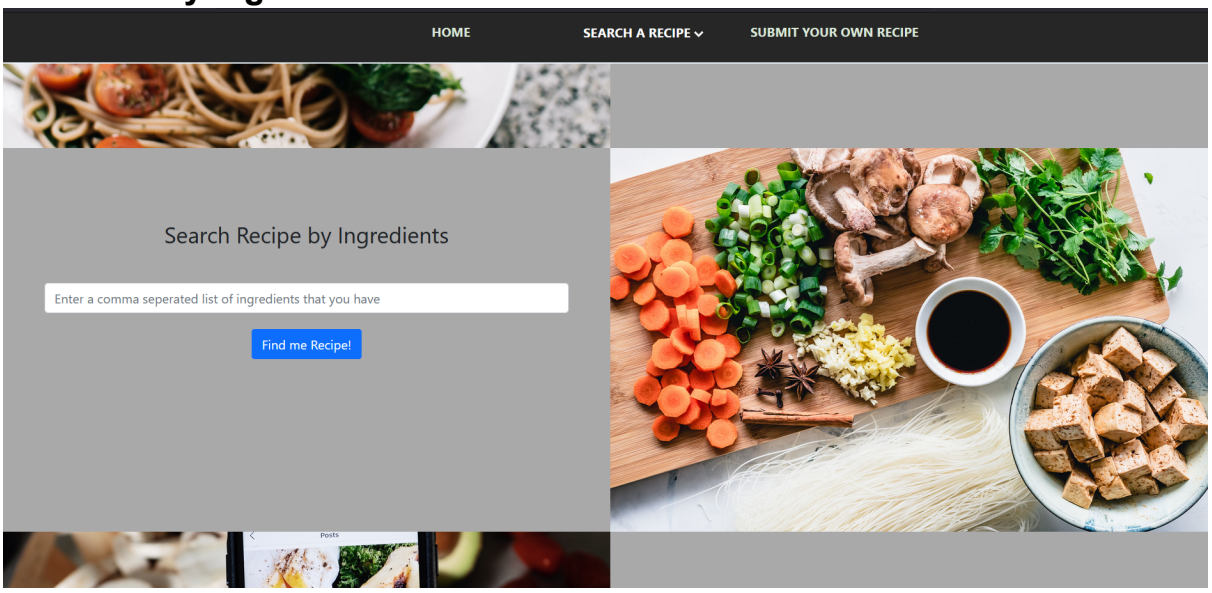
1. Home page: When the user opens this app, this is the home screen that he sees:

It shows a navigation bar with the functionalities that the web app provides.

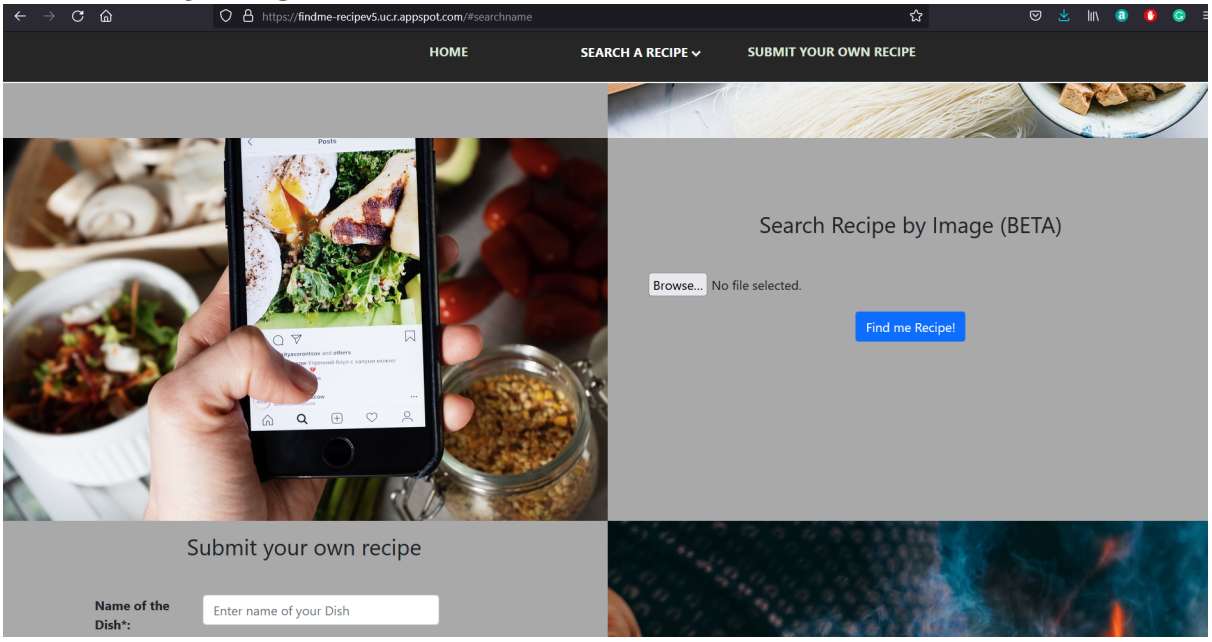
2. Search by name and Search by ingredients:



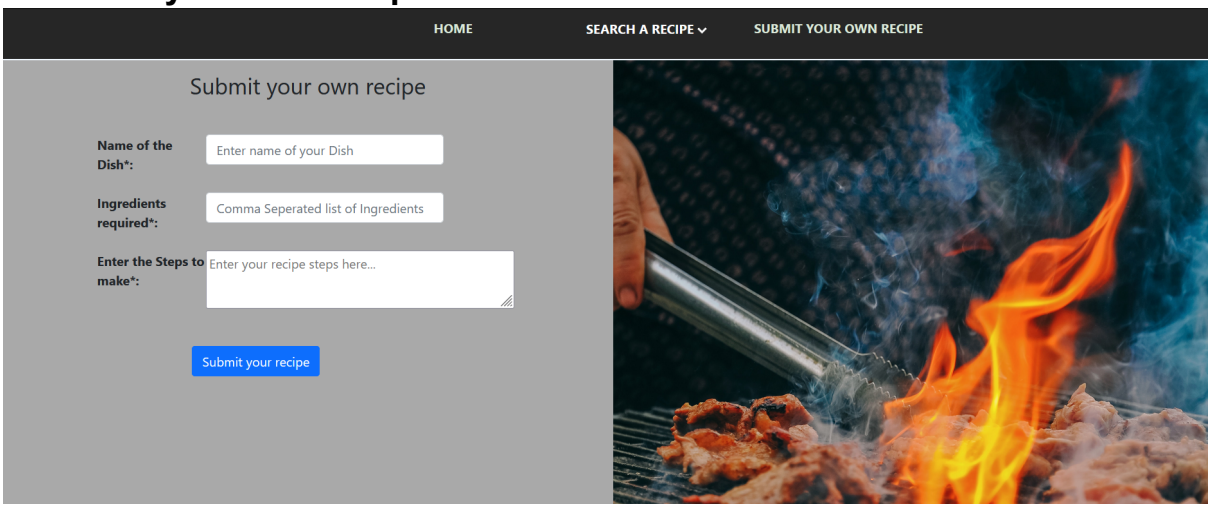
3. Search by Ingredients:



4. Search by Image:



5. Submit your own recipe:



6. **Results:** After running a query, the user gets results on this page: