

# 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.

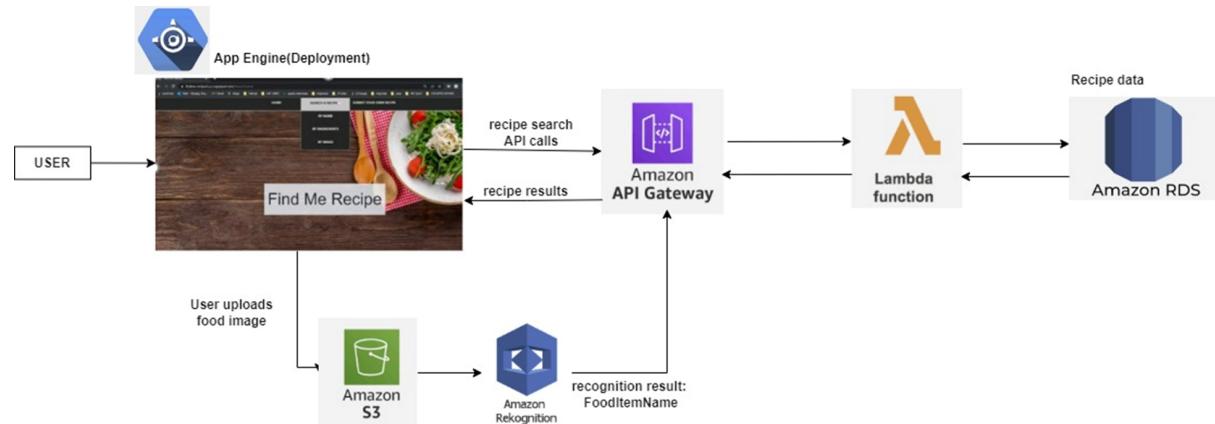
## Components used:

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:

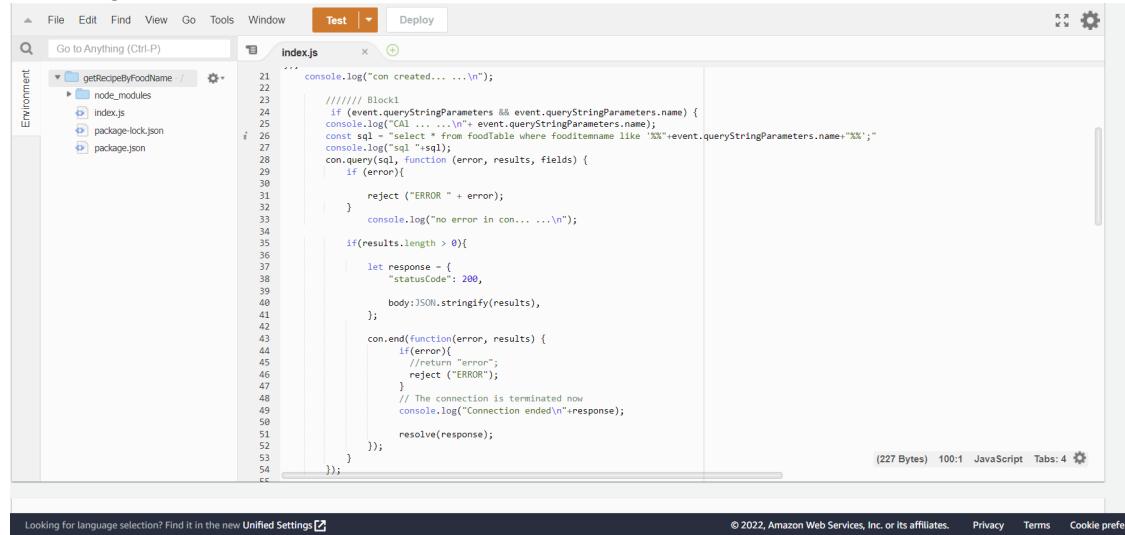
1. Configure IAM role and give permission for lambda execution

The screenshot shows the AWS IAM Roles page. A role named "mysql\_Lab" is selected. The "Permissions" tab is active, showing one managed policy attached: "AWSLambdaEdgeExecutionRole-19a3de53-d5e3-44a1-ab03-f056f22a8f6c" (Customer managed). Other tabs include "Trust relationships", "Tags", "Access Advisor", and "Revoke sessions".

- 2.Under permission configure role:

The screenshot shows the AWS Lambda Functions page. A function named "getRecipeByFoodName" is selected. The "Configuration" tab is active. On the left, a sidebar menu has "Permissions" highlighted. In the main panel, under the "Execution role" section, the "Role name" dropdown is set to "mysql\_Lab". Other tabs include "Code", "Test", "Monitor", "Aliases", and "Versions".

### 3. Configure source code for backend api



The screenshot shows the AWS Lambda function editor interface. The left sidebar displays the project structure with files: node\_modules, index.js, package-lock.json, and package.json. The main editor tab is titled 'index.js' and contains the following code:

```
21  ...
22  ...
23  ...
24  ...
25  ...
26  ...
27  ...
28  ...
29  ...
30  ...
31  ...
32  ...
33  ...
34  ...
35  ...
36  ...
37  ...
38  ...
39  ...
40  ...
41  ...
42  ...
43  ...
44  ...
45  ...
46  ...
47  ...
48  ...
49  ...
50  ...
51  ...
52  ...
53  ...
54  ...
55  ...

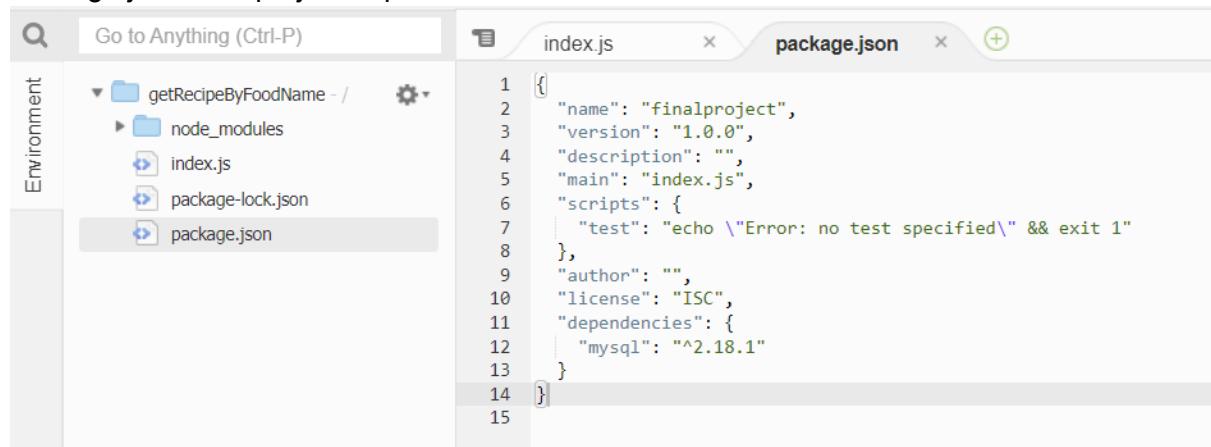
(227 Bytes) 100:1 JavaScript Tabs: 4
```

Below the editor, there is a footer bar with links: 'Looking for language selection? Find it in the new Unified Settings' (with a link icon), '© 2022, Amazon Web Services, Inc. or its affiliates.', 'Privacy', 'Terms', and 'Cookie preferences'.

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



The screenshot shows the AWS Lambda function editor interface. The left sidebar displays the project structure with files: node\_modules, index.js, package-lock.json, and package.json. The right editor tab is titled 'package.json' and contains the following JSON code:

```
1  {
2      "name": "finalproject",
3      "version": "1.0.0",
4      "description": "",
5      "main": "index.js",
6      "scripts": {
7          "test": "echo \\"Error: no test specified\\" && exit 1"
8      },
9      "author": "",
10     "license": "ISC",
11     "dependencies": {
12         "mysql": "^2.18.1"
13     }
14 }
```

## 4. Setup API Gateway using RESI API

For this project we created a REST API:

The screenshot shows the AWS API Gateway 'APIs' list. There are three APIs listed:

Name	Description	ID	Protocol	Endpoint type
getFoodItem		ukkvq64u78	REST	Regional
http-crud-fetch-recipe-api		4nwy9e8qj0	HTTP	Regional
LambdaSimpleProxy	LambdaSimpleProxy	yswql0o4gd	REST	Regional

Under API, created a resource named helloworld:

The screenshot shows the AWS API Gateway 'Resources' section for the 'LambdaSimpleProxy' API. A new resource named '/helloworld' has been created under the '/' path. The 'Actions' dropdown is open, and the 'Method' tab is selected, showing an 'ANY' method configuration.

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:

The screenshot shows the AWS API Gateway 'Method Execution' configuration for the '/helloworld' resource. It illustrates the flow from a 'Method Request' (Client side) through an 'Integration Request' (Lambda\_PROXY type) to a 'Method Response' (Client side). The 'Integration Request' panel shows the Lambda function ARN: arn:aws:execute-api:us-east-1:672393558415:yswql0o4gd/\*/\*helloworld.

## 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).

Name	Type	Last modified	Size	Storage class
icecream.jpg	jpg	December 4, 2022, 17:51:21 (UTC-06:00)	42.9 KB	Standard
istockphoto-1278029909-612x612.jpeg	jpeg	December 4, 2022, 16:57:16 (UTC-06:00)	31.0 KB	Standard
istockphoto-1302679757-612x612.jpeg	jpeg	December 4, 2022, 16:57:16 (UTC-06:00)	43.3 KB	Standard

```
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_access_key)

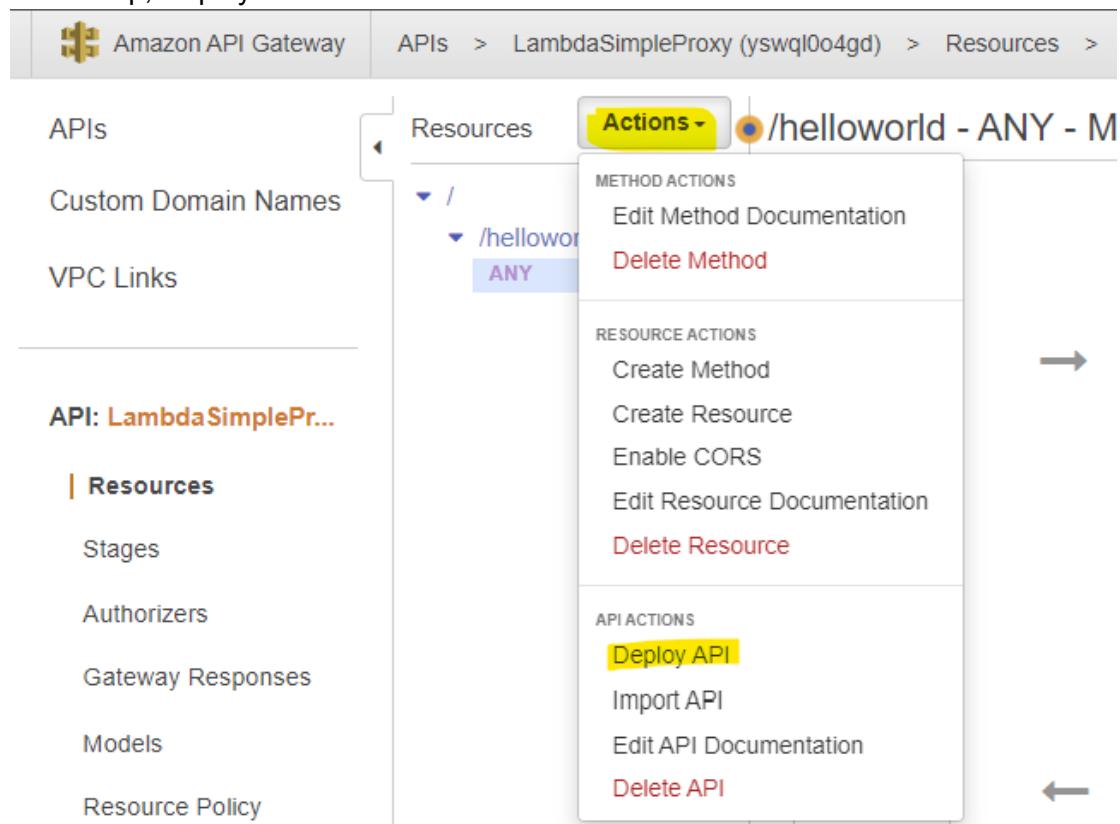
response= client.detect_labels(Image={
    'S3Object': {
        'Bucket': 'food-recok-buck',
        'Name': photo
    }},MaxLabels=10,MinConfidence=95)

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:

The screenshot shows the 'test' Stage Editor in the AWS API Gateway console. The 'Invoke URL' field at the top contains the value `https://yswql0o4gd.execute-api.us-east-1.amazonaws.com/test`. Below this, the 'Settings' tab is selected, showing configuration for 'Default Method Throttling' (Rate: 10000 requests per second, Burst: 5000 requests) and 'Web Application Firewall (WAF)' (None selected). Other tabs include Logs/Tracing, Stage Variables, SDK Generation, Export, Deployment History, Documentation History, and Canary.

## Testing URL Endpoint: Returns JSON

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

```
[{"foodid":1,"fooditemname":"Grilled Burger","ingredients":"Bread butter lettuce onion","instructions":"Heat the olive oil in a large pot over medium heat. Stir in the onion, and season with bay leaves, cumin, oregano, and salt. Cool and stir until onion is tender, then mix in the celery, green bell peppers, jalapeno peppers, garlic, and green chile peppers. When vegetables are heated through, mix in the vegetarian burger crumbles. Reduce heat to low, cover pot, and simmer 5 minutes.\nMix the tomatoes into the pot. Season chili with chili powder and pepper. Stir in the kidney beans, garbanzo beans, and black beans. Bring to a boil, reduce heat to low, and simmer 45 minutes. Stir in the corn, and continue cooking 5 minutes before serving."}, {"foodid":2,"fooditemname":"Chilli Burger","ingredients":"Bread, Wheat, Herbs, Garlic","instructions":"bla bla bla..."}, {"foodid":3,"fooditemname":"Chilli Burger","ingredients":"Bread, Wheat, Herbs, Garlic","instructions":"bla bla bla..."}, {"foodid":4,"fooditemname":"TypeA Burger","ingredients":"chicken, butter, salt","instructions":"bla bla bla..."}, {"foodid":5,"fooditemname":"TypeA Burger","ingredients":"chicken, butter, salt","instructions":"bla bla bla..."}]
```

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>

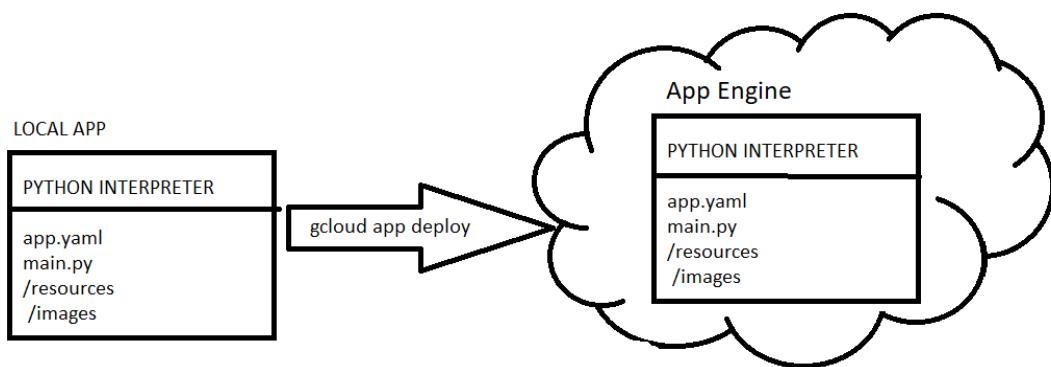
## Logs:

For logging all the logs are monitored using CloudWatch

The screenshot shows the CloudWatch Log Groups interface for the log group `/aws/lambda/getRecipeByFoodName`. The log group details include:

- ARN:** arn:aws:logs:us-east-2:672393558415:log-group:/aws/lambda/getRecipeByFoodName\*
- Metric filters:** 0
- Subscription filters:** 0
- Creation time:** 23 hours ago
- Retention:** Never expire
- Stored bytes:** -
- Data protection - new:** Inactive
- Sensitive data found - new:** -
- KMS key ID:** -

## 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

A screenshot of a code editor showing the "app.yaml" file. The file contains a single line: "runtime: python37".

```
! app.yaml
1   runtime: python37
```

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

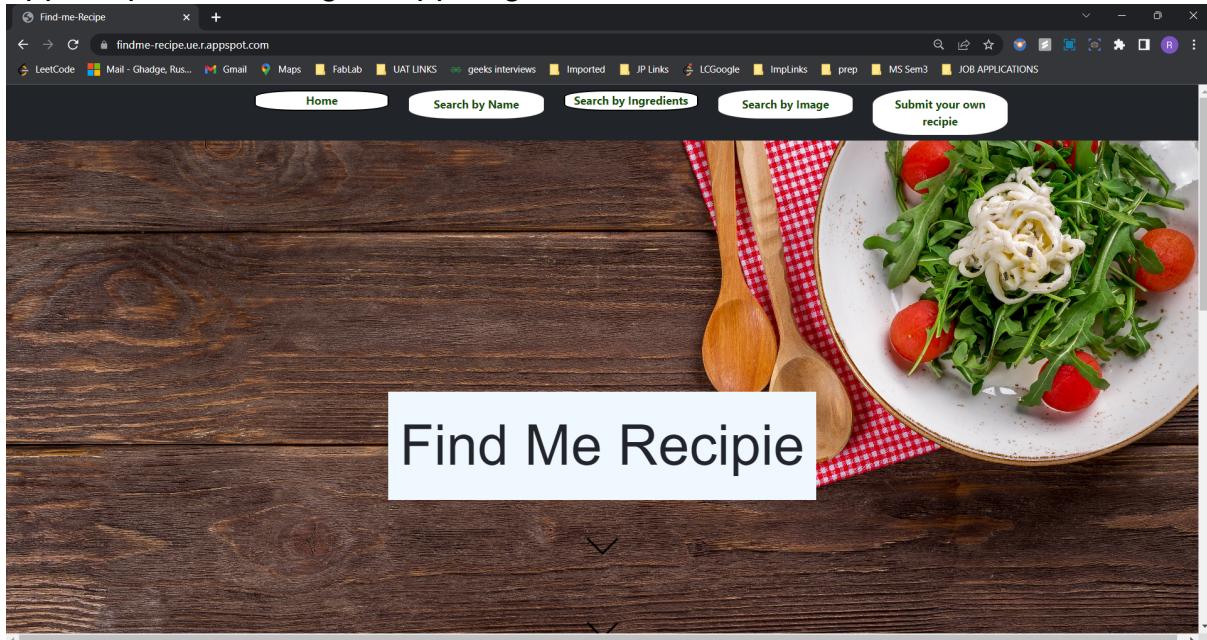
```
C:\Users\rushi\Downloads\Flsk-app\Flsk-app>gcloud app deploy
Services to deploy:

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:

The top screenshot shows the 'Log group details' for the log group '/aws/lambda/getRecipeByFoodName'. The log ARN is arn:aws:logs:us-east-2:672393558415:log-group:/aws/lambda/getRecipeByFoodName:. The creation time is 23 hours ago. The retention is Never expire. The stored bytes are 0. Metric filters, Subscription filters, and Contributor Insights rules are all 0. Data protection is set to 'new' with an inactive status. Sensitive data found and KMS key ID are both listed as '-'.

The bottom screenshot shows the 'Log streams' page. It lists several log streams with their creation times and last event times:

Log stream	Creation time	Last event time
2022/12/04/[\$LATEST]ae24a6d80084edc937fae0635c6e907	2022-12-04 17:14:05 (UTC-06:00)	
2022/12/04/[\$LATEST]2fee59e9ec404cd6ab5d5932c05ccad1	2022-12-04 16:47:20 (UTC-06:00)	
2022/12/04/[\$LATEST]91429246d83c437e8ee623b567368b70	2022-12-04 16:33:11 (UTC-06:00)	
2022/12/04/[\$LATEST]2b76f5d9dc463d9ea011b739412e96	2022-12-04 15:41:24 (UTC-06:00)	
2022/12/04/[\$LATEST]fd3d27e147441a08141dcecf0deef5	2022-12-04 15:22:37 (UTC-06:00)	
2022/12/04/[\$LATEST]6a5024cb19944f0852ae6881295583b	2022-12-04 15:07:25 (UTC-06:00)	
2022/12/04/[\$LATEST]50b4205d8abd45ff95800a7c42511518	2022-12-04 14:25:27 (UTC-06:00)	
2022/12/04/[\$LATEST]da16e0f95034c2fbce0ad345e65b27	2022-12-04 14:18:44 (UTC-06:00)	

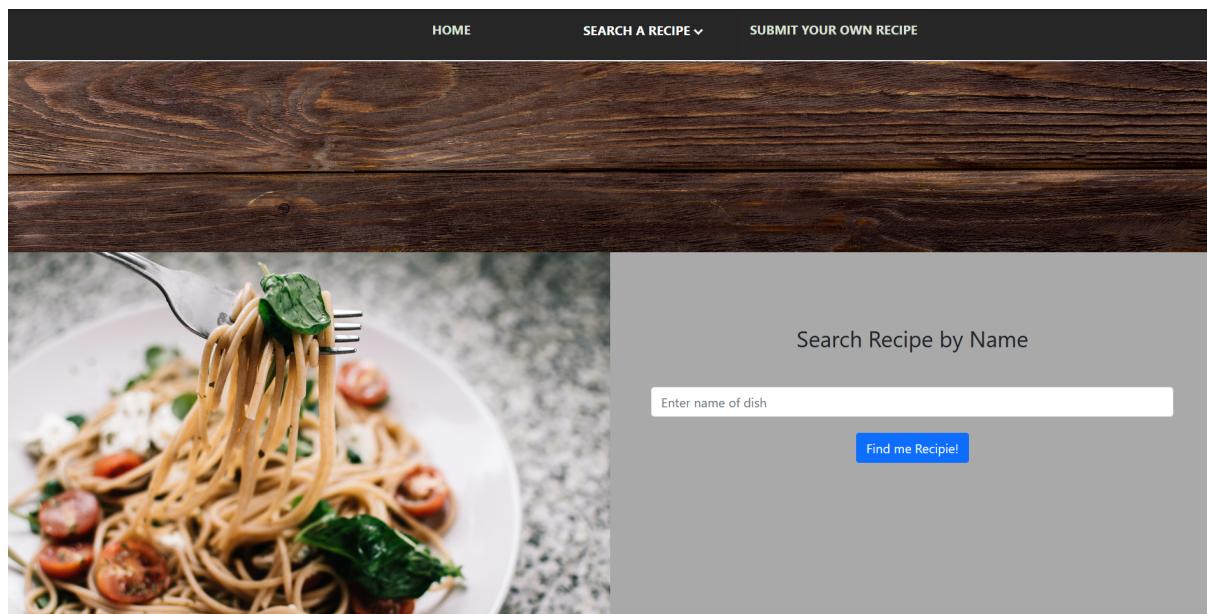
## Application UI Screen Shots:

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

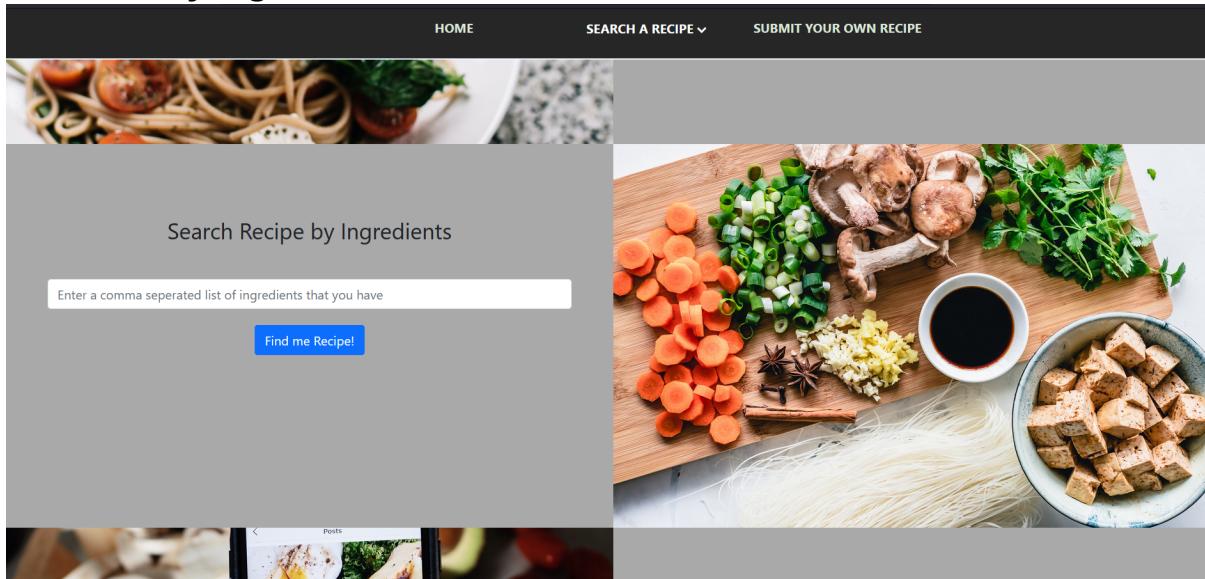


It shows a navigation bar with the functionalities that the web app provides.

2. Search by name and Search by ingredients:

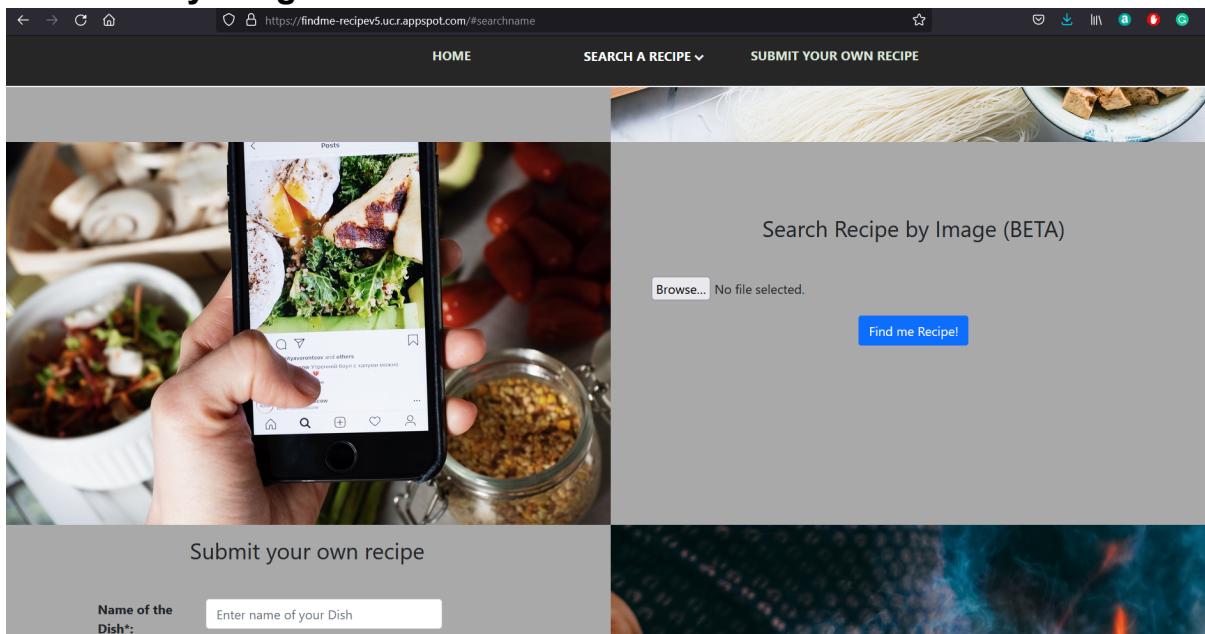


### 3. Search by Ingredients:



The interface for searching by ingredients features a top navigation bar with links for "HOME", "SEARCH A RECIPE", and "SUBMIT YOUR OWN RECIPE". Below the navigation is a large image of a dish. To the left of the image is a search form with the placeholder "Enter a comma seperated list of ingredients that you have" and a blue "Find me Recipe!" button. To the right of the image is another image showing various ingredients like carrots, mushrooms, and tofu.

### 4. Search by Image:



The interface for searching by image includes a browser header with the URL "https://findme-recipev5.ucr.appspot.com/#searchname". The top navigation bar is identical to the previous section. On the left, there's a "Submit your own recipe" section featuring a smartphone displaying a food photo from Instagram and a field for entering the dish name. On the right, there's a "Search Recipe by Image (BETA)" section with a file input field ("Browse... No file selected."), a "Find me Recipe!" button, and a small image of a dish.

## 5. Submit your own recipe:

Submit your own recipe

Name of the Dish\*:

Ingredients required\*:

Enter the Steps to make\*:



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

Found The Following Recipes

**1. Bulgur Veggie Burgers with Lime Mayonnaise**

**Ingredients:** 1/2 cup chopped onion, divided,1 tablespoon olive oil plus additional for brushing,1/2 cup bulgur,1 cup water,1 cup canned pinto beans, rinsed and drained,1 1/2 tablespoon soy sauce,3/4 cup walnuts (2 1/2 ounces),2 garlic cloves, coarsely chopped,1/2 cup packed cilantro sprigs,3/4 teaspoon ground cumin,1/4 teaspoon cayenne,1/4 cup mayonnaise,1/4 teaspoon grated lime zest,1/2 teaspoon fresh lime juice,4 slices multi-grain bread,toasted,Equipment: a perforated grill sheet,Accompaniments: lettuce

**Instructions:** Cook half of onion with 1/4 teaspoon salt in oil in a small heavy saucepan over medium heat, stirring occasionally, until golden, 5 to 7 minutes. Add bulgur and water and cook, covered, over low heat until water is absorbed, 15 to 18 minutes. Transfer to a bowl and stir in beans and soy sauce. Pulse bulgur mixture, walnuts, garlic, cilantro, cumin, cayenne, a rounded 1/4 teaspoon salt, 1/2 teaspoon pepper, and remaining onion in a food processor until finely chopped. Form rounded 1/2 cups of mixture into 4 (3 1/2-inch-diameter) patties. Chill at least 10 minutes. While patties chill, stir together mayonnaise, zest, and juice. Prepare grill for direct-heat cooking over medium-hot charcoal (medium heat for gas). Put perforated grill sheet on grill and preheat 10 minutes. Brush patties all over with oil. Oil grill sheet, then grill burgers on grill sheet, covered only if using a gas grill,

**2. Tex-Mex Veggie Burgers**

**Ingredients:** 1 15 1/4-ounce can whole kernel corn%2C drained%2C 1/2 cup liquid reserved%2C1/2 cup plus 1 tablespoon cornmeal%2C1/2 cup finely chopped onion%2C1/3 cup finely chopped red bell pepper%2C1/2 teaspoon grated lime peel%2C1/4 cup cooked

